Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Comments on Notation used in this text

I will use subscripts to indicate the dimensionality of matrices. This facilitates the conversion of a theoretical equation into code, since the dimensionality and indexing of variables can be easily visualized. It also helps to visualize and confirm the steps within a derivation.

For example, if I have an array, X, of n rows and m columns I can write it as

$$X_{n,m} = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,m} \\ x_{2,1} & x_{2,2} & \dots & x_{2,m} \\ \dots & \dots & \dots & \dots \\ x_{n,1} & x_{n,2} & \dots & x_{n,m} \end{pmatrix} \tag{1.1}$$

I will write the transpose of $X_{n,m}$ with the indices reversed, $X^T_{m,n}$ so that it is easy to see the rank of the resulting matrix.

$$C_{n,n} = X_{n,m} \cdot X^T_{m,n} \tag{1.2}$$

When using an array lanquage, such as MATLAB, IDL, or FORTRAN-90 one must make sure that an array variable within a lanquage, x(n,m), is ordered properly. For example, FORTRAN physically arranges the variables in memeory with the first index representing columns. Therefore, when encountering a new lanquage it is best to test the order of matrix multiplication.

1.2 What is Computational Physics?

Computation physics is also known as "numerical analysis" and "scientific computing." The salient features that distinguish computational physics from computer science or mathematical physics are:

- The problem requires in depth knowledge of the scientific discipline (*e.g.*, Physics, Astronomy, etc.)
- The problem has significant complexity and scope. The problem may be intractable by conventional means or may require approximation to solve.
- The problem has a precise mathematical statement, and quantities can be a combination of continuous functions (*e.g.*, theoretical), discrete functions (*e.g.*, modeled), or measured data. Modeled functions and measured data include uncertainties due to noise and computational errors.

When reviewing the various methods you will typically see many names of great mathematicians and physicists. This is because, they also needed computational methods to make their computations tractable on the machines of their day (*e.g.*, abacus, slide rule (invented 1621 by William Oughtred (1574-1660), simple adding machines). Many of the same issues are relevant today. There are methods by Gregory & Newton (1642-1727), Gauss (1777-1855), Sterling, Bessel (1784-1846), Everett, and Lagrange (1736-1813), just to name a few.
Some of the problems that utilize computational physics are

- human genome mapping
- climate or weather modeling, modeling of biophysical processes (*e.g.*, photo-synthesis)
- simulation of galactic structure, supernovas, stellar interiors, etc.
- simulation of material properties
- modeling human dimensions (*e.g.*, traffic simulations)
- calibration and/or modeling of instrumental measurements.
- analysis of models or measurements.

This class is designed to instill good programming habits and to build a foundation for your future computational work. While the physical and mathematical concepts in this class will tend to be simple, you will find that the time it takes to make these methods work can be exhaustive. Therefore, you should avoid procrastinating on the homework and projects. If you can build a hierarchy of tools then the time it takes to use those tools will reduce significantly.

I will use many examples encountered from my own work. In some cases this may be frustrating, since there may not be a correct "answer." We will employ real data files which will include instrumental measurement noise. I hope to emulate how science is really done within this class to give you some exposure to the "real" world and to help build some good habits for your future career.

I assume that Physics students have a strong bias towards understanding of a computer from the inside out. To envision how the computer "thinks" is critical to writing code that is efficient, robust, and will reasonably emulate the physical problem to be solved. I would hope that at the end of this class you would see the computer for what it is: a versatile tool for visualization of physical concepts, a test-bed for theoretical constructs of the physical world, and a robust calculator with a tremendous memory.

Two references of the use of computers in the classroom are:

- Luehrmann, A.W. 1974. Orbits in the solar wind - a mini-research problem. Amer. J. Phys. v.42 p.361-371.
- Merrill, J.R. and R.A. Morrow 1979. An introductory scattering experiment by simulation. Amer. J. Phys. v.38 p.1104-1107.

1.3 Recommended Syllabus

Table 1.1: Syllabus for PHYS 640, Spring 2002, pg. 1/2

	day	Date	Topic(s)	Project Milestones	Homework Due Date
1	Tues	Jan. 29	Introduction, Requirements		
2	Thur	Jan. 31	Computer Architecture - registers, clock, CPU, Bus, I/O - Example Processors - Cache, Optimization		
3	Tues	Feb. 5	Representation, Error & Accuracy - integer, real, character - random numbers		HW #1(FTP)
4	Thur	Feb. 7	underflow & overflow - error propagation		
5	Tues	Feb. 12	File I/O - ASCII vs BINARY		
6	Thur	Feb. 14	Discretization and Sampling (DS) - Nyquist criterion		HW #2 (B_ν)
7	Tues	Feb. 19	Interpolation methods (IM) - piecewise linear, 1d & 2d - polynomial & spline	ideas due	
8	Thur	Feb. 21	- polynomial & spline - rational functions		
			Differentiation (DI) - finite differences - Laplacian		
9	Tues	Feb. 26	Digital Image Processing (IP)		
10	Thur	Feb. 28	Quadrature Methods (QU) - midpoint & trapezoidal rules - Simpsons		HW #3 (FP)
11	Tues	Mar. 5	QU: Gaussian Quadrature	2pg summary	HW #4 (spline)
12	Thur	Mar 7	Monte Carlo Methods (MC) - Probability Distributions - Definite Integrals		HW #5 (DS,IM)
13	Tues	Mar 12	Monte Carlo (continued) - Random Walk Processes - Simulated Annealing		HW #6 (DI,QU)

Table 1.2: Syllabus for PHYS 640, Spring 2002, pg. 2/2

	day	Date	Topic(s)	Project Milestones	Homework Due Date
14	Thur	Mar 14	Ordinary Diff. Eqn's (DE) - Runge-Kutta		
15	Tues	Mar 19	Partial Diff. Eqn's (DE) - Relaxation Methods - Shallow Water Model		
16	Thur	Mar. 21	Fourier Series (FS) - Harmonic Analysis		
	Tues	Mar. 26	n/a (spring break)		
	Thur	Mar. 28	n/a (spring break)		
17	Tues	Apr. 2	Discrete Fourier Transforms (DFT) - Sampling & Aliasing		HW #7 (DE)
18	Thur	Apr. 4	Fast Fourier Transforms (FFT) - Decimation - Performing FFT in hardware		
19	Tues	Apr. 9	FFT Symmetry - Composite FFT - "Real" FFT's		
20	Thur	Apr. 11	FFT Applications - De-convolution - Interferometers - Mass Spectrometers		
21	Tues	Apr. 16	Apodization (AP)	progress report	
22	Thur	Apr. 18	Linear Least Squares (LSQ) - Weighting & Binning		HW #8 (DFT)
23	**Fri**	Apr. 19	LSQ error estimation		
24	Thur	Apr. 25	Linear Regression systems		
25	Tues	Apr. 30	Finding Roots, Optimization		HW #9 (AP)
26	Thur	May 2	Non-linear least squares (NL) - Marquart-Levenberg - Linear Constrained		
27	Tues	May 7	Eigenvector Decomposition (EVD) - Information content analysis - Non-Linear "recipe"		HW #10 (LSQ)
28	Thur	May 9	EOF's & SVD	**final report**	
29	Tues	May. 14	Atmospheric Retrieval Methods - Minimium Variance - Maximum Entropy - Optimal Estimation		HW #11 (NL)
30	Tues	May. 21	**presentations**	**12 min. talk**	
	Fri	May. 24	Grades are Due		

1.4 Some Examples of numerical algorithms

1.4.1 An algorithm to evaluate square roots

Suppose we want to find y where $y^2 = x$, that is we want to solve $f(y) = 0$ as given by

$$f(y) = y^2 - x = 0 \tag{1.3}$$

which has the analytic derivative equal to

$$f'(y) = 2y \tag{1.4}$$

One method to compute the square root is Newton-Raphson iteration (Section 14.1.3). We begin with an initial value, y_0, we can compute an approximation for the derivative about the initial value and a perturbaton. We will write the derivative so that $f(y)$ goes to zero. In general, for any iteration n this can be written

$$f'(y) \simeq \frac{f(y_n) - f(y_{n+1})}{y_n - y_{n+1}} = \frac{f(y_n) - 0}{y_n - y_{n+1}} = \frac{f(y_n)}{y_n - y_{n+1}} \tag{1.5}$$

Equating Eqn. 1.4 and Eqn. 1.5 and solving for y_{n+1} will yield

$$y_{n+1} \simeq y_n - \frac{f(y_n)}{2y_n} = \frac{1}{2} \cdot \left(y_n + \frac{x}{y_n} \right) \tag{1.6}$$

The ability to perform the square-root efficiently depends on having a reasonable first guess, y_0. For example, let's say that $x = 16$ and we start with an initial guess of $y_0 = 1$. The results of the iterations are shown in Table 1.3 and illustrated in Fig. 1.1.

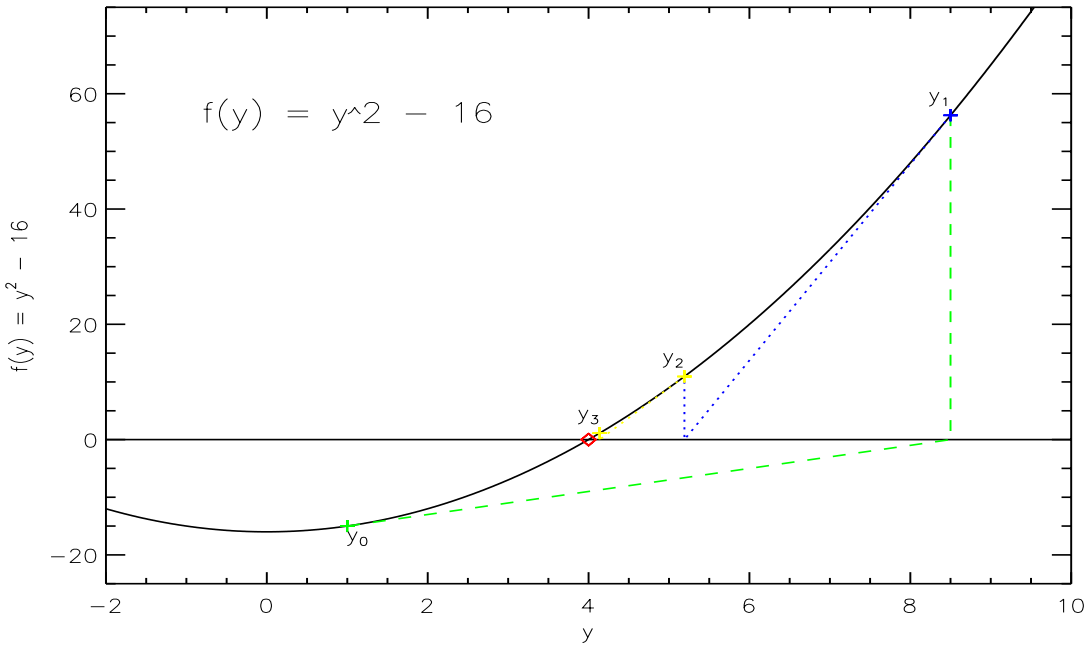


Figure 1.1: Illustration of Newton-Raphson iteration with the square root function

Notice that the method improves the estimate by a couple of decimal places in each iteration if the estimate is close to the correct answer. In Section 14.1.4 the implementation of this in machine code is

Table 1.3: Example of Newton-Raphson iteration of the square root function

y_n	$f(y_n)$	y_{n+1}	$f(y_{n+1})$
1.0	-15.0	$\frac{1}{2}(1+16/1) = 8.5$	56.25
8.5	56.25	$\frac{1}{2}(8.5+16/8.5) = 5.19$	10.95
5.19	10.95	$\frac{1}{2}(5.19+16/8.5) = 4.14$	1.11
4.14	1.11	$\frac{1}{2}(4.14+16/4.14) = 4.002$	0.02
4.002	0.02	$\frac{1}{2}(4.002+16/4.002) = 4.0000005$	$0.5 \cdot 10^{-6}$

discussed using a computing machines native floating point format. This algorithm is the core algorithm for most modern floating point packages and can compute the desired accuracy in a small number of iterations.

1.4.2 Elliptic Integral's of the 2nd kind

This section taken from Burden and Faires 1985, pg. 136. Elliptic integral's of the 2nd kind are very common in many science and engineering applications. For example, suppose we want to compute the amount of metal necessary to construct a specific length, L, of corrugated roofing. In our example, let's say the amount of roofing desired, L, is 4 feet and the amplitude of the corrugation (a sine wave) is 1 inch, $A = 1$, with a 6 inch period, $P = 6$. The cross-section of the corrugated metal can be described by

$$f(x) = 1 \cdot \sin\left(2\pi \cdot \frac{x}{6}\right) \tag{1.7}$$

The amount of flat metal needed, S, is given by the integral of the desired surface of the horizonal length of the corrugated metal, L, of the

$$S = \int_0^L \sqrt{1 + (f'(x))^2} \cdot dx \tag{1.8}$$

$$= \int_0^{48} \sqrt{1 + \left(\frac{A \cdot P}{2\pi} \cos\left(2\pi \cdot \frac{x}{P}\right)\right)^2} \cdot dx \tag{1.9}$$

$$\tag{1.10}$$

In general, this integral is called an elliptic integral of the 2nd kind and can be written

$$E(k, \phi) = \int_0^L \sqrt{1 + (k^2 \cos^2(\phi))} \cdot dx \tag{1.11}$$

It is usually presented in tabular form (*e.g.*, CRC Standard Mathematical Tables, pg. 408). If we wanted to write a program to solve this problem we would have to

1. read the table into the program and then write a program to interpolate the table (see section 4.3)
2. compute the integral directly using numeric methods (see section 5)

1.5 Overview of Computer Hardware

Computers are devices that can decode instructions and perform operations on data. In general, the architecture of most computed systems can be described in terms of a set of interconnections canned buses. In Fig. 1.2 we show an example of a computed with the address bus, data bus, and control logic bus. All computer logic is build upon the following basic elements.

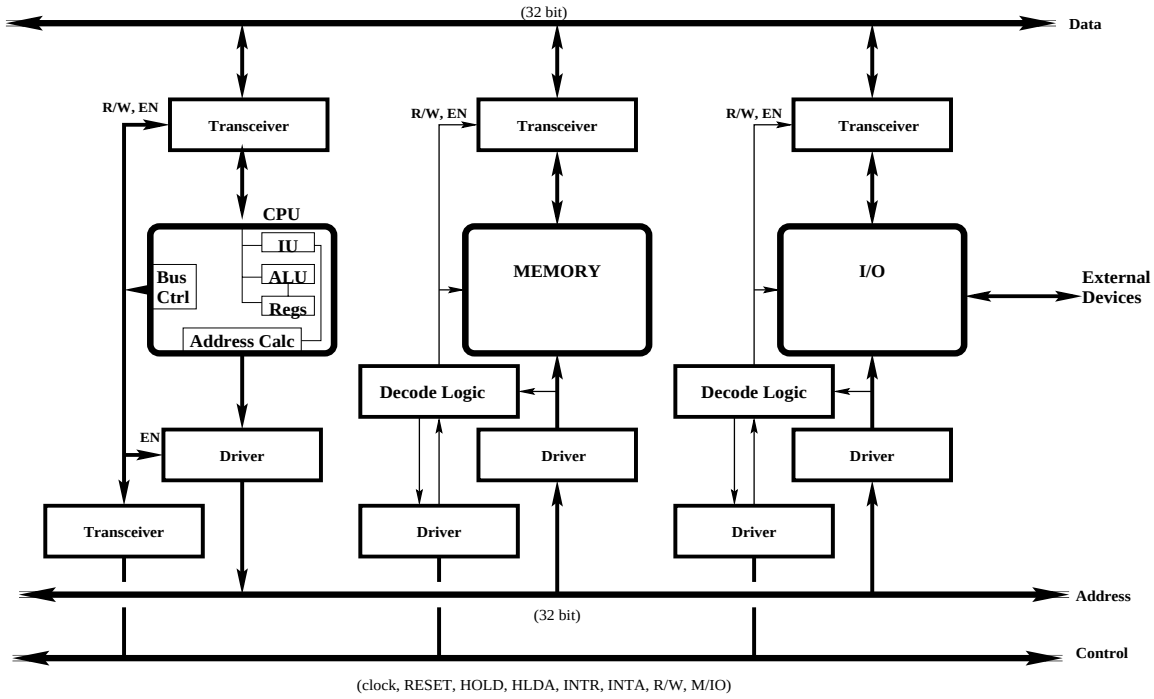


Figure 1.2: Illustration of a computer bus

1.5.1 Inverter: logical NOT

Fig. 1.3 is taken from "The TTL Data Book for Design Engineers, 2^{nd} Edition, Texas Instruments, Inc., 1976, pg. 5-7 and shows the circuitry of a logical NOT operator.

in	out
1	0
0	1

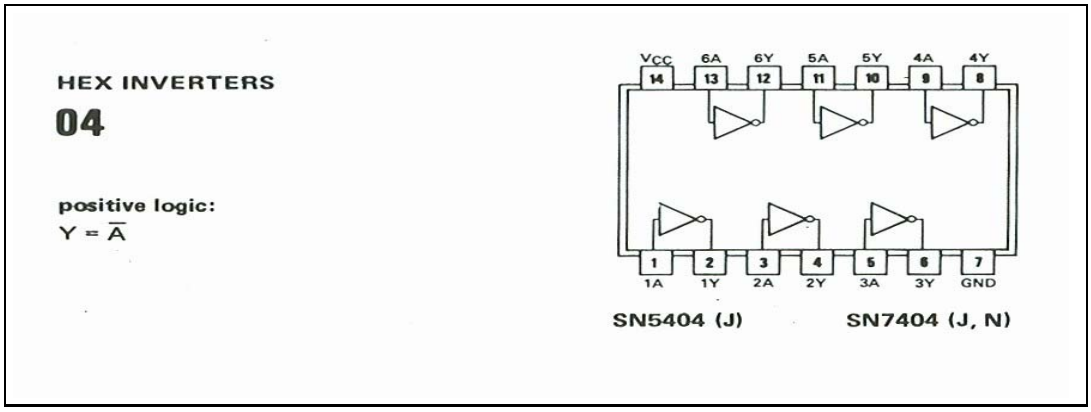


Figure 1.3: Logical NOT circuit

1.5.2 NAND gate: inverted logical AND

A logical AND circuit is shown in Fig. 1.4, taken from "The TTL Data Book for Design Engineers, 2^{nd} Edition, Texas Instruments, Inc., 1976, pg. 5-11 and 6-4.

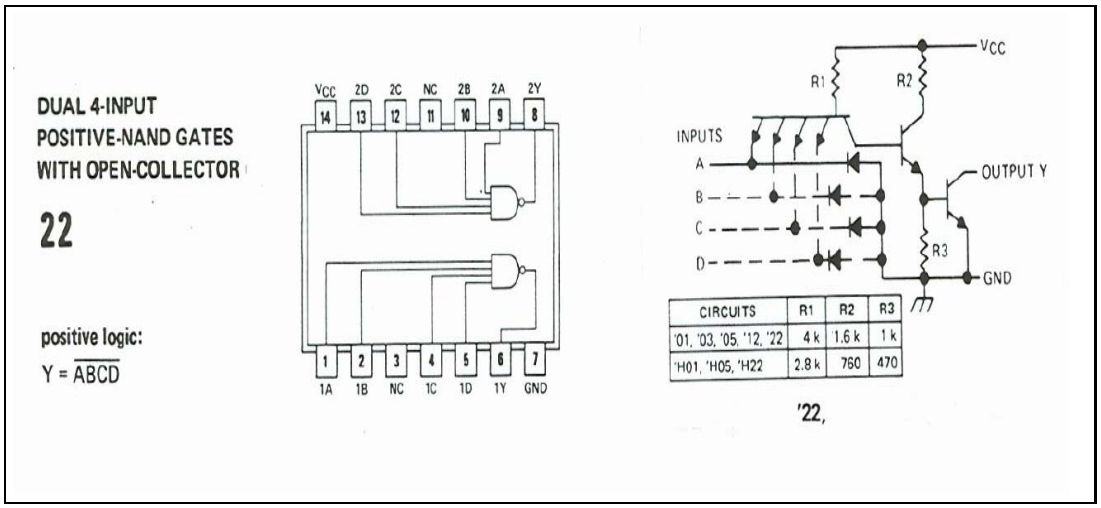


Figure 1.4: Logical AND circuit

a	b	out
0	0	1
0	1	1
1	0	1
1	1	0

1.5.3 NOR gate: inverted logical OR

A logical OR circuit is shown in Fig. 1.5, taken from "The TTL Data Book for Design Engineers, 2^{nd} Edition, Texas Instruments, Inc., 1976, pg. 5-12 and 6-8.

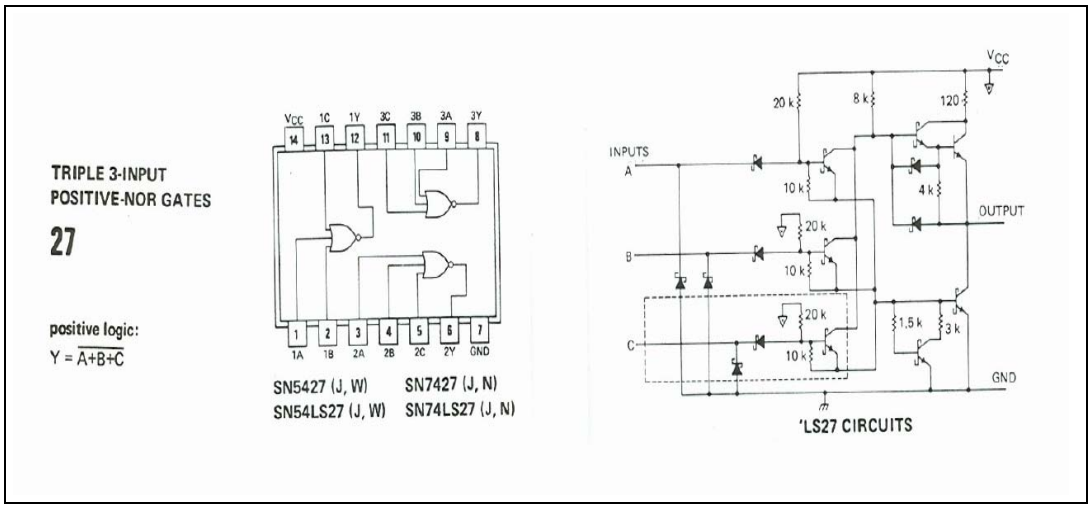


Figure 1.5: Logical OR circuit

a	b	out
0	0	1
0	1	0
1	0	0
1	1	0

1.5.4 bi-stable latch: memory & syncronization

A bi-stable latch is shown in Fig. 1.6, taken from "The TTL Data Book for Design Engineers, 2^{nd} Edition, Texas Instruments, Inc., 1976, pg. 6-48.

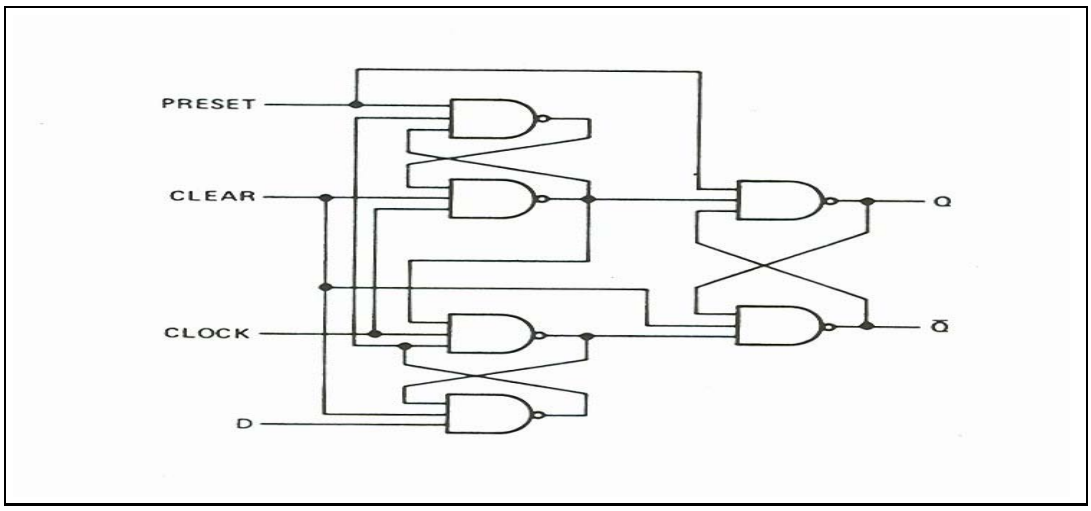


Figure 1.6: Bi-stable latch circuit

in	clk	Q	$\overline{Q}$
0	↑	0	1
1	↑	1	0

1.5.5 Address Decoder

An address decoded circuit is shown in Fig. 1.7, taken from "The TTL Data Book for Designa Engineers, 2^{nd} Edition, Texas Instruments, Inc., 1976, pg. 7-134,135.
Example of an address decoder chip (S138). This circuit will select 1 of 8 banks of memory from 3-bits of the address.

1.5.6 Bus Transceiver

A bus driver circuit is shown in Fig. 1.8, taken from "The TTL Data Book for Design Engineers, 2^{nd} Edition, Texas Instruments, Inc., 1976, pg. 7-349.
Example of a bus transceiver chip (S245). If ENABLE is 5V (H) then the chip effectively does not exist. If ENABLE is set to 0V then the chip passes data from the A $\rightarrow$ B side (DIR = 5V) or from B $\rightarrow$ A (DIR = 0V).

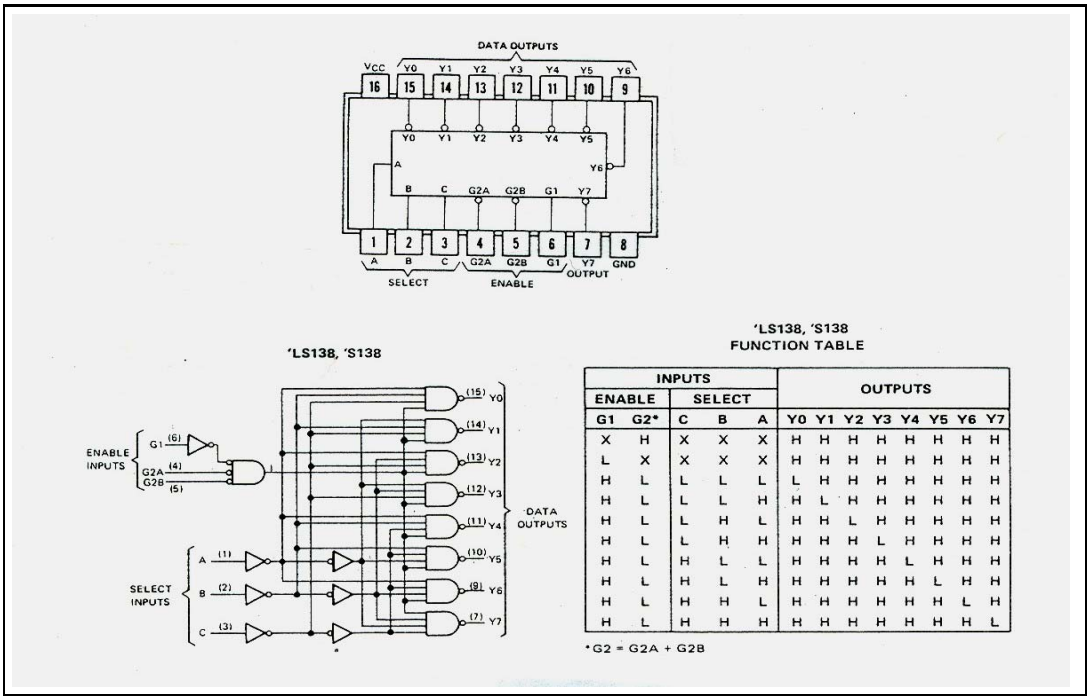


Figure 1.7: Address decoder circuit

1.6 Components of a computer

- Clock: All processes are **synchronized** by this clock. Address, Data, and Control are presented to the bus and on next clock cycle the event is **registered**. Thus an electronic delay and settling time is allowed.
- The **control**, **data**, and **address** bus is a set of wires to transmit information. The control signals determine which device transmits and which device(s) receive the information.
- Processor(s) are in control of the bus. One processor controls the address register and presents control information to the bus such as whether this is a memory or I/O cycle (M/IO) and whether it is a read or write cycle (R/W). The memory or I/O devices on the bus can respond to the CPU with a BUSY signal (to slow it down), or present data to the bus, or acquire data presented by other sources. Processors can issue a HOLD operation to other processors and those processors decouple themselves from the BUS and issue a HOLD acknowledge signal (HLDA) signal.
  - Bus Logic (R/W, M/IO). The processors "registers" signals on rising or falling edges of the clock.
  - Address Unit (w/registers): Addressing modes can be quite complex. There is direct, indirect, indirect with offset, indirect with register offset, etc. The INTEL family has a complicated scheme to segment memory into 64K chunks, which makes the AU a more complicated piece of silicon.
  - Arithmetic Logic Unit: Add, Shift, Logical (*e.g.*, XOR, AND, OR, NEG).
  - Status register(s) is an internal register used mainly for testing the output of the ALU for branching (*e.g.*, CARRY, EQ, GT, LT)
  - Instruction Unit (IU) and microcode memory: Microcode is a "language" stored in memory on-board the processor to perform the processor's instructions. It is a hardware interface to the processor's "*assembler*" language. For example, to perform an assembler "MOV CX, @mem" command (move register CX to location at the address stored in memory location mem), the CPU

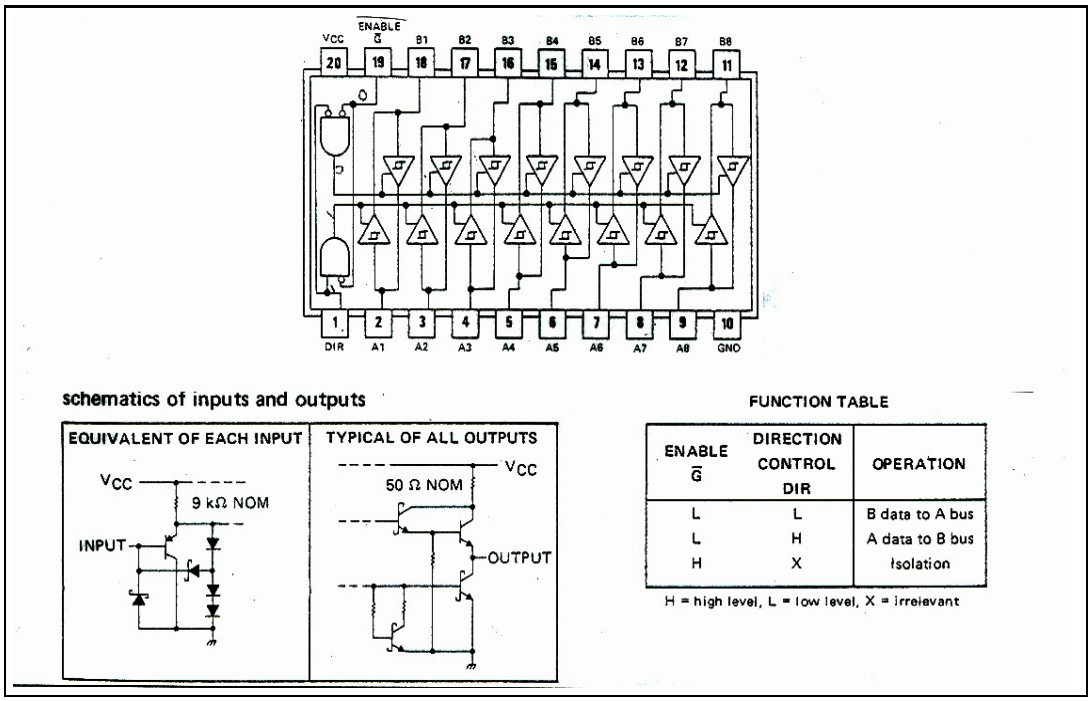


Figure 1.8: BUS driver circuit

must enable the data bus for a read operation, present the address to the address bus, configure its internal bus to select register CX and then register the data with a strobe. Many microcode operations are done together so that this entire process takes place within 2 clock cycles. If the memory issues a WAIT via the control bus, then the microcode will loop or be halted. The more complex commands (*e.g.*, multiply and divide) can use many lines of microcode (in older chips); however, more recent chips have hardware multipliers in their ALU or as co-processor chips (co-processors are external to the CPU but are physically and electronically coupled so that I/O and control is much faster.

A schematic of the internal structure of a 80286 chip is shown in Fig. 1.9. (taken from the Advanced Micro Devices 80286 data sheet, Jan. 1985, pg. 1).

Example of '8086 CPU assembler instructions	
Data transfer	MOV
bit manipulation	SHL, SHR, AND, OR, XOR, NOT (1's complement)
arithmetic	ADD, SUB, MUL, DIV, NEG (2's complement), CMP
loops and jumps	JMP, JEQ, ...
subroutine & interrupt	CALL, RET, POP, PUSH

  - interrupt and external control
    - reset (RESET)
    - exception handling
    - critical event timing (NMI, INTR, INTA)
    - other processors, disk controllers, etc. (HOLD, HLDA)
  - address and/or data cache

- Memory determines when it is accessed via the address bus and control signals. For example, when the physical memory is addressed the upper bits of that address signify that that memory should connect to the bus. The R/W control line determines whether data is directed into or out of the memory.

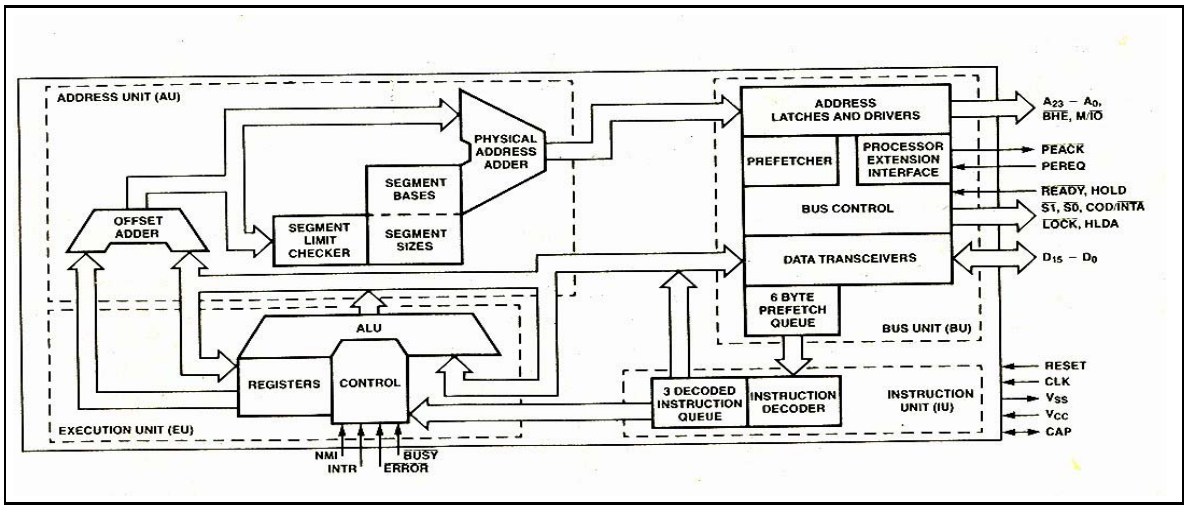


Figure 1.9: Schematic of the '286 chip

  - Read Only Memory (ROM)
  - Programmable Read Only Memory (PROM)
  - Erasable Programmable Read Only Memory (PROM)
  - Random Access Memory (RAM)
    * dynamic RAM (DRAM)
    * static RAM
  - Non-volatile memory (NVM)

- Input/Output (I/O)
  - Serial: asynchronous or synchronous
  - Parallel: drivers and interfaces for switches, motors, valves. Can be either simple driver interface or more complex protocols such as GPIB, SCSI, TCP/IP, etc.
  - Digital-to-analog (D/A): output a voltage, resolution depends on number of bits. D/A conversion is done via a tree of comparators.
  - Analog-to-digital (A/D): read a voltage (*e.g.*, thermocouples, thermistors, strain gauges). One method of A/D conversion (successive approximation) is done by converting a digital guess to analog using a D/A converter and then using a comparator to decide if the guess is high or low. If we begin with the most significant digit then we would need to make N guesses, where N is the number of bit of precision. With redundant circuitry the operation can be made faster (flash A/D's).
  - Video: video processors are CPU's in themselves. They usually have their own memory for pixel maps and color tables. To the system CPU the video board can look like memory or I/O or both. The memory on this board is shared between the internal and external buses. This "multi-porting" of memory requires extra circuitry. For example, if the internal video CPU is reading the memory to display it to the monitor at the same time as the system is writing new display data the video CPU may tell the system CPU to WAIT for a cycle.
  - disk interfaces: disk interfaces typically have their own processor and memory and most invoke what is known as direct memory access (DMA) which effectively takes control of the bus in between system cycles (*i.e.*, while system ALU is busy) to fetch system memory.

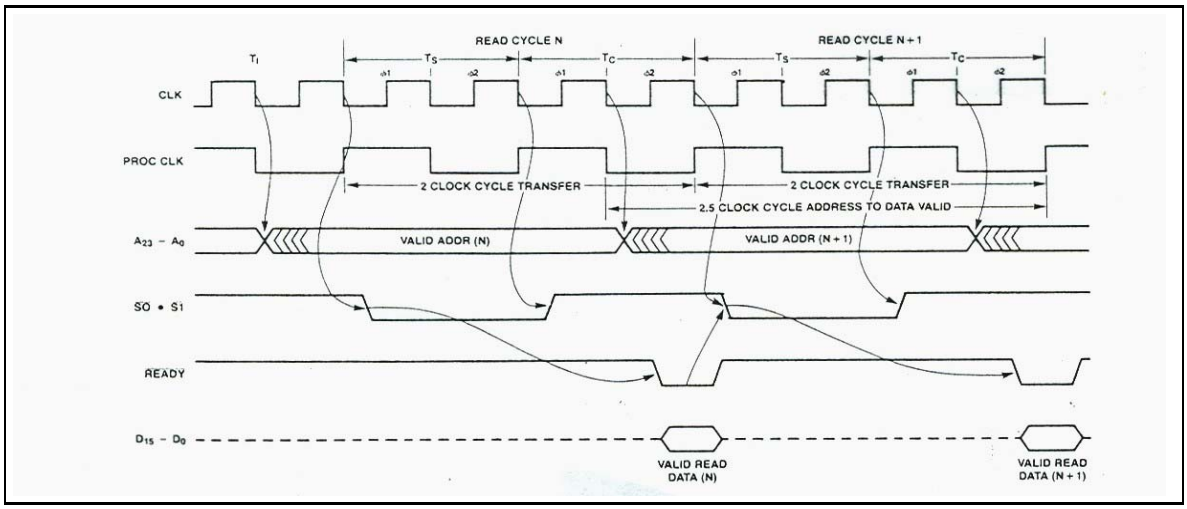


Figure 1.10: Timing diagram of the '286 chip

Example of the timing of the READ operation for a 80286 processor (taken from the Advanced Micro Devices 80286 data sheet, Jan. 1985, pg. 26).

CPU's are classified in terms of their speed, instruction sets, internal structure and special features. A useful gauge of CPU performance is MIPS and MFLOPS; however, caution must be taken in how these are used. Both of these are algorithm dependent and usually only simple instructions are utilized in advertised values. If you are comparing CPU's you should compute the MIPS and/or MFLOPS based on your application. Usually the tough part is estimating the number of instructions.

- **MIPS**: Millions of Instructions per Second is computed by taking dividing the total time of an algorithm into the number of instructions executed. The result divided by 10^6 is the number of MIPS.
- **MFLOPS**: Millions of floating point operations per second. Floating point can either be done in hardware or software and involves many clock cycles. Hardware floating point is usually must faster because addition, multiplication, and division are done with massive logic chips whereas in software the operations are done by utilizing the CPU integer math (16 or 32 bit) and operating on longer words in memory. An exponential, for example, is done by Taylor expansion and consists of many floating point additions and multiplications.

$$\mathrm{EXP}(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots \tag{1.12}$$

Example: On a certain computer I can compute 10000, 2^{nd} order polynomials ($y = a_0 + a_1 \cdot x + a_2 \cdot x \cdot x$) in 0.1 seconds. How many mega-floating point operations (MFLOPS) per second does this equate to (show computation)?

$$\frac{5 \text{ floating operations} \cdot 10000}{0.1 \text{ seconds}} = 500,000 \text{ FLOPS} = 0.5 \text{ MEGAFLOPS} \tag{1.13}$$

1.7 History of Computing

The images in Fig. 1.11 show the ENIAC (c.1946) which was the first electronic computer. It could store 20, 10 decimal digit numbers and perform addition (0.200 ms), multiplication (2.6 ms), division (25 ms), and

Table 1.4: Highlights in the history of computing

1889	Herman Hollerith used 80 character cards to store census data.
1944	MARK-I, electromagnetic relays
1946	ENIAC, 18,000 vacuum tubes, 160 KW, 5000 MIPS, 1000 sq. feet.
1952	UNIVAC-
1957	FORTRAN
1958	Integrated Circuits
1964	IBM 360
	BASIC created at Dartmouth College
1965	DEC PDP-8, $18,000.
1969	UNIX created by AT&T Bell Laboratories.
1971	Shugart 8" floppy disk
1974	Intel 8080 developed
1976	Shugart 5 $\frac{1}{4}$" floppy disk
1977	Apple-II, 6502 CPU, 16K RAM, 16 K ROM
1978	VAX 11/780, 4.3 GB
1979	Motorola 68000
1981	Sony 3 $\frac{1}{3}$" floppy disk
	IBM-PC, 4.77 MHz 8088, MS-DOS 1.0/1.1
1985	Sony & Phillips CD
	C++ created
1987	EXABYTE introduces 8mm helican scan tape drive
1989	use of e-mail emerges rapidly
1991	world-wide-web with internet browsers emerged from CERN.
1996	advanced metal evaporated (AME) tape emerges in AIT's

square root (25 ms). It had 19,000 vacuum tubes and 1,500 relays and consumed 200 kilo-Watts. On the right hand side the boards for ENAIC, EDVAC, ORDVAC, and BRLESC-I are shown

(from http://ftp.arl.army/ftp/historic-computers)

The image in Fig. 1.12 is of a INTEL 8088 8bit processor processor (c.1979) which has 29,000 transistors, 3 μm technology, and could be clocked at 4.77 MHz. 16-bit interger addition took 2 μs. (from http://www/intel.com/intel/museum/25anniv/hof/hof-main.html)

The image in Fig. 1.13 is a 8MHz 286 processor (c.1982). With 80287 co-processor the chip-set could add a 7 decimal digit value in 1 μs, multiply in 1 μs, divide in 10 μs, and perform a square root in 9 μs. A cosine function would take 5-31 μs.
(from http://www/intel.com/intel/museum/25anniv/hof/hof-main.html)

The image in Fig. 1.14 is a INTEL Pentium III 32 bit processor (1999) which has 28.1 million transistors, 0.18 μm technology and can be clocked at 866 MHz. (from http://www/intel.com /intel/museum/25anniv/hof/hof-main.html). It has a build-in math co-processor, 8 KB cache memory for instructions and data, dual path 486 CPU's, and branch prediction.

There have been interesting things done in hardware over the recent years.

- In 1979 Motorola built the 68000 chip (Fig. 1.15 68,000 transistors on a 0.246x0.281 inch substrate) which had a 16 bit data and 24 bit address space. The internal architecture was a full 32 bit address and data space. The 68020 (Fig. 1.16, 1984, 33 MHz, 200,000 transistors) provided the full 32 bit interface and the code was *upward compatible*. That is, one could write 32 bit instructions on the 68000 and it would parse them onto the 16 bit bus. If one plugged in the 68020 then the instructions would operate faster. They also added had 256 byte cache, etc. The chip could separate supervisor and user, as well as data or program memory to provide up to 16 GB of addressable space. 68030 (1987) and 68040 (1990, 1.2 million transistors, can process 20 million instructions per second) has the same evolutionary path as the INTEL family of chips.

- An interesting chip of the past is the Texas Instruments 9900, introduced in June of 1976. In the era

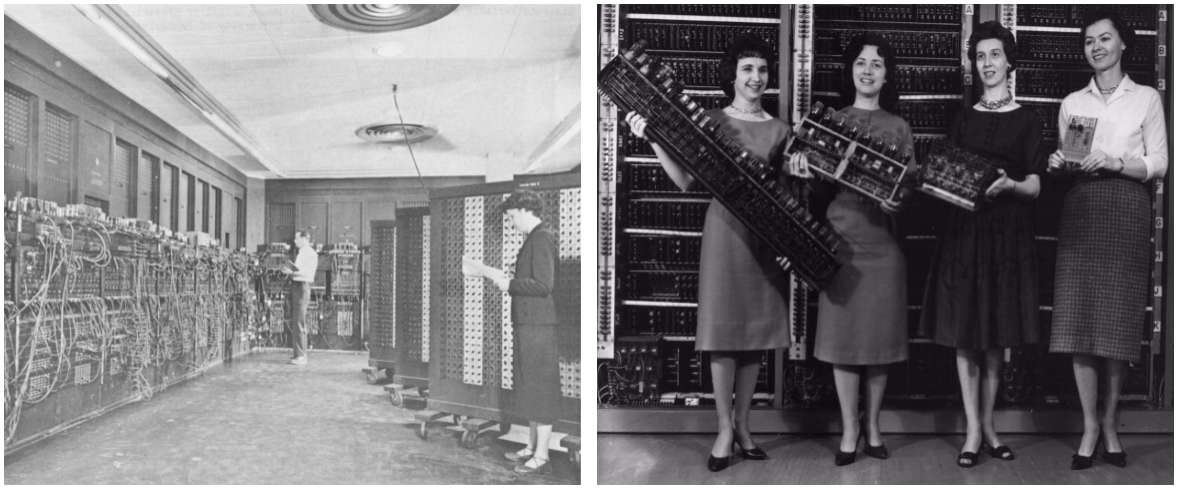


Figure 1.11: The first computer, the Eniac (1946)

Table 1.5: Clock Cycles for Intel Pentium Instructions

instruction	type	# cycles
ADD, NEG, SHIFT	reg = reg + reg	1
	reg = mem + reg	2
	mem = mem + reg	3
integer MUL	byte	11
	16-bit	11
integer DIV	byte	17
	16-bit	25
FADD/FSUB		3
FMUL		3
FDIV		39
FCOS		18-124
FSQRT		70

when CPU's were slower than memory it had an interesting solution. The chip did not have on-board user registers. It required the system to provide fast memory (usually static RAM which was physically located next to the CPU) for registers. This allowed the CPU to access a large number of registers. The CPU had a status register (ST), a program counter (PC), and a workspace pointer (WP). You could call a subroutine (branch and link with pointers, or BLWP) which would 1) read a new PC from the next instruction 2) read a new WP from the next instruction and then using the new workspace of 16 registers it would deposit the old ST,PC,and WP. The new routine could use R1-R12 and also utilize R13-R15 to fetch data or test the environment of the calling program. The deepest subroutine could trace itself all the way back to the reset command! The CPU also had a a 4 bit prioritized interrupt handler, so that interrupts could be handled efficiently (an interrupt was a BLWP triggered by an external device and the user provided the PC,WP in system memory). It also had 64 K of addressable memory and a 4 K synchronous serial communications register unit (CRU) which could individually test or set groups of bits. This chip was ideal for micro-controllers where devices had to be turned on and off (*e.g.*, valves, indicator lights) and single bits had to be monitored *e.g.*, report back switches on valves). The CRU could also be used to control things like memory, so that the 64K address limit could be expanded via paging of memory.

- Advanced Microdevices introduced the AMD2901 in 1975, which was a 4 bit CPU which could be connected together to build any size of word (bit sliced processor). If you needed a 20-bit data bus, just

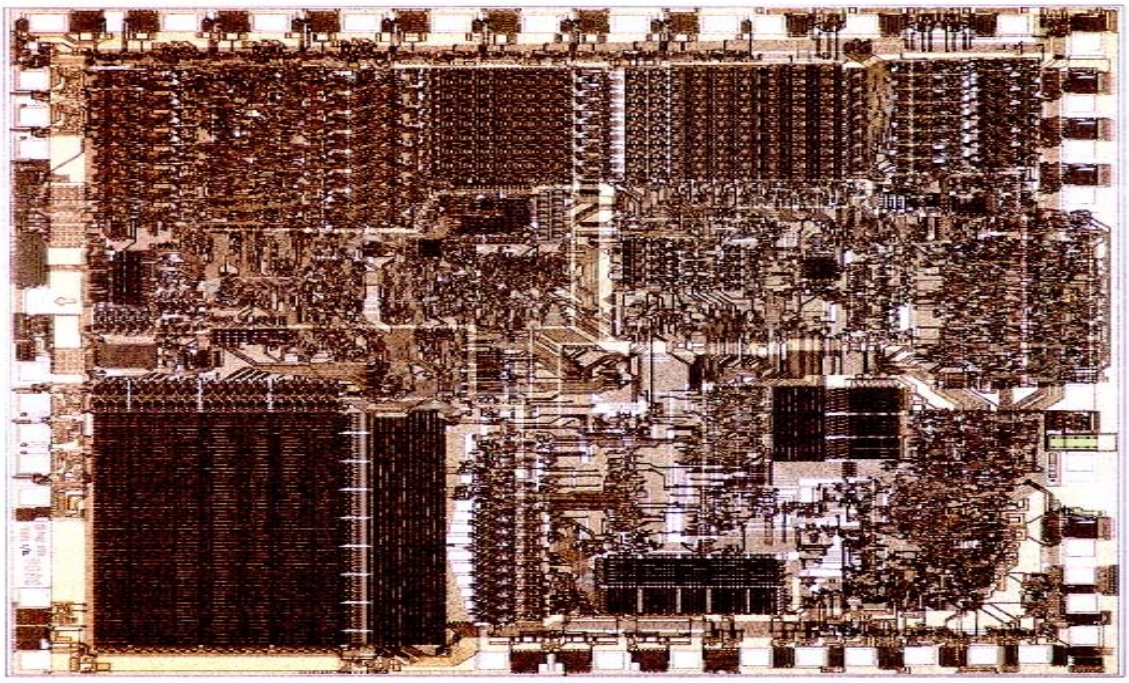


Figure 1.12: An intel 8088 (c.1979) chip

connect 5 of them together. It featured sixteen 4-bit registers and a 4-bit ALU, and operation signals to allow carry/borrow or shift operations and such to operate across any number of other 2901s. An address sequencer (such as the 2910) could provide control signals with the use of custom microcode in ROM.

- 1990's SGI Origin 2000 allows up to 512 2-processor units, each operating at 250 MHz. The internal bus is 64 bits (32 data, 32 address) and each procssing unit can access up to 4 GB of "local" memory. The system can access up to 512·4 GB or 1 TB of memory. All I/O can DMA into entire memory.

For more on the history of computers (especially mainframes) you should visit the National Cryptologic Museum on Rt. 32 near 295. It is open weekdays 9:00-3:00 and weekends 10:00-2:00. Heading east on Rte. 32 make a left at the light (more like a U-turn) onto Colony 7 road after crossing/exiting the BW Parkway (295). Pass the Shell station and make the right hand bend. There is a small museum and parking lot up ahead. They have some old technology including tape silos and Cray hardware.

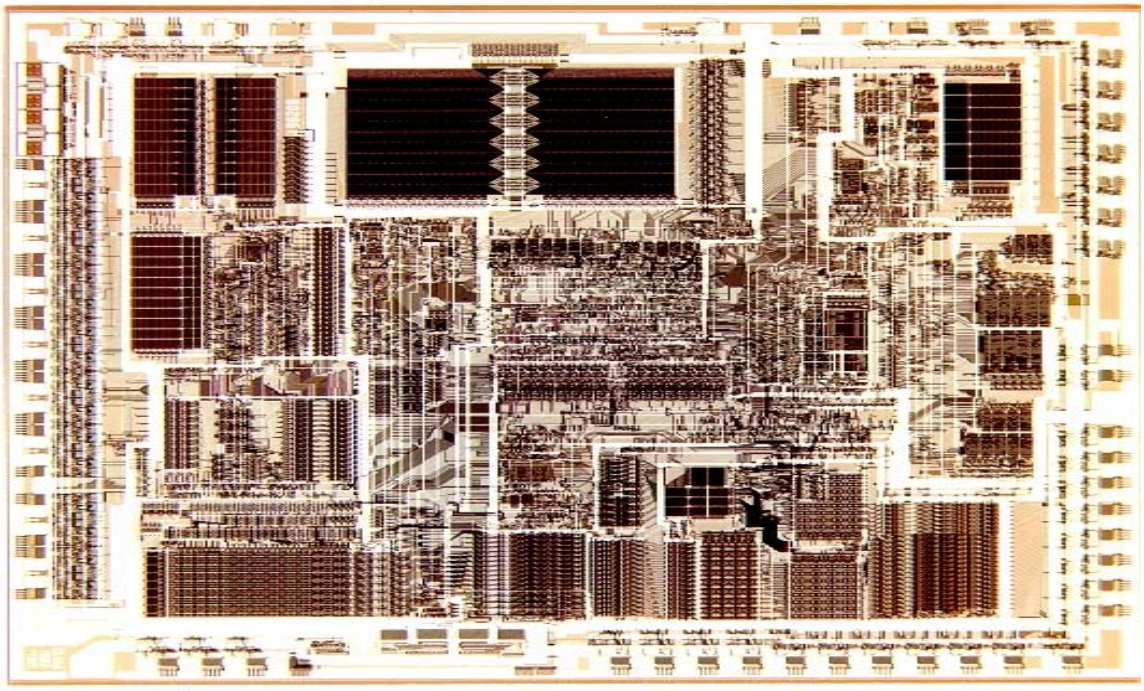


Figure 1.13: An intel 80286 (c.1982) chip

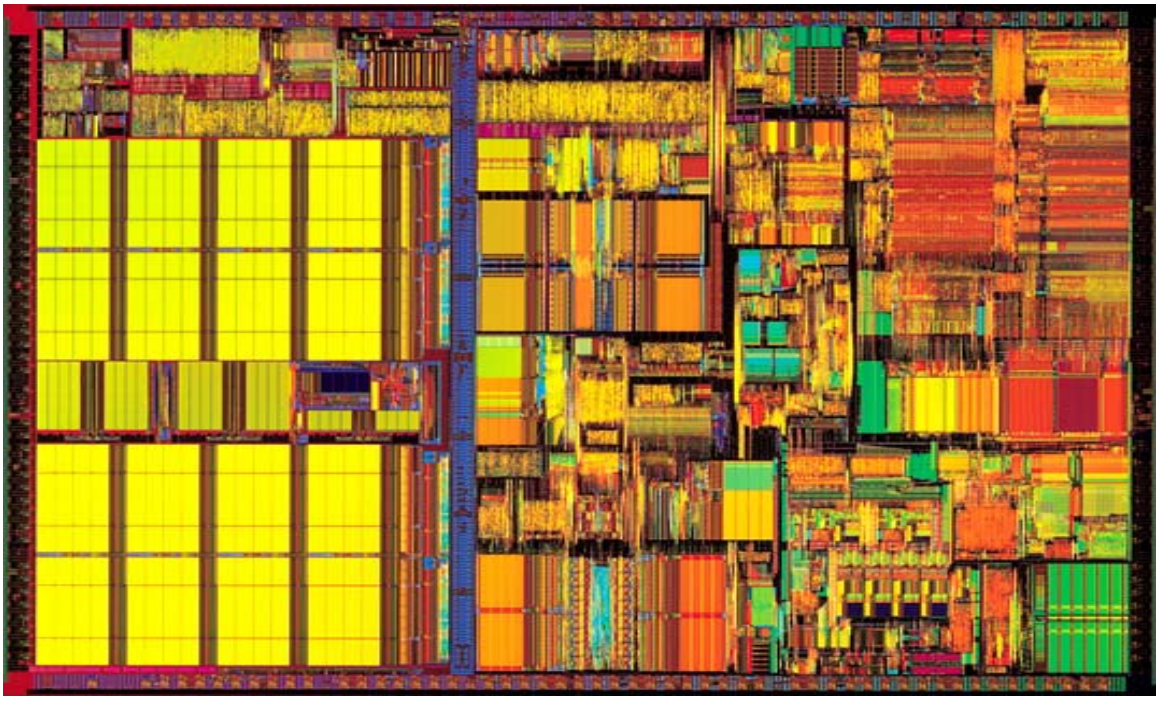


Figure 1.14: An Pentium-III (c.1999) chip

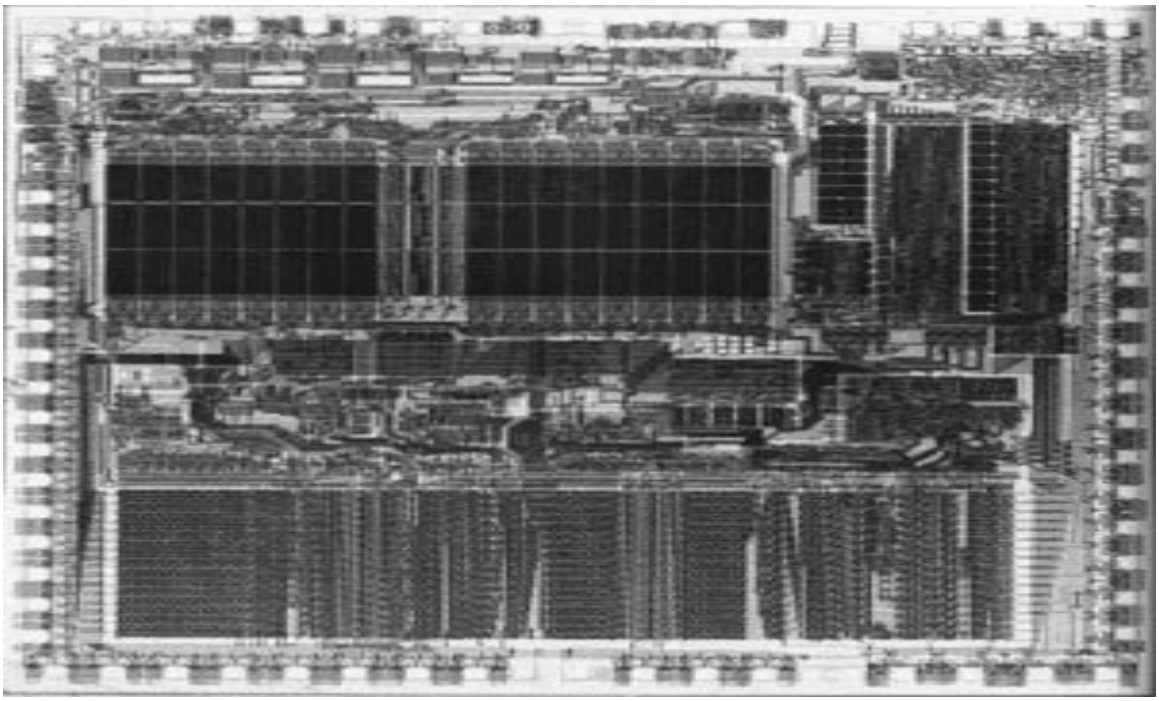


Figure 1.15: The Motorola 68000 microprocessor (c.1979).

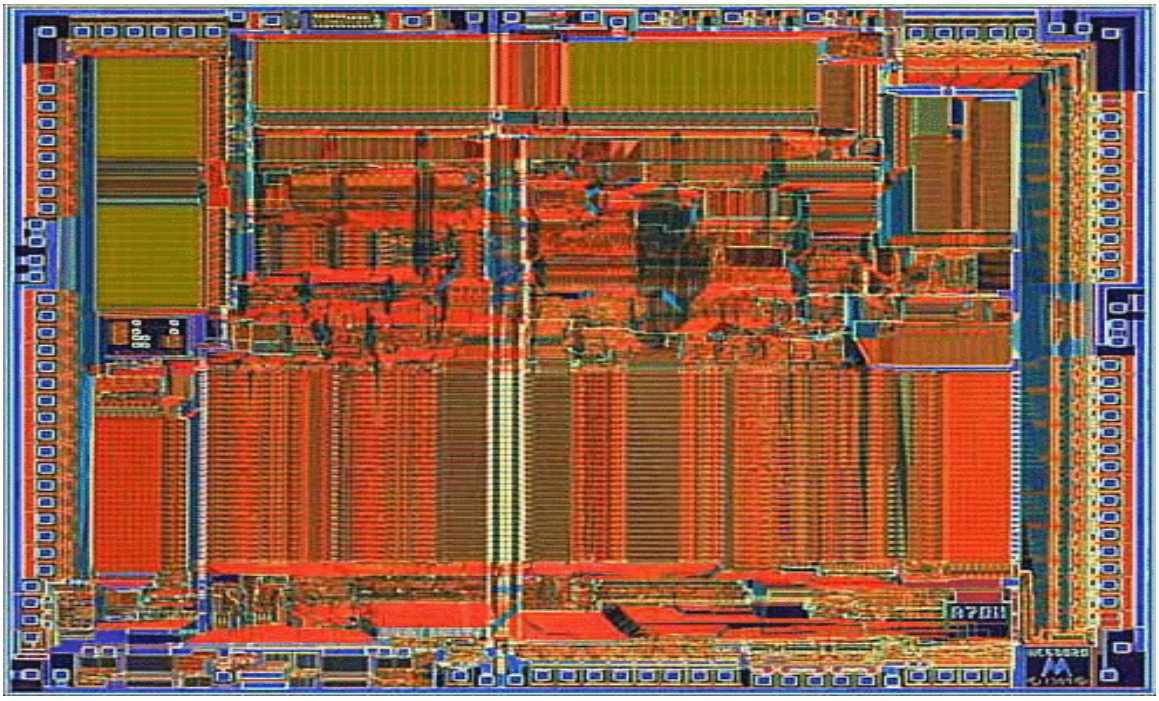


Figure 1.16: The Motorola 68020 microprocessor (c.1984).

Chapter 2

Binary Representation of Variables

Computers work in binary because that is the simplest base for hardware. A "latch" circuit, which is made up with a pair of transistors can hold a "0" state, which is a voltage less than a threshold (0.6 V for TTL logic) or a "1" state which is a voltage greater than 2.5 V.

A single BInary digiT is called a BIT, a four bit grouping of bits is called a nibble and 8 bits are called a BYTE. A word is generally attributed to 16 bits, but in general it is machine dependent.

Table 2.1: Conversion of decimal, binary, octal, and hexadecimal

decimal	binary	octal	hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11

For a base B, the value at each digit has a weight of B^i where i is the index of the digit, starting at $i = 0$ for the least significant digit. Thus, in decimal a value of 56 is decoded as

$56 \equiv 5 \cdot 10^1 + 6 \cdot 10^0 = 50 + 6$

In other bases the same decoding is applied. For 56h the value is

$56h \equiv 5 \cdot 16^1 + 6 \cdot 16^0 = 80 + 6 = 86d$

For hardware applications and assembler programming the use of hexadecimal numbers is convenient. We will use hexadecimal in the next few pages to illustrate how integers, real numbers, and character strings are stored within a binary computer.

Table 2.2: Example of integer formats for common numbers

# of bits	hexadecimal range of numbers	decimal range of numbers	AKA
8	$0 \rightarrow$ FF	$0 \rightarrow 255$	BYTE
12	$0 \rightarrow$ FFF	$0 \rightarrow 4{,}095$	
16	$0 \rightarrow$ FFFF	$0 \rightarrow 65{,}535$	"word"
20	$0 \rightarrow$ FFFFF	$0 \rightarrow 1{,}048{,}576$	1MB
24	$0 \rightarrow$ FFFFFF	$0 \rightarrow 16{,}777{,}215$	16MB
32	$0 \rightarrow$ FFFFFFFF	$0 \rightarrow 4{,}294{,}967{,}296$	4G
40	$0 \rightarrow$ FFFFFFFFFF	$0 \rightarrow 1{,}099{,}511{,}627{,}775$	1T
N	$0 \rightarrow$	$0 \rightarrow 2^N - 1$	

2.1 Binary Representation of Variable Types

2.1.1 Integer Formats

The following are possible "formats" to represent binary in a word of data.

- unsigned integer: see table above
- sign-magnitude: 1^{st} bit is a sign bit, number is negative if it is a "1". Range is $1-2^{N-1} \leq v \leq 2^{N-1}-1$, for an N bit integer.
- 1's complement: The bits are all reversed if negated. Zero has a redundant value. Examples: -0000 $\rightarrow$ FFFF, -0001 $\rightarrow$ FFFE, etc. Range: $1 - 2^{N-1} \leq v \leq 2^{N-1} - 1$.
- 2's complement: Inverted word plus one. This makes negatives defined such that adding a value and it's negative will result in 0000. Examples: -0000 = 0000, -0001 = FFFF, -7FFF=8000, 0001+FFFF=0000 Range: $-2^{N-1} \leq v \leq 2^{N-1} - 1$. This system has the advantage that the number can easily be extended to higher precision by filling the binary from the left with the 1st bit. For example, 0001 can be converted to 32 bits as 00000001 and FFFF is FFFFFFFF.
- Offset binary: We could define zero to be anywhere within the binary number range. For example, if we had a 16-bit system the value of 8000 could be defined as zero. 8001 is +1, 7FFF is -1, 0000 is -7FF, FFFF is +7FF.
- Binary Coded Decimal (BCD): Each hexadecimal digit can have a value of $0 \rightarrow 9$. This wastes 6/16 of the memory but was useful in the early days for display drivers and calculator functions (*i.e.,* very simple binary circuits for decimal interfaces). There are BCD functions in many microprocessor ALU's (*e.g.,* INTEL '86 family) for quick conversion of ASCII I/O to binary.

2.1.2 Character Formats

- Extended Binary Coded Decimal Interchange Code (EBCDIC) was a format used as an extension to BCD by IBM.
- American Standard Code for Information Interchange (ASCII)
- DOS has an extended format
- almost every software package has its own extended set, and there is no standard.

Table 2.3: ASCII definition

Hexadecimal values of ASCII Character Set								
hex	char	function	hex	char	hex	char	hex	char
00	null		20	space	40	@	60	‘
01	ctrl-A		21	!	41	A	61	a
02	ctrl-B	STX	22	”	42	B	62	b
03	ctrl-C	ETX	23	#	43	C	63	c
04	ctrl-D		24	$	44	D	64	d
05	ctrl-E		25	%	45	E	65	e
06	ctrl-F		26	&	46	F	66	f
07	ctrl-G	BELL	27	’	47	G	67	g
08	ctrl-H	BS	28	(	48	H	68	h
09	ctrl-I	TAB	29	)	49	I	69	i
0A	ctrl-J	LF	2A	*	4A	J	6A	j
0B	ctrl-K		2B	+	4B	K	6B	k
0C	ctrl-L	FF	2C	,	4C	L	6C	l
0D	ctrl-M	CR	2D	-	4D	M	6D	m
0E	ctrl-N		2E	.	4E	N	6E	n
0F	ctrl-O		2F	/	4F	O	6F	o
10	ctrl-P		30	0	50	P	70	p
11	ctrl-q		31	1	51	Q	71	q
12	ctrl-r		32	2	52	R	72	r
13	ctrl-s		33	3	53	S	73	s
14	ctrl-T		34	4	54	T	74	t
15	ctrl-U		35	5	55	U	75	u
16	ctrl-V		36	6	56	V	76	v
17	ctrl-W		37	7	57	W	77	w
18	ctrl-X		38	8	58	X	78	x
19	ctrl-Y		39	9	59	Y	79	y
1A	ctrl-Z		3A	:	5A	Z	7A	z
1B	ESC		3B	;	5B	[	7B	{
1C			3C	<	5C	back-slash	7C	|
1D			3D	=	5D	]	7D	}
1E			3E	>	5E	hat	7E	tilde
1F			3F	?	5F	_	7F	DELETE

2.1.3 Real Formats

Real numbers are encoded into binary. The word size for floating point depends on the application's requirements for dynamic range and precision. Two categories for storing floating point are:

- Fixed Point: We could build a representation where we assume a “binary point”. For example, if we had a problem where all the values were KNOWN to be between 0.0 and 1.0 (*i.e.*, for example a SIN() or COS() lookup table) we could assume that 0001 was .0001. Analogous to decimal where digits to the right of the decimal point are weighted by 10^{-i}, in general the digits are weighted by B^{-i}, where B is the base. Fixed point has a fixed precision, defined by the number of digits and is relatively easy to implement with an integer ALU.
- sign, exponent, mantissa formats: Here the 1$^{\text{st}}$ bit is a sign (s) bit. The next group of bits are an exponent (e), usually given in offset-binary integer format, and then a fixed point mantissa (f) is given in the remaining bits.

Table 2.4: ASCII and EBCDIC

ASCII: Hexadecimal Values of Symbols

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0									BS	TAB		LF		FF	CR	
1											ESC					
2	SP	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

EBCDIC: Hexadecimal Values of Symbols

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0					TAB								FF	CR		
1																
2		LF														
3																
4	SP										cent	.	<	(	+	|
5											!	$	*	)	;	NOT
6	/											,	%	_	>	?
7											:	#	@	’	=	"
8		a	b	c	d	e	f	g	h	i						
9		j	k	l	m	n	o	p	q	r						
A			s	t	u	v	w	x	y	z						
B										‘						
C		A	B	C	D	E	F	G	H	I						
D		J	K	L	M	N	O	P	Q	R						
E			S	T	U	V	W	X	Y	Z						
F	0	1	2	3	4	5	6	7	8	9						

In the 50’s to 70’s there were a plethora of sign/exponent/mantissa formats. FORTRAN specified a 4 byte (real*4) and 8-byte format (real*8) for floating point, but various vendors made their own assumptions about the number of bits in the exponent and mantissa, choice of base-2, 10, or 16, the offset for the exponent, and even how the mantissa should be formatted (IBM, DEC VAX-D, VAX-G, etc.). As chip manufacturers were using very large scale integration (VLSI) to build floating point processors it was evident a single format was needed. The IEEE (see IEEE Computers, Mar. 1981, pg. 51) proposed a format that has become the standard, due to both its robustness and also because it has been etched into silicon by many chip manufacturers. Their proposal addressed

- for 32 bit formats, precision was deemed most important and they
  - base-2 is used to avoid wasting memory on leading zeros,
  - use normalized mantissas with “hidden” leading bit (decrement exponent until the leading “1” passes the binary point). This adds 1 extra bit of precision.
- for 64 bit numbers, range was considered most important. Multiplication of any two 32-bit numbers will not overflow the 64-bit format.

- underflow could be handled easily by allowing the mantissa to become de-normalized (non-zero numbers which lie between the largest negative number and the smallest positive number) with a flag in the exponent field.
- exceptions, such as not-a-number (NaN, *e.g.*, $\sqrt{-5}$, log(0)) and infinity (Inf, *e.g.*, 1.0/0.0) could be handled.

format	sign(s)	exponent(e)	mantissa(f)
32-bit	1st bit	8 bits	23 bits
64-bit	1st bit	11 bits	52 bits

IEEE 32-bit format

The range of well-behaved numbers for the 32-bit format is $2^{-126}(1.18 \cdot 10^{-38}) \rightarrow 2 \cdot 2^{+127}(3.4 \cdot 10^{38})$ with a relative precision of $2^{-23}(1.2 \cdot 10^{-7})$ (NOTE: $2^k = 10^{k \cdot \ln(2)/\ln(10)}$). Underflow can represent numbers down to $2^{-149}(1.4 \cdot 10^{-45})$ with reduced precision.

Table 2.5: IEEE 32-bit format

type	e	f	value
NaN	$e = 255$	$f \neq 0$	v=NaN
InF $= \infty$	$e = 255$	$f = 0$	$v = (-1)^s \infty$
normal	$0 < e < 255$		$v = (-1)^s \cdot 2^{e-127} \cdot (1.f)$
underflow	$e = 0$	$f \neq 0$	$v = (-1)^s \cdot 2^{-126} \cdot (0.f)$
zero	$e = 0$	$f = 0$	$v = (-1)^s \cdot 0$ (zero)

```
Example of a 32-bit floating point format:

10.0 = 2^3 * 1.25 = 2^3 * (1 + 1/4)

exponent = 3 + offset
mantissa = 1/4 (since normalize bit it "hidden"

32-bit (real*4) IEEE format
---------------------------
1-bit sign = 0
8-bit exp  = 127+3 = 130 = 82h = 1000 0010
23-bit mantissa = .0100 0000 0000 0000 0000 000

put it all together  0100 0001 0010 0000 0000 0000 0000 0000 = 4120 0000h
```

IEEE 64-bit format

The range of well-behaved numbers for 64-bit format is $2^{-1022}(2.2 \cdot 10^{-308}) \rightarrow 2 \cdot 2^{+1023}(1.8 \cdot 10^{308})$ with a relative precision of $2^{-52}(2.2 \cdot 10^{-16})$. Underflow can represent numbers down to $2^{-1074}(5 \cdot 10^{-324})$ with reduced precision.

```
Example of a 64-bit floating point format:

10.0 = 2^3 * 1.25 = 2^3 * (1 + 1/4)
```

Table 2.6: IEEE 64-bit format

type	e	f	value
NaN	$e = 2047$	$f \neq 0$	v=NaN
InF $= \infty$	$e = 2047$	$f = 0$	$v = (-1)^s \infty$
normal	$0 < e < 2047$		$v = (-1)^s \cdot 2^{e-1023} \cdot (1.f)$
underflow	$e = 0$	$f \neq 0$	$v = (-1)^s \cdot 2^{-1022} \cdot (0.f)$
zero	$e = 0$	$f = 0$	$v = (-1)^s \cdot 0$ (zero)

```
64-bit (real*8) IEEE format
---------------------------
1-bit sign = 0
11-bit exp = 1023+3 = 1026 = 402h = 100 0000 0010
52-bit mantissa = .0100 0000 0000 ... 0000

put it all together   0100 0000 0010 0100 0000 0000 ... 0000
                    = 4024 0000 0000 0000h
```

Examples IEEE floating point (hexadecimal)

decimal	int*2	int*4	real*4	real*16
1.0	01	0001	3F80 0000	3FF0 0000 0000 0000
2.0	02	0002	4000 0000	4000 0000 0000 0000
-2.0	FE	FFFE	C000 0000	C000 0000 0000 0000
e	02	0002	402D F854	4005 BF0A 8B14 5769
π	03	0003	4049 0FDB	4009 21FB 5444 2D18
$1.0 \cdot 10^{36}$	-	-	799A 130C	4733 4261 72C7 4D82
$1.0 \cdot 10^{141}$	-	-	-	5D00 CB70 D24B 7379

Problem: Design Your Own Floating Point

1. Write all the binary digits of the IEEE 32-bit floating point representation of the real value 114688.0 (NOTE: $114688.0 = 1.75 \cdot 2^{16}$). Then, write the number as hexadecimal.

   $114688.0 = 1.75 \cdot 2^{16} = 2^{16} \cdot \left(1 + 2^{-1} + 2^{-2}\right)$

   Therefore, $s = 0$, $f = 1.11b$, $e - 127 = 16$

   $e = 143\text{d} = 8{\cdot}16 + 15 = 8\text{Fh} = 10001111\text{b}$, and the entire binary value is

   0|100 0111 1|110 0000 0000 0000 0000 0000b = 47E00000H

2. Design your own floating point system (*i.e.*, specify the binary format) that has more than 4 decimal digits of precision, a dynamic range better than $10^{\pm 18}$, and fits within 24 bits. Be sure that you can handle the exceptions *NaN* and *Inf* as well as a specification of gradual underflow. Give examples of the hexadecimal representation of the special cases as well as the extreme decimal values that your system can represent.

   Usually, the representation of the mantissa is where we want to place the most bits. Therefore, we first determine the minimum number of bits that can represent the dynamic range of our number system. We chose a power of two because any other exponent would make the number of bits in the mantissa too variable.

   We note that $10^{\pm 18}$ can be represented as $2^{\pm k}$ where $-63 \leq k \leq 62$. $2^{-63} = 1.084 \cdot 10^{-19}$ and $2^{62} = 4.6 \cdot 10^{18}$. Therefore, a binary number between 0 and 127 (7 bits) can be used to represent the exponent in offset binary in the following format:

My 24-bit format			
type	e	f	value
NaN	$e = 127$	$f \neq 0$	v=NaN
InF $= \infty$	e=127	$f = 0$	$v = (-1)^s \infty$
normal	$0 < e < 127$		$v = (-1)^s \cdot 2^{e-64} \cdot (1.f)$
underflow	$e = 0$	$f \neq 0$	$v = (-1)^s \cdot 2^{-63} \cdot (0.f)$
zero	$e = 0$	$f = 0$	$v = (-1)^s \cdot 0$ (zero)

This leaves 16 bits for the mantissa, f, where 2^{-16} which equals a decimal precision of 0.00001526.

NaN = 7FxxxxH where xxxx can be anything but 0000

$+\infty$ = 7F0000H

$-\infty$ = FF0000H

The maximum value is given by $2^{62}\left(1 + \frac{65536}{65536}\right) = 9.2233 \cdot 10^{18}$ or 7E0001H.

The minimum value with full precision is given by $2^{-63}\left(1 + \frac{0}{65536}\right) = 1.0842 \cdot 10^{-19}$ or 010000H

The minimum value with reduced precision ($e = 0, f \neq 0$) is given by $2^{-63}\left(\frac{1}{65536}\right) = 1.654 \cdot 10^{-24}$ or 000001H.

3. Using your new floating point format, what is the approximate result of each of the following floating point arithmetic operations?

   a) $1 + 10^{-7} = 1$

   b) $1 + 10^{+3} = 1001$

   c) $1 + 10^{+7} = 10^7$

   d) $10^{-10} + 10^{+3} = 10^3$

   e) $10^{+10}/10^{-15}$ = overflow (NaN)

   f) $10^{-10} \cdot 10^{-15}$ = underflow (NaN) (both normal and gradual would fail)

   NOTE: If a 9-bit exponent (and 14-bit mantissa) had been used in the previous problem then both (e) and (f) would be able to be represented in that system.

2.1.4 Complex formats

In FORTRAN there exists a complex data type that is really a grouping of two real formats. In complex*8 there are two real*4 values, the first represents the real component and the second represents the imaginary component. Implicit and explicit functions can operate on the complex numbers and produce complex results. Conversion from real to complex is done with the CMPLX(real*4, real*4) command and conversion back to real is done with the REAL() and IMAG() commands.

In double precision, the complex*16 variable is really a pair of real*8 variables.

2.1.5 Logical formats

Logicals can only have a value of true or false. In FORTRAN the variable can have 1,2,or 4 bytes; however, in all cases the bits are all set to 0 for TRUE and the least significant bit is set to 1 for FALSE, which makes the logical value equivalent to an integer +1 for false. For logical*2 the 16-bit binary would be 0000 0000 0000 0000 for TRUE and 0000 0000 0000 0001 for false. The following FORTRAN program shows how the memory is used:

```
      program test
      logical*4 i
      integer*4 j
      equivalence (i,j)
      i = .TRUE.
      print *, 'TRUE=', j
      i = .false.
      print *, 'FALSE=', j
      end

 TRUE= 1
 FALSE= 0
```

Results from logical operators can be stored into logical variables, for example, the result of (x.ge.0.0) .and. (y.le.100.0) could be stored into logical data type for later testing.

```
logical*4 L1
real*4    x,y
...
L1 = (x.ge.0.0).and.(y.le.100.0)
if(L1 .and. y.eq.0.0) then
```

The use of logicals is appealing, since their meaning cannot be corrupted by machine precision or interpretation. For example, if your were writing a program where you needed to determine if a satellite was observing in daylight or not you could have a flag called DAYLIGHT. If TRUE it is daylight and if FALSE it would be nighttime. The algorithm elements that depended on daytime would be easier to understand compared to a conditional on solar zenith angle (solzang), which would

a) avoids errors due to machine precision. A poorly constructed conditional might not work floating point values. (For example, the use of equal, might never work *e.g.*, if(solzang.eq.90.0) should be if(solzang.ge.90.0).

b allow for a more sophisticated test to result in a single flag. For example, the Earth is still in daylight when solzang is greater than 90° due to refraction, which is a function of latitude, time of year, and topography.

2.1.6 Variable Declarations: Comparison of FORTRAN, IDL, MATLAB

MATLAB works internally with an 8 byte real format. Complex variables are specified with an "i" for the imaginary component (*e.g.*, c16 = 1+2i). When writing binary files the variables can be converted to other types.

Typing in fread() or fwrite()

type	MATLAB
1 byte signed integer	
2 byte signed integer	short, int32
4 byte signed integer	long, int64
4 byte real	float, float32, real*4
8 byte real	double, float64, real*8
8 byte complex	
16 byte complex	
4 byte logical	
k byte character	char

IDL and FORTRAN are quite similar in their typing of variables. In the following table the variable types are summaries with the FORTRAN and IDL syntax.

IDL does not support the logical type; however, any variable can be treated as a logical. False is indicated by any EVEN integer or zero real or zero complex. True is indicated by any ODD integer or non-zero real or complex.

In IDL strings are dynamic (terminated with a null) whereas FORTRAN must have the string length declared. FORTRAN fills character strings with spaces (ASCII=32).

Value Declaration		
type	FORTRAN	IDL
1 byte signed integer	integer*1 i1	i1 = 0B
2 byte signed integer	integer*2 i2	i2 = 0
4 byte signed integer	integer*4 i4	i4 = 0L
4 byte real	real*4 r4	r4 = 0.0
8 byte real	real*8 r8	r8 = 0.0d00
8 byte complex	complex*8 c8	c8 = (0.0,0.0)
16 byte complex	complex*16 c16	c16 = (0.0d00,0.0d00)
4 byte logical	logical*4 L4	...
k byte character	character*k sk	sk = '123...k'

Type conversion		
type	FORTRAN	IDL
1 byte signed integer		i1 = byte(r)
2 byte signed integer		i1 = float(r)
4 byte signed integer	i4 = int(r4)	i4 = long(r)
4 byte real	r4 = float(i)	r4 = float(i)
8 byte real	r8 = dble(r4)	r8 = double(r4)
8 byte complex	c8 = cmplx(r4,r4)	c8 = complex(r4,r4)
16 byte complex	c16 = dcmplx(r8,r8)	c16 = dcomplex(r8,r8)
4 byte logical	logical*4 L4(N)	...
k byte character	write(sk,'(fk.2)')	s8 = string(r,format='(fk.2)')

Array Declaration, N=dimension(s)		
type	FORTRAN	IDL
1 byte signed integer	integer*1 i1(N)	i1 = BYTARR(N)
2 byte signed integer	integer*2 i2(N)	i2 = INTARR(N)
4 byte signed integer	integer*4 i4(N)	i4 = LONARR(N)
4 byte real	real*4 r4(N)	r4 = FLTARR(N)
8 byte real	real*8 r8(N)	r8 = DBLARR(N)
8 byte complex	complex*8 c8(N)	c8 = COMPLEX(N)
16 byte complex	complex*16 c16(N)	c16 = DCOMPLEX(N)
4 byte logical	logical*4 L4(N)	...
k byte character	character*k sk(N)	sk = STRARR(N)

2.2 Scientific Computing Approximations

Consider the computation of the surface area of the Earth. There are a number of levels of approximation that occur

1) The shape of the Earth must be modeled, which introduces errors. For example, if we assume a sphere the model would be

$$A_E = 4 \cdot \pi \cdot R_E^2 \tag{2.1}$$

2) The parameters of the model (*e.g.*, $R_E = 6370$) are based on measurements which have

- random errors
- systematic biases
- finite sample size

3) Value of constants (*e.g.*, π) must be truncated to precision of the number system.

4) Results are rounded in floating point mathematics and errors propagate through the equation.

Example of problems in BINARY representation

Here are some examples of a number whose decimal representation has a finite number of digits, while the binary representation cannot be represented with a finite number of digits.

$1.2 = 1.\overline{0011} = 1.00110011001100110011\ldots$
$1.4 = 1.\overline{0110} = 1.01100110011001100110\ldots$
$1.6 = 1.\overline{1001} = 1.10011001100110011001\ldots$
$1.8 = 1.\overline{1100} = 1.11001100110011001100\ldots$

There are many other examples. For example, 1.6 has the same repeat pattern as 3.2, 0.8, 0.4, 0.2, 0.1 since the only difference is the exponent.

2.2.1 Absolute Error

$$\text{Abs.error} = f(x+h) - f(x) \simeq h \cdot f'(x) \tag{2.2}$$

2.2.2 Relative Error

The ratio of absolute error to the true value can be expressed as a fraction or per-cent and is equal to

$$\text{Rel.error} = \frac{f(x+h) - f(x)}{f(x)} \simeq h \cdot \frac{f'(x)}{f(x)} \tag{2.3}$$

2.2.3 Definition of Condition number

$$\text{Condition} \equiv \frac{\text{rel.error of solution}}{\text{rel.error of input}} = \frac{h \cdot f'(x)/f(x)}{h/x} = x \cdot \frac{f'(x)}{f(x)} \tag{2.4}$$

If condition is $\gg 1$ then the problem is "ill-conditioned" or "sensitive". For example, the $\cos(x)$ near $x \approx \frac{\pi}{2}$ has a relative error of $-h \cdot \tan(x) \to \infty$.

$$\begin{aligned} \cos(1.57079) &= 0.63268 \cdot 10^{-5} \\ \cos(1.57078) &= 1.63268 \cdot 10^{-5} \\ \text{condition} &= \frac{1.58}{6.37 \cdot 10^{-6}} = 2.5 \cdot 10^{5} \end{aligned}$$

Remember that the number systems have the following inherent uncertainty due to their binary representation:

Summary of precision and dynamic range of computation number systems			
	absolute precision	relative precision	dynamic range
int*1	1	$1/v$	$\pm$ 127
int*2	1	$1/v$	$\pm$ 32767
int*4	1	$1/v$	$\pm$ 2,147,483,647
real*4	$v \cdot 10^{-7}$	10^{-7}	$\pm 10^{\pm 308}$
real*8	$v \cdot 10^{16}$	10^{-16}	$\pm 10^{\pm 308}$

2.2.4 Rounding Error:

Difference with exact computation due to rounding arithmetic (precision of floating point). That is the value of x is represented by $\hat{x}$, which has small differences due to machine precision ($|x - \hat{x}| \approx \epsilon_{mach}$).

$$f(x) \simeq f(\hat{x}) \tag{2.5}$$

For example, representation of trancendentals, such as π and e are always truncated. In addition, many decimal values are not represented exactly in binary.

Small errors due to machine precision can propagate. Consider, for example, the equation $x^2 - 4 \cdot x + 4 = 0$ has the exact roots of $x \pm 2$. If there is a rounding error in the constant, $x^2 - 4 \cdot x + 4 \pm 10^{-8} = 0$ then this can introduce larger errors in the roots of $2 \pm 2 \cdot 10^{-4}$

Problems:

1. Which method of computing the standard deviation is better and explain why.

$$\sigma = \left[\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})^2 \right]^{\frac{1}{2}} \qquad \text{Two - pass formula} \tag{2.6}$$

$$\begin{aligned} \sigma &= \left[\frac{1}{n-1} \left(\left(\sum_{i=1}^{n} x_i^2 \right) - n \cdot \overline{x}^2 \right) \right]^{\frac{1}{2}} \\ &= \left[\frac{1}{n-1} \left(\left(\sum_{i=1}^{n} x_i^2 \right) - n \cdot \left(\frac{1}{n} \sum_{i=1}^{n} x_i \right)^2 \right) \right]^{\frac{1}{2}} \qquad \text{One - pass formula} \end{aligned} \tag{2.7}$$

The two pass formula will tend to be more stable in that the square root will tend to be more behaved. A test for negative arguments will still be necessary; however. When it fails the answers will have sums of underflows, which may cause some systems to fail.

The one pass formula will tend to exaggerate the loss of precision and can have dramatic errors. See the example below. For this reason the one pass formula should be avoided.

```
program stdtest
implicit none

integer*4 n, k
real*8 x(7), real*8 avg, sum1
real*8 sum2a, sum2b, arg, sum2

do k = 15,19
```

```
        do n = -3, 3
          x(n+4) = 10.0d00**k + DBLE(n)
        enddo

c       example of how two pass method fails.
c       -------------------------------------
        avg = x(1)
        do n = 2,7
          avg = avg + x(n)
        enddo
        avg = avg/7.0d00
        sum1 = 0.0
        do n = 1, 7
          sum1 = sum1 + (x(n)-avg)**2
        enddo
        sum1 = DSQRT(sum1/6.0d00)

c       example of how one pass method fails.
c       -------------------------------------
        sum2a = 0.0d00
        sum2b = 0.0d00
        do n = 1, 7
          sum2a = sum2a + x(n)**2
          sum2b = sum2b + x(n)
        enddo
        sum2b = sum2b/7.0d00
        arg = sum2a-7.0d00*sum2b**2
        if(arg.gt.0.0d00) then
          sum2 = DSQRT(arg/6.0d00)
          write(*,200) k, sum1, sum2
        else
          write(*,210) k, sum1, arg
        endif
      enddo
 200  format(' 10^',i2,': ', 2e26.16)
 210  format(' 10^',i2,': ', e26.16, '   SQRT(',e26.16,'/6)')
      end
```

2. Devise an input data sequence (for a small number of points) that dramatically demonstrates the numerical difference between the two formula, given the precision of the particular machine you are using. Write a program to compute the standard deviation by both methods and show that the computation you devised does indeed fail.

   In real*8 math the can make an array where the x values are

   $x = \overline{x} + n$ for n=-3,3

   $$\sigma(x) = \left[\frac{1}{6}\left(\sum_{n=-3}^{3} n^2\right)\right]^{\frac{1}{2}} = 2.1602469$$

   One pass method fails because the argument of the square root is equal to a negative value. The two pass method returns an incorrect value.

   Here are some examples:

$\overline{x}$	one pass	two pass
10^{15}	2.1602469	$8.37 \cdot 10^6$
10^{16}	2.7080128	$1.61 \cdot 10^8$
10^{17}	0.0	$\sqrt{-0.96 \cdot 10^{19}/6}$
10^{18}	0.0	$\sqrt{-0.44 \cdot 10^{21}/6}$
10^{19}	0.0	$2.28 \cdot 10^{10}$

3. The exponential function is usually computed by a MacLauren series:

$$\exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \tag{2.8}$$

a) Show how you can use the series, in the form shown, to get accurate results for $x < 0$?

$$\exp(-x) = \frac{1}{\exp(|x|)} = \frac{1}{1 + |x| + \frac{|x|^2}{2!} + \frac{|x|^3}{3!} + \dots} \tag{2.9}$$

b) Can you rearrange the series or regroup the terms in any way to get more accurate results for $x < 0$.

$$\exp(-x) = \left(1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \dots\right) - \left(\frac{|x|^3}{3!} + \frac{|x|^5}{5!} + \dots\right) \tag{2.10}$$

or, alternatively

$$\exp(-x) = (1 - x) + \frac{x^2}{2!}\left(1 - \frac{x}{3}\right) + \frac{x^4}{4!}\left(1 - \frac{x}{5}\right) + \frac{x^6}{6!}\left(1 - \frac{x}{7}\right) + \dots \tag{2.11}$$

2.2.5 Discretization or Truncation Error:

This error arises due to a computational approximation due to finite amount of resources. For example

- truncating an infinite series
- finite differencing approximations for derivatives
- polynomial fitting for interpolation
- convergence criteria in an interation computation.

$$f(x) \simeq \hat{f}(x) \tag{2.12}$$

For example, the exponential can be represented by the first few terms of the Taylor expansion.

$$e^x \simeq 1 + x + \frac{x^2}{2} + \frac{x^3}{6} \tag{2.13}$$

Or a continuous function is sampled by an instrument at specific moments in time producing a discrete and finite number of points.

2.2.6 Unstable Algorithms:

An algorithm can be computed in a manner where finite machine precision will make that algorithm have severe errors. Heath, 1997, gives four examples

1. The quadratic equation:

$$a \cdot x^2 - b \cdot x + c = 0 \tag{2.14}$$

has roots given by

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2.15}$$

but if $|b|$ is roughly equal to $\sqrt{b^2 - 4ac}$ then the numerator can vanish for one of the roots (*e.g.*, $a \Rightarrow 0$). This condition can be trapped and an alternate solution equation for the roots can be used

$$x = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}} \tag{2.16}$$

In addition, the algorithm should test if the argument of the square root becomes negative.

2. A divergent summation is a good example of how unexpected instabilities can arise.

$$\sum_{n=1}^{\infty} \frac{1}{n} \tag{2.17}$$

One might think that $\frac{1}{n}$ will eventually underflow or $\sum \frac{1}{n}$ will eventually overflow, but the reality is this summation will converge in a numeric computation due to finite precision of the floating point representation. It converges, when $\frac{1}{n} < \epsilon_{mach} \cdot \sum_{k=1}^{n-1} \frac{1}{k}$ For a IEEE 32-bit calculation this will occur a N =2,097,152 and will have have a value of 15.40368 (4176 757C in Hexadecimal).

3. The exponential function for $x < 0$ can become unstable. For very small values of x the negative terms will be lost and the function will converge to an improper value. For very large x the accumulated sum will be too small to be added to the higher order terms.

$$\mathrm{EXP}(-x) = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \ldots \tag{2.18}$$

4. An example that occurs frequently is computing standard deviation of an array of numbers. The standard deviation is related to the root-mean-square (RMS) and average value (BIAS) of the array of numbers and is computed with the equations:

$$\begin{aligned} \sigma &= \left[\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2 \right]^{\frac{1}{2}} && (2.19) \\ &= \sqrt{\frac{n}{n-1}} \cdot \left[\frac{1}{n} \sum_{i=1}^{n} x_i^2 - (\bar{x})^2 \right]^{\frac{1}{2}} && (2.20) \\ &= \sqrt{\frac{n}{n-1}} \cdot \left[\mathrm{RMS}^2 - \mathrm{BIAS}^2 \right]^{\frac{1}{2}} && (2.21) \end{aligned}$$

where,

$$\text{RMS} \equiv \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} \tag{2.22}$$

$$\text{BIAS} \equiv \bar{x} \equiv \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.23}$$

The first form of the equation for standard deviation requires two summations

```
s1 = 0.0
do i = 1, n
  s1 = s1 + x(i)
enddo
bias = s1/float(n)
S2 = 0.0
do i = 1, n
  s2 = s2 + (x(i)-bias)**2
enddo
sdv = SQRT(s2/float(n-1))
```

while the second method can be done with one loop

```
s1 = 0.0
s2 = 0.0
do i = 1, n
  s1 = s1 + x(i)
  s2 = s2 + x(i)**2
enddo
bias = s1/float(n)
rms  = SQRT(s2/float(n))
sdv = float(n)*SQRT(s2/float(n) - bias**2)/float(n-1)
```

The author of the textbook uses these illustrations as examples of why one should use canned software. While this may be true, I would still warn you to check all software, no matter how reliable the source. I have found the following bugs in packages - and these were NOT easy to isolate!

- SQRT(0.0001) = -0.01
- complex exponential, $\exp(i \cdot x) = \cos(x) + i \cdot \sin(x)$, was wrong for certain values of x. One of their branches or approximations dropped the $\sin(x)$ component.
- literals passed to subroutine are buffered so they cannot be modified. A compiler bug made the following code print zeros for both the value of "2" and for the value of "j".

```
program test
 call x(2)   ! a literal is passed
 j = 2*5
 print, 2, j
end

subroutine x(i)
i = 0        ! subroutine modifies argument
return
end
```

Every programmer has their horror stories of bugs they have found in compilers and algorithms. My recommendation is you never assume that a canned or borrowed routine is bug free. Thoroughly test all routines that are critical to your task and build a hierarchy of trustworthy routines.

2.3 Problems: Polynomials & Accuracy of Numbers

1. Which is a better way to compute a polynomial

   a) $a_0 + a_1 \cdot x + a_2 \cdot x^2$ or

   b) $(a_2 \cdot x + a_1) \cdot x + a_0$

   This is a trick question because the answer depends on assumptions about the coefficients of the polynomial. Pick (b) because

   - (a) has 3 adds and 2 multiplies and either another multiply or a $\exp(2 \cdot \log(x))$ to perform x^2 (depending on the compiler) versus 2 adds and 2 multiplies for (b). So (b) is faster and has less operations, therefore, is less prone to round-off error.
   - (b) is more accurate if $a_2 \cdot x$ is significantly smaller than a_1 (a case that is nearly linear)

   but,

   - (a) can be written as array math

     $$B_{i,j} \cdot A_j = Y_i \tag{2.24}$$

     where, B is the *Vandermonde* matrix

     $$B_{i,j} = \begin{pmatrix} 1 & x(1) & x(1)^2 \\ 1 & x(2) & x(2)^2 \\ \dots & \dots & \dots \\ 1 & x(N) & x(N)^2 \end{pmatrix} \tag{2.25}$$

   - (a) might be more precise if the equation is strongly quadratic and added from left to right (the compiler convention).

2. Fill in the table below

type	accuracy	abs. or rel.	dynamic range
8-bit signed integer	1	ABS	$\pm 2^7$ or -128 $\rightarrow$ +127
16-bit signed integer	1	ABS	$\pm 2^{15}$ or -32768 $\rightarrow$ +32767
32-bit signed integer	1	ABS	$\pm 2^{31}$ or -2·10^9 $\rightarrow$ +2·10^9
32-bit IEEE real (real*4)	$2^{-23} \approx 10^{-7}$	REL	$\approx 2^{\pm 127} \approx 10^{\pm 38}$
64-bit IEEE real (real*8)	$2^{-52} \approx 10^{-15}$	REL	$\approx 2^{\pm 1023} \approx 10^{\pm 308}$

2.4 Random Number Generators

Within your computer there are random number generator(s) which produce a sequence of uniform deviates on an interval of $[0, 1]$. Usually you must specify a *seed*. The simplest functions (*e.g.*, RAN(seed)) will use *linear congruential generators* which will produce a sequence

$$I_{j+1} = (a \cdot I_j + c) \quad \text{MOD} \quad m \tag{2.26}$$

where m is the *modulus* (usually set by the size of the machine word), a and c are positive integers called the *multiplier* and the *increment*, respectively. The sequence will repeat itself with a period that is no greater than m. The RAN() function divides the integer by m to give a real uniform deviate between 0 and 1. The *seed* is simply the value of I_j and is returned as I_{j+1}.

Example congruential generators			
	a	m	c
Scheid pg. 450	25173	65536	13849
IMSL RAN1	16807	2^{31}-1=2147483647	0
Sobol, pg 71	5^{17}	2^{40}	0
IMB 360 RANDU	65539	2^{29}	0

Sequential correlations can exist with simple random number generators and usually the sequences have noticable repetitive patterns. This is easily noticed when one uses these generators for card games. The brain has a remarkable ability to visualize patterns in the cards.

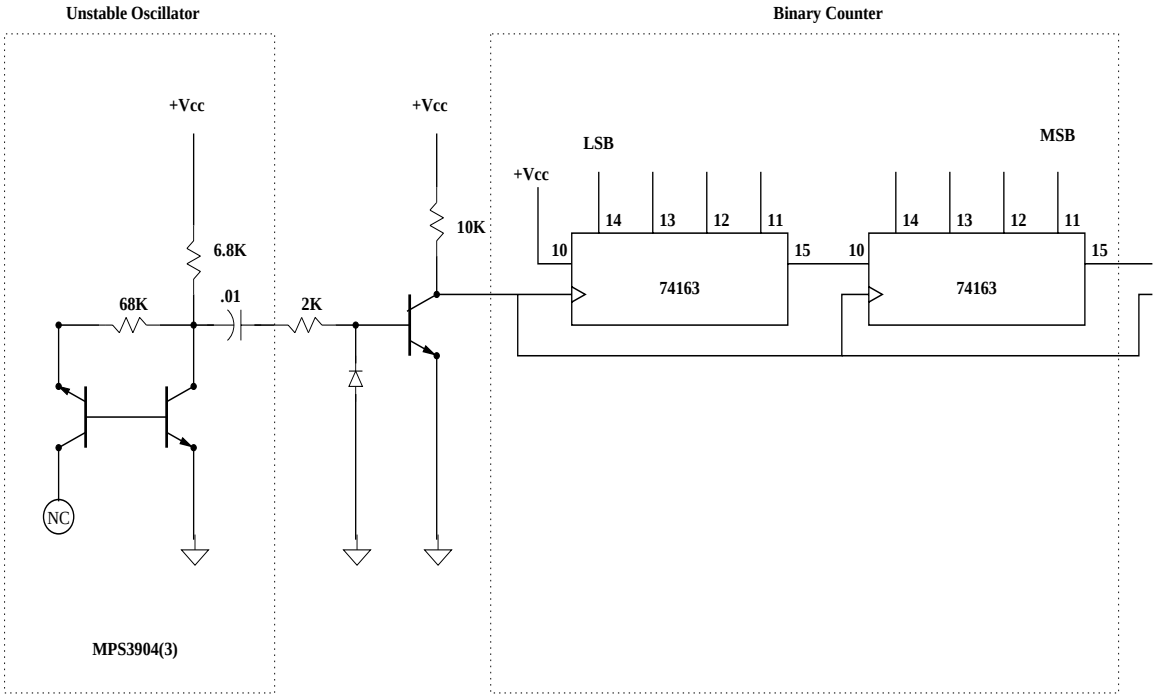


Figure 2.1: An example of a hardware random number generator

An example of a circuit to generate truly random numbers is shown in Fig. 2.1. The basic idea is to amplify an unstable noise source (*i.e.*, a uniform distribution of frequencies) and drive an electronic counter. In the circuit above a back biased transistor is used as a very unstable, high-frequency oscillator. The output is wired into a counter and the frequency is such that the counter is also unstable (missing counts, etc.). The high-speed counter chip will get hot and the unstable transistor is in thermal contact. The transciever interface is an asynchronous interface.

For many applications a truly random number is not desired. We want a pseudo-random sequence that has good statistical properties but also is repeatable if we begin with the same *seed*. Different machines and languages use different constants and techniques. If you want to build a pseudo-random sequence that runs

the same on all platforms, you will need your own random generator.

The code in section B.15 for a robust random number generator (ftp/ran3.pro, ftp/ran3.F) is from Numerical Recipes. As with all code, you should test the code yourself. If the output of the random number is binned over the interval of $[0, 1]$ then the bins should fill evenly. Sequential correlations are more difficult to detect.

One warning with this code, the common block holds values that must be passed between calls to the routine. If you are programming in another lanquage you must insure that those values are maintained between calls.

A uniform probability distribution is one with probability of generating a number between x and $x + \delta x$ given by

$$p(x) \cdot \delta x = \delta x \quad \text{for} \quad 0 \leq x < 1 \tag{2.27}$$

$$p(x) \cdot \delta x = 0 \quad \text{otherwise} \tag{2.28}$$

The probability function is normalized

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{2.29}$$

In many applications there is a need for other probability distributions. If we generate a uniform deviate and then take a prescribed function of it, $y(x)$, then the probability of y, denoted $p(y) \cdot \delta y$ is determined by the fundamental transformation law of probabilities.

$$|p(y)\delta y| = |p(x)\delta x| \tag{2.30}$$

or

$$p(y) = p(x)\left|\frac{\delta x}{\delta y}\right| \tag{2.31}$$

For the uniform deviate

$$p(y) \cdot dy = \left|\frac{\delta x}{\delta y}\right| dy \tag{2.32}$$

If we desire an exponential distribution, we can compute $x = -\log_e(r)$. This expression is found by inversion of the descied distribution function by integration over an interval

$$\int_{x=0}^{x} 1 \cdot dx = \int_{0}^{y} f(y) \cdot dy = \int_{0}^{y} \log_e(y) \cdot dy = e^{-y} \cdot dy \tag{2.33}$$

2.4.1 Example: exponential deviates

A computation was done using a uniform random deviate, r, and computing $x = \log_e(r)$. A histogram of the result is shown in the following figure.

2.4.2 Gaussian Deviates

The most common is the Gaussian (normal) distribution. Random noise in instruments and many Monte-Carlo applications utilize Gaussian deviates.

$$p(y)dy = \frac{1}{\sqrt{2\pi}} \cdot e^{-y^2/2} \cdot dy \tag{2.34}$$

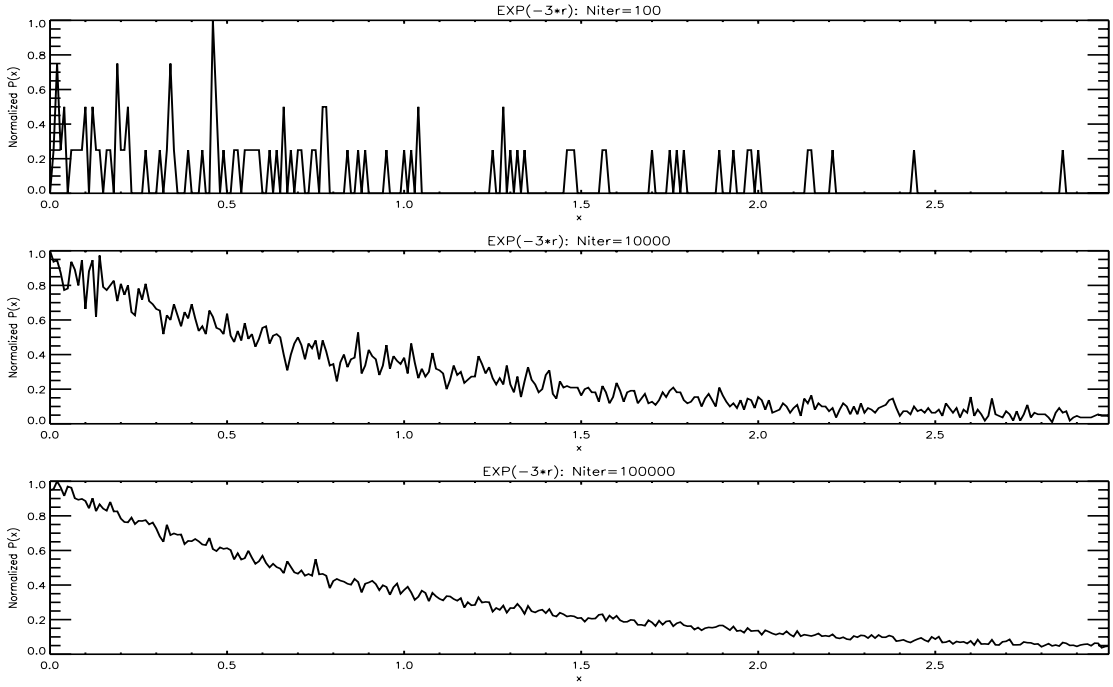


Figure 2.2: Example of probability density function $x = \log_e(r)$ generated from a uniform deviate

But integration of the probability distribution is problematic. Schied pg. 456 discusses a simple algorithm to compute the Gaussian distribution using

$$y = \sum_{i=1}^{1} 2x_i - 6 \tag{2.35}$$

where x_i is a random uniform deviate.

The routine in section B.5 (gasdev.pro or gauss.F) uses the *Box-Muller* transformation to compute a Gaussian deviate. See Numerical Recipes (Press *et al.*, 1986) for a more detailed description.

I have provided the output of a program (gastest.pro) that I used to test the Numerical Recipes (Press *et al.*, 1986) Gaussian deviate (which called their uniform deviate supplied here). The routine simply bins the occurance of random numbers and builds the probability distribution. For the Gaussian deviate one would expect a Gaussian function to emerge. In Fig. 2.3 the routine was called 5000 times. We would expect a bias tending towards zero and a standard deviation of 1.0 is all is working well. After 500,000 calls, shown in Fig. 2.4, the random number generator we begin to see the Gaussian distribution emerge and the statistics (bias and standard deviation) approach the expected values.

For arbitrary distributions see the Monte Carlo acceptance/rejection method in section 6.1.

2.5 Storage of Data on Disk

2.5.1 Introduction to DISK I/O

- its all "0's and 1's": disks are formatted in tracks and sectors. The track and sector ID is written and a space is cleared out. Usually sectors are 512 bytes and there are 10-20 sectors per track and many tracks per disk.

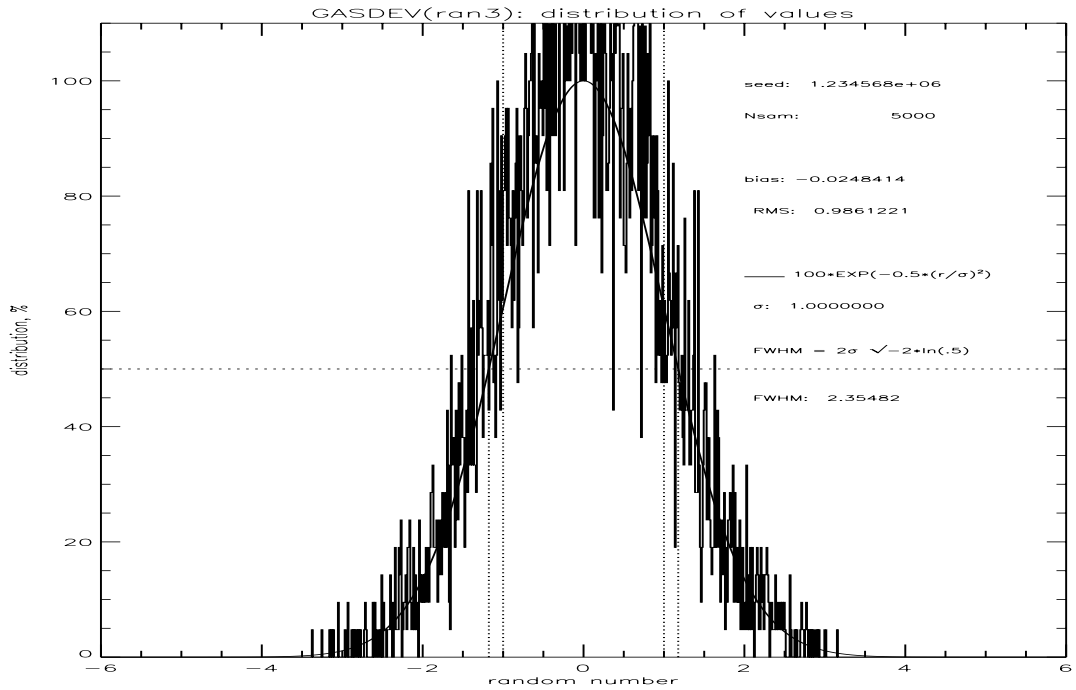


Figure 2.3: Gaussian probability density function after 5000 calls.

- To read a file the disk-head steps to the track and begins reading the bits, verifies it located the track by reading track ID, it waits for sector=0 to cross under and then begins transferring the sector contents to internal buffers. The disk controller begins transfer of the data in the buffers to the system via DMA.
- The OS determine how the disk is allocated. Each file it stores has a directory entry, which is file information (name, size, attributes, date/time, etc.) stored in a sector or two on the first track(s).
- The OS usually marks bad regions and keeps a record in the first track of the disk (the file allocation table (FAT) table) and these regions are avoided by the disk operating system.
- The disk can be thought of, and used, as memory. In the old days this was typical.
- memory can be partitioned to emulate a disk. A virtual disk is a block of memory that the disk operating system treats as hardware.
- ASCII (TEXT) files (free or formatted output)

  forces system to convert ASCII to binary (integer, logical, real complex), therefore, it takes longer to load.

  to represent the dynamic range of variables takes more bytes (see table below)

- BINARY files

  BINARY files are system dependent. There is a pseudo standard for FORTRAN 77 files format (FORTRAN, IDL, MATLAB?); however, even within a recognized standard there are differences that a user must wrestle with:

  – Big Endian, Little Endian
  – DEC minimum record unit is 4 bytes

  The options available for writing BINARY files are:

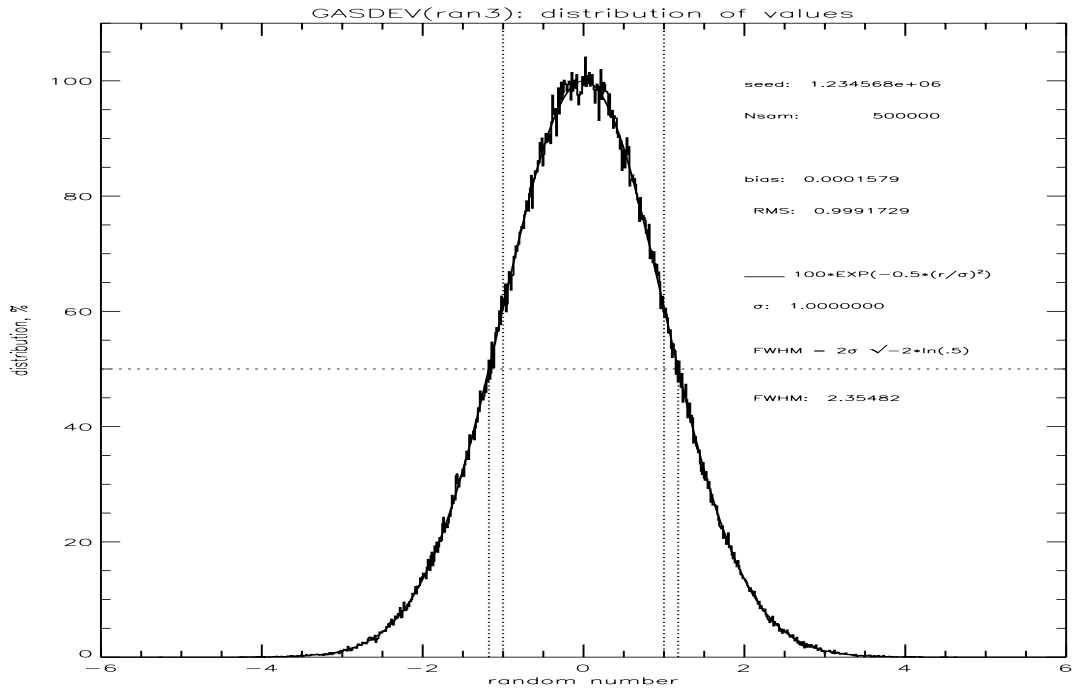


Figure 2.4: Gaussian probability density function after 500,000 calls.

- sequential
  - formatted
  - unformatted
- direct
  - unformatted

File size requirements for ASCII versus binary files.

variable type	# of bytes for binary	ASCII
character	N	N
8-bit integer	1	3
16-bit integer	2	6
32-bit integer	4	11
32-bit floating point	4	13
64-bit floating point	8	23
32-bit complex	8	30
64-bit complex	16	46

It should be noted; however, that one could encrypt binary information into ASCII with a more efficient storage. This might be useful to do in order to send images or other binary file formats over the internet via e-mail. Care must be taken to not use ASCII or binary special symbols that would be interpreted by the transfer protocols or word processors. One method could use the standard ASCII symbols minus special characters to represent groups of 6-bit binary as an 8-bit character. In this way, binary could be encoded, stored and/or transmitted as ASCII and then decoded as binary on the other end. The storage would take 8/6 bytes per byte.

2.5.2 How to read ASCII

The reading of ASCII files requires parsing strings. One useful command in FORTRAN is the internal formatted read.

You can read a file into a string and then parse the string. This allows multiple reads of a line or multiple conditionals.

```
    The file looks like:
    xxxx
    xxxx
    begin xxxxx
    end   xxxxx

    In FORTRAN:

      character*4  c4
      character*80 cline

 100   read(10,'(a)') cline
       read(cline,'(a4)') c4
       if(c4.ne.'begi') goto 100
       if(c4.eq.'end ') goto 200

     Or in IDL:

     cline = ' '
lp1: readf, 1, cline
     reads, cline, c4, format= (a4)   ; <== (a4) was double quotes here
     if(c4 eq 'begi') then goto, lp1

     it can also be done in IDL as follows:

     cline = ' '
lp1:    readf, 1, cline
     c4 = STRMID(cline,0,4)
     if(c4 eq 'begi') then goto, lp1
```

2.5.3 Big endian versus little endian

In Gulliver's travels the lilliputians were divided into two camps. The big-endians ate their eggs by opening the "big" end and the little-endians ate their's by opening the little end.

Well, in our world INTEL and DEC went "little-endian" and stored multi-byte words with the least significant byte at the lowest address. Motorola (Apple, SGI, SUN) went "big-endian" and stores multi-byte words with the most significant byte at the lowest address.

For example, let's say that we have a IEEE floating point value like we discussed in class, say -1.0. In hex this is BF80 0000 where the bits and bytes are ordered from most significant to least significant.

But if this were all there was to it we could all be software engineers. INTEL and DEC needed extra job security so they would write the hexadecimal floating point as 000080BF. Much better.

So if I write a file on an SGI (see homework) and try to read in floating point on a PC, then my binary world collapses. I have to flip all the bytes so the PC can understand the coding of integer's, real's, and character's.

```
                SGI/SUN     PC/DEC
     int*2  +1    0001        0100
     int*4  +1   0000 0001  0100 0000
    real*4  +1.0 3F80 0000  0000 803F
```

Fortran code necessary to undo this:

```
      subroutine swap4(in)
      implicit none
      real*4 in

      real*4 r4a, r4b
      integer*4 j
      integer*1 i1a(4),i1b(4)   !  <-- SGI,SUN,PC
c     byte      i1a(4),i1b(4)   !  <-- HP

      equivalence (i1a,r4a),(i1b,r4b)

      r4a = in
      do j = 1, 4
       i1b(j) = i1a(5-j)
      enddo
      in = r4b

      return
      end

      subroutine swap8(in)
      implicit none
      real*8 in

      real*8 r8a, r8b
      integer*4 j
      integer*1 i1a(8),i1b(8)   !  <-- SGI,SUN,PC
c     byte      i1a(8),i1b(8)   !  <-- HP

      equivalence (i1a,r8a),(i1b,r8b)

      r8a = in
      do j = 1, 8
       i1b(j) = i1a(9-j)
      enddo
      in = r8b

      return
      end
```

Many languages (*e.g.*, MATLAB, IDL, FORTRAN) support big-endian f77 file I/O. Many early PC compilers did not. Their equivalent uses bytes instead of 16-bit integers for the appended words and they

Figure 2.5: Example of a paper tape

didn't allow records greater than 130 bytes. You will need to refer to "BINARY I/O" in the manuals for your language and machine to understand how to read I/O.

2.5.4 Binary versus ASCII I/O

1. If I have a file of a large quantity of data (e.g., the solar spectrum, sampled at 0.01 cm^{-1}, from 1000 to 100,000 cm^{-1}) what kind of I/O should I use (ASCII or BINARY)? Why?

   There are approximately 10 million values. A BINARY file would be the smallest file size and would not require conversion from decimal to binary (a non-trivial amount of computer time) and binary would be an EXACT representation of the digital number.

	size of			
data type	BINARY value (bytes)	BINARY file (MB)	ASCII value (bytes)	ASCII file (MB)
32-bit integer (±2147483647)	4	40	11	110
32-bit (±1.0000001E±38)	4	40	14	140
64-bit (±1.0000000000000001E±308)	8	80	24	240

   However, ASCII is always transportable across platforms.

2.5.5 FORTRAN f77 FORMATS

2.5.6 FITS, NETCDF, and HDF file formats

2.5.7 Storage Media

- Paper tape evolved from mid-19th century telegraphy. It was used until the 1970's. The most popular was 1 inch wide and had 8 holes per row, although 5, 6, and 7 hole versions were available. Hole density allowed 400 rows/meter and mechanical readers could operate at about 10-300 rows/second. Electo-optic readers could read 2000 rows/second.
- paper cards: 8.3 x 18.8 cm card. Cards are about 0.007" thick. 2000 cards/14" box or 55 cards/cm. 80 bytes with parity error detection can be stored on a single card. A stack of cards 1 inch thick would contain about 11.5K bytes.

  2000 cards/box = 160,000 ASCII bytes/box
- 9-track tape: up to 2400' long, 0.5" wide. 800, 1600, and 6250 bits per inch. Data was formatted in blocks and required about 0.6" for inter-block gaps and 3.5" for inter-file gaps. A typical block size was 4800 bytes.

bpi	capacity w/ 2400'	inches per 4800b block	# of blocks	data capacity
1600	46 MB	3.60"	8000	38.4 MB
6250	180 MB	1.37"	21052	101 MB

- Floppy Disks: media thickness = 0.004" package thickness = 0.055".

diameter	drive rpm	tracks /inch	sectors /track	bytes/ sector	# sides	data capacity
SSSD 8"	360	48	26	128	1	160 KB
SSDD 5 $\frac{1}{4}$"	300	80	9	512	1	360 KB
DSDD 5 $\frac{1}{4}$"	300	80	9	512	2	720 KB
DSHD 5 $\frac{1}{4}$"	300	80	15	512	2	1200 KB
3 $\frac{1}{2}$"	300	80	18	512	1	737 KB
ZIP	2941					100 MB

- CD's

  A CD is a 120 mm diameter disk of polycarbonate with a 15mm diameter hole in the center. Tracks are written from the center to the outside in the active data area, from 46 to 117 mm diameter.

  The edge's of pits represents 1's. The pits are 0.5 μm wide and 0.83 to 3.56 μm long. Each track is separated by 1.6 μm. The width of the pit is in the green part of the spectrum.

  Commercial CD players use a AlGaAs laser diode with a wavelength in airs of 780 nm; however, due to the index of refraction (1.55) of the polycarbonate the wavelength in the media is 500 nm. The pits are read from the back of the CD so that the unpitted surface is 1/4 λ further away. Thus light traveling to the unpitted surface exactly 1/2 of a wavelength further than the inverted pit so that the light destructivelyinterfere.

  In both audio and data CD's the signals are processed with error detection and correction algorithms. Audio format is called EFM (Eight to Fourteen Modulation) to minimize 0→1 and 1→0 transitions, thus avoiding small pits. The encoding is governed by a standard called IEC-908.

  Data CD's are broken into 2352 byte sectors which contain a user data size of 2048 bytes. 75 sectors can be read per second giving a fundamental data rate of 4.3218 MB/sec. Data CD's have additional error detection and correction called EDC coding governed by IES-10149. The reason is that a failure of the error correction with audio CD's might make an annoying click; however, in a data CD it could destroy the file.

- 8mm tape

  9.5 x 6.2 x 1.5 cm tape (same as 8mm VIDEO tape). 3 MB/s.

  Sony QC112m is 112 meters and has 2.5/5.0 GB per tape capacity

  Maxell HS-8/160 is 160 meters and has 3.5/7.0 GB per tape capacity

- AIT: Advanced Intelligent Tape (Sony & Seagate)

  8mm media format. Digital 170m tape has 25 GB capacity.

Density of Digital Media			
media	storage capacity	packed size (cm^3)	density (B/cm^3)
1" paper tape	400 B/meter	2	200
80 column cards	160 KB/box	5550	28
8" DSDD floppy	1.2 MB	56	21,000
5 $\frac{1}{4}$" Floppy	1.2 MB	25.1	48,000
3 $\frac{1}{2}$" Floppy	1.2 MB	27.7	52,000
6250 bpi 9-track tape	180 MB/2400'	1290	140,000
ZIP Disk	100 MB/disk	50	2,000,000
RAID Tower	400 GB	17,000	22,600,000
CD	650 MB/disk	25.6	25,400,000
8mm Tape	7 GB/160 meter	88.4	80,000,000
AIT Tape	25 GB/170 meter	88.4	280,000,000

Chapter 3

Solving Sytems of Linear Equations

NOTE: This Chapter was provided by John Blaisdell (GSFC) and is a summary of Chapter X in Heath, 1997.

Goals for this chapter are:

- Remember how matrices work,
- Recognize common applications of matrices, such as solving a set of coupled equations.
- Examine the details of a matrix inversion routine,
- See where you can run into trouble using "canned" routines
- Recognize special types of matrices
- Learn the syntax associated with matrix methods

You will probably never need to WRITE a matrix inversion routine; but it is useful to become familiar with the individual steps that go on in the process of solving a set of simultaneous equations.

The principal goal here is to show you enough that you will be able to select the appropriate function call from a large mathematical library for the application that you have, and to understand what the various error codes it might return actually mean.

3.1 Solving a system of Equations

Suppose you were asked to solve a set of three equations in three unknowns:

$$\begin{array}{lrcrcrcr} \text{Eq. 1:} & 2x_1 & + & 4x_2 & - & 2x_3 & = & 2 \\ \text{Eq. 2:} & 4x_1 & + & 9x_2 & - & 3x_3 & = & 8 \\ \text{Eq. 3:} & -2x_1 & - & 3x_2 & + & 7x_3 & = & 10 \end{array}$$

After grumbling for awhile, you look for ways to add and subtract the equations from each other. This is legal, because you are adding equals to equals. Notice that if one subtracts twice the first equation (Eq. 1) from the second equation (Eq. 2) to get rid of the x_1's. The result is called Eq. 2'.

$$\begin{array}{rrcrcrcr} \text{Eq. 2:} & 4x_1 & + & 9x_2 & - & 3x_3 & = & 8 \\ -2\cdot\text{ Eq. 1:} & -4x_1 & - & 8x_2 & + & 4x_3 & = & -4 \\ \hline = \text{Eq. 2':} & & & x_2 & + & x_3 & = & 4 \end{array}$$

Also one can add Eq. 1 to Eq. 3 to build the third equation:

$$\begin{array}{rrcrcrcr} \text{Eq. 1:} & 2x_1 & + & 4x_2 & - & 2x_3 & = & 2 \\ \text{+ Eq. 3:} & -2x_1 & - & 3x_2 & + & 7x_3 & = & 10 \\ \hline \text{= Eq. 3':} & & & x_2 & + & 5x_3 & = & 12 \end{array}$$

Collecting what we have so far:

$$\begin{array}{rrcrcrcr} \text{Eq. 1:} & 2x_1 & + & 4x_2 & - & 2x_3 & = & 2 \\ \text{Eq. 2':} & & & x_2 & + & x_3 & = & 4 \\ \text{Eq. 3':} & & & x_2 & + & 5x_3 & = & 12 \end{array}$$

Similarly, we can subtract Eq. 2' from Eq. 3' and replace Eq. 3' with the result:

$$\begin{array}{rrcrcrcr} \text{Eq. 1:} & 2x_1 & + & 4x_2 & - & 2x_3 & = & 2 \\ \text{Eq. 2':} & & & x_2 & + & x_3 & = & 4 \\ \text{Eq. 3':} & & & & & 4x_3 & = & 8 \end{array}$$

So Eq. 3" has only one variable, so we can solve it trivially:

$$x_3 = 2 \tag{3.1}$$

Substituting $x_3 = 2$ into Eq. 2': yields

$$\begin{aligned} x_2 + 2 &= 4 && (3.2) \\ x_2 &= 2 && (3.3) \end{aligned}$$

And back-substitute $x_3 = 2$ and $x_2 = 2$ into Eq. 1 will yield:

$$\begin{aligned} 2x_1 + 8 - 4 &= 2 && (3.4) \\ 2x_1 &= -2 && (3.5) \\ x_1 &= -1 && (3.6) \end{aligned}$$

3.2 Gaussian Elimination

This methodology of solution is very algorithmic and should be easy to program. Let's recast the original problem in matrix notation:

$$\mathbf{A}\mathbf{x} = b \tag{3.7}$$

or

$$\begin{pmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 8 \\ 10 \end{pmatrix} \tag{3.8}$$

Remember how matrix multiplies (dot products) work. In general,

$$\sum_j \mathbf{A}_{i,j}\mathbf{B}_{j,k} = \mathbf{C}_{i,k} \tag{3.9}$$

In matrix notation the original problem can be written as

$$\begin{pmatrix} 2 & 4 & -2 \\ 4 & 9 & -3 \\ -2 & -3 & 7 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 8 \\ 10 \end{pmatrix} \tag{3.10}$$

Convince yourself that this matrix equation represents the three equations that we started with. The goal again is to solve for x. All we have to do is rewrite the steps we've already done.

$$\begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ -2 & -3 & 7 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 10 \end{pmatrix} \tag{3.11}$$

$$\begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 1 & 5 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 12 \end{pmatrix} \tag{3.12}$$

$$\begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 8 \end{pmatrix} \tag{3.13}$$

This procedure is call **Gaussian elimination**. The process of solving trivially for x_3, x_2, x_1 is called **back-substitution** because it starts at the end of the array and works backwards toward the first element.

3.3 LU decomposition

The transformations we just made are equivalent to a series of matrix multiplies on both sides of the matrix equation. Changing the second equation by subtracting twice the first corresponds to a matrix multiply by

$$\begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.14}$$

Then we multiplied by

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \tag{3.15}$$

and then by

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \tag{3.16}$$

Each of these is an **elementary elimination** as it causes one of the elements in the matrix we operate on to become zero. The product of these three operations (remember to operate right to left) is

$$\begin{aligned} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} &= \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & -1 & 1 \end{pmatrix} \end{aligned} \tag{3.17}$$

and our Gaussian-eliminated solution (Eq. 3.13) can be re-written using the transformed right hand side of Eq. 3.10

$$\begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & -1 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 8 \\ 10 \end{pmatrix} \tag{3.18}$$

It turns out that an elementary elimination matrix has an easy inverse: change the sign of the single non-zero off-diagonal element. Why is this so?

The inverse of a matrix

$$\mathbf{M} = \mathbf{M}_3 \cdot \mathbf{M}_2 \cdot \mathbf{M}_1 \tag{3.19}$$

is given by the inverses taken in the opposite order:

$$\mathbf{M}^{-1} = \mathbf{M}_1^{-1} \cdot \mathbf{M}_2^{-1} \cdot \mathbf{M}_3^{-1} \tag{3.20}$$

In this case the 3 inverses can be multiplied

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix}$$
$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix} \tag{3.21}$$

and if we multiply both sides of Eq. 3.18 on the left by this matrix we get

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & -1 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 8 \\ 10 \end{pmatrix}$$
$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 8 \\ 10 \end{pmatrix} \tag{3.22}$$

and we are back to our original equation $Ax = b$. But we have expressed A as a product of a **lower triangular** times an **upper triangular** matrix. This is called **LU decomposition** and is a very powerful tool for computation implementation of matrix math routines.

We already saw how to solve for x by backsubstitution in $U \cdot x = M \cdot b$ where U is our upper triangular matrix and M is the inverse of L, the lower triangular matrix. In exactly the same way, you can solve $L \cdot y = b$ by **forward substitution**, starting with x_1 first, and then solve $U \cdot x = y$ by the back-substitution. So here is a complete algorithm to solve the linear system we started with. But it is useful for several other reasons.

Suppose you now want to solve

$$A \cdot x = b' \tag{3.23}$$

where b' is a new vector of constants. This is a very common situation in data reduction where you measure one set of variables to determine a second set of variables. You could go ahead and calculate A^{-1} as we will in a moment and then just do a matrix multiply

$$x = A^{-1} \cdot b' \tag{3.24}$$

However, if you have the LU decomposition available, you only need to do the forward-and- backsubstitutions with the new vector. What else can you do with the LU decomposition? You can directly get the inverse of A. The set of b vectors can be thought of as a matrix B and the set of x vectors as a matrix X

$$A \cdot X = B \tag{3.25}$$

since the definition of matrix multiplication works for each column of X matched to B. A particularly interesting choice of B is the identity matrix I

$$A \cdot X = I \tag{3.26}$$

in which case X will perforce equal A^{-1} and we find the inverse just forward-substituting and back- substituting for each column of I.
How many FLOPS (floating point operations) does the matrix inversion cost? Generally people count the multiplies and assume the additions are quicker, but with modern processors both are important.

- The LU decomposition has about $n^3/3$ multiplies.
- Each call to the forward-and-backsubstitutions has about n^2 multiplies.
- Doing the inverse by columns might require n sets of forward-and-backsubstitutions, or another n^3 multiplies. However, because of all the zeroes in the solution vectors, the number is actually only $2n^3/$

so that the total number of multiplies for the matrix inversion goes as n^3. If you then think of solving $x = A^{-1} \cdot b'$, that matrix multiply requires another n^2 operations.

Therefore, it is more efficient to save the L and U components of A instead of saving A^{-1}, because the forward-and-backsubstitutions are about the same cost as the additional matrix multiply, and you never have to do the matrix inverse in most situations. And very importantly, fewer calculations means less accumulated roundoff error.

3.4 Computing an Inverse Matrix from the LU decomposition

However, there are times you would like to have the inverse, so let's continue and find the inverse of A from our LU decomposition.

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix} \cdot A^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.27}$$

First solve for the product $U \cdot A^{-1}$

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix} \cdot U \cdot A^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.28}$$

by forward-substitution ($X_{11} = 1$; 2 + $X_{21} = 0$; -1 - 2 + $X_{13} = 0$, etc.) to give

$$U \cdot A^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & -1 & 1 \end{pmatrix} \tag{3.29}$$

and then solve this equation for A^{-1}

$$\begin{pmatrix} 2 & 4 & -2 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix} \cdot A^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & -1 & 1 \end{pmatrix} \tag{3.30}$$

by back-substitution we can find the components of the inverse
$4 \cdot A^{-1}_{13} = 3 \quad \rightarrow \quad A^{-1}_{13} = 3/4$
$A^{-1}_{12} + A^{-1}_{13} = -2 \quad \rightarrow \quad A^{-1}_{12} = -11/4$
$2 \cdot A^{-1}_{11} + 4 \cdot A^{-1}_{12} - 2 \cdot A^{-1}_{13} = 1 \quad \rightarrow \quad A^{-1}_{11} = 27/4$
etc., to get

$$A^{-1} = \begin{pmatrix} 27/4 & -11/4 & 3/4 \\ -11/4 & 5/4 & -1/4 \\ 3/4 & -1/4 & 1/4 \end{pmatrix} \tag{3.31}$$

You should verify that this is indeed the inverse. This has all been presented as a cookbook approach. Can anything go wrong? The answer is yes, in general there need to be several refinements added for this computation to work for most matrices. We have not taken any care to limit growth of roundoff errors, as we have calculated everything exactly. For large matrices, accumulated error is of great importance and commercial routines do everything they can to prevent it.

The two major categories of refinement involve **pivoting** and **scaling**. You also have to worry about discovering **singular** matrices; that is, matrices which have no inverse. There is no black magic here; doing it by hand these are things you would naturally do anyway if you had your wits about you.

Suppose you get partway through a LU decomposition and you find a zero on the diagonal:

$$\begin{pmatrix} 1 & 2 & -2 & 1 \\ 0 & 0 & 1 & -3 \\ 0 & 5 & 0 & -3 \\ 0 & 7 & 2 & -3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 7 \\ 5 \\ 3 \\ -9 \end{pmatrix} \tag{3.32}$$

Because of the zero in the (2,2) spot, you can't eliminate the element in the (3,2) spot. If you were doing it by hand, you might see to swap the second and fourth equations because it really doesn't matter what order you wrote them down in. Putting the 7 on the diagonal means multiplying by a number less than 1 to eliminate the 5, and multiplying by a smaller number limits the error growth.

Swapping rows of the matrix is called **pivoting** and it is NECESSARY for Gaussian elimination to be stable. You can also pivot columns if you want to, but that scrambles the order of the x's and is more work to keep track of.

A second kind of numerical problem occurs because the coefficients differ in magnitude:

$$\begin{pmatrix} 1 & 2 & -2 \\ 5 & 9 & 6 \\ 0.0001 & 0.0003 & 0.0010 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 3 \\ -2 \\ 0.0005 \end{pmatrix} \tag{3.33}$$

You might very well multiply the third row through by 10000 before you started. This is called **scaling** and there are many variations in how to implement it. A common scaling method is to make the largest coefficient in each row unity.

What if you find a whole row or column is zero, so you can't find anything to pivot? You are in trouble. This arises from cases like

$$\begin{aligned} x_1 + x_2 &= 3 \\ x_1 + x_2 &= 4 \end{aligned} \tag{3.34}$$

and when you subtract the equations, you get

$$0 = 1 \tag{3.35}$$

Obviously there is no solution because the matrix is **singular**:

- Its determinant is zero
- It has no inverse
- Its rows (and columns) are not linearly independent

You may not discover that the matrix is singular until you are quite far along in the calculation.

At the end of this chapter are two published routines, with some added comments, which do the LU decomposition and forward-and-backsubstitutions. They are published in the book Numerical Recipes (Press *et al.* 1986). There is every reason to think that canned matrix routines in any package will have more or less the same components. These example routines handle pivoting and scaling well, but they try to continue in the case of singular matrices, assuming they might be singular only because of roundoff. This does not seem to be the ideal plan, but the only changes I have made are added comments.

Notice that you have to study the calling sequences very carefully. In the good old days, memory usage was extremely important. So the clever folks writing these routines found ways to reuse storage. In the LU decomposition routine, the clever folks pack L and U into one matrix so as not to store all those zeroes. This requires assuming that the L matrix has a unit diagonal, so that doesn't need to be stored either. For a 3x3 matrix, they pass back

$$\begin{pmatrix} U_{11} & U_{12} & U_{13} \\ L_{21} & U_{22} & U_{23} \\ L_{31} & L_{32} & U_{33} \end{pmatrix} \tag{3.36}$$

and just for good measure, they do the calculations in a tricky order so that they can overwrite A in place! So if you still want your original matrix, you have to store it yourself. Also, they wreorder the equations for pivoting as they go along, storing the permutation as an index array so that you can undo it later. Not all packages will be so unfriendly, but you should be prepared for apparently unusual setups and read the documentation. . .

To solve a linear set of equations $A \cdot x = b$ you call the routines as follows

```
! NOTE: FORTRAN stores arrays by columns, not true for C
      real*4 A(np,np)  ! input matrix
      integer*4 indx(np)
      real*4    D, B(np)

      call LUDCMP(A,n,np,indx,D)
      call LUBKSB(A,n,np,indx,B)
```

where A is the original matrix dimensioned as A(n,n) with np elements filled, indx is a dummy integer array dimensioned to indx(n). x is returned in b; the original matrix A is no longer intact.

To find the inverse of a matrix the routines in section B.7 and B.8 are used as follows:

```
      real*4    A(np,np)  ! input matrix
      real*4    y(np,np)
      integer*4 indx(np)
      real*4    D, B(np)

      do i = 1,n       ! build an identity matrix
        do j = 1,n
          y(i,j) = 0.0
        enddo
        y(i,i) = 1.0
      enddo
      call LUDCMP(A,n,np,indx,d)        ! decompose A, just once
      do j=1,n                        ! loop over columns
        call LUBKSB(A,n,np,indx,y(1,j)) ! find one column of inverse
      enddo
```

Now to touch briefly on some related topics. You may find matrix math routines that use these terms and it helps to have seen them defined.

Gauss-Jordan elimination starts just like Gaussian elimination, but also eliminates the elements ABOVE the diagonal in the U matrix, leaving a diagonal matrix. It is then trivial to read off the solution vector and the inverse is created on the other side of the equals sign. This method is equally efficient (n^3) for the matrix inversion but is not as efficient for the solution of the linear system.

Cholesky factorization applies only to **symmetric positive definite** matrices. It is a LU decomposition where U is the transpose of L. It requires only about half as many steps as a general LU decomposition.

Band matrices have non-zero elements only on the diagonal and n rows above and below the diagonal. In particular the **tridiagonal** matrix is a band matrix with only three non- zero entries in each row, on and adjacent to the diagonal. Tridiagonal matrices are an important reduction step in eigenvalue problems. In general, not having to store all the zeroes makes larger matrices tractable.

Hessenberg matrices are almost triangular; they have non-zero elements one row beyond the diagonal.

The **Vandermonde** matrix has columns whose elements are successive powers. It is useful in linear least squares fitting.

$$\begin{pmatrix} 1 & a & a^2 \\ 1 & b & b^2 \\ 1 & c & c^2 \\ 1 & d & d^2 \end{pmatrix} \tag{3.37}$$

The **Toeplitz** matrix has the following form:

$$\begin{pmatrix} E & F & G & H & I \\ D & E & F & G & H \\ C & D & E & F & G \\ B & C & D & E & F \\ A & B & C & D & E \end{pmatrix} \tag{3.38}$$

A good example of Toeplitz matrices is given by the blurring operator in image processing.

3.5 Norms and Condition Numbers

A **norm** is a measure of the size of a vector or a matrix. At least three different norms are in common usage:

For vectors:

the 2-norm x 2, is defined as the usual vector length, the square root of x x

the 1-norm x 1, is defined as the sum of the absolute values of the elements

the 0-norm x 0, is defined as the largest absolute value of the elements

For matrices, each norm is defined in terms of the corresponding vector norm:

$$A = MAX(A \cdot x/x) \tag{3.39}$$

where the maximum is taken over all non-zero x.

We are primarily interested in norms because they determine the **condition number** of a matrix:

$$cond(A) = A \cdot A - 1 \tag{3.40}$$

The larger the condition number, with respect to whichever norm is being used, the more **ill-conditioned** a matrix is. By convention, a singular matrix has condition number; the identity matrix can be seen to have condition number 1.

The condition number measures how close a matrix is to being singular and is a useful tool in assessing the accuracy of solutions to linear systems.

Two additional applications of matrices will follow later in the course. These are LINEAR LEAST SQUARES FITTING and SINGULAR VALUE DECOMPOSITION or solving for EIGENVALUES. Although many of the steps involved in those problems look like those we have covered today, try to keep them

distinct in your mind. We have been working with any old matrix multiplication because we have operated on both sides of an equation. Singular value decomposition works only on a matrix, not on an equation, and only similarity transformations are allowed operations. Least squares fitting is an extension of the linear systems solution to non- square matrices, where there are more (or fewer) equations than unknowns.

Chapter 4

Sampling of Data & Interpolation

4.1 Discretization of Continuous Functions and Nyquist Sampling

- Information is lost when a continuous function, $f(x)$ is sampled at a discrete number of points.
- "Aliasing" can occur if continuous function is periodic with a higher frequency than the sampling interval. Low sample rates convert the higher frequency into a lower frequency.
- Must sample with enough points to represent highest frequency within the data.

We will assume that the tabulated points are **sampled** on a uniform spacing.

$$f_d(n) \quad \text{associated with} \quad x(n) = n \cdot \Delta x \quad \text{for} \quad n = -\infty, \ldots, -1, 0, +1, \ldots, \infty \tag{4.1}$$

The relationship between the sampled data, $f_d(n)$, and the continuous function, $f(x)$, can be written mathematically as

$$f_d(n) = \sum_{n=-\infty}^{\infty} f(x) \cdot \delta(x - n \cdot \Delta x) \tag{4.2}$$

where $\delta()$ is the Dirac delta function, which has the property

$$\int_x f(x) \cdot \delta(x_0, x) \cdot dx = f(x_0) \tag{4.3}$$

4.1.1 Intuitive Sampling Theorem

We will begin with an intuitive argument for the maximum frequency that can be sampled by discretely sampled data. First assume we have a simple cosine function

$$y = \cos(2\pi \cdot f \cdot x) \tag{4.4}$$

where f is the frequency which represents the number of cycles per unit value of x. In the following figure we show the case when $f = 5$ cycles over the range of $0 \leq x \leq 1$ as a solid line. If we sampled the data with two points per sample, $\Delta x = 0.1$, then we could reasonably reproduce the cosine wave, shown as a

dotted line between the points. If we required 2 points per cycle then the minimum sampling would be given by

$$\Delta = \frac{1}{2 \cdot f_{max}} \tag{4.5}$$

Given a fixed sampling, for example $\Delta x = 0.1$, if we exceed the maximum frequency given by Eqn. 4.5 then the higher frequencies will be measured as lower values. In the second panel of the following figure a frequency of $f = 10$ is shown as a solid line and you can see that the sampled signal appears exactly like the $f = 5$ case. In the final panel we show a frequency of 6 cycles as a solid line and the measured sampling as a dotted line. This effect is called *aliasing* and occurs whenever a signal is *undersampled*.

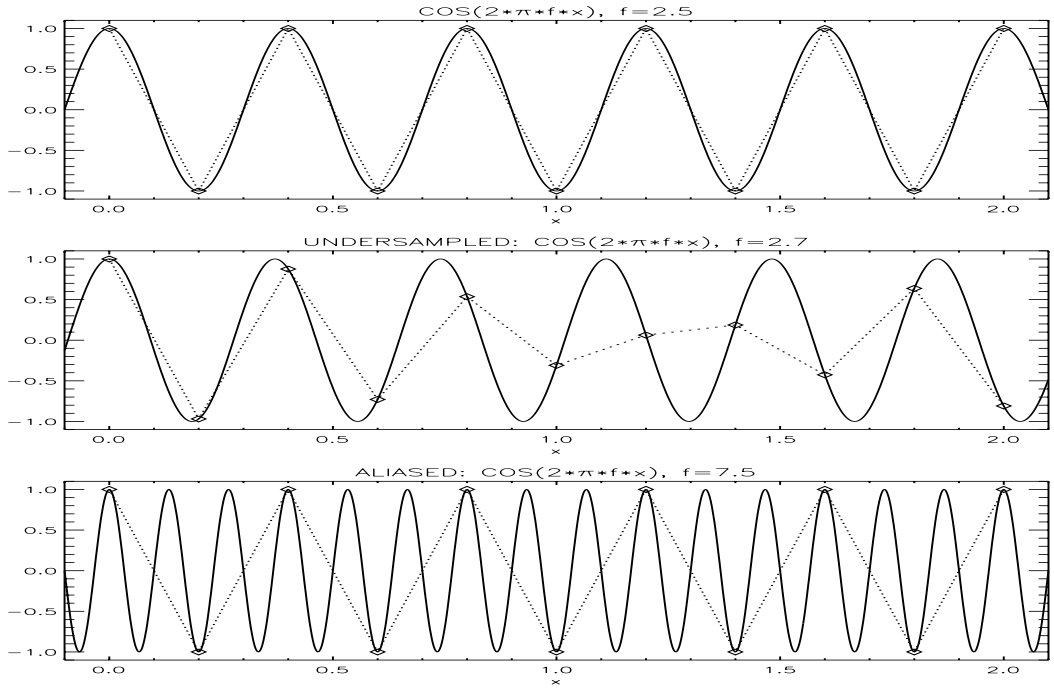


Figure 4.1: Illustration of an intuitive argument for Nyquist sampling

In Fig. 4.1 we shown an illustration of the intuitive argument for the maximum frequency represented by discretely sampled data. The Nyquist sampling theorem, given by Eqn. 4.5 is an important concept in computational physics and deserved to be discussed in more detail.

4.1.2 Formal Sampling Theorem

To understand the information content of a discretely sampled function versus it's continuous counterpart we can Fourier transform the continuous and discrete functions and analyze the information content in the frequency domain. The continuous Fourier transform is defined as

$$F(k) = \frac{1}{\sqrt{2 \cdot \pi}} \int_{-\infty}^{\infty} f(x) \cdot e^{i \cdot k \cdot x} \partial x \tag{4.6}$$

Example Transform Pairs

spectral(ν)	spatial(x)
$f(\nu) = 2\int_0^\infty F(x)\cdot\cos(\pi\cdot\nu\cdot x)dx$	$F(x) = \int_{-\infty}^{\infty} f(\nu)\cdot\cos(\pi\cdot\nu\cdot x)d\nu$
$f(\nu) = \exp(-a\cdot\nu^2)$ $a = -4\log_e(0.5)/\mathrm{FWHM}_\nu^2$	$F(x) = B\cdot\exp(-b\cdot x^2), B = \sqrt{a/\pi}$ $b = -4\log_e(0.5)/\mathrm{FWHM}_x^2 = \pi^2/a$ $\mathrm{FWHM}_x = -4\log_e(0.5)/\pi/\mathrm{FWHM}_\nu$
$f(\nu) = 2\cdot[x\cdot(\nu-\nu_0)+x\cdot(\nu+\nu_0)]$	$F(x) = \cos(\pi\cdot\nu_0\cdot x)$

The continuous Fourier transform will have a continuous frequency spectrum and, in general, will not have any upper frequency bound. For example, the Fourier transform of a Gaussian function has a Gaussian envelope in the frequency domain. Since the k-space Gaussian never truly goes to zero, this implies that there is an infinite frequency spectrum for that function. For discretely sampled function, we will compute the continuous Fourier transform as follows:

$$F(k) \simeq F_d(k) = \frac{1}{\sqrt{2\cdot\pi}}\int_{-\infty}^{\infty}\sum_{n=-\infty}^{\infty} f(x)\cdot\delta(x-n\cdot\Delta x)\cdot e^{i\cdot k\cdot x}\cdot\partial x \tag{4.7}$$

$$F_d(k) = \frac{1}{\sqrt{2\cdot\pi}}\sum_{n=-\infty}^{\infty} f_d(n)\cdot e^{i\cdot k\cdot n\cdot\Delta x} \tag{4.8}$$

Now we notice that $F_d(k)$ is a periodic function of k for the discretized function but not for the continuous Fourier transform, $F(k)$. Replacing k with $k+\frac{2\pi\cdot m}{\Delta x}$ in Eqn. 4.8 for an arbitrary value of m yields

$$F_d\left(k+\frac{2\pi\cdot m}{\Delta x}\right) = \frac{1}{\sqrt{2\cdot\pi}}\sum_{n=-\infty}^{\infty} f_d(n)\cdot e^{i\cdot k\cdot n\cdot\Delta x}\cdot e^{i\cdot\frac{2\pi}{\Delta x}\cdot n\cdot\Delta x\cdot m} \tag{4.9}$$

but, the discrete orthogonality condition results in

$$e^{i\cdot\frac{2\pi}{\Delta x}\cdot n\cdot\Delta x\cdot m} = e^{2\pi\cdot n\cdot m} \equiv 1 \qquad \text{for all integer values of } n \text{ and } m \tag{4.10}$$

Therefore, the Fourier transform of the discrete function is periodic in k with a period equal to $\frac{2\pi}{\Delta x}$.

$$F_d\left(k+\frac{2\pi\cdot m}{\Delta x}\right) = F_d(k) \tag{4.11}$$

Frequency is given by $k = 2\pi\cdot f$. Therefore, sampling limits the maximum frequency "band-width" to

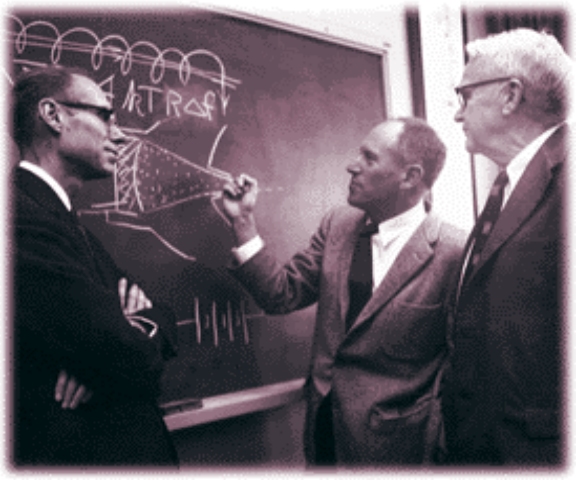
$$f_c = \frac{\pm 1}{2\cdot\Delta x} \tag{4.12}$$

Conversely if f_{max} is the maximum frequency component in $f(x)$ then we can completely determine $f(x)$ by sampling at an interval of

$$\Delta x \le \frac{1}{2\cdot f_{max}} \tag{4.13}$$

In many physical applications the system is bandwidth limited by either detector performance, amplifiers, etc. There is usually a practical upper limit of sample, beyond which additional sampling is unnecessary.

Only sampling two points per cycle; however, can be dangerous if the system is sensitive to out of band signal. This occurs for many interferometer designs. If an interferometer is designed to measure in a certain band, one might sample that band at the desired Nyquist frequency. Out of band signals can propagate into the band due to non-linear amplifier and optical effects (*for example* the beam-splitter). Therefore, one may want to sample the raw interferogram at many times the Nyquist sampling to isolate the in-band signal (remove the out-of-band signal via Fourier transforms or digital filters) and then transmit the filters in-band signal at the Nyquist sampling.



Harry Nyquist (1889-1976, right) talking with John Pierce (left) and Rudolf Kompfner (center) at Bell Labs in 1960. In 1928 Nyquist described his sampling theory in a paper entitled "Certain Topics in Telegraph Transmission Theory"

Figure 4.2: Picture of Harry Nyquist

Example

$$f(x) = e^{-\left(\frac{x-1}{0.2}\right)^2} \tag{4.14}$$

The transform of the continuous function is a Gaussian with an infinite frequency spectrum. If we sample the Gaussian with data spaced at $\Delta x = 0.2$ then the highest frequency we can capture is $k_{max} = \frac{2\pi}{\Delta x}$ which in this case is a critical frequency of $f_c = \frac{1}{2 \cdot \Delta x} = 2.5$ cycles per value of x. This is illustrated in Fig. 4.3

Example of the power spectrum of the transformation of a Gaussian for a continuous transform (solid line) and a discrete transform (dashed line).

4.2 binning, sampling, re-sampling

Sometimes data is acquired with over-sampling or on a sub-optimal set of points. The techniques described in this section are used to alter the sampling of data.

First, if the data is noisy or too-large a volume for modeling it can either be sub-sampled, which throws away information, or binned. Binning can capture more information within an interval; however, it will still lower the maximum frequency within the dataset. One would chose binning in problems where the integral of the data is more important (*e.g.*, energy transport, heat balance, etc). For example, in Fig. 4.4 we show a high spectral resolution spectrum (sampling = $\Delta\nu = 0.25$ cm^{-1} that has bin binned to 10 cm^{-1}.

For example, if the instrument is a spectrometer the data can be binned to simulate a lower resolution instrument and still maintain accuracy in terms of the total energy contained in the spectrum. See the IDL program "binner.pro" in Section B.1.

If the data is Nyquist sampled, then there are techniques to re-sample the data via Fourier transforms. If the original data is transformed into k-space and then that space it zero filled to a larger frequency the original data content will be unchanged, but upon transforming the data will be sampled at a finer sampling. For example, if the k-space is zero filled by a factor of 10 the upon computing the inverse transform value of Δx will be reduced by a factor of 10. This form of re-sampling is not an interpolation, but is equivalent to sampling the original instrument at those points. Sampling at the Nyquist sampling is identical to an instrument with a resolution equal to twice the value of Δx. For a complicated spectrum, for example, interpolation could not capture the effects of a deep absorption feature whereas re-sampling would.

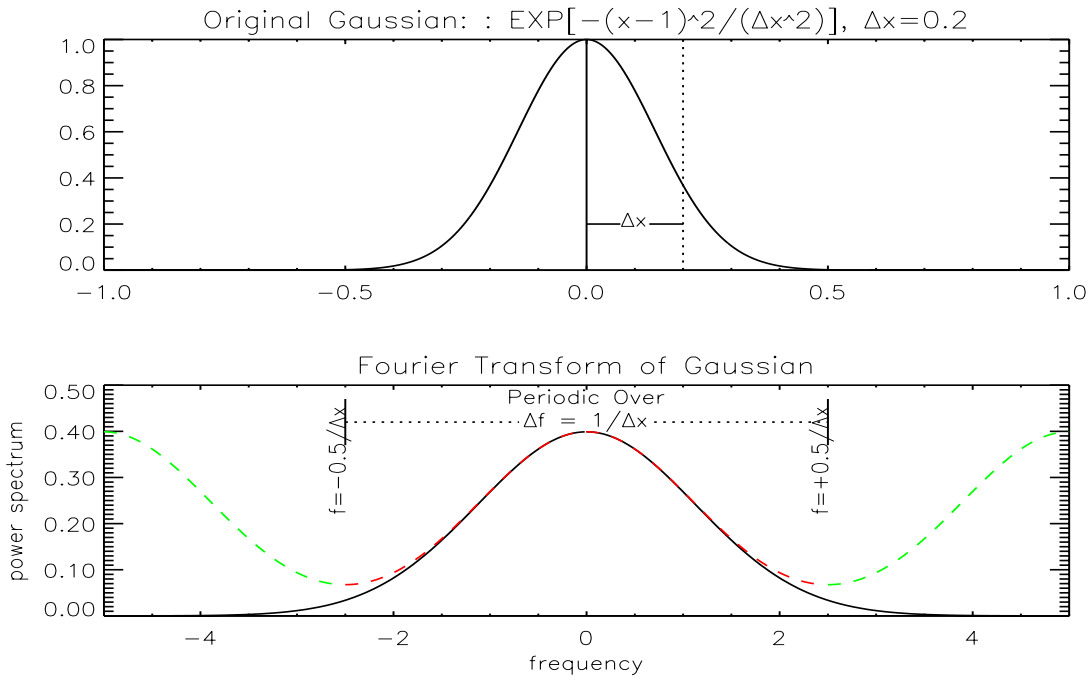


Figure 4.3: Example of the power spectrum of a Fourier transform of a Gaussian

4.3 Interpolation of discrete data

By definition, interpolation within a set of data points, $x(n), y(n)$ must fit the given data points exactly. This is in contrast to approximation, discussed earlier.

Why do we need to determine $f(x)$ between sampled points?

- merging data sets which are sampled on different grids
- interpolating a measurement between calibrated values *e.g.*, thermistors, thermocouples, pressure transducers
- interpolating a table of measurements for the purpose of plotting or determining properties such as derivatives.
- extrapolating outside a finite range of data. NOTE: that for higher order polynomials this is a dangerous thing to do.

When reviewing the methods you will see many names of great mathematicians and physicists. This is because, they were using interpolation methods to extend logarithm and trigonometric functions. There are methods by Gregory & Newton (1642-1727), Gauss (1777-1855), Sterling, Bessel (1784-1846), Everett, and Lagrange (1736-1813), just to name a few.

When choosing an interpolating function consider the following:

- What form should the function have. If there is a physical basis or knowledge of the coarse behavior then this can be the basis within the interval. For example:
  - sometimes it is best to interpolate in the log() of a parameter that is varying rapidly.
  - sometimes a parameter needs to be converted to a monotonic function to remove rapidly varying terms.

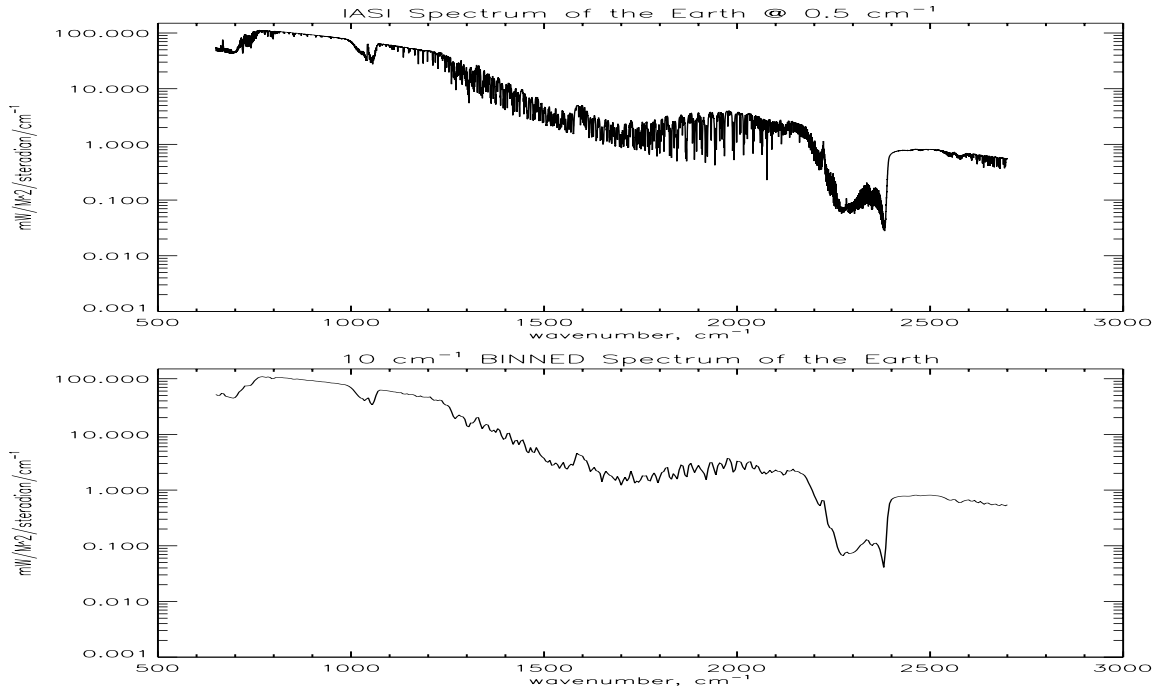


Figure 4.4: Example of re-sampling of data via binning. Resolution is reduced by factor of 40 while maintaining the spectral integral

4.4 Example of interpolating atmospheric moisture

Radiation transfer models (neutrons, stars, planets) typically represent the distribution of absorbers on a fixed grid. If we need to interpolate the modelled values it is best to first convert the functions to a monotonically increasing function and, if the function to be interpolated is highly non-linear, it is best to attempt to linearize the function before applying the interpolation.

In the following figure an example from Earth atmospheric modelling is shown. Moisture on the Earth is a strong function of altitude due to the rapid decrease of temperature with altitude and surface boundary effects (*i.e.*, interaction with lakes and oceans in the convective part of the atmosphere). The first panel shows moisture as it is represented in the model, $\Delta c(L)$, where the number of molecules/cm^2 of H_2O per atmospheric layer is given. The atmospheric layers are shown by the dotted horizontal lines. This function has a lot of vertical structure and would be difficult to interpolate reliably.

We can build a monotonic function by summing the layers down to a level. This is the number of molecules/cm^2 from the top-of-atmosphere (TOA, L=1) to level L is given by $c(L)$ and is related to the layer column density by

$$c(L) = \sum_{i=1}^{L} \Delta c(i) \tag{4.15}$$

In the case of moisture this is still a difficult, albeit monotonic, function to interpolate. Therefore, it is best to perform the interpolation in $\log_e(c(L))$ space. This function is shown in the 3$^{\text{rd}}$ panel. A simple linear interpolation can be performed and then the inverse processes can be applied, if desired.

Conversion to/from mixing ratio's from/to column densities are typically done via this method to ensure

- the atmospheric structure can be accurately represented
- kinks and spurious "features" can be avoided.

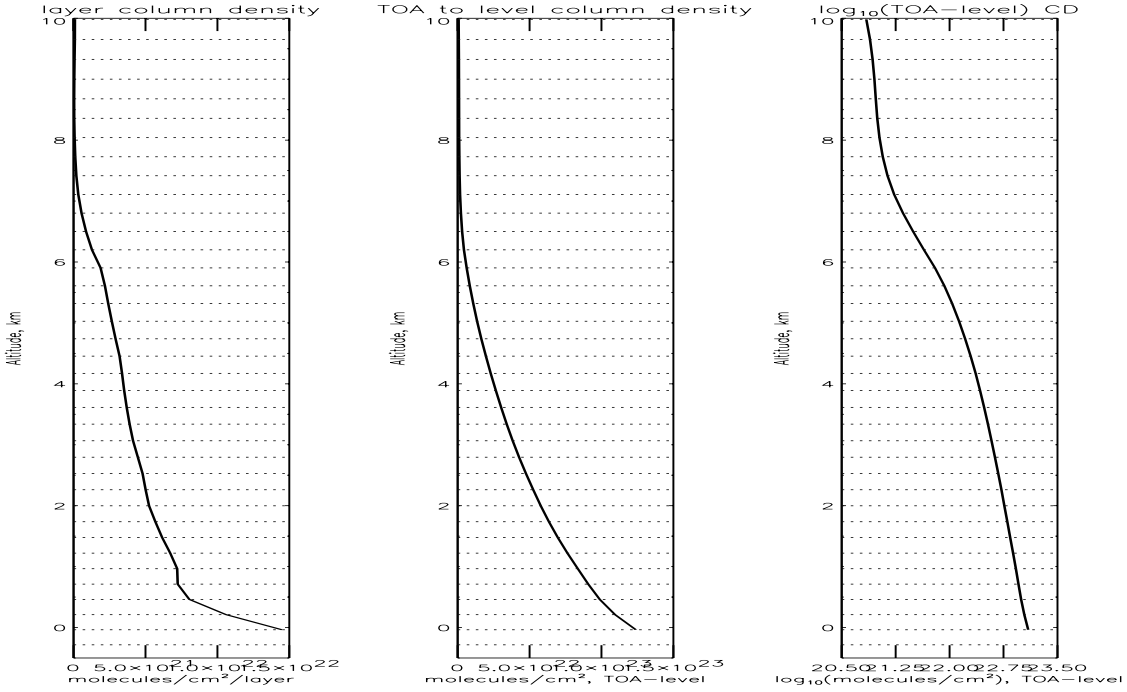


Figure 4.5: Example of converting a function to a quasi-linear monotonic function prior to interpolating

4.5 polynomial approximation (over the full tabulated range)

For N data points we can fit exactly to a $M = N - 1$ polynomial. While higher order polynomials will fit the data well, they can be quite unstable inside and outside the range of the tabulated data. The order of the polynomial can be reduced, in which case the least square solution to the data can be computed. This will probably not exactly represent the original data; however, for tabulated data with measurement error this may not be a criterion for the interpolation.

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_N \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^M \\ 1 & x_2 & x_2^2 & \dots & x_2^M \\ 1 & x_3 & x_3^2 & \dots & x_3^M \\ 1 & \dots & \dots & \dots & \dots \\ 1 & x_N & \dots & \dots & N_N^m \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \dots \\ a_M \end{pmatrix} \tag{4.16}$$

which can be written as a matrix equation

$$Y_n = X_{n,j} \cdot A_j \tag{4.17}$$

The unweighted least squares solution (see Chapter 13, Eqn. 13.30 or Eqn. 13.36 with $W_{n,n} = I_{n,n}$) is given by

$$A_j = \left[X_{j,n}^T \cdot X_{n,j}\right]^{-1} \cdot X_{j,n}^T \cdot Y_n = A_j \tag{4.18}$$

where, $X_{j,n}^T$ is the transpose of $X_{n,j}$. Care must be taken to ensure that the covariance of the *Vandermonde* matrix, $B_{j,n}' \cdot B_{n,j}$ is an invert-able matrix. For functions which are monotonically increasing the polynomial method can work well for a small tabulated array of data (*e.g.*, a set of calibration points for thermistors, thermocouples, etc.).

The polynomial method will work for physical fitting functions. In one knows the form of the function, $Y(x)$, then a least squares fit can be performed to that function. If the function can be written as a linear equation then it can be solved with the methods above. For example, if we were fitting to

$$Y_n = a_0 \cdot \sin(\pi \cdot x_n) + a_1 \cdot \cos(\pi \cdot x_n) \tag{4.19}$$

$$B_{j,n} = \begin{pmatrix} \sin(\pi \cdot x_1) & \cos(\pi \cdot x_1) \\ \sin(\pi \cdot x_2) & \cos(\pi \cdot x_2) \\ \cdots & \cdots \\ \sin(\pi \cdot x_N) & \cos(\pi \cdot x_N) \end{pmatrix} \tag{4.20}$$

If the function can not be written as a linear set of equations, for example,

$$Y_n = b_0 \cdot \sin(a_0 \cdot x_n) + b_1 \cdot \cos(a_0 \cdot x_n) \tag{4.21}$$

then the coefficients must be solved for via non-linear least squares analysis, which is a topic to be discussed in Chapter 14. This method must be applied with caution.

- **Reduce the number of free parameters to only physically meaningful parameters.**
  - **Eliminate redundancy of parameters.**
  - **Eliminate parameters which measurements are insensitive to.**
- **Always plot the interpolation values to look for instabilities.**

For example, in fitting a Mössbauer spectrum one could chose to fit the intensity, position, and width of each spectral line. For a spectrum with Zeeman line splitting this could result in many parameters. Instead, one could solve for the magnetic field strength, B, a line broadening parameter related to the temperature of the sample, $\gamma(T)$, and a single intensity function.

4.5.1 Example of Polynomial Fits to the Runge function

In the figure below we show a polynomial fit to the Runge function. This function is chosen because it is a very **difficult** function to fit with a polynomial.

$$\text{Runge}(x) = \frac{1}{1 + 25 \cdot x^2} \tag{4.22}$$

for a tabulated array of values (without noise) at 11 points. A 5^{th}, 9^{th}, and 10^{th} order polynomial fit is made to these points. The Runge function is difficult to fit with a polynomial and it illustrates the failure of this method to provide a reasonable interpolation.

We can improve the fit at the endpoints by using a non-uniform spacing. The Chebyshev spacing, defined by

$$x(k) = -\cos\left(\frac{(2k-1)\cdot\pi}{2n}\right) \tag{4.23}$$

k	equal $x(k)$	Chebyshev $x(k)$	k	equal $x(k)$	Chebyshev $x(k)$
1	-1.0	-0.98982	7	+0.2	+0.28173
2	-0.8	-0.90963	8	+0.4	+0.54064
3	-0.6	-0.75575	9	+0.6	+0.75575
4	-0.4	-0.54064	10	+0.8	+0.90963
5	-0.2	-0.28173	11	+1.0	+0.98982
6	+0.0	+0.00000			

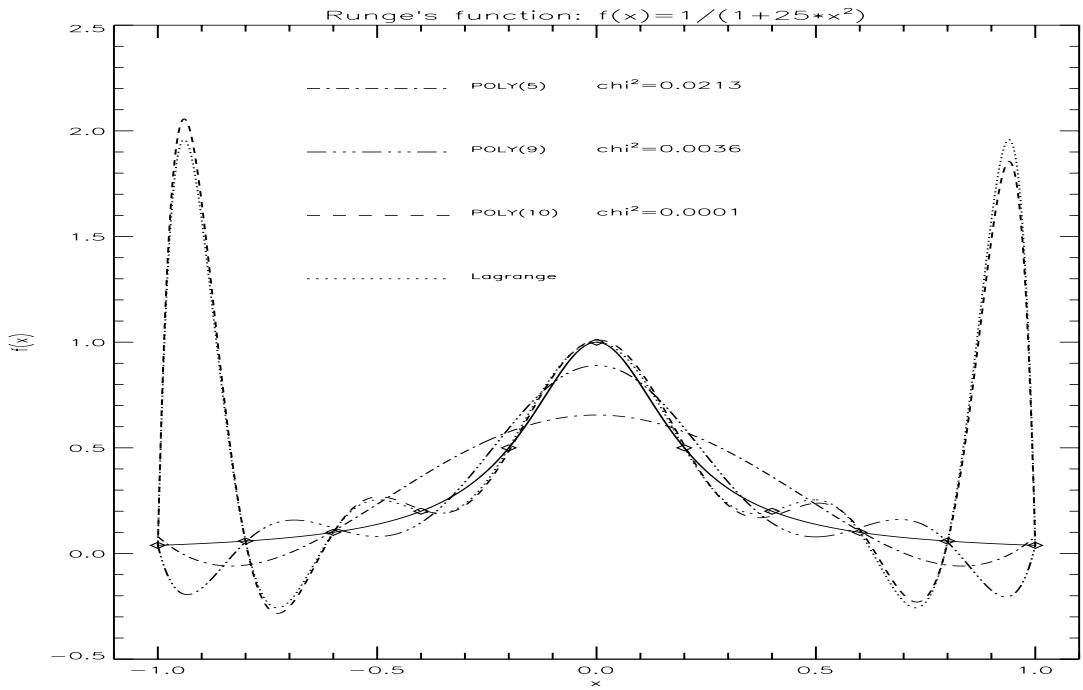


Figure 4.6: Example of polynomial fits to a Runge function

4.5.2 Example of program to compute the position of the Sun

In section B.19 is an example of using polynomial interpolation to provide an accurate position (few arc seconds = .7 parts per million = real*4 accuracy) of the Earth (or Sun) over a large range of time (-4000 to +2800). The author of this code (Bretagnon, 1986) utilized numerical integrations of the multi-body solar system to compute accurate positions and then fit the orbital components to polynomials of the cyclic arguments. The result is a simple, yet accurate, routine.

4.6 Piecewise linear interpolation

To linearly interpolate between two points, $(x_1, y_1), (x_2, y_2)$ we can fit a line to the two points. This is the simplest method of interpolation but the derivatives resulting interpolation are not continuous.

To derive the equation for linear interpolation we can "pivot" about the first point in the pair, (x_1, y_1), we see that the slope is given by $(y_2 - y_1)/(x_2 - x_1)$

$$y(x) = y_1 + (y_2 - y_1) \cdot \frac{x - x_1}{x_2 - x_1} \tag{4.24}$$

We could have also solved the two equations for our pair of points simultaneously to determine the coefficients of the polynomial.

$$y_1 = a_0 + a_1 \cdot x_1 \tag{4.25}$$
$$y_2 = a_0 + a_1 \cdot x_2 \tag{4.26}$$

by substituting the equations for y_1 for the a_0 term in the the equation for y_2, as follows:

$$y_2 = [y_1 - a_1 \cdot x_1] + a_1 \cdot x_2 \tag{4.27}$$

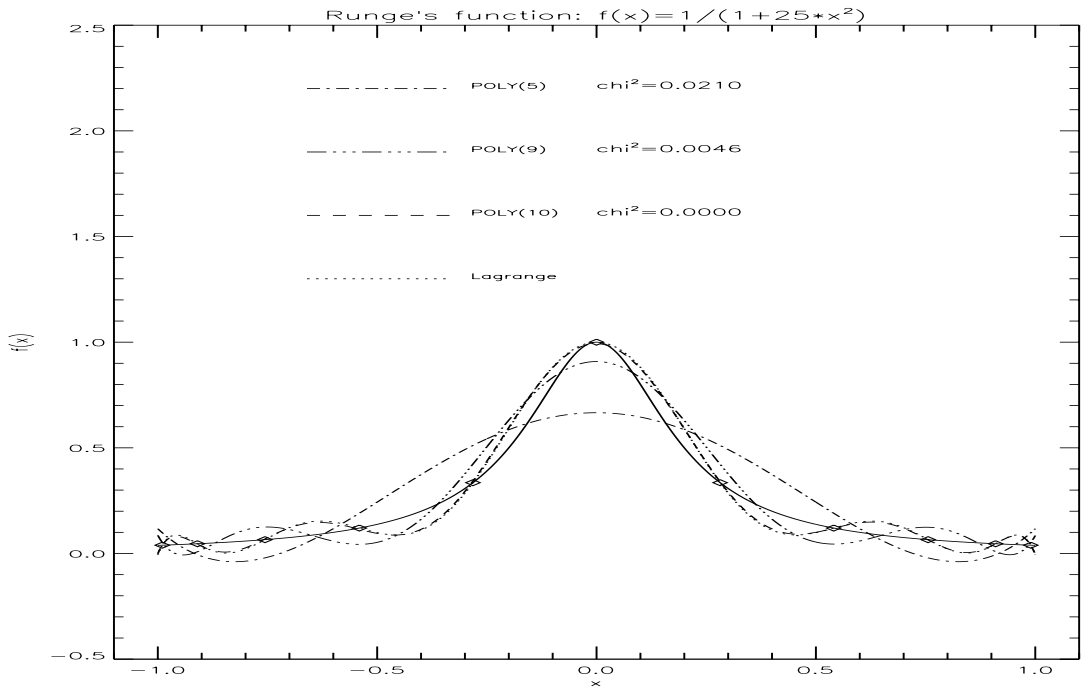


Figure 4.7: Improving the fit to a Runge function by using Chebyshev spacing

$$y_2 - y_1 = a_1 \cdot (x_2 - x_1) \tag{4.28}$$

$$a_1 = \frac{y_2 - y_1}{x_2 - x_1} \tag{4.29}$$

$$a0 = y_1 - a_1 \cdot x_1 \tag{4.30}$$

$$a0 = y_1 - x_1 \cdot \frac{y_2 - y_1}{x_2 - x_1} \tag{4.31}$$

If we re-arrange Eqn. 4.24 we can get the same solution

$$y(x) = y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1} \right) \cdot (x - x_1) \tag{4.32}$$

4.7 Double Linear Interpolation

We desire to find an arbitrary value, $z(x, y)$ from a tabulated grid of values $z(i, j)$. In Fig. 4.9 the •'s represent the tabulated locations. Double linear interpolation first computes the interpolated values along one axis, $z(x, y_j)$ and $z(x, y_{j+1})$, and then interpolates those computed values to the desired location, $z(x, y)$.

In image processing it is often necessary to interpolate between the pixels of the image. If the image is arranged as Y lines consisting of X samples per line, then if we wish to interpolate the brightness (called "DN") to a value with the grid box defined by (x_1, y_1) and (x_2, y_2) we first compute the offset from (x_1, y_1) as follows

$$\Delta x = \frac{x - x_1}{x_2 - x_1}, \qquad 0 \le \Delta x \le 1 \tag{4.33}$$

$$\Delta y = \frac{y - y_1}{y_2 - y_1}, \qquad 0 \le \Delta y \le 1 \tag{4.34}$$

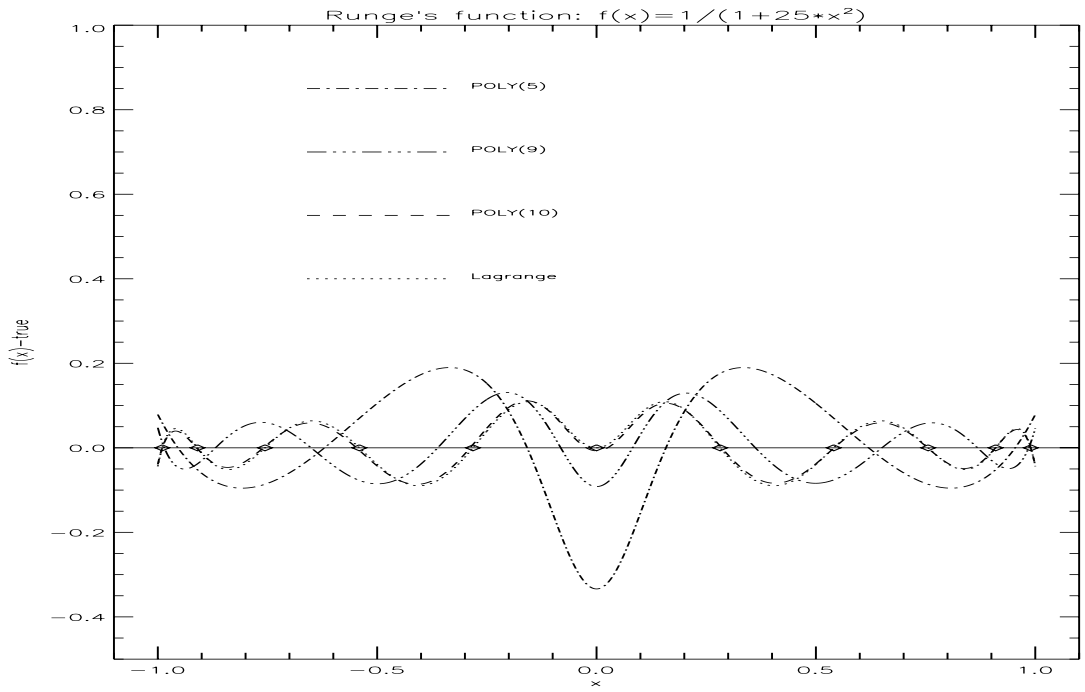


Figure 4.8: Error in Runge fit w/ Chebyshev spacing (difference of Fig. 4.7 from truth)

we also assume that we are counting pixels, such that $x_2 = x_1 + 1$ and $y_2 = y_1 + 1$.
The linear interpolation along the upper and lower line is given by

$$DN(x, y_1) = DN(x_1, y_1) + \Delta x \cdot (DN(x_2, y_1) - DN(x_1, y_1)) \quad (4.35)$$
$$DN(x, y_2) = DN(x_1, y_2) + \Delta x \cdot (DN(x_2, y_2) - DN(x_1, y_2)) \quad (4.36)$$

and we can write the interpolated value at any given point within this grid box as

$$DN(x, y) = DN(x, y_1) + \Delta y \cdot (DN(x, y_2) - DN(x, y_1)) \quad (4.37)$$

which is given by

$$\begin{aligned} DN(x,y) &= DN(x_1, y_1) \\ &+ \Delta x \cdot (DN(x_2, y_1) - DN(x_1, y_1)) \\ &+ \Delta y \cdot (DN(x_1, y_2) - DN(x_1, y_1)) \\ &+ \Delta x \Delta y \cdot (DN(x_2, y_2) + DN(x_1, y_1) - DN(x_1, y_2) - DN(x_2, y_1)) \end{aligned} \quad (4.38)$$

4.8 Higher order interpolation of a 1-d array

In the previous sections we were working with a linear interpolation methods to linearly interpolate between points, $(x_1, y_1), (x_2, y_2)$ by fitting a line between the two points. If we "pivot" about (x_1, y_1) we see that the slope is given by $(y_2 - y_1)/(x_2 - x_1)$

$$y(x) = y_1 + (y_2 - y_1) \cdot \frac{x - x_1}{x_2 - x_1} \quad (4.39)$$

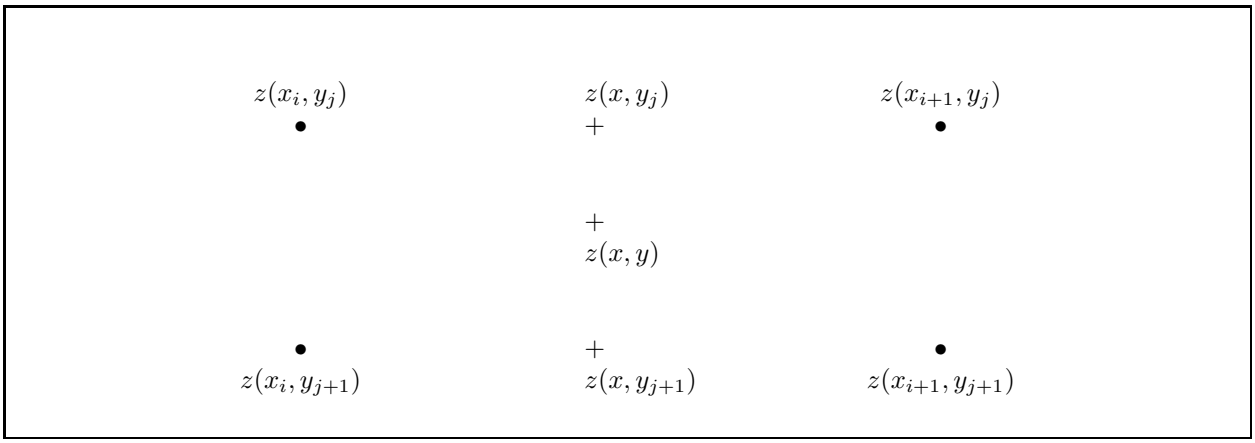


Figure 4.9: Illustration of bi-linear interpolation

$$= \quad y_1 \cdot \frac{x_2 - x_1}{x_2 - x - 1} + (y_2 - y_1) \cdot \frac{x - x_1}{x_2 - x_1} \tag{4.40}$$

$$= \quad y_1 \cdot \frac{x_2 - x}{x_2 - x - 1} \quad + \quad y_2 \cdot \frac{x - x_1}{x_2 - x_1} \tag{4.41}$$

The Lagrange interpolation formula fits a polynomial of order $N-1$ to the set of N data points, *without the need for a matrix inversion.* Polynomial fitting is discussed in Chapter 13. For large numbers of points polynomial fitting requires the inversion of a matrix (the Vandermonde matrix) and this can become numerically unstable whereas the Lagrange method is still stable. As with polynomial fitting, the Lagrangian interpolation formula does not require points to be equally sampled. As an illustration, we can rearrange Eqn. 4.39 so that the polynomial equation is written in terms of polynomials that have zeros at the tabulated values.

$$y(x) = y_1 \cdot \frac{x_2 - x}{x_2 - x_1} + y_2 \cdot \frac{x - x_1}{x_2 - x_1} \quad \text{for} \quad n = 2 \tag{4.42}$$

In general, the Lagrange formulation solves for a set of polynomial expressions, $a_m(x)$, where each polynomial is equal to zero for all the tabulated arguments except for x_m, as follows

$$\begin{aligned} a_m(x_k) &= 1 \qquad \text{for} \quad k = m \\ &= 0 \qquad \text{for} \quad k \neq m \end{aligned} \tag{4.43}$$

and the complete interpolation formula is then given by

$$y(x) = \sum_{m=1}^{n} a_m(x) \cdot y_m \tag{4.44}$$

In the case of Eqn. 4.42 it would be written

$$a_1(x) = \tfrac{x - x_2}{x_1 - x_2} \qquad\qquad a_2(x) = \tfrac{x - x_1}{x_2 - x_1} \tag{4.45}$$

which can be shown numerically (and analytically if you have the time) to be equal to a polynomial fit to the tabular data of order $N-1$ (see previous notes)

$$y(x) = \sum_{m=1}^{N} b_m(x) \cdot x^{m-1} \tag{4.46}$$

From Eqn. 4.43 it follows that $a_m(x)$ has $N-1$ zeros $(x_1, x_2, \ldots, x_{m-1}, x_{m+1}, \ldots x_N)$, and hence, in general

$$a_m(x) = A_m \cdot (x - x_1) \cdot (x - x_2) \cdot \ \ldots \ \cdot (x - x_{m-1}) \cdot (x - x_{m+1}) \cdot \ \ldots \ \cdot (x - x_n) \tag{4.47}$$

where A_m is a constant. But since $a_m(x_m) \equiv 1$ then we can solve for the constant

$$A_m = \frac{1}{(x_m - x_1) \cdot (x_m - x_2) \cdot \ \ldots \ \cdot (x_m - x_{m-1}) \cdot (x_m - x_{m+1}) \cdot \ \ldots \ \cdot (x_m - x_n)} \tag{4.48}$$

and set the equations for the m^{th} term

$$a_m(x) = \frac{(x - x_1) \cdot (x - x_2) \cdot \ \ldots \ \cdot (x - x_{m-1}) \cdot (x - x_{m+1}) \cdot \ \ldots \ \cdot (x - x_n)}{(x_m - x_1) \cdot (x_m - x_2) \cdot \ \ldots \ \cdot (x_m - x_{m-1}) \cdot (x_m - x_{m+1}) \cdot \ \ldots \ \cdot (x_m - x_n)} \tag{4.49}$$

$$= \frac{\prod_{m \neq n}^{N} (x - x_n)}{\prod_{m \neq n}^{N} (x_m - x_n)} \tag{4.50}$$

$$= \prod_{m \neq n}^{N} \left(\frac{x - x_n}{x_m - x_n} \right) \tag{4.51}$$

which can be shown to be equivalent to

$$a_m(x) = \sum_{n=1}^{N} b_{m,k} \cdot x^{k-1} \tag{4.52}$$

Lagrange interpolation can therefore be done with a very small piece of code and does not require any matrix inversion code. Here is an example program in FORTRAN to compute the Lagrange interpolation

```
c      Npts = # of input points
c      x(i) = input sample locations  (recommend double precision)
c      y(i) = input f(x(i))           (recommend double precision)
c
c      Npts_c = # of desired output points
c      x_c(j) = output sample locations
c      y_c(j) = interpolated f(x_c(j)) by Lagrange interpolation

       real*8 x1, a_m, b_m

       do i = 1, Npts_c
          x1 = x_c(i)                 ! interpolate to this value of x
          y_c(i) = 0.0d00
          do m = 1, Npts              ! build a_m(x1)
            a_m = 1.0d00
            b_m = 1.0d00
            do n = 1, Npts do begin
              if(m.ne.n) then
                a_m = a_m*(x1-x(n))
                b_m = b_m*(x(m)-x(n))
              endif
            enddo
            y_c(i) = y_c(i) + a_m*(y(m))/b_m ! interpolated polynomial
          enddo
        enddo
```

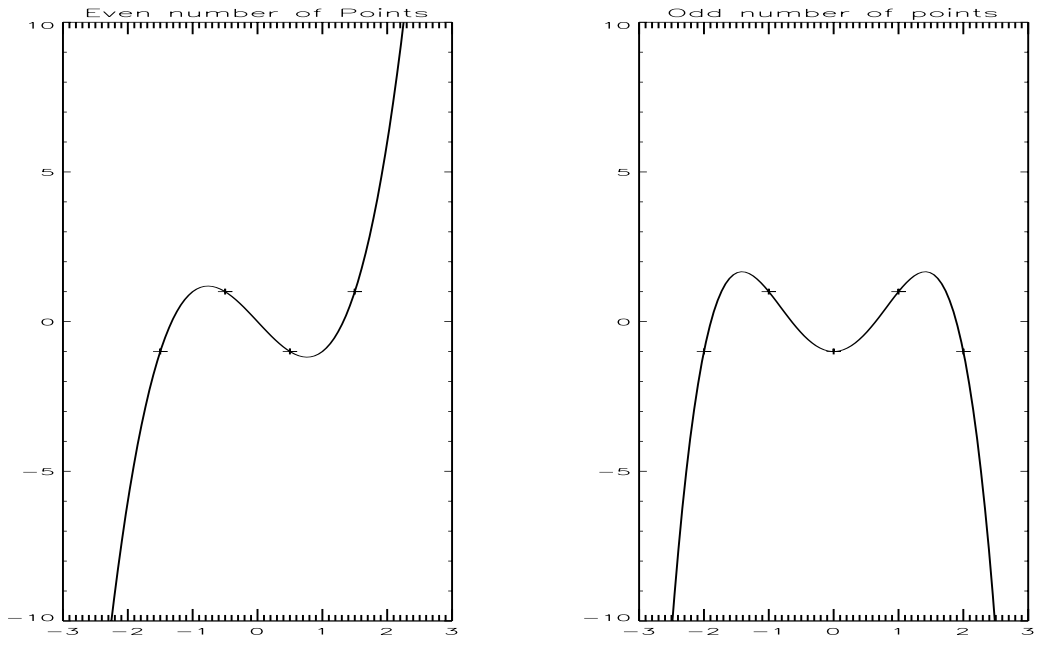


Figure 4.10: Illustration of extrapolation of a polynomial fit for odd and even functions

Polynomial fitting has disadvantages; however. For a polynomial of order $N-1$ there are $N-2$ turning points. Therefore, caution must be employed when extrapolating past the constraining points. Here is an example of using polynomial fitting to extrapolate outside the tabulated data range.

- Extrapolation past the endpoints can result in wild departures.
- High order polynomials can be unstable and can "blow up" (as seen in the figures from the last class, plot annotation="Lagrange").

4.9 Quadratic Piecewise Interpolation

This is simply a special case of the Lagrange formula for 3 adjacent points; however it is a good example of the use of a piecewise polynomial interpolation of higher orders.

$$f_i(x) = \sum_{m=1}^{2} a_{i,m}(x) \cdot y_{i+m-2} \tag{4.53}$$

$$a_m(x) = \prod_{m \neq n} \left(\frac{x - x_{i+n-2}}{x_{i+m-2} - x_{i+n-2}} \right) \tag{4.54}$$

$$\begin{aligned} f_i(x) &= \frac{(x-x_i)(x-x_{i+1})}{(x_{i-1}-x_i)(x_{i-1}-x_{i+1})} \cdot y_{i-1} \\ &+ \frac{(x-x_{i-1})(x-x_{i+1})}{(x_i-x_{i-1})(x_i-x_{i+1})} \cdot y_i \\ &+ \frac{(x-x_i)(x-x_{i-1})}{(x_{i+1}-x_{i-1})(x_{i+1}-x_i)} \cdot y_{i+1} \end{aligned} \tag{4.55}$$

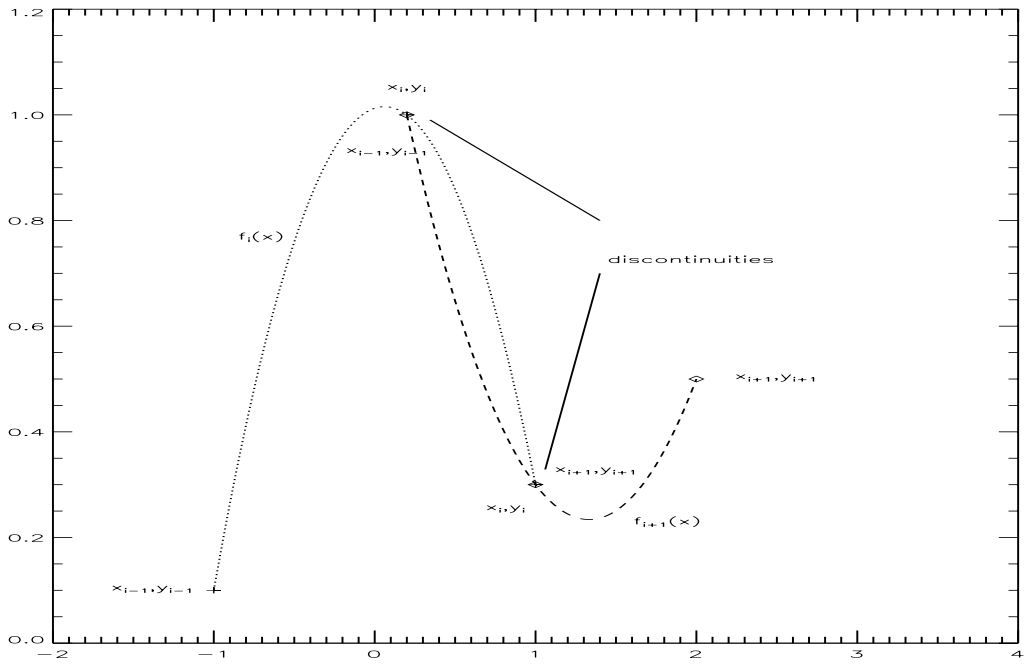


Figure 4.11: Illustration of quadratic (3 point) interpolation

Note, in the figure, that the derivatives at (x_i, y_i) and $(x_{i+1}, y_{i+1}$ are discontinuous between $f_i(x)$ and $f_{i+1}(x))$

4.10 Splines: interpolation with continuous derivatives

A spline is a piecewise polynomial interpolant of degree k that is continuously differentiable $k-1$ times. A linear spline is a piecewise linear polynomial that has degree one and is continuous but not differentiable. A *cubic* spline is a piecewise cubic polynomial that is twice continously differentiable.

$$f(x) = a_0 + a_1 \cdot x^1 + a_2 \cdot x^2 + a_3 \cdot x^3 \tag{4.56}$$

Within any two points of a tabular data set, (x_i, y_i) and (x_{i+1}, y_{i+1}) the cubic equation requires 4 pieces of information to solve for the 4 unknowns (a_0, a_1, a_2, a_3). Two pieces of information come from the two points and the other two pieces of information come from:

$$\left.\frac{df_{i-1}(x)}{dx}\right|_{x_i} = \left.\frac{df_i(x)}{dx}\right|_{x_i} \tag{4.57}$$

and

$$\left.\frac{df_i(x)}{dx}\right|_{x_{i+1}} = \left.\frac{df_{i+1}(x)}{dx}\right|_{x_{i+1}} \tag{4.58}$$

4.10.1 Derivation of the cubic spline

A cubic spline fits data $y_j = f(x_j))$ by fitting a cubic function over each interval $[x_j, x_{j+1}]$. The cubic spline is subject to the following constraints

1. Each interval has a continuous function defined by 4 coefficients

$$S_j(x) = a_j + b_j \cdot (x - x_j) + c_j \cdot (x - x_j)^2 + d_j \cdot (x - x_j)^3 \qquad \text{for} \quad j = 1, n-1 \tag{4.59}$$

The first and second derivative of each segment is given by

$$S'_j(x) = \frac{d\ S_j(x)}{d\ x} = b_j + 2 \cdot c_j \cdot (x - x_j) + 3 \cdot d_j \cdot (x - x_j)^2 \qquad \text{for} \quad j = 1, n-1 \tag{4.60}$$

$$S''_j(x) = \frac{d^2\ S_j(x)}{d\ x^2} = 2 \cdot c_j + 6 \cdot d_j \cdot (x - x_j) \qquad \text{for} \quad j = 1, n-1 \tag{4.61}$$

2. The functions are equal to the data values at the sampled locations.

$$S_j(x_j) = f(x_j) \qquad \text{for} \quad j = 1, n-1 \tag{4.62}$$

3. The piecewise functions are equal at the sampled positions, x_j, (*i.e.*, continuous function).

$$S_{j+1}(x_{j+1}) = S_j(x_{j+1}) \qquad \text{for} \quad j = 1, n-1 \tag{4.63}$$

This constraint will result (see below) in a relationship between a_{j+1} and a_j if we set $S_n(x_n) \equiv a_n$.

4. The first derivatives are equal at the sampled positions, x_j, (*i.e.*, continuous 1st derivative).

$$S'_{j+1}(x_{j+1}) = S'_j(x_{j+1}) \qquad \text{for} \quad j = 1, n-1 \tag{4.64}$$

This constraint will result (see below) in a relationship between b_{j+1} and b_j if we set $S'_n(x_n) \equiv b_n$.

5. The second derivatives are equal at the sampled positions, x_j, (*i.e.*, continuous 2nd derivative).

$$S''_{j+1}(x_{j+1}) = S''_j(x_{j+1}) \qquad \text{for} \quad j = 1, n-1 \tag{4.65}$$

This constraint will result (see below) in a relationship between c_{j+1} and c_j if we set $S''_n(x_n) \equiv 2 \cdot c_n$.

6. The boundary condition is specified at either

   - Free or natural boundary

$$S''_1(x_1) = 0 \quad \Rightarrow \quad c_n = 0 \tag{4.66}$$

$$S''_n(x_n) = S''_{n-1}(x_n) = 0 \quad \Rightarrow \quad c_n = 0 \quad \text{(see below)} \tag{4.67}$$

   - Clamped boundary

$$S'_1(x_1) = f'(x_1) = b_1 \tag{4.68}$$

$$S'_n(x_n) = f'(x_n) = b_n \quad \text{(see below)} \tag{4.69}$$

From constraint #1 we see that $S_j(x_j) = a_j$ and then from constraint #2 we note that

$$S_j(x_j) = a_j = f(x_j) \qquad \text{for} \quad j = 1, n-1 \tag{4.70}$$

Contraint #3 can be written as a set of equations as follows

$$S_{j+1}(x_{j+1}) = a_{j+1} = S_j(x_{j+1}) = a_j + b_j \cdot h_j + c_j \cdot h_j^2 + d_j \cdot h_j^3 \qquad \text{for} \quad j = 1, n-1 \tag{4.71}$$

if we can specify a_{j+1}. If we specify a_{j+1} using the final data point, $f(x_n)$ then contraint #3 can be written as

$$\begin{aligned} a_n &\equiv f(x_n) && (4.72)\\ a_{j+1} &= a_j + b_j \cdot h_j + c_j \cdot h_j^2 + d_j \cdot h_j^3 \quad \text{for} \quad j = 1, n-1 && (4.73)\\ h_j &\equiv (x_{j+1} - x_j) && (4.74) \end{aligned}$$

Contraint #4 can be written as a set of equations as follows

$$S'_{j+1} = b_{j+1} = S'_j(x_{j+1}) = b_j + 2 \cdot c_j \cdot h_j + 3 \cdot d_j \cdot h_j^2 \quad \text{for} \quad j = 1, n-1 \qquad (4.75)$$

If we set the term b_n from the boundary condition, $f'(x_n) = S'(x_n) = b_n$ then constraint #4 can be written

$$\begin{aligned} b_n &\equiv f'(x_n) = S'(x_n) && (4.76)\\ b_{j+1} &= b_j + 2 \cdot c_j \cdot h_j + 3 \cdot d_j \cdot h_j^2 \quad \text{for} \quad j = 1, n-1 && (4.77) \end{aligned}$$

If $f''(x_n) = S''(x_n) = 2 \cdot c_n$ then constraint #5 can be written

$$S''_{j+1} = 2 \cdot c_{j+1} = S''_j(x_{j+1}) = 2 \cdot c_j + 6 \cdot d_j \cdot h_j \quad \text{for} \quad j = 1, n-1 \qquad (4.78)$$

which can be rewritten as

$$c_{j+1} = c_j + 3 \cdot d_j \cdot h_h \quad \text{for} \quad j = 1, n-1 \qquad (4.79)$$

and used to solve for d_j

$$d_j = \frac{1}{3 \cdot h_j}(c_{j+1} - c_j) \quad \text{for} \quad j = 1, n-1 \qquad (4.80)$$

Substituting Eqn. 4.80 into Eqn. 4.73 yields

$$\begin{aligned} a_{j+1} &= a_j + b_j \cdot h_j + c_j \cdot h_j^2 + \frac{(c_{j+1} - c_j) \cdot h_j^2}{3} \quad \text{for} \quad j = 1, n-1 && (4.81)\\ a_{j+1} &= a_j + b_j \cdot h_j + \frac{h_j^2}{3} \cdot (c_{j+1} + 2 \cdot c_j) \quad \text{for} \quad j = 1, n-1 && (4.82) \end{aligned}$$

which results in the following conditions for b_j

$$\begin{aligned} b_j &= \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2 \cdot c_j + c_{j+1}) \quad \text{for} \quad j = 1, n-1 && (4.83)\\ b_{j-1} &= \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2 \cdot c_{j-1} + c_j) \quad \text{for} \quad j = 1, n-1 && (4.84) \end{aligned}$$

Substituting Eqn. 4.80 into Eqn. 4.77 yields

$$\begin{aligned} b_{j+1} &= b_j + 2 \cdot c_j \cdot h_j + (c_{j+1} - c_j) \cdot h_j \quad \text{for} \quad j = 1, n-1 && (4.85)\\ b_{j+1} &= b_j + (c_{j+1} c_j) \cdot h_j \quad \text{for} \quad j = 1, n-1 && (4.86) \end{aligned}$$

which results in another condition for b_j

$$b_j = b_{j-1} + (c_j + c_{j-1}) \cdot h_{j-1} \qquad \text{for} \quad j = 1, n-1 \tag{4.87}$$

Substitution of the pair of equations given in Eqn. 4.83 into Eqn. 4.87 yields

$$b_j = b_{j-1} + (c_j + c_{j-1}) \cdot h_{j-1} \tag{4.88}$$

$$\frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2 \cdot c_j + c_{j+1}) = \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2 \cdot c_{j-1} + c_j) + h_{j-1}(c_j + c_{j-1}) \tag{4.89}$$

$$\frac{3}{h_j} \cdot (a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}) = h_j(2 \cdot c_j + c_{j+1}) - h_{j-1}(2 \cdot c_{j-1} + c_j) + 3h_{j-1}(c_j + c_{j-1}) \tag{4.90}$$

$$\frac{3}{h_j} \cdot (a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}) = c_{j-1}(h_{j-1}) + c_j(2h_{j-1} + 2h_j) + c_{j+1}(h_j) \tag{4.91}$$

The final form of the equation to be solved is

$$c_{j-1}(h_{j-1}) + c_j(2h_{j-1} + 2h_j) + c_{j+1}(h_j) = \frac{3}{h_j} \cdot (a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}) \qquad \text{for} \quad j = 2, n-1 \tag{4.92}$$

Note that h_j is specified from the sampling for $j = 1, n-1$ and $a_j = f(x_j)$ are known for $j = 1, n$. Therefore, Eqn 4.92 is a set of linear equations that can be solved for $c_j, j = 1, n$ if can use boundary conditions for the Equations for $j = 1$ and $j = n$.

Solution for Natural Boundary Conditions

For example, if the *natural* cubic spline is desired, then the values of $f''(x_1) = f''(x_n) = 0$ and, therefore from Eqn. 4.61,

$$c_1 = 0 \tag{4.93}$$

and Eqn. 4.76 will yield.

$$c_n = 0 \tag{4.94}$$

The set of linear equations can be written as a matrix operator, $A \cdot x = b$. For example, if $n = 5$ the matrices would be

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & 0 \\ 0 & 0 & h_3 & 2(h_3+h_4) & h_4 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 3(a_3-a_2)/h_2 - 3(a_2-a_1)/h_1 \\ 3(a_4-a_3)/h_3 - 3(a_3-a_2)/h_2 \\ 3(a_5-a_4)/h_4 - 3(a_4-a_3)/h_3 \\ 0 \end{bmatrix} \tag{4.95}$$

Solution for Clamped Boundary Conditions

The boundary conditions are illustrated in Fig. 4.12.
For the clamped condition we specify $f'(x_1)$ and $f'(x_n)$. The value of $f'(x_1)$ is given by Eqn. 4.60

$$b_1 = f'(x_1) \tag{4.96}$$

and from Eqn. 4.83 we can determine the conditions for the c's.

$$b_1 = \frac{1}{h_1}(a_2 - a_1) - \frac{h_1}{3}(2 \cdot c_1 + c_2) \tag{4.97}$$

Therefore, this boundary condition results in the following equation

$$2 \cdot h_1 \cdot c_1 + h_1 \cdot c_2 = \frac{3}{h_1}(a_2 - a_1) - 3 \cdot f'(x_1) \tag{4.98}$$

The boundary condition $f'(x_n)$ was used to define b_n in Eqn. 4.76 and we can use Eqn. 4.87 to related this condition to an equation for c_n.

$$f'(x_n) = b_n = b_{n-1} + (c_n + c_{n-1}) \cdot h_{n-1} \tag{4.99}$$

and using the $(j-1)^{\text{th}}$ Eqn. from Eqn. 4.83 we can write

$$b_{n-1} = \frac{1}{h_{n-1}}(a_n - a_{n-1}) - \frac{h_{n-1}}{3}(2 \cdot c_{n-1} + c_n) \tag{4.100}$$

and then substitute into our version of Eqn. 4.87 above will yield

$$\begin{aligned} f'(x_n) &= b_{n-1} + (c_n + c_{n-1}) \cdot h_{n-1} && (4.101)\\ &= \frac{1}{h_{n-1}}(a_n - a_{n-1}) + \frac{h_{n-1}}{3}(3 \cdot c_n + 3 \cdot c_{n-1} - 2 \cdot c_{n-1} - c_n) && (4.102)\\ &= \frac{1}{h_{n-1}}(a_n - a_{n-1}) + \frac{h_{n-1}}{3}(c_{n-1} + 2 \cdot c_n) && (4.103) \end{aligned}$$

$$c_{n-1} \cdot h_{n-1} + c_n \cdot 2 \cdot h_{n-1} = 3 \cdot f'(x_n) - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \tag{4.104}$$

$$\begin{bmatrix} 2h_1 & h_1 & 0 & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & 0 \\ 0 & 0 & h_3 & 2(h_3+h_4) & h_4 \\ 0 & 0 & 0 & h_4 & 2h_4 \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} = \begin{bmatrix} 3(a_2-a_1)/h_1 - 3f'(x_1) \\ 3(a_3-a_2)/h_2 - 3(a_2-a_1)/h_1 \\ 3(a_4-a_3)/h_3 - 3(a_3-a_2)/h_2 \\ 3(a_5-a_4)/h_4 - 3(a_4-a_3)/h_3 \\ 3f'(x_n) - 3(a_5-a_4)/h_4 \end{bmatrix} \tag{4.105}$$

Solution for mixed Boundary Conditions

Of course we could specify one boundary condition as natural and one as clamped. In the case where the natural boundary condition is used at x_1 we would solve the following simultaneous linear equations

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & 0 \\ 0 & 0 & h_3 & 2(h_3+h_4) & h_4 \\ 0 & 0 & 0 & h_4 & 2h_4 \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 3(a_3-a_2)/h_2 - 3(a_2-a_1)/h_1 \\ 3(a_4-a_3)/h_3 - 3(a_3-a_2)/h_2 \\ 3(a_5-a_4)/h_4 - 3(a_4-a_3)/h_3 \\ 3f'(x_n) - 3(a_5-a_4)/h_4 \end{bmatrix} \tag{4.106}$$

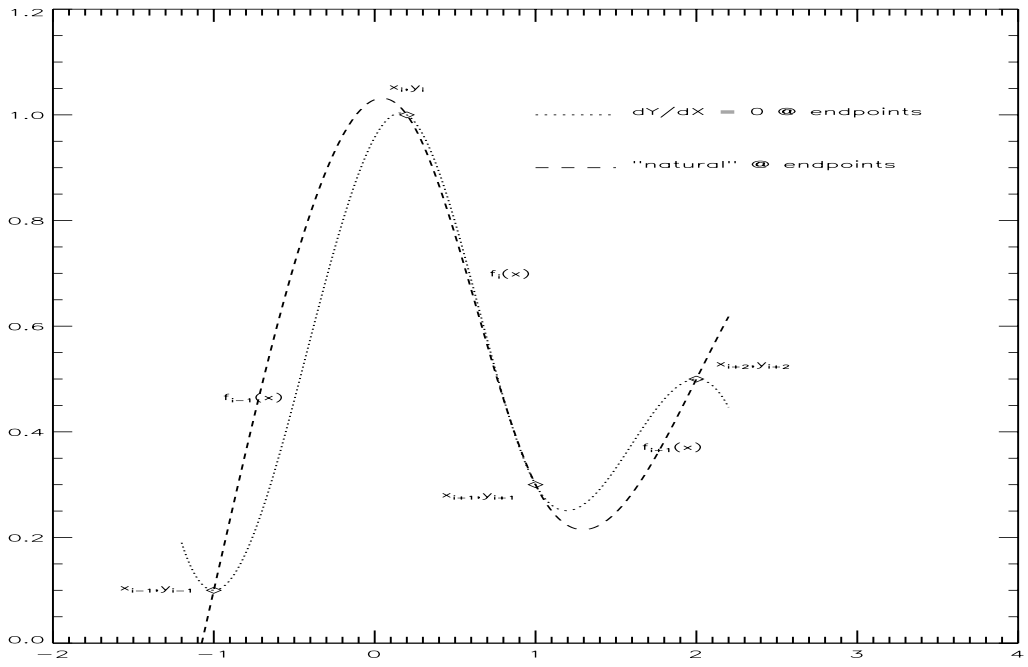


Figure 4.12: Illustration of end-point constraints for cubic splines

4.10.2 Comparison of interpolation methods

4.11 Rational Functions

Some functions are not well approximated by polynomials but are well approximated by rational functions, that is quotients of polynomials. The rational functions can model functions with poles, that is, zeros in the denominator.

$$R(x_k) = \frac{P_M(x)}{Q_N(x)} = \frac{\sum_{m=0}^{M} p_m \cdot x^m}{\sum_{n=0}^{N} q_n \cdot x^n} \tag{4.107}$$

Given the order of the polynomials, N and M there are $N + M + 1$ unknowns (since p_m and q_n are degenerate) and there must be $K = M + N + 1$ points (usually values and derivatives of the functions) to determine the polynomial coeffients.

$\Rightarrow$ Usually can write a recursion relation to solve for the polynomial coefficients, see Scheid [1986, pg. 292-304] and Press *et al.* [1987]

4.12 Example of Rational Functions used in Approximating Fundamental Functions

The following example is based on a math package for 10 digit floating point functions based only on the basic floating point operations (*, /, +, -) and can be written easily in assembler language. The necessary mathematical functions for a math library are SQRT, SIN, COS, ATN, EXP, LOG. Given that these functions

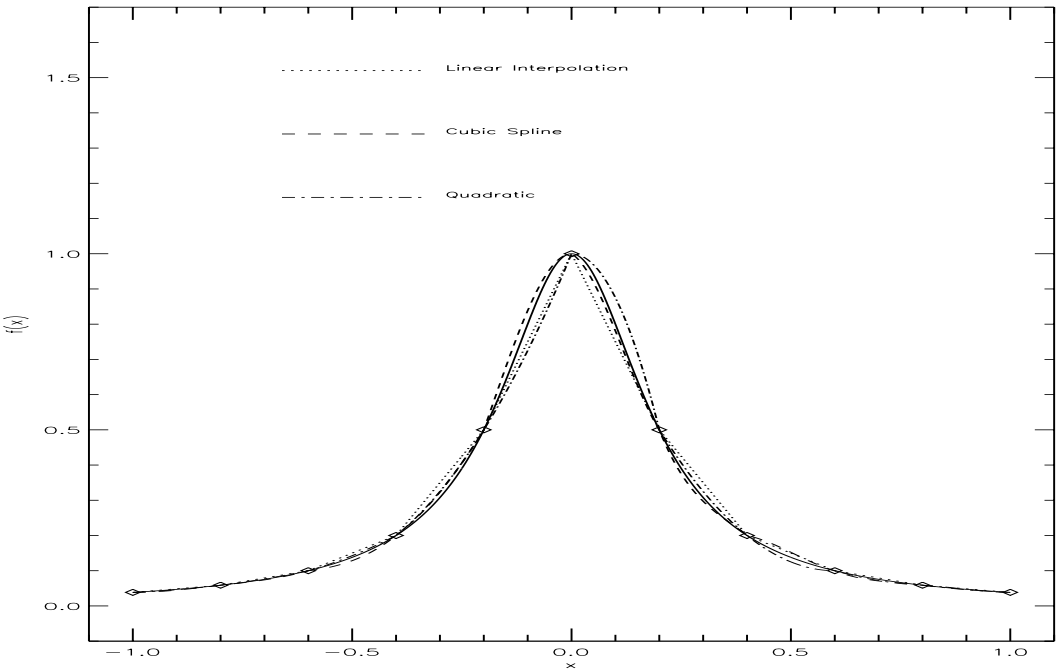


Figure 4.13: Linear and Quadratic, Cubic Spline Interpolation of Runge Function ($f(x) = 1/(1 + 25 \cdot x^2)$ on 11 Equally Spaced Points

are utilized heavily by other library functions it is worth the investment to make them as quick as possible. Here we will show an example for the sine function. The operations for determining quadrants appear complicated; however, in assembler code they are simple and quick instructions.

4.12.1 $\sin(a), a$ given in radians

$$\mathrm{SIN}(x) = 1 + x - \frac{x^3}{3!} + \frac{x^5}{5!} \ldots \tag{4.108}$$

$$\mathrm{COS}(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots \tag{4.109}$$

If we wanted a result accurate to 10 decimal digits of accuracy we would need to extend the series to the terms $x^{14}/14!$ since $1/14! = 0.115 \cdot 10^{-10}$ for values of x of order unity. Analysis of the most efficient implementation of the series solutions taken to the n^{th} factorial term would have the following operations

operation	real math	32 bit integer math
multiply	$\frac{n-1}{2} + 1 = 8$	n-1 = 14
addition	$\frac{n-1}{2} + 1 = 8$	n-1 = 14
divide	$\frac{n-1}{2} = 7$	0

Recognizing that a floating point divide is usually the longest operation, followed by multiplication and then addition it is desirable to see if there are quicker solutions to these functions.

The following rational operator will determine the SINE function to 10 decimal digits of accuracy. A FORTRAN program, used to test and mimic an ASSEMBLER program is given in *ftp*/primfunc.F.

$$\sin\left(\frac{2}{\pi} \cdot |x|\right) \simeq f_1(x) \equiv \frac{x \cdot \sum_{k=0}^{2} a_k \cdot x^{2 \cdot k}}{\sum_{k=0}^{5} b_k \cdot x^{2 \cdot k}} \quad \text{for} \quad 0 \le x \le 1 \tag{4.110}$$

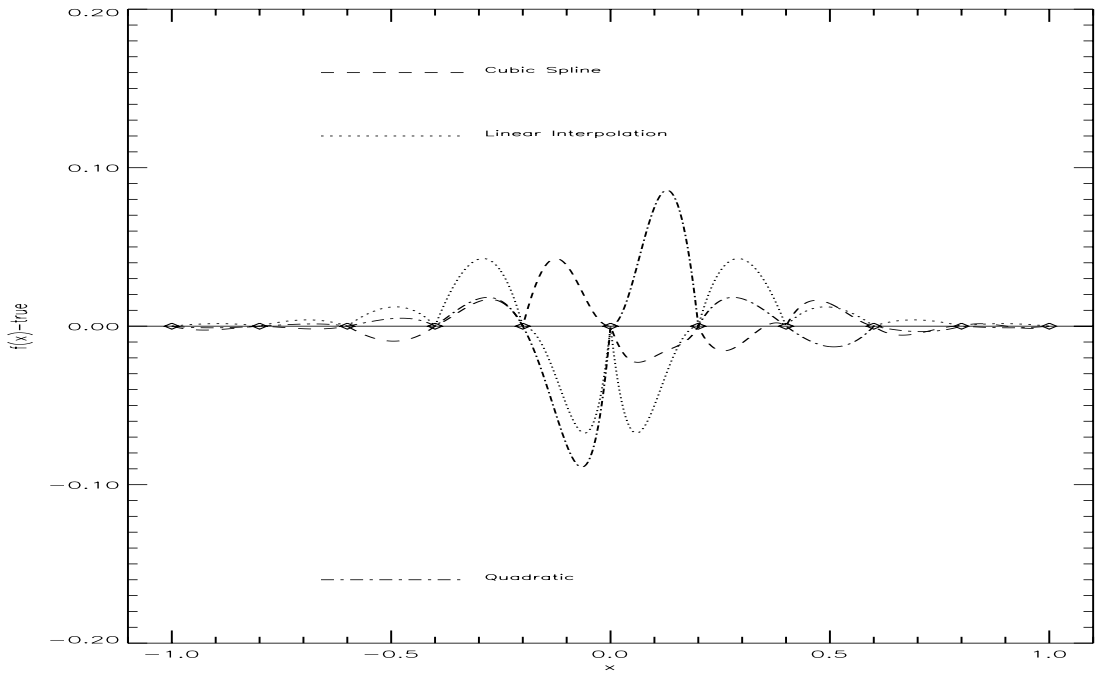


Figure 4.14: Error in Linear and Quadratic, Cubic Spline Interpolation $f(x) = 1/(1 + 25 \cdot x^2)$ on 11 Equally Spaced Points

Here the number of divisions is drastically reduced and the number of overall multiplications. On most machines this would be significantly faster.

operation	real math	integer math
multiply	12	0
addition	7	0
divide	1	0

The SINE function for other quadrants can be computed by adjusting the argument. For example, the

Table 4.1: Rational function coefficients for a SIN() function
coefficients for SIN(a)

	a_k	b_k
0	1.0	-.227677966d-02
1	1.15303871d02	0.25738461d00
2	4.47214584d03	-1.23945831d01
3	-	2.83485682d02
4	-	-2.70772675d03
5	-	7.02483026d03

To obtain the value of the SINE or COSINE for other quadtrants first make sure the argument is within 2π radians with $a = \text{MOD}(a, 2\pi)$ and then the function $f_1(x)$ can be used as follows:

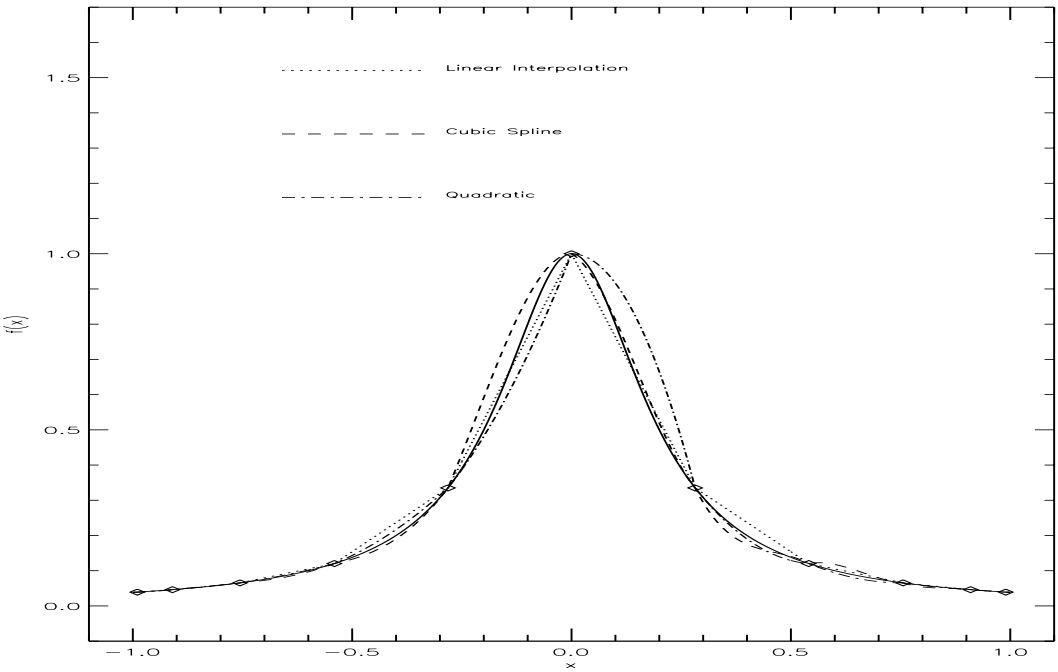


Figure 4.15: Linear and Quadratic, Cubic Spline Interpolation of Runge Function ($f(x) = 1/(1 + 25 \cdot x^2)$ 11 Points with Chebyshev Spacing

Table 4.2: Rational function coefficients for a COS() function

a	$\sin(a)$	$\cos(a)$
$0 \le a < \frac{\pi}{2}$	$f_1(\frac{2}{\pi} \cdot a)$	$f_1(1 - \frac{2}{\pi} \cdot a)$
$\frac{\pi}{2} \le a < \pi$	$f_1(2 - \frac{2}{\pi} \cdot a)$	$f_1(\frac{2}{\pi} \cdot a - 3)$
$\pi \le a < \frac{3\pi}{2}$	$f_1(\frac{2}{\pi} \cdot a - 2)$	$-f_1(\frac{3-2}{\pi} \cdot a)$
$\frac{3\pi}{2} \le a < \frac{2}{\pi}$	$-f_1(\frac{4-2}{\pi} \cdot a)$	$f_1(\frac{2}{\pi} \cdot a - 3)$

4.12.2 $\tan^{-1}(a), a$ given in radians

$$f(z) = z^2 \cdot \frac{\sum_{k=0}^{3} b_k \cdot z^{2k}}{\sum_{k=0}^{4} a_k \cdot z^{2k}} \tag{4.111}$$

a	z	$\tan^{-1}(a)$
0		0
$\lvert a \rvert \le \tan(\frac{\pi}{12})$	$z = \lvert a \rvert$	$s \cdot \lvert f(z) \rvert$
$\tan(\frac{\pi}{12} \le a \le 1$	$z = \frac{\lvert a \rvert \cdot \sqrt{3}-1}{\lvert a \rvert + \pi/6}$	$s \cdot \lvert f(z) \rvert + \pi/6$
a > 1	$z = \frac{\lvert 1/a \rvert \cdot \sqrt{3}-1}{\lvert 1/a \rvert + \pi/6}$	$s \cdot \lvert f(z) \rvert + \pi/6 - \pi/2$

4.12.3 $\log_e(a)$

$$\ln(1 + x) = x - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \dots \tag{4.112}$$

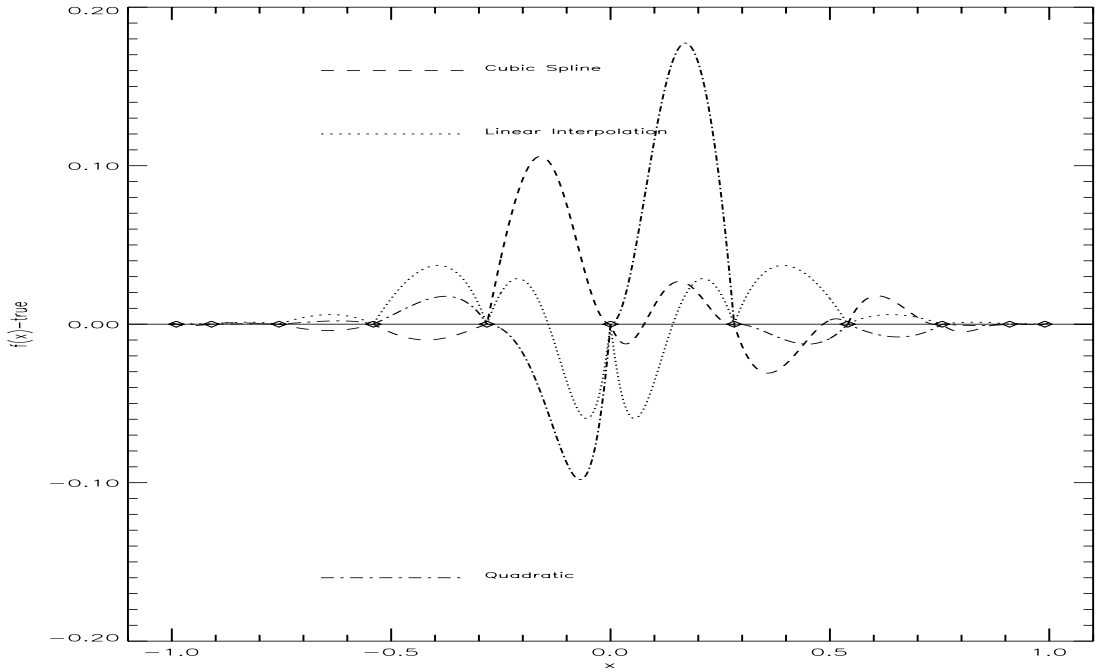


Figure 4.16: Error in Linear and Quadratic, Cubic Spline Interpolation $f(x) = 1/(1+25 \cdot x^2)$ 11 Points with Chebyshev Spacing

Table 4.3: Rational function coefficients for a ATAN() function
coefficients for ATAN(a)

	a_k	b_k
0	1.0	1.25958023d01
1	3.70660897d01	1.25579166d02
2	2.07692682d02	2.98928038d02
3	3.646624d02	1.97203096d02
4	1.97203096d02	-

First it is useful to separate the mantissa from the exponent. In floating point we write

$$a = 2^i \cdot m \tag{4.113}$$

where i is the exponent and m is the mantissa. Then note that

$$\ln(x) = \ln(m * 2^i) = \ln(m) + i * \ln(2) \tag{4.114}$$

we can use rational functions of the form

$$f(z) = z \cdot \frac{\sum_{k=0}^{2} b_k \cdot z^{2k}}{\sum_{k=0}^{3} a_k \cdot z^{2k}} \tag{4.115}$$

Table 4.4: Rational function coefficients for a ln() function
coefficients for $\ln(a)$

	a_k	b_k
0	1.0	-1.83278700d01
1	-2.07334879d01	9.34639000d01
2	6.17610656d01	-9.01746917d01
3	-4.50873458d01	

a	z	$\ln(a)$
≤ 0		NaN
$a \leq \frac{1}{\sqrt{2}}$	$z = \frac{a-\frac{1}{2}}{a+\frac{1}{2}}$	$(i-1)\cdot\ln(2) + f(z)$
$a > \frac{1}{\sqrt{2}}$	$z = \frac{a-1}{a+1}$	$i\cdot\ln(2) + f(z)$

4.12.4 $\exp(a)$

$$\exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots \tag{4.116}$$

Noting that the $\exp(x) = 2^{\frac{x}{\log_e(2)}}$. Therefore, we can transform the floating point number to a log of base 2 and then separate the exponent, i, and mantissa, m.

$$2^i \cdot m = \frac{|x|}{\log_e(2)} \tag{4.117}$$

The variable used in the rational function will be z, defined as

$$z = m - \frac{1}{2} \tag{4.118}$$

$$f(z) = \sqrt{2} \cdot \frac{\sum\limits_{k=0,3} a_k \cdot z^{2k} + z \cdot \sum\limits_{k=0}^{2} b_k \cdot z^{2k}}{\sum\limits_{k=0}^{3} a_k \cdot z^{2k} - z \cdot \sum\limits_{k=0,2} b_k \cdot z^{2k}} \tag{4.119}$$

Table 4.5: Rational function coefficients for a EXP) function
coefficients for $\exp(a)$

	a_k	b_k
0	1.00000000d00	6.06148533d01
1	1.74928769d03	3.02869717d04
2	3.27725152d05	2.08038435d06
3	6.00272036d06	

and the exponential can be evaluated as

- x =0, $\exp(x) = 0$
- x >0, $\exp(x) = 2^i \cdot f(z)$
- x <0, $\exp(x) = \frac{1}{2^i \cdot f(z)}$

4.13 Problem: Extrapolation of World Population

The following table was down-loaded a couple of years ago from http://www.popin.org (unfortunately it is no longer there). I will e-mail this file to everyone in ASCII.

Table 4.6: World population model

Real Data to be used for fitting				Projected Data	
year	population	year	population	year	population
1000	0.31	1930	2.07	2010	6.79
1250	0.40	1940	2.30	2020	7.50
1500	0.50	1950	2.52	2030	8.11
1750	0.79	1960	3.02	2040	8.58
1800	0.98	1970	3.70	2050	8.91
1850	1.26	1980	4.44		
1900	1.65	1990	5.27		
1910	1.75	1998	5.90		
1920	1.86	2000	6.06		

In all cases determine your interpolation function using only the real data above (*i.e.*, those before 2002). Evaluate your interpolation function over the range of 1000 $\leq$ year $\leq$ 2200. To evaluate the quality of your interpolation model prediction, use the UN projection values above as a guide.

1. Compute an interpolated/extrapolated model using linear interpolation. Plot your interpolation function.

2. Compute an interpolated/extrapolated model using quadratic interpolation. Plot the results (on the same graph as #1, if possible).

3. Compute a polynomial fit to the real data. Determine the coefficients for an arbitrary degree N. You may use either built in functions to compute the polynomial or compute the Vandermonde matrix and solve the least squares equations for the coefficients.

   a) Does a lower order (*e.g.*, N=3) polynomial reproduce the data points?
   No, the function has too much variability to be represented by such as small order.

N	a_0	a_1	a_2	a_3	a_4	a_5	χ^2
3	-105.08	235.79	-170.92	40.264898			0.4596
4	-3284.39	11647.9	-16276.2	11212.4	-3811.3	511.89	0.0358

   b) Does a higher order remain stable? Why?
   No. The function oscillates wildly between points to match the exponential shape.

   c) Does scaling the data or taking logarithms help? That is one can fit $\log(\text{population}) = a_0 + a_1 \cdot t$ and then exponentiate the result of the polynomial.
   The logarithm fit is quite a bit better. It helps to keep the plot from oscillating for the first 1500 years and it is a better extrapolation, but again there is too much variability to match the shape. Notice that the fit to the data points is also quite a bit better as illustrated by the χ^2 value.

$$\chi^2 = \frac{1}{N} \cdot \sum i = 1^N \left(p(i) - f(i)\right)^2 \tag{4.120}$$

N	a_0	a_1	a_2	a_3	a_4	a_5	χ^2
3	-29.62	63.721	-46.598	11.278			0.1406
4	-496.02	1753.0	-2451.4	1693.14	-578.08	78.176247	0.0093

   d) Plot the "best" example of your polynomial fit.
   A low order polynomial does the least damage, however, "best" is still quite poor.

4. Compute the Lagrange interpolation function.

   a) Does it reproduce the data points? Why?
   Yes, it matches the points exactly.

   b) Is it well behaved in between? Why?
   Same problems as polynomial fitting to $n-1$ degrees. It oscillates *wildly* in between the real data points. Taking logarithms does not help.

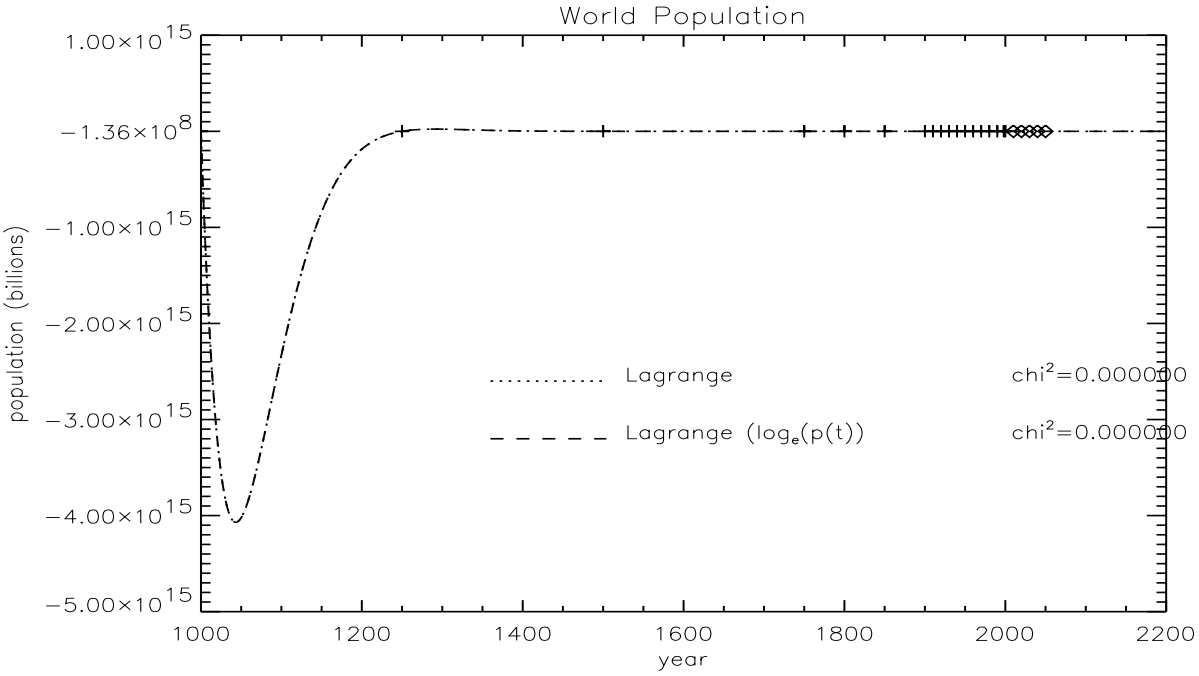


Figure 4.17: Example of a fit to the world population with Lagrangian interpolation

5. Compute the Cubic Spline function for the real data points and .

   a) Does it reproduce the data points? Why?
   Yes, because it is constrained to do so.

   b) Is it well behaved in between? Why?
   Yes, the curve in between the points is well behaved and is reasonable.

   c) What is the "best" choice for your boundary conditions? Why?
   The natural boundary condition $(d^2p(t)/dt^2 = p''(t) = 0$ appeared to work reasonable well within the range of real data.

   Outside the range of the fit the cubic spline behaves poorly because it uses a cubic constraint (4 degrees of freedom) to fit within the final interval, however, the degrees of freedom outside the interval are meaningless and produce trash. The population function is an extreme example of what can go wrong.

   Alternatively, one could set the clamped boundary condition. This is only slightly different than the natural boundary condition and cannot improve the extrapolation outside the fitting range. If it does, it is by shear coincidence.

$$\left.\frac{dp(t)}{dt}\right|_{t=1000} = \frac{0.09}{250} = 0.0036 \quad \text{billion/year} \tag{4.121}$$

For the upper boundary I drew a line through the points and took the sides of the triangle to compute the slope

$$\left.\frac{dp(t)}{dt}\right|_{t=2000} = \frac{17}{320} = 0.053125 \quad \text{billion/year} \tag{4.122}$$

One could use the last two points to derive the slope $f'(2000) = 0.08$.

d) Plot the "best" example of the cubic spline fit.

I used the natural boundary condition ($p''(t) = 0$) which is equivalent to letting the slope, $p'(t)$, at the boundaries be a constant.

6. Given the UN estimate of future population, which interpolation method would you feel best represents the population of the world?

   Linear interpolation is the least damaging. Quadratic interpolation on $\log_e$(population) had a tendency to project well into the near future, but this is probably coincidental for this particular set of data.

   A cubic spline works best as long as one doesn't want to estimate future population.

   Interpolation using the UN projections is the best way to go. In this way the model of future population can be based on many indicators. A cubic spline (natural boundary conditions) within the range of the fit is the "best" representation of the UN real and projected data.

7. Outline another method which might improve your ability to use the UN data outside of the range given.

   - Use cubic spline within the bounds of the data, 1000 $\leq$ year $\leq$ 2000 or better yet, 1000 $\leq$ year $\leq$ 2050 and use linear interpolation outside of the range of the cubic spline.
   - Find an analytic equation and fit the population to that function. While this most likely will not reproduce the exact numbers it can either spliced with a cubic spline within the range of observations or used for the entire domain. Rational functions could be a viable candidate. For example,

     $$p(t) = 10^{[a_0 + a_1 \cdot t + a_2/(a_3 - t)]} \tag{4.123}$$

     A fit with a_0 =-0.69, $a_1 = 0.00004$, $a_2 = 200$, and $a_3 = 2150$ was reasonable.
   - Use a model (or guess) to predict the future population and then use a cubic spline within the model range.

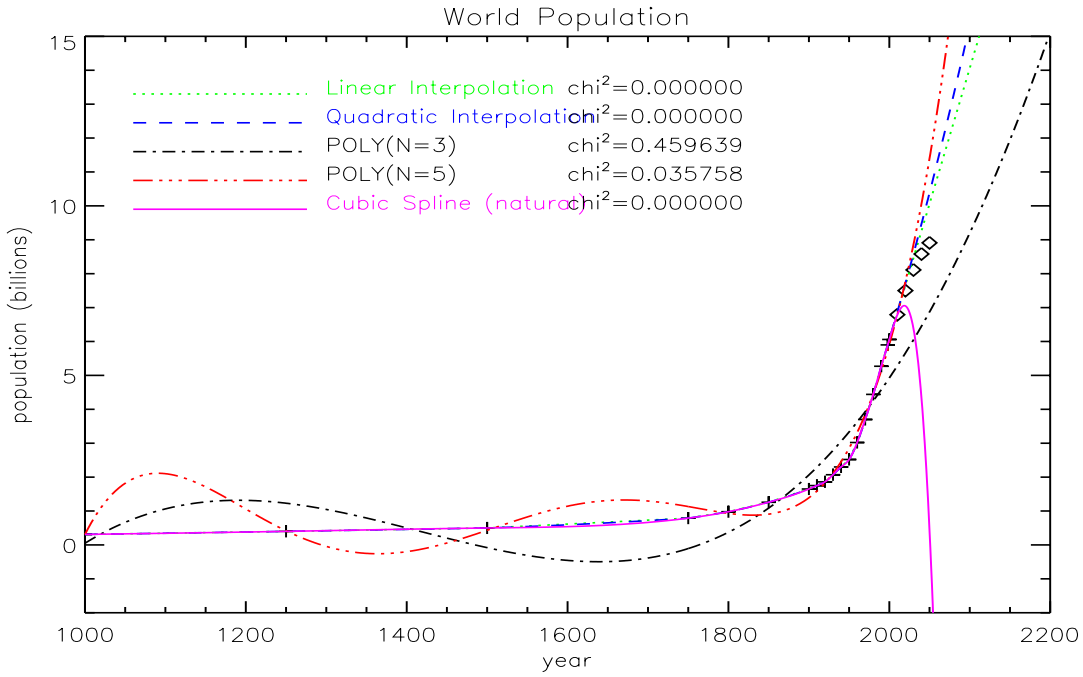


Figure 4.18: Comparison of linear, quadratic, polynomial, and spline fits to the world population using only real data

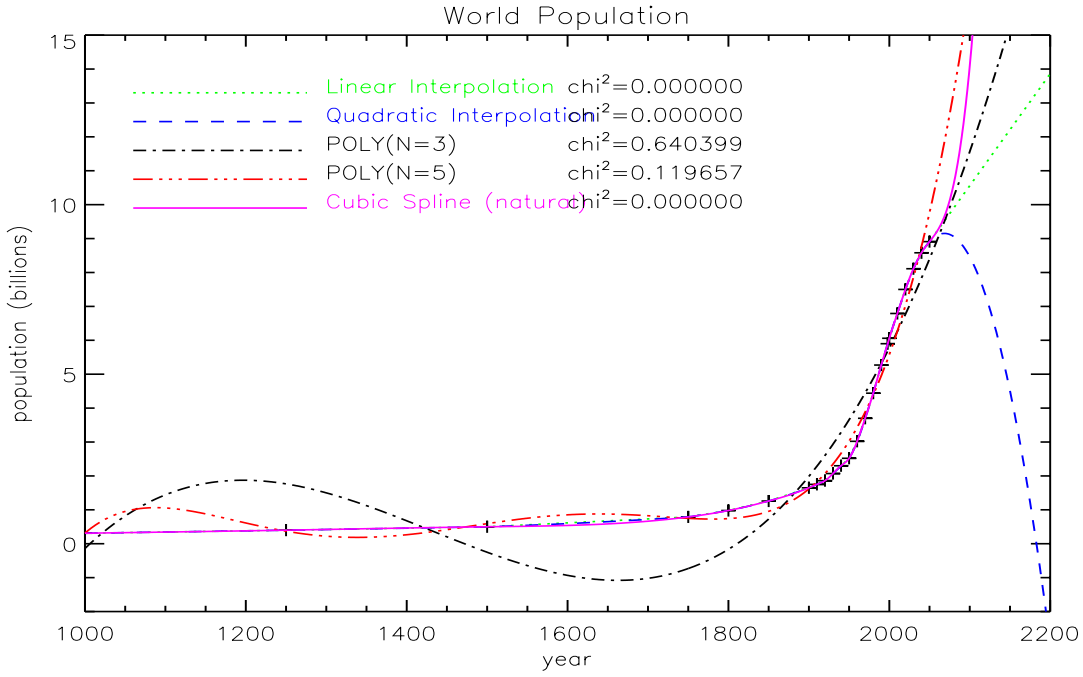


Figure 4.19: Comparison of linear, quadratic, polynomial, and spline fits to the world population using real and *projected* data

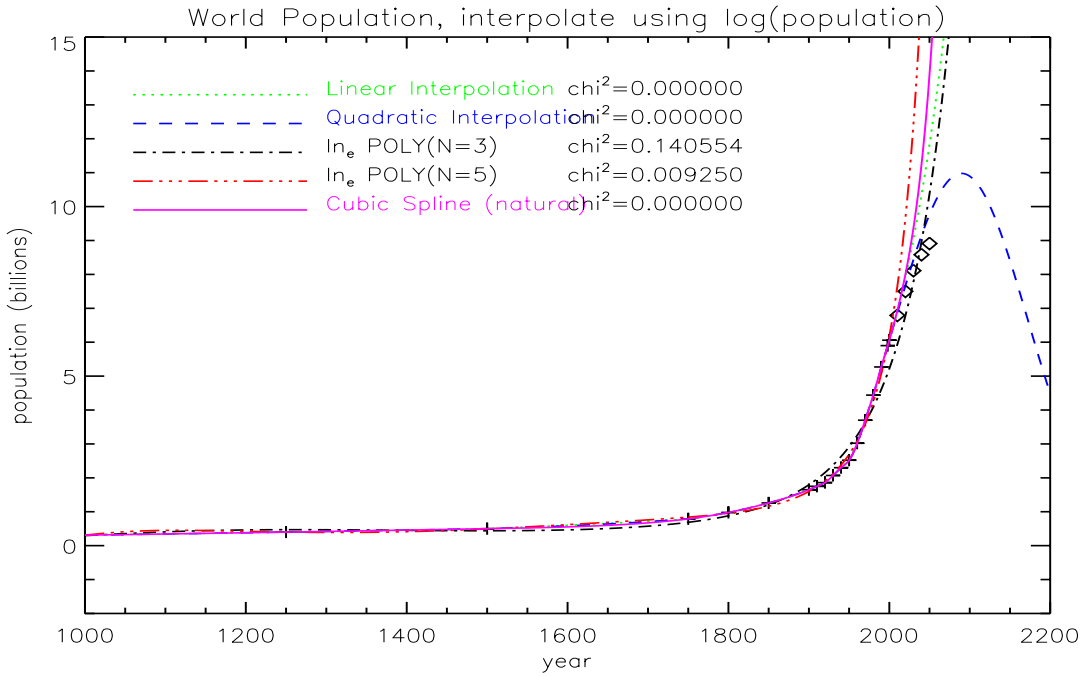


Figure 4.20: Comparison of linear, quadratic, polynomial, and spline fits to the logarithm of world population using only real data

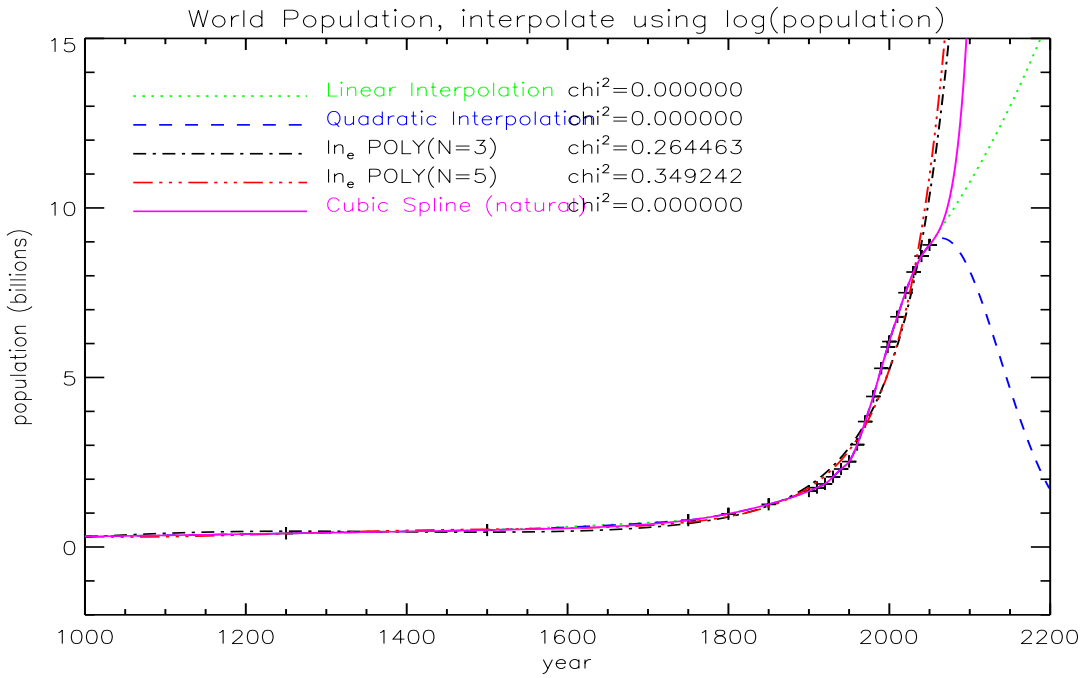


Figure 4.21: Comparison of linear, quadratic, polynomial, and spline fits to the logarithm of world population using real and *projected* data

Chapter 5

Quadrature

Numerical integration is a solution of a differential equation. This will be covered in Chapter 9. Quadrature means the evaluation of a definite integral of a function. Interpolation and quadrature are related; however, in quadrature we optimize for the integral quantity whereas in interpolation we optimize towards the value itself. We can write the analytic integral as follows

$$I = \int_a^b f(x) \cdot dx \tag{5.1}$$

One evaluates I on a computer by converting the integral to a sum.

$$I = \int_a^b f(x) \cdot dx \simeq \sum_{i=1}^{N} (w_i \cdot f(x_i)) + R_n \tag{5.2}$$

integrand is "sampled" at N points
$x_i \equiv$ the nodes
$w_i \equiv$ the weights
$R_n \equiv$ the remainder

5.1 Quadrature Rules: general considerations

A quadrature rule is a recipe for choosing a particular set of nodes and weights. We want a quadrature rule that

- minimizes the trunction error, R_n
- minimizes the number of nodes, N, because Eqn. 5.2 takes a run-time that scales linearly with N.
- Newton-Cotes rules

  Uses equally spaces nodes. For example, the integrand is from a table of values or the output from an instrument or experiment. This form is analygous to piecewise interpolation.
- Other rules

  Sample the integrand at strategic points. Only useful if $f(x)$ can be calculated or measured for any value of x. This form is analygous to polynomial fitting in the context of interpolation.

- Closed rules *e.g.*, Gaussian
  Rules that sample the endpoints, a and b.

- Open rules
  Rules that do not sample the endpoints. For example,

$$I = \int_0^1 \frac{dx}{\sqrt{x}} \tag{5.3}$$

  The integrand has a singularity at $x = 0$, therefore, $x = 0$ cannot be a node. Nevertheless, $\frac{1}{\sqrt{x}}$ has an integrable singularity at $x = 0$ because

$$I = \int_0^1 \frac{dx}{\sqrt{x}} = \left[2 \cdot \sqrt{x}\right]_0^1 = 2 \tag{5.4}$$

  Therefore, we must use an open quadrature rule.

5.2 Newton-Cotes rules

$$I = \int_a^b f(x) \cdot dx = \sum_{n=0}^{N-1} \int_{x_n=a+n\cdot h}^{x_{n+1}=a+(n+1)\cdot h} f(x) \cdot dx \tag{5.5}$$

If the panels are "small" then we can find a simple formula to evaluate the integral accurately within each panel. "Small" h means that the integrand varies slowly between x_n and x_{n+1} and it can be approximated by another function, $g(x) \simeq f(x)$ over that interval. The trick is to choose a function $g(x)$ which can be easily integrated. An easy function to integrate is a polynomial.

- Midpoint Rule (an open Newton-Cotes rule): We assume that $g(x)$ is a constant (polynomial of order 0) within the panel, $g(x) \simeq f(x_n + \frac{h}{2})$. This means that the positions of x_n and the evaluation of the function, $y(x + h/2)$, are NOT at the same position. The integral becomes

$$\int_{x_n}^{x_{n+1}} f(x) \cdot dx \quad \simeq \quad h \cdot f\left(x_n + \frac{h}{2}\right) \tag{5.6}$$

  and for a uniform sampling the integral over N points is given by

$$\int_a^b f(x) \cdot dx \quad \simeq \quad h \sum_{n=1}^{N-1} f\left(x_n + \frac{h}{2}\right) \tag{5.7}$$

- Trapezoidal Rule (a closed Newton-Cotes rule): We assume a linear (polynomial of order 1) approximation over the interval,

$$g(x) = a_0(n) + a_1(n) \cdot x = f(x_n) + \frac{f(x_n + h) - f(x_n)}{h} \cdot x \tag{5.8}$$

  the integral over the panel is then

$$\int_{x_n}^{x_n+h} f(x)\cdot dx \simeq \int_{x_n}^{x_n+h} g(x)\cdot dx = \frac{h}{2}\left(f(x_{n+1}) - f(x_n)\right) \tag{5.9}$$

so that

$$\begin{aligned}\int_a^b f(x)\cdot dx &\simeq \frac{h}{2}\sum_{n=1}^{N-1} f\left(f(x_n+h) - f(x_n)\right) \\ &= \frac{h}{2}\cdot\left(f(a)+f(b)\right) + h\sum_{n=2}^{N-1} f(x_n)\end{aligned} \tag{5.10}$$

Here is a program (trapezoid.F) to compute the trapezoidal integral for an array of data.

```
      function trapezoid(npts,x0,x1,Ydata)
      integer*2 npts
      real*4    trapezoid,x0,x1,Ydata(*)

c     calculates the integral of npts equally spaced points
c      Ydata(i),   1 <= i <= npts
c     over the limits of   x0 <= x <= x1

      integer*2 i,inpt
      real*8    sum,dx0,dx1

      inpt = npts - 1
      dx0 = dble(x0)
      dx1 = dble(x1)

      sum = (dble(Ydata(1)) + dble(Ydata(npts)))/2.0d00

      do i = 2,inpt
        sum = sum + dble(Ydata(i))
      enddo
      sum  = sum*(dx1-dx0)/float(npts-1)

      trapezoid = sngl(sum)

      return
      end
```

- Simpson's Rule (a closed Newton-Cotes rule): We assume a quadratic (polynomial of order 2) approximation over the interval, $g_n(x) = a_0(n) + a_1(n)\cdot x + a_2(n)\cdot x^2$. To find $g_n(x)$ we use 3 nodes, which are the end-points and mid-point of the interval.

  We can use the Lagrange interpolation formula (see Eqn. 4.55) to solve the quadratic equation for a set of three points

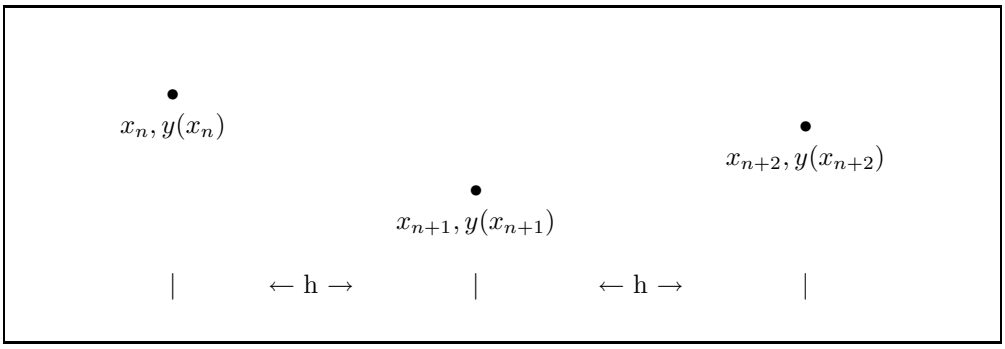


Figure 5.1: Illustration of Simpson's rule

$$\begin{aligned} g_n(x) &= \frac{(x-x_{n+1})(x-x_{n+2})}{(x_n-x_{n+1})(x_n-x_{n+2})}\cdot y(x_n) \\ &+ \frac{(x-x_n)(x-x_{n+2})}{(x_{n+1}-x_n)(x_{n+1}-x_{n+2})}\cdot y(x_{n+1}) \\ &+ \frac{(x-x_n)(x-x_{n+1})}{(x_{n+2}-x_n)(x_{n+2}-x_{n+1})}\cdot y(x_{n+2}) \end{aligned} \tag{5.11}$$

For a uniform spacing ($h = (x_{n+1} - x_n) = (x_{n+2} - x_{n-1})$) the quadratic interpolation formula can be solved for a group of 3 points. If we define a new variable, $z = x - x_n$, then $x - x_{n+1} = z - h$, $x_n - x + n + 1 = h$, etc. so that Eqn. 5.11 can be written as

$$\begin{aligned} g_n(z) &= \frac{(z-h)(z-2h)}{(-h)(-2h)}\cdot y(x_n) + \frac{(z)(z-2h)}{(h)(-h)}\cdot y(x_{n+1}) \\ &+ \frac{(z)(z-h)}{(2h)(h)}\cdot y(x_{n+2}) \end{aligned} \tag{5.12}$$

simplifying we have

$$g_n(z) = \frac{(z-h)(z-2h)}{(-h)(-2h)}\cdot y(x_n) + \frac{(z)(z-2h)}{(h)(-h)}\cdot y(x_{n+1}) + \frac{(z)(z-h)}{(2h)(h)}\cdot y(x_{n+2}) \tag{5.13}$$

Expanding we have

$$g_n(z) = \frac{z^2 - hz - 2hz + 2h^2}{2h^2}\cdot y(x_n) + \frac{2hz - z^2}{h^2}\cdot y(x_{n+1}) + \frac{z^2 - hz}{2h^2}\cdot y(x_{n+2}) \tag{5.14}$$

Now combining common terms of $1 \equiv z^0$, z, and z^2 we have

$$\begin{aligned} g_n(z) &= y(x_n) + z\cdot\left[\frac{-h-2h}{2h^2}\cdot y(x_n) + \frac{2}{h}\cdot y(x_{n+1}) - \frac{y(x_{n+2})}{2h^2}\right] \\ &+ z^2\cdot\left[\frac{y(x_n)}{2h^2} - \frac{y(x_{n+1})}{h^2} + \frac{y(x_{n+2})}{2h}\right] \end{aligned} \tag{5.15}$$

simplifying we have

$$\begin{aligned} g_n(z) &= y(x_n) + \frac{z}{2h} \cdot [-3 \cdot y(x_n) + 4 \cdot y(x_{n+1}) - y(x_{n+2})] \\ &+ \frac{z^2}{2 \cdot h^2} \cdot [y(x_n) - 2 \cdot y(x_{n+1}) + y(x_{n+2})] \end{aligned} \tag{5.16}$$

$$g_n(z) = a_0 + a_1 \cdot z + a_2 \cdot z^2 \tag{5.17}$$

$$a_0 = y(x_n) \tag{5.18}$$

$$a_1 = \frac{[-3 \cdot y(x_n) + 4 \cdot y(x_{n+1}) - y(x_{n+2})]}{2h} \tag{5.19}$$

$$a_2 = \frac{[y(x_n) - 2 \cdot y(x_{n+1}) + y(x_{n+2})]}{h^2} \tag{5.20}$$

The integral over these 3 points is given by

$$\int_{x_n}^{x_n+2h} f(x) \cdot dx \simeq \int_{x_n}^{x_n+2h} g_n(x) \cdot dx = \int_0^{2h} g_n(z) \cdot dz \tag{5.21}$$

$$= \int_0^{2h} \left[a_0 + a_1 \cdot z + a_2 \cdot z^2\right] \cdot dz \tag{5.22}$$

$$= \left[a_0 \cdot z + a_1 \cdot \frac{z^2}{2} + a_2 \cdot \frac{x^2}{3}\right]_0^{2h} \tag{5.23}$$

$$= 2h \cdot a_0 + 2h^2 \cdot a_1 + \frac{8h^3}{3} \cdot a_2 \tag{5.24}$$

$$\begin{aligned} &= 2h \cdot y(x_n) + [4y(x_{n+1}) - 3y(x_n) - y(x_{n+2})] \cdot h \\ &+ [y(x_n) - 2y(x_{n+1}) + y(x_{n+2})] \cdot \frac{4h}{3} \end{aligned} \tag{5.25}$$

$$\int_{x_n}^{x_n+2h} f(x) \cdot dx = \frac{h}{3} [y(x_n) + 4 \cdot y(x_{n+1}) + y(x_{n+2})] \tag{5.26}$$

To integrate over N points we require that the total number of points, N, must be odd. We can sum the integrals over sets of 3 points (Eqn. 5.26) by re-indexing out arrays by groups of 3 points, $n(k) = 1 + (k-1) \cdot 2$, for the N points. The Simpson algorithm reduces to an accounting problem. Keep in mind that the endpoints repeat (*i.e.*, $x_{n(k=2)} = x_{n(k=1)+2}$) and $x_{n(k)+1}$ does not repeat.

$$\int_{a \equiv x_1}^{b \equiv x_N} f(x) \cdot dx = \sum_{k=1}^{(N-1)/2} \left(\int_{x_n(k)}^{x_{n(k)}2h} f(x) \cdot dx \right) \tag{5.27}$$

$$\begin{aligned} &\simeq \sum_{k=1}^{(N-1)/2} \frac{h}{3} \cdot \left(y(x_n(k)) + 4 \cdot y(x_{n(k)+1}) + y(x_{n(k)+2})\right) \\ &= \frac{h}{3} \cdot (f(a) + f(b)) + \frac{4h}{3} \cdot \sum_{k=1}^{(N-1)/2} y(x_{2 \cdot k}) \\ &+ \frac{2h}{3} \cdot \sum_{k=1}^{(N-3)/2} y(x_{2 \cdot k - 1}) \end{aligned} \tag{5.28}$$

In section B.17 is a program (simpson.F) to compute the Simpson's quadrature rule for an array of data.

5.3 Accuracy of Newton-Cotes Rules

For a given interval, h, we expect higher order polynomials to better represent $f(x)$, and therefore, presumably, the integral of $f(x)$. We can derive and estimate of the truncation error by a Taylor series.

For the midpoint rule

$$\int_{x_n}^{x_n+h} f(x)dx \simeq h \cdot f(x_n + \frac{h}{2}) = h \cdot f(x_m) \tag{5.29}$$

and we use a Taylor expansion for $f(x)$ about x_m

$$f(x) = f(x_m) + f'(x_m) \cdot (x - x_m) + \frac{1}{2!}f''(x_m) \cdot (x - x_m)^2 + \frac{1}{3!}f'''(x_m) \cdot (x - x_m)^3 + \dots \tag{5.30}$$

we can integrate $f(x)$ term-by-term remembering that $f(x_m)$, $f'(x_m)$, and $f''(x_m)$ are all constants

$$\begin{aligned} \int_{x_n}^{x_n+h} f(x)dx \quad \simeq \quad & \int_{x_m-h/2}^{x_n+h/2} f(x_m)dx + \int_{x_m-h/2}^{x_m+h/2} f'(x_m)\,(x - x_m)\,dx \\ & + \frac{1}{2}\int_{x_m-h/2}^{x_m+h/2} f''(x_m)\,(x - x_m)^2\,dx + \dots \end{aligned} \tag{5.31}$$

NOTE: that

$$\left.\frac{f'\,(x - x_m)^2}{2}\right|_{x_m-h/2}^{x_m+h/2} \Rightarrow 0 \tag{5.32}$$

and

$$\left.\frac{f''\,(x - x_m)^3}{6}\right|_{x_m-h/2}^{x_m+h/2} = \frac{h^3}{8 \cdot 3} \cdot f'' \tag{5.33}$$

$$\int_{x_n}^{x_n+h} f(x)dx = h \cdot f(x_m) \quad + \quad 0 \quad + \quad \frac{h^3}{24} \cdot f''(x_m) \quad + \quad \dots \tag{5.34}$$

so that the error for the midpoint rule is of order $O(h^3)$. This is the error for one panel, so that the whole integral over interval $[a, b]$ of N panels is

$$R_n \approx N \cdot \frac{h^3}{24} \cdot \overline{f}''(x_m) \tag{5.35}$$

since $h = \frac{b-a}{N}$ then

$$R_n \approx \frac{1}{N^2} \cdot \frac{(b-a)^3}{24} \cdot \overline{f}'' \quad \text{which is proportional to } \frac{1}{N^2} \tag{5.36}$$

R_n(Midpoint)	R_n(Trapezoidal)	R_n(Simpson's)
$\frac{(b-a)^3 \cdot \overline{f}''}{24N^2}$	$\frac{(b-a)^3 \cdot \overline{f}''}{12N^2}$	$\frac{(b-a)^5 \cdot \overline{f}''''}{90N^4}$

Efficiency:

- Simpson's truncation error, R_n(Simpson's) is proportional to $\frac{R_n(\text{midpoint})}{N^2}$
  For example, if a given integral requires 1000 panels using the midpoint rule to obtain a certain accuracy then for the same error, Simpson's rule needs $M = \sqrt{N/2} \approx 22$.
  The relative error of the Newton-Cotes rules is summarized as follows:

$$\frac{R_n(\text{Simpson}'\text{s})}{4 \cdot M^4} \approx \frac{R_n(\text{Midpoint})}{N^2} \tag{5.37}$$

$$M \approx \left(N^2/4\right)^{\frac{1}{4}} = \sqrt{N/2} \tag{5.38}$$

- CPU-time to evaluate integral is proportional to $N \cdot$ # of function evaluations.

5.3.1 Example #1: Comparison of Integration accuracy

$$f(x) = 5 \cdot x^2 - 2.5 \cdot x + \exp\left(\frac{-x}{3}\right) \tag{5.39}$$

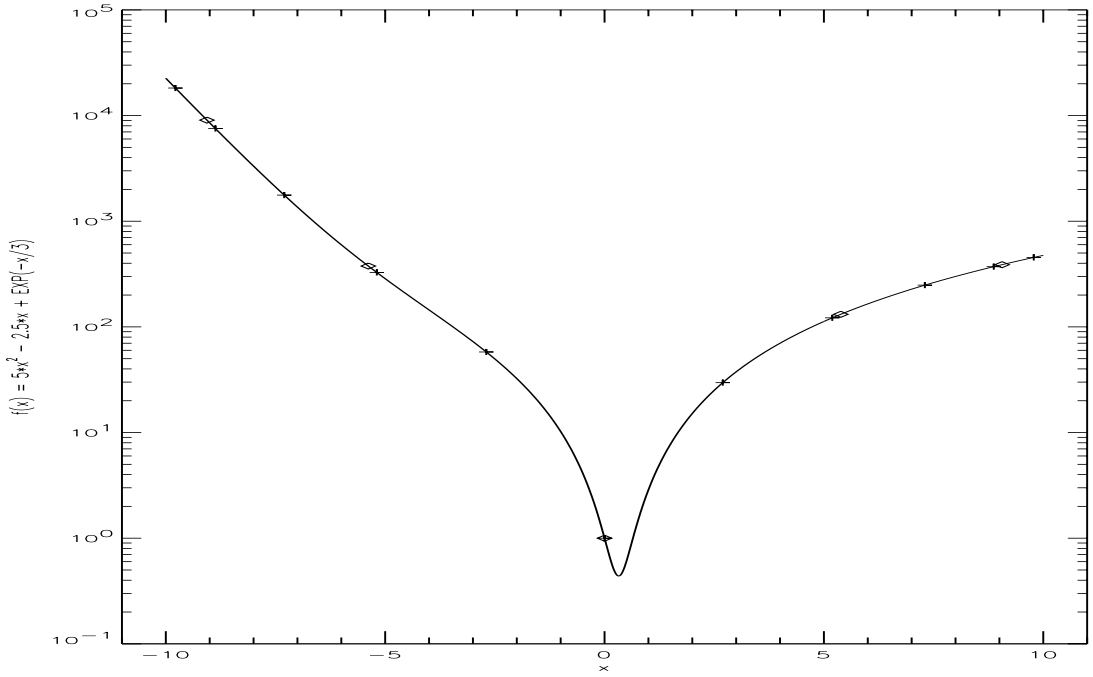


Figure 5.2: Example of the Gaussian integration of $5 \cdot x^2 - 2.5 \cdot x + \exp\left(\frac{-x}{3}\right)$

which has the analytical solution of

$$\int_a^b f(x) \cdot dx = \left[\frac{5}{3} \cdot x^3 + \frac{5}{4} \cdot x^2 - 3 \cdot \exp\left(\frac{-x}{3}\right)\right]_a^b \tag{5.40}$$

a	b	N	Analytic Value	Midpoint error(%)	Trapezoidal error(%)	Simpsons error(%)	Guassian error(%)
0	10	11	1544.560	-.271	.541	.000016	.000000
-10	10	5	3417.321	-6.36	12.74	.0784	-.000050
-10	10	11	3417.321	-1.02	2.04	.0026	.000007
-10	10	101	3417.321	-.0102	.0204	.000007	.000007
-100	100	11	.8987e+15	-76.2	234.2	123.4	-.412

5.3.2 Example #2, Integration accuracy using Runge's function

We can examine the integral of Runge's function in Fig. 5.3

$$f(x) = \frac{1}{1 + 25 \cdot x^2} \tag{5.41}$$

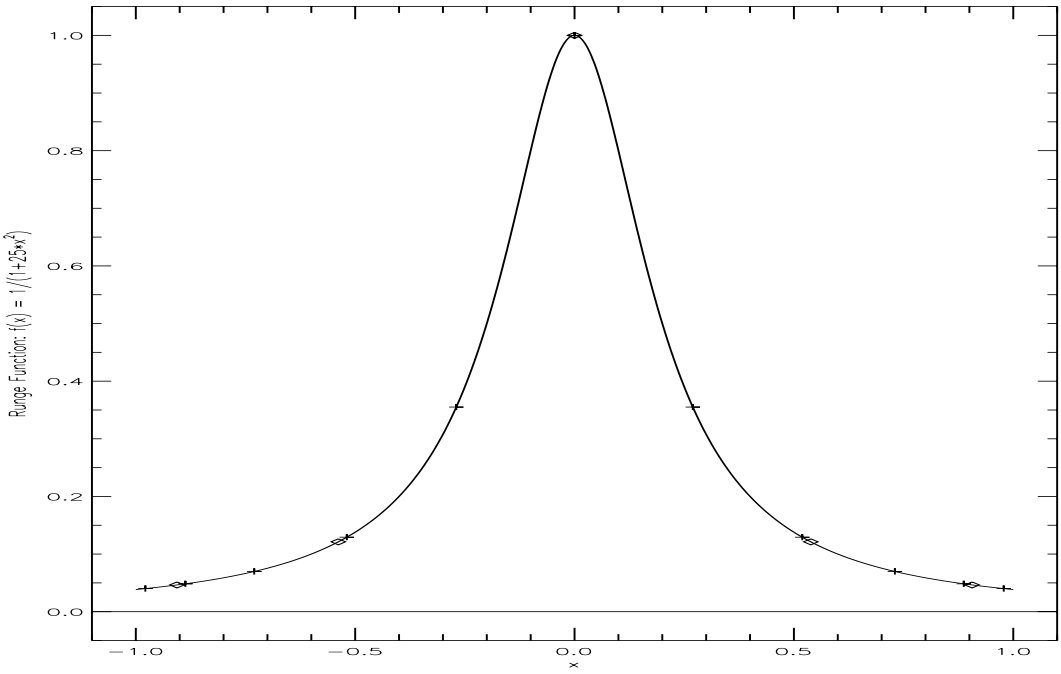


Figure 5.3: Example of the Gaussian integration of $\frac{1}{1+25 \cdot x^2}$

over various limits. The analytic integral is given by (CRC 26$^{\text{th}}$ Edition, Integral #62, pg. 293)

$$\int_a^b \frac{dx}{1 + 25 \cdot x^2} = 0.2 \cdot \left[\tan^{-1}(5 \cdot x)\right]_a^b \tag{5.42}$$

a =0, b =1, Analytic Value=0.274680

N	Midpoint error(%)	Trapezoidal error(%)	Simpsons error(%)	Guassian error(%)	Spline error(%)
11	0.0104	-0.022	-0.142	-0.00014	-.504
21	0.0028	-0.006	-0.0003	-0.00001	-0.058

a =-1, b =1, Analytic Value = 0.549360

N	Midpoint error(%)	Trapezoidal error(%)	Simpsons error(%)	Guassian error(%)	Spline error(%)
5	-16.9	19.6	-3.51	28.7	16
7	-5.03	5.15	17.23	12.15	5.94
9	1.42	1.37	-4.71	5.34	1.49
11	-0.38	0.34	3.73	2.38	0.44
21	0.010	-0.022	-0.142	0.0445	0.0026
51	0.0028	-0.006	-0.0028	0.000011	0.00023
101	0.00041	-0.00090	-0.000011	-0.000011	0.000011

5.3.3 Example #3, Integration accuracy of the Normal Equation

Suppose we wish to numerically compute the integral of

$$\int_0^1 e^{-x^2} dx \tag{5.43}$$

While this is trivial to do numerically the analytic answer is very difficult. To arrive at an approximate analysic result we can use a table in Bevington page 308 of the error functon, which is tabulated for the function

$$f(x) = \int_{-x}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} dx \tag{5.44}$$

We can compute the true value of our numerical integral using the following modification of Bevington's interpolated value to arrive at the analytic value of 0.746824. We need to make the substition of $u = \sqrt{2} \cdot x$ and $du = \sqrt{2} \cdot dx$ so that

$$\int_0^1 e^{-x^2} dx = \frac{1}{2} \int_{-1}^{1} e^{-x^2} dx \tag{5.45}$$

$$= \int_{-\sqrt{2}}^{+\sqrt{2}} e^{-\frac{1}{2}u^2} \sqrt{2} du \tag{5.46}$$

$$= \frac{\sqrt{\pi}}{2} \int_{-\sqrt{2}}^{+\sqrt{2}} e^{-\frac{1}{2}u^2} \sqrt{2} du \tag{5.47}$$

Therefore, we need to know the error function for the values of $x = \sqrt{2}$, which is taken from Bevington page 308's table

f(1.41)	0.84145
f(1.42)	0.84438

If we wish to compute the error function for $f(\sqrt{2})$ we can interpolate Bevington's table to arrive at $f(\sqrt{2}) \simeq 0.842685$ or $y(x) = \frac{\sqrt{\pi}}{2} \cdot 0.842685 = 0.74681$. Upon experimenting numerically we can find with large number of points that the correct value is actually 0.746824.

$$y(x) = EXP(-x^2) \tag{5.48}$$

a =0, b =1, Analytic Value = 0.746824

N	Midpoint error(%)	Trapezoidal error(%)	Simpsons error(%)	Guassian error(%)	Spline error(%)
5	0.257494	-0.514173	0.004190	0.000008	-0.065
7	0.114217	-0.228259	0.000846	0.000008	-0.019
9	0.064200	-0.128336	0.000271	0.000008	-0.008
11	0.041087	-0.082117	0.000120	0.000008	-0.004
21	0.010288	-0.020519	0.000016	0.000008	-0.00050
51	0.001652	-0.003280	0.000008	0.000008	-0.000024
101	0.000407	-0.000814	0.000008	0.000008	0.000000
1001	0.000040	0.000000	0.000008	0.000008	0.000008

NOTE: we could experiment with $\int_{-1}^{1} e^{-x^2} dx$ and take $\frac{1}{2}$ of that result; however, it does not appear to be necessary.

5.3.4 Example #4, Integration accuracy using Planck's function

The integral of the Planck function can be compared to the analytic value.

$$\int_0^\infty B_\nu(T) d\nu = \frac{\sigma}{\pi} \cdot T^4 \tag{5.49}$$

where $\sigma = 5.6705 \cdot 10^5$ milli-Watt$\cdot$ Meter$^{-2}\cdot$ Kelvin and ν is the frequency expressed in wavenumbers, which is related to the frequency in Hertz, f, as $\nu \equiv f/c$. The Planck function, $B_\nu(T)$ can be written as a function of wavenumber

$$B_\nu(T) = \frac{2 \cdot h \cdot c^2 \nu^3}{e^{(hc\nu/kT)} - 1} \tag{5.50}$$

where h is Planck's constant, c is the speed of light, k is Boltzmann's constant. Here is an example of the Planck function at the temperature of deep space, T =2.73 K, the temperature of the Earth, T =300 K, and the temperature at the surface of the Sun, T =5600 K. Since the Sun only extend $\frac{1}{2}^\circ$ we need to multiply the Planck function by the number of steradians

$$\Omega = \pi \cdot \left(\frac{R_\odot}{D_\odot} \right)^2 \tag{5.51}$$

where $R_\odot$ is the radius of the Sun and $D_\odot$ is the distance to the Sun. Here is an example of what the figure should look like:

The Planck function is interesting because it covers many orders of magnitude. If we use a linear quadrature in ν we can introduce large errors, or, conversely require a large number of quadrature points.

We can substitute a log scale by substitution of $\nu' = e^\nu$ so that

$$\int_a^b f(x) dx = \int_{ln(a)}^{ln(b)} f(x') \cdot e^{x'} \cdot dx' = \int_{ln(a)}^{ln(b)} g(x') dx' \tag{5.52}$$

We can then pick our finite limits of integration to correspond to which temperature we are computing. For the case of

temperature (K)	ν_1 cm^{-1}	ν_2 cm^{-1}	$ln(\nu_1)$ cm^{-1}	$ln(\nu_2)$ cm^{-1}	$\frac{\sigma}{\pi} T^4$ (mW/M^2/steradian)
2.73 K	0.001	100	-11.5	5.0	$1.00257 \cdot 10^{-3}$
300 K	0.01	10,000	-10.0	11.5	$1.462 \cdot 10^5$
5600 K	0.1	100,000	-2.3	13.8	$1.7751 \cdot 10^{10}$

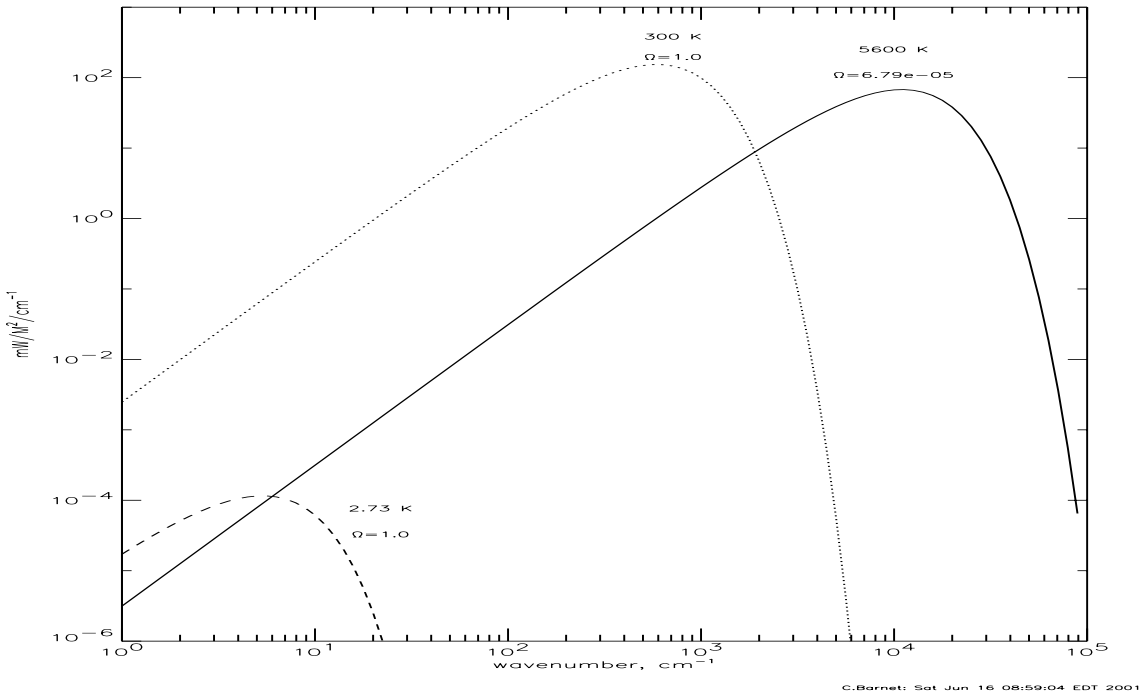


Figure 5.4: Example of the Planck function for black bodies at 2.73, 300, and 5600 K

In the tables below we compare the accuracy of the integral of the Planck function with the theoretical value. In the first table a linear quadrature was used and in the second table a logarithmic quadrature was used.

Planck Function @ T= 2.73

a	b	N	Analytic value	Midpoint error(%)	Trapezoid error(%)	Simpsons error(%)	Gaussian error(%)
0.0001	100.00	5	0.001003	-20.0	-99.1	-98.8	35.7
0.0001	100.00	7	0.001003	44.2	-85.9	-81.3	-15.5
0.0001	100.00	9	0.001003	44.8	-59.6	-46.4	-1.3
0.0001	100.00	11	0.001003	29.9	-36.0	-16.4	0.489491
0.0001	100.00	21	0.001003	2.7	-3.1	7.9	-0.000093
0.0001	100.00	51	0.001003	0.069240	-0.079285	0.316662	-0.000070
0.0001	100.00	101	0.001003	0.004250	-0.005028	0.019728	-0.000081
0.0001	100.00	1001	0.001003	-0.000151	-0.000081	-0.000070	-0.000070

Planck(ln(x)) Function @ T= 2.73

a	b	N	Analytic value	Midpoint error(%)	Trapezoid error(%)	Simpsons error(%)	Gaussian error(%)
-11.50	5.00	5	0.001003	-69.9	-36.1	-14.8	18.2
-11.50	5.00	7	0.001003	-57.4	80.7	140	-68.3
-11.50	5.00	9	0.001003	51.4	-53.0	-58.7	49.1
-11.50	5.00	11	0.001003	-8.0	7.8	-25.7	-32.0
-11.50	5.00	21	0.001003	0.114747	-0.114932	-2.7	0.827449
-11.50	5.00	51	0.001003	-0.000046	-0.000070	-0.022782	-0.000081
-11.50	5.00	101	0.001003	-0.000058	-0.000070	-0.000058	-0.000070
-11.50	5.00	1001	0.001003	-0.000035	-0.000070	-0.000058	-0.000070

Notice that the midpoint, trapezoidal, and Simpson's rule all improve dramatically with the logarithmic substitution. The Gaussian quadrature out performs the other quadratures; however, it has difficulty with the logarithmic substitution because the shape of the Planck function is too complex for small numbers of points. That is, the symmetric Legrendre polynomials do not fit function well. We could experiment with the extent of the limits or make a symmetric function by computing the integral from $-\infty \leq \nu \leq +\infty$ and $B_{-\nu}(T) = B_{\nu}(T)$. With the finite integral the limits of $-b \leq \nu \leq +b$ would be used and for the logarithmic the limits of $-ln(b) \leq \nu \leq +ln(b)$ would be used with the interval of $-ln(a) \leq \nu \leq +ln(a)$ removed from the computation.

For $T = 300$ K we multiply the integral by 10^{-6} so that the answer is easy to print in our table. In this case, all quadratures improve with the logarithmic substitution for moderate number of points.

1.0E-6*Planck Function @ T= 300.0

a	b	N	Analytic value	Midpoint error(%)	Trapezoid error(%)	Simpsons error(%)	Gaussian error(%)
0.01	9999.99	5	0.146201	-0.632339	-98.024612	-97.366211	21.145548
0.01	9999.99	7	0.146201	48.425793	-78.704773	-71.644615	-13.240739
0.01	9999.99	9	0.146201	39.059116	-49.328472	-33.096432	0.010294
0.01	9999.99	11	0.146201	23.303120	-27.544102	- 6.360752	0.251678
0.01	9999.99	21	0.146201	1.854861	-2.120492	6.354045	-0.000071
0.01	9999.99	51	0.146201	0.047496	-0.054478	0.217167	-0.000061
0.01	9999.99	101	0.146201	0.002915	-0.003486	0.013515	-0.000071
0.01	9999.99	1001	0.146201	-0.000163	-0.000061	- 0.000061	-0.000061

1.0E-6*Planck(ln(x)) Function @ T= 300.0

a	b	N	Analytic value	Midpoint error(%)	Trapezoid error(%)	Simpsons error(%)	Gaussian error(%)
-10.00	11.50	5	0.146201	-99.745110	140.339905	220.453171	260.781158
-10.00	11.50	7	0.146201	60.238514	-94.566589	-94.242630	-79.089493
-10.00	11.50	9	0.146201	-48.024426	20.297398	-19.716776	-19.899742
-10.00	11.50	11	0.146201	-3.351351	0.961763	-26.477079	47.568653
-10.00	11.50	21	0.146201	1.203677	-1.194799	-1.913650	9.407701
-10.00	11.50	51	0.146201	-0.000031	-0.000163	0.017011	-0.001457
-10.00	11.50	101	0.146201	-0.000092	-0.000071	-0.000082	-0.000061
-10.00	11.50	1001	0.146201	-0.000071	-0.000061	-0.000061	-0.000071

For $T = 5600$ K we multiply the integral by 10^{-6} so that the answer is easy to print in our table.

1.0E-6*Planck Function @ T= 5600.0

a	b	N	Analytic value	Midpoint error(%)	Trapezoid error(%)	Simpsons error(%)	Gaussian error(%)
0.10	99999.99	5	17750.7	43.388065	-56.820290	-42.795555	-11.610079
0.10	99999.99	7	17750.7	16.495897	-19.192627	2.464940	0.445811
0.10	99999.99	9	17750.7	5.862456	-6.716107	9.985284	-0.005061
0.10	99999.99	11	17750.7	2.441294	-2.791104	7.489061	-0.001012
0.10	99999.99	21	17750.7	0.152899	-0.174905	0.697154	-0.000077
0.10	99999.99	51	17750.7	0.003862	-0.004566	0.017825	-0.000077
0.10	99999.99	101	17750.7	0.000154	-0.000363	0.001045	-0.000088
0.10	99999.99	1001	17750.7	-0.000286	-0.000077	-0.000077	-0.000077

1.0E-6*Planck(ln(x)) Function @ T= 5600.0

a	b	N	Analytic value	Midpoint error(%)	Trapezoid error(%)	Simpsons error(%)	Gaussian error(%)
-2.30	13.80	5	17750.7	-90.055649	184.880310	279.819519	80.917351
-2.30	13.80	7	17750.7	90.938339	-64.239487	-76.069290	30.215246
-2.30	13.80	9	17750.7	-41.868862	47.412346	1.589668	-65.108810
-2.30	13.80	11	17750.7	25.320093	-24.959747	-7.863991	23.217407
-2.30	13.80	21	17750.7	-0.180329	0.180175	8.560177	-2.167549
-2.30	13.80	51	17750.7	-0.000066	-0.000077	-0.009364	-0.000088
-2.30	13.80	101	17750.7	-0.000077	-0.000077	-0.000066	-0.000055
-2.30	13.80	1001	17750.7	-0.000110	-0.000077	-0.000088	-0.000077

5.4 Gaussian quadrature rule

For a small panel, h, the integrand $f(x)$ can be well represented by a low-order polynomial, $p(x)$. Since, $p(x)$ can be integrated analytically we can obtain an estimate of

$$\int_{x_n}^{x_n+h} f(x) \cdot dx \simeq \int_{x_n}^{x_n+h} p(x) \cdot dx \tag{5.53}$$

However, the midpoint rule, $p(x) = a_0$, gives a better estimate of the integral than the trapezoidal rule, $p(x) = a_0 + a_1 \cdot x$. We want to compute $\int f(x)dx$, not $f(x)$ so we should seek a quadrature rules that are best at representing the integral rather than $f(x)$.

If $p_n(x)$ is a polynomial of degree n such that

$$\int_a^b p_n(x) \cdot x^k dx = 0 \qquad \text{for} \qquad k = 0, \ldots, n-1 \tag{5.54}$$

and hence p is *orthogonal* on $[a, b]$ to all polynomials of degree less that n, then it can be shown that

1. The n zeros of the polynomial are real, simple and lie in the open interval [a,b].
2. The n point interpolatory quadrature rule on $[a, b]$ whose nodes are the zeros of $p(x)$ is a polynomial of degree $2n - 1$

The Gaussian quadrature rule uses zeroes of orthogonal polynomials to compute the integral over the interval $[a, b]$ as given in Eqn. 5.2; however, the spacing of the points is now non-uniform

$$I = \int_a^b w(x) \cdot f(x) \cdot dx \simeq \sum_{i=1}^{N} (c_i \cdot f(x_i)) + R_n \tag{5.55}$$

Here is a program (gausquad.F) to compute integral if the weights, w_i, and sample values, $f(x_i)$ are given.

```
      function gausquad(npts,Wgt,Ydata)
      implicit none
      integer*2 npts
      real*4    gausquad,Wgt(*),Ydata(*)
```

```
c     input:
c       npts = # of data points
c       Wgt(1)..Wgt(npts) = weights
c       Ydata(1)..Ydata(npts) are the user supplied values
c          corresponding to Xpts(1)..Xpts(npts)
c
c     output:
c       gausquad = the value of the integral over the limits
c                  specified in call gauleg()

      integer*2 i
      real*8    sum

      sum = 0.0d00
      do i = 1,npts
        sum = sum  + dble(Wgt(i))*dble(Ydata(i))
      enddo
      gausquad = sngl(sum)
      return
      end
```

Some orthogonal polynomials which are useful for Guassian quadrature are:

Table 5.1: Polynomial and recurrence relations of Gauss quadrature formula

$[a,b]$	$w(x)$	Gauss-	Recurrence Relation
$[-1,1]$	1	Legendre	$(i+1)P_{i+1} = (2i+1)xP_i - iP_{i-1}$
$[-1,1]$	$(1-x^2)^{1/2}$	Chebyshev	$T_{i+1} = 2xT_i - T_{i-1}$
$[0,\infty]$	Laguerre(c)	$x^c e^{-x}$	$(i+1)L^c_{i+1} = (-1+2i+c+1)L^c_i - (i+c)L^c_{i-1}$
$[-\infty,\infty]$	e^{-x^2}	Hermite	$H_{i+1} = 2xH_i - 2iH_{i-1}$

The weights can be determined by the method of undetermined coefficients (*e.g.*, Heath, see Simpson's example on pg. 242). If we force the quadrature rule to integrate each of the polynomial basis functions exactly, then by linearity, it will integrate any polynomial of degree $N-1$ exactly. As an example, consider a 3 point Guassian quadrature. The integral is given by

$$I = \sum_{i=1}^{N} (c_i \cdot f(x_i)) = c_1 \cdot f(x_1) + c_2 \cdot f(x_2) + c_3 \cdot f(x_3) \tag{5.56}$$

In the previous quadrature methods we presumed we knew x_1, x_2, x_3. We can insert our monomial functions $1, x, x^2$ and force them to fit the following set of 3 equations with 3 unknowns

$$c_1 \cdot 1 + c_2 \cdot 1 + c_3 \cdot 1 = \int_a^b 1 \cdot dx = [x]_a^b = b - a \tag{5.57}$$

$$c_1 \cdot x_1 + c_2 \cdot x_2 + c_3 \cdot x_3 = \int_a^b x \cdot dx = \left[\frac{x^2}{2}\right]_a^b = \frac{b^2 - a^2}{2} \tag{5.58}$$

$$c_1 \cdot x_1^2 + c_2 \cdot x_2^2 + c_3 \cdot x_3^2 = \int_a^b x^2 \cdot dx = \left[\frac{x^3}{3}\right]_a^b = \frac{b^3 - a^3}{3} \tag{5.59}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ x_1^2 & x_2^2 & x_3^2 \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} b-a \\ \frac{b^2-a^2}{2} \\ \frac{b^3-a^3}{3} \end{bmatrix} \tag{5.60}$$

For example, in Simpson's quadrature rule, $x_1 = a, x_2 = (b-a)/2, x_3 = b$ and for an integral over the limits $a = -1, b = 1$ the matrix becomes

$$\begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 2/3 \end{bmatrix} \tag{5.61}$$

and we find that $c_1 = c_3 = 1/3$ and $c_2 = 4/3$. This is exactly the same form as Simpson's quadrature found in Eqn. 5.26.

$$I == \frac{1}{3} \cdot f(-1) + \frac{4}{3} \cdot f(0) + \frac{1}{3} \cdot f(+1) \tag{5.62}$$

Gaussian integration relies on the fact that allowing x_i to be a free-parameter allows us to "fit" to a polynomial of higher degree. Again, take the example of a 3 point Gaussian quadrature, we can insert our monomial functions $1, x, x^2, x^3, x^4, x^5$ and force them to fit the following set of 6 equations with 6 unknowns

$$\begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ x_1^2 & x_2^2 & x_3^2 \\ x_1^3 & x_2^3 & x_3^3 \\ x_1^4 & x_2^4 & x_3^4 \\ x_1^5 & x_2^5 & x_3^5 \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} b-a \\ \frac{b^2-a^2}{2} \\ \frac{b^3-a^3}{3} \\ \frac{b^4-a^4}{4} \\ \frac{b^5-a^5}{5} \\ \frac{b^6-a^6}{6} \end{bmatrix} \tag{5.63}$$

which can be solved by Gaussian elimination or least squares methods. Note that an n point Gaussian quadrature allows us to solve an integrand up to a polynomial order of $2n-1$ instead of order $n-1$ as with Newton-Cotes rules.

5.4.1 Truncation Error for Gaussian quadrature

When approximating a set of n points, $y(x_i)$, with a polynomial of order $n-1$, $g(x)$, the error is of the form given by

$$y(x) - g(x) = \frac{y^{(n)}(\zeta)}{n!} \cdot \prod_{i=1}^{n} (x - x_i) \tag{5.64}$$

where $y^{(i)}(\zeta)$ means the i^{th} derivative of $y(x)$ evaluated at ζ which is an value of x at which the i^{th} derivative has an extrema within the interval of $[a, b)$. For an integral expression the polynomial expression for the derivative is given by

$$y'(x) - g'(x) = \prod_{i=1}^{n} (x - x_i) \tag{5.65}$$

$$y(x) - g(x) = C \cdot \left[\prod_{i=1}^{n} (x - x_i) \right]^2 \tag{5.66}$$

and the constant can be exaluated by differentiation of the polynomial and comparison with the Taylor expansion of the function to give

$$C = \frac{y^{(2n)}(\zeta)}{(2n)!} \tag{5.67}$$

The trucation error for an n-point Gaussian guadrature can be then found by noting that the error is given by

$$\int_a^b w(x) \cdot y(x)\, dx - \sum_{i=1}^{n} w_i \cdot f(x_i) = \frac{y^{(2n)}(\zeta)}{(2n)!} \cdot \int_a^b w(x) \cdot \left[\prod_{i=1}^{n} (x - x_i) \right]^2 \tag{5.68}$$

5.5 Gauss-Legendre quadrature rule

5.5.1 properties of Legendre polynomials

The n^{th} Legendre polynomial provides a suitable polynomial over the limits

$$P_0(x) = 1 \tag{5.69}$$
$$P_1(x) = x \tag{5.70}$$
$$P_2(x) = \frac{1}{2}\left(3 \cdot x^2 - 1\right) \tag{5.71}$$
$$P_3(x) = \frac{1}{2}\left(5 \cdot x^3 - 3 \cdot x\right) \tag{5.72}$$
$$P_4(x) = \frac{1}{8}\left(35 \cdot x^4 - 30 \cdot x^2 + 3\right) \tag{5.73}$$
$$P_5(x) = \frac{1}{8}\left(65 \cdot x^5 - 70 \cdot x^3 + 15 \cdot x\right) \tag{5.74}$$
$$P_{i+1}(x) = \frac{(2i+1) \cdot x \cdot P_i(x) - i \cdot P_{i-1}(x)}{i+1} \tag{5.75}$$

Note that the recussion relation in Eqn 5.75 is simple to program and display (see Fig. 5.5). In IDL the Legendre polynomials over an equally spaced interval of N points can be computed as follows

```
x = -1 + 2*make_array(N,/index)/(N-1)  ; an array of x(N) values
y_0 = replicate(1.0, N)                ; initial polynomial, P_0(N)=1
plot, x, y_0
y_1 = x                                ; P_1(N) = x
plot, x, y_1
for i = 1, K do begin
  y = ((2i+1)*x*y_1 - y_0*i)/(i+1)     ; compute P_{i+1}
  plot, x, y
  y0 = y_1
  y1 = y
endfor
```

The Legendre polynomials are orthogonal, such that

$$\int_{-1}^{+1} P_n(x) \cdot P_m(x) dx = \frac{2}{2n+1} \cdot \delta_{m,n} \tag{5.76}$$

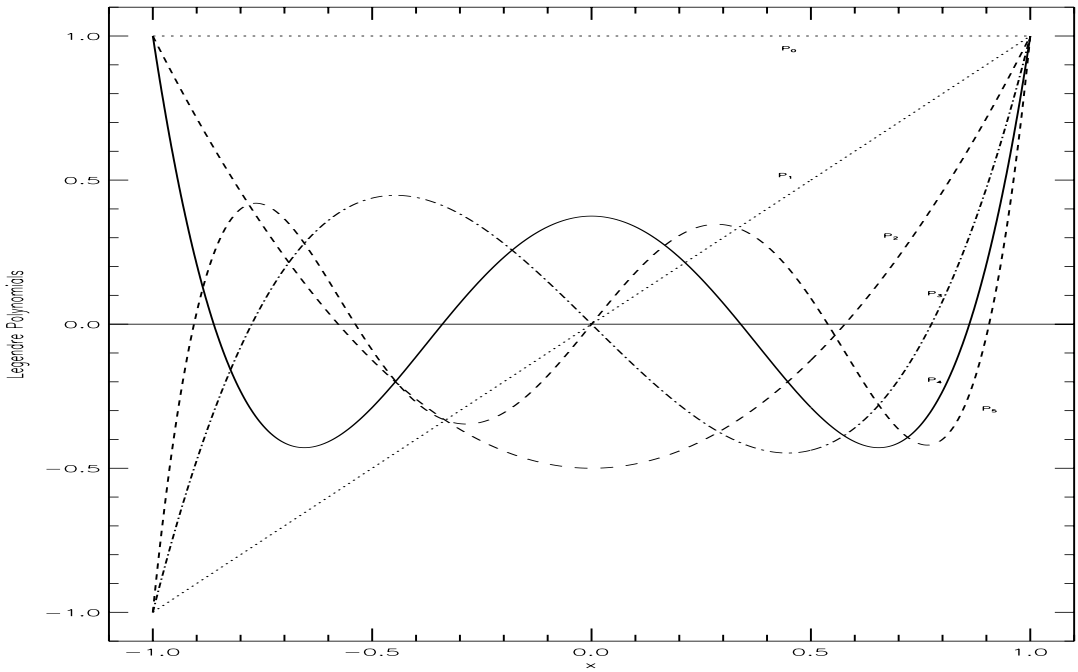


Figure 5.5: The Legendre polynomials

5.5.2 Gauss Quadrature Solution for Two Points

The 2-point Guassian formula is written as:

$$\int_{-1}^{1} f(x)\ dx = c_1 \cdot f(x_1) + c_2 \cdot f(x_2) \tag{5.77}$$

We can select x's & w's so that 2 point integral formula is exact for a polynomial up to order 3. We can solve the following four equations for the four unknowns:

$$f = 1 \quad \Rightarrow \quad 2 = c_1 + c_2 \tag{5.78}$$
$$f = x \quad \Rightarrow \quad 0 = c_1 \cdot x_1 + c_2 \cdot x_2 \tag{5.79}$$
$$f = x^2 \quad \Rightarrow \quad \frac{2}{3} = c_1 \cdot x_1^2 + c_2 \cdot x_2^2 \tag{5.80}$$
$$f = x^3 \quad \Rightarrow \quad 0 = c_1 \cdot x_1^3 + c_2 \cdot x_2^3 \tag{5.81}$$

Eqn. 5.79 can be written

$$c_1 = -c_2 \cdot x_2 / x_1 \tag{5.82}$$

and Eqn. 5.81 can be written as

$$c_2 \cdot x_2 \cdot x_1^2 = c_2 \cdot x_2^3 \tag{5.83}$$
$$x_1^2 = x_2^2 \tag{5.84}$$

so that the non-trivial answer of the pair of equations (Eqn. 5.79 & 5.81) yields

$$x_1 = -x_2 \tag{5.85}$$
$$c_1 = c_2 \tag{5.86}$$

and from Eqn. 5.78 we find $c_1 = c_2 = 1$ and from Eqn. 5.80 we see that

$$\frac{2}{3} = c_1 \cdot x_1^2 + c_2 \cdot x_2^2 \tag{5.87}$$
$$\frac{2}{3} = x_1^2 + x_1^2 \tag{5.88}$$
$$x_1 = \frac{\pm 1}{\sqrt{3}} \tag{5.89}$$

$$x_1 = \frac{-1}{\sqrt{3}} \tag{5.90}$$
$$x_2 = \frac{+1}{\sqrt{3}} \tag{5.91}$$
$$c_1 = c_2 = 1 \tag{5.92}$$

These two points can integrate cubic polynomials over the inteval ± 1 exactly. One can also notice that these points are the zeroes of the 2nd Legendre polynomial, $P_2(x)$.

5.5.3 Gauss Quadrature Solution for Three Points

We can select x's & w's so that 3 point integral formula is exact for a polynomial up to order 5. Now we solve the following six equations for the six unknowns:

$$f = 1 \quad \Rightarrow \quad 2 = c_1 + c_2 + c_3 \tag{5.93}$$
$$f = x \quad \Rightarrow \quad 0 = c_1 \cdot x_1 + c_2 \cdot x_2 + c_3 \cdot x_3 \tag{5.94}$$
$$f = x^2 \quad \Rightarrow \quad \frac{2}{3} = c_1 \cdot x_1^2 + c_2 \cdot x_2^2 + c_3 \cdot x_3^2 \tag{5.95}$$
$$f = x^3 \quad \Rightarrow \quad 0 = c_1 \cdot x_1^3 + c_2 \cdot x_2^3 + c_3 \cdot x_3^3 \tag{5.96}$$
$$f = x^4 \quad \Rightarrow \quad \frac{2}{5} = c_1 \cdot x_1^4 + c_2 \cdot x_2^4 + c_3 \cdot x_3^4 \tag{5.97}$$
$$f = x^5 \quad \Rightarrow \quad 0 = c_1 \cdot x_1^5 + c_2 \cdot x_2^5 + c_3 \cdot x_3^5 \tag{5.98}$$

Again, we can find that $x_1 = -x_3$, $c_1 = c_3$, $c_2 = 2 \cdot (1 - c_1)$ $x_2 = 0$. These will yield the solution of

$$x_1 = -\sqrt{\frac{3}{5}} = -0.77459666 \tag{5.99}$$
$$x_2 = 0 \tag{5.100}$$
$$x_3 = +\sqrt{\frac{3}{5}} = +0.77459666 \tag{5.101}$$
$$c_1 = c_3 = \frac{5}{9} = 0.55555556 \tag{5.102}$$
$$c_2 = \frac{8}{9} = 0.88888889 \tag{5.103}$$

One can also notice that these points are the zeroes of the 3rd Legendre polynomial, $P_3(x)$. In general, we find that for n points, we get an exact formula for a polynomial of degree $2n-1$.

Some examples of zeros of Legendre polynomials (for $N = 3, 5,$ and 11) are given here:

Table 5.2: The zeroes, x_i, and weights, w_i, for Gauss-Legendre Quadrature

n	$x(i)$	$w(i)$	n	$x(i)$	$w(i)$
2	± 0.57735027	1.00000000	11	± 0.97822866	0.05566857
				± 0.88706337	0.12558037
3	± 0.77459667	0.55555556		± 0.73015201	0.18629021
	0.0	0.88888889		± 0.51909613	0.23319376
				± 0.26954316	0.26280454
4	± 0.86113631	0.34785485		0.000000	0.27292509
	± 0.33998104	0.65214515			
			12	± 0.98156063	0.04717534
5	± 0.90617985	0.23692689		± 0.90411726	0.10693933
	± 0.53846931	0.47862867		± 0.76990267	0.16007833
	0.00000000	0.56888889		± 0.58731795	0.20316743
				± 0.36783150	0.23349254
6	± 0.93246951	0.17132449		± 0.12523341	0.24914705
	± 0.66120939	0.36076157			
	± 0.23861919	0.46791393	14	± 0.98628381	0.03511946
				± 0.92843488	0.08015809
7	± 0.94910791	0.12948497		± 0.82720132	0.12151857
	± 0.74153119	0.27970539		± 0.68729290	0.15720317
	± 0.40584515	0.38183005		± 0.51524864	0.18553840
	0.00000000	0.41795918		± 0.31911237	0.20519846
				± 0.10805495	0.21526385
8	± 0.96028986	0.10122854			
	± 0.79666648	0.22238103	16	± 0.98940093	0.02715246
	± 0.52553241	0.31370665		± 0.94457502	0.06225352
	± 0.18343464	0.36268378		± 0.86563120	0.09515851
				± 0.75540441	0.12462897
10	± 0.97390653	0.06667134		± 0.61787624	0.14959599
	± 0.86506337	0.14945135		± 0.45801678	0.16915652
	± 0.67940957	0.21908636		± 0.28160355	0.18260342
	± 0.43339539	0.26926672		± 0.09501251	0.18945061
	± 0.14887434	0.29552422			

In section B.6 is a program (gauleg.F) to compute the zeroes of the Legendre polynomial, $P_N(x)$, for any number of sample points, N=Npts, over an interval (x0=x_1,x1=x_N). It is a little tricky in that it uses a special formula (Press *et al.*, 1986) to compute the weights, rather than the method of undetermined coefficients described above.

$$w_i = \frac{2}{(1-x_i^2)\cdot[P_n'(x_i)]^2} \tag{5.104}$$

For the Gauss-Legendre form, $w(x) = 1$. Using the definition of Legendre polynomials,

$$P_n(x) = \frac{1}{2^n n!} \frac{\partial^n}{\partial x^n} \left(x^2 - 1\right)^n \tag{5.105}$$

we can show that (*e.g.*, see Scheid, pg. 147)

$$\prod_{i=1}^{n} (x - x_i) = \frac{2^n (n!)^2}{(2n)!} \cdot P_n(x) \tag{5.106}$$

where $x_1, x_2, \ldots, x_n$ are the zeroes ofe the Legendre polynomals. Substitution of this form into Eqn. 5.68. Finally, using the orthogonality of the Legendre polynomails results in

$$\int_a^b [P_n(x)]^2 = \frac{2}{2n+1} \tag{5.107}$$

$$e = \frac{f^{(2n)}(\zeta)}{(2n)!} \cdot \left[\frac{2^n (n!)^2}{(2n)!}\right]^2 \cdot \frac{2}{2n+1} \tag{5.108}$$

Another common form of this expression is given as

$$e = \frac{f^{(2n)}(\zeta)}{(2n)!} \left(\frac{2}{2n+1} - \sum_{i=0}^{n} w_i \cdot x_i^{2n}\right) \tag{5.109}$$

5.6 Gauss-Hermite Quadrature Rule

The Gauss-Hermite quadrature formulas are used to solve equations of the form

$$\int_{-\infty}^{\infty} e^{-x^2} \cdot y(x) \cdot dx \quad = \quad \sum_{i=1}^{n} w_i \cdot y(x_i) \tag{5.110}$$

where the arguments, x_i, are zeros of the n^{th} Hermite polynomial

$$H_n(x) = (-1)^n \cdot e^{x^2} \cdot \frac{d^n}{dx^n} \left(e^{-x^2}\right) \tag{5.111}$$

and the coefficients, w_i, are given by

$$w_i = \frac{2^{n+1} \cdot n! \cdot \sqrt{\pi}}{[H_n'(x_i)]^2} \tag{5.112}$$

and for a n term approximation the quadrature has a truncation error of

$$E = \frac{n! \cdot \sqrt{\pi} \cdot y^{(2n)}(\zeta)}{2^n \cdot (2n)!} \tag{5.113}$$

where $y^{(2n)}(\zeta)$ is the $(2n)^{\text{th}}$ derivative of y evaluated at an extremum, $x = \zeta$.

5.6.1 Example to determine Gauss-Hermite coefficients

Determine the values of the coefficients (*i.e.* determine x_i, w_i) for $n = 2$ for the Gauss-Hermite quadrature using the formula given on the previous page.

Using Eqn. 5.111 for $n = 2$ yields

$$H_2(x) = (-1)^2 e^{x^2} \frac{d^2}{dx^2} \left(e^{-x^2}\right) = 4x^2 - 2$$

Table 5.3: The zeroes, x_i, and weights, w_i, for Gauss-Hermite Quadrature

n	x_k	w_k		n	x_k	w_k
2	± 0.70710678	0.88622693		12	± 0.31424038	0.57013524
					± 0.94778839	0.26049231
4	± 0.52464762	0.80491409			± 1.59768264	0.05160799
	± 1.65068912	0.08131284			± 2.27950708	0.00390539
					± 3.02063703	0.00008574
6	± 0.43607741	0.72462960			± 3.88972490	0.00000027
	± 1.33584907	0.15706732				
	± 2.35060497	0.00453001		14	± 0.29174551	0.53640591
					± 0.87871379	0.27310561
8	± 0.38118699	0.66114701			± 1.47668273	0.06850553
	± 1.15719371	0.20780233			± 2.09518326	0.00785005
	± 1.98165676	0.01707798			± 2.74847072	0.00035509
	± 2.93063742	0.00019960			± 3.46265693	0.00000472
					± 4.30444857	0.00000001
10	± 0.34290133	0.61086263				
	± 1.03661082	0.24013861				
	± 1.75668365	0.03387439				
	± 2.53273167	0.00134365				
	± 3.43615912	0.00000764				

The zeros of this polynomial are then at $x^2 = \frac{1}{2}$ or

$$x_{1,2} = \frac{\pm 1}{\sqrt{2}} = \pm\frac{\sqrt{2}}{2} = \pm 0.70710678$$

and w_i are given by Eqn. 5.112 (NOTE: $H_2'(x) = 8x$)

$$c_1 = c_2 = \frac{16\sqrt{\pi}}{\left[4\sqrt{2}\right]^2} = \frac{\sqrt{\pi}}{2} = 0.88622693$$

5.6.2 Using Gauss-Hermite to Evaluate $\int_{-\infty}^{\infty} e^{-x^2}x^2dx$

Use the 2-point Gauss-Hermite formula to determine the integral of $\int_{-\infty}^{\infty} e^{-x^2}x^2dx$. Show the steps in your computation. Compare your answer to the analytic value of $\sqrt{\pi}/2$. Does this agree with the truncation error estimate (Eqn. 5.113)?

Using Eqn. 5.110 we have

$$\begin{aligned}
\int_{-\infty}^{\infty} e^{-x^2}x^2dx &= \sum_{i=1}^{2} w_i \cdot y(x_i), \qquad y(x) = x^2 \\
&= \frac{\sqrt{\pi}}{2} \cdot \left[y\left(\frac{+\sqrt{2}}{2}\right) + y\left(\frac{-\sqrt{2}}{2}\right)\right] \\
&= \frac{\sqrt{\pi}}{2} \cdot \left[\frac{2}{4} + \frac{2}{4}\right] = \frac{\sqrt{\pi}}{2}
\end{aligned}$$

Using Eqn. 5.113 we obtain

$$E = \frac{2 \cdot \sqrt{\pi} \cdot y^{(4)}(\zeta)}{4 \cdot (24)} = 0$$

which is zero because the derivative is zero

$$\frac{d^4(\zeta^2)}{d\zeta^2} = 0$$

5.6.3 Using Gauss-Hermite to Evaluate $\int_{-\infty}^{\infty} e^{-x^2} \sin^2(x) dx$

Evaluate the integral the integral of $\int_{-\infty}^{\infty} e^{-x^2} \sin^2(x) dx$ using a 6 point Gauss-Hermite formula. The analytic value is $\sqrt{\pi}\left(1 - e^{-1}\right)/2 = 0.56020226$. Show the steps in your computation.

Note that $y(x_i) = \sin^2(-x_i) = \sin^2(x_i)$. A 6 term formula can be written as the sum of three terms

$$\begin{aligned}
\int_{-\infty}^{\infty} e^{-x^2} \sin^2(x) dx &\simeq \sum_{i=1}^{3} 2 \cdot w_i \cdot \sin^2(x_i) \\
&= 1.449259 \cdot \sin^2(.436077) + 0.314135 \cdot \sin^2(1.335849) \\
&\quad +0.009060 \cdot \sin^2(2.350605) \\
&= 1.449259 \cdot 0.178411 + 0.314135 \cdot 0.945808 + 0.009060 \cdot 0.505589 \\
&= 0.560255
\end{aligned}$$

How does the error estimate in Eqn. 5.113 compare with the error found in problem # 5.6.3?

The error in problem 5.6.3 is 0.560255-0.560202 = $0.000032 = 5.2 \cdot 10^{-5}$

Using Eqn. 5.113 we have

$$\begin{aligned}
E &= \frac{n!\sqrt{\pi} \cdot y^{(2n)}(\zeta)}{2^n \cdot (2n)!} = \frac{6!\sqrt{\pi} \cdot \frac{d^{12}(\sin^2(\zeta))}{d\zeta^{12}}}{2^6 \cdot (12)!} \\
&= \frac{720 \cdot \sqrt{\pi} \cdot 2^{11}}{2^6 \cdot 479001600} = \frac{1275 \cdot 2^5}{479001600} = 8.5 \cdot 10^{-5}
\end{aligned}$$

where we approximated the derivative as

$$\begin{aligned}
y^{(1)}(\zeta) &= 2\sin(\zeta)\cos(\zeta) = \sin(2\zeta) \\
y^{(2)}(\zeta) &= 2\cos(2\zeta) \\
y^{(3)}(\zeta) &= -4\sin(2\zeta) \\
y^{(4)}(\zeta) &= +8\cos(2\zeta) \\
\ldots &= \ldots \\
y^{(2n)}(\zeta) &= 2^{2n-1}\cos(2\zeta) \simeq 2^{2n-1}
\end{aligned}$$

5.7 Gauss-Laguerre Quadrature Rule

The Gauss-Laguerre quadrature formulas are used to solve equations of the form

$$\int_0^\infty e^{-x} \cdot y(x) \cdot dx \quad = \quad \sum_{i=1}^{n} w_i \cdot y(x_i) \tag{5.114}$$

where the arguments, x_i, are zeros of the n^{th} Laguerre polynomial

$$L_n(x) = e^x \cdot \frac{d^n}{dx^n}\left(e^{-x} \cdot x^n\right) \tag{5.115}$$

$$\begin{aligned} L_0(x) &= 1 && (5.116)\\ L_1(x) &= 1 - x && (5.117)\\ L_2(x) &= x^2 - 4 \cdot x + 2 && (5.118)\\ L_3(x) &= -x^3 + 9 \cdot x^2 - 18 \cdot x + 6 && (5.119) \end{aligned}$$

and the coefficients, w_i, are given by

$$w_i = \frac{(n!)^2}{x_i \cdot \left[L_n'(x_i)\right]^2} \tag{5.120}$$

and for a n term approximation the quadrature has a truncation error of

$$E = \frac{(n!)^2 \cdot y^{(2n)}(\zeta)}{(2n)!} \tag{5.121}$$

where $y^{(2n)}(\zeta)$ is the $(2n)^{\text{th}}$ derivative of y evaluated at an extremum, $x = \zeta$.

5.8 Gauss-Chebyshev Quadrature Rule

The Gauss-Chebyshev quadrature formulas are used to solve equations of the form

$$\int_{-1}^{1} \frac{y(x)}{\sqrt{1-x^2}} \cdot dx \quad = \quad \sum_{i=1}^{n} w_i \cdot y(x_i) \quad = \quad \frac{\pi}{n} \sum_{i=1}^{n} w_i \cdot y(x_i) \tag{5.122}$$

where the arguments, x_i, are zeros of the n^{th} Chebyshev polynomial

$$T_n(x) = \cos\left[n \cdot \cos^{-1}(x)\right] \tag{5.123}$$

which has the zeroes given by

$$x_i = \cos\left[\frac{(2i-1)\pi}{2n}\right] \tag{5.124}$$

and all the coefficients, w_i, are equal and given by

$$w_i = \frac{\pi}{n} \tag{5.125}$$

and for a n term approximation the quadrature has a truncation error of

$$E = \frac{2\pi \cdot y^{(2n)}(\zeta)}{2^{2n} \cdot (2n)!} \tag{5.126}$$

where $y^{(2n)}(\zeta)$ is the $(2n)^{\text{th}}$ derivative of y evaluated at an extremum, $x = \zeta$.

Table 5.4: The zeroes, x_i, and weights, w_i, for Gauss-Laguerre Quadrature

n	x_k	w_k		n	x_k	w_k
2	0.58578644	0.85355339		12	0.11572212	0.26473137
	3.41421356	0.14644661			0.61175748	0.37775928
					1.51261027	0.24408201
4	0.32254769	0.60315410			2.83375134	0.09044922
	1.74576110	0.35741869			4.59922764	0.02010238
	4.53662030	0.03888791			6.84452545	0.00266397
	9.39507091	0.00053929			9.62131684	0.00020323
					13.00605499	0.00000837
6	0.22284660	0.45896467			17.11685519	0.00000017
	1.18893210	0.41700083			22.15109038	0.00000000
	2.99273633	0.11337338			28.48796725	0.00000000
	5.77514357	0.01039920			37.09912104	0.00000000
	9.83746742	0.00026102				
	15.98287398	0.00000090		14	0.09974751	0.23181558
					0.52685765	0.35378469
8	0.17027964	0.36918859			1.30062912	0.25873461
	0.90370178	0.41878678			2.43080108	0.11548289
	2.25108663	0.17579499			3.93210282	0.03319209
	4.26670017	0.03334349			5.82553622	0.00619287
	7.04590540	0.00279454			8.14024014	0.00073989
	10.75851601	0.00009077			10.91649951	0.00005491
	15.74067864	0.00000085			14.21080501	0.00000241
	22.86313174	0.00000000			18.10489222	0.00000006
					22.72338163	0.00000000
10	0.13779347	0.30844112			28.27298172	0.00000000
	0.72945455	0.40111993			35.14944366	0.00000000
	1.80834290	0.21806829			44.36608171	0.00000000
	3.40143370	0.06208746				
	5.55249614	0.00950152				
	8.33015275	0.00075301				
	11.84378584	0.00002826				
	16.27925783	0.00000042				
	21.99658581	0.00000000				
	89.92069701	0.00000000				

5.9 Problems: Gaussian Quadrature

1. Apply the Gauss-Laguerre quadrature to solve

$$\int_0^\infty e^{-x} \cdot (a + b \cdot x)\, dx = a + b \tag{5.127}$$

How many points are required to solve the integral exactly. Where must the point be sampled (*i.e.*, compute x_i). Estimate the error for the n point Gauss-Laguerre quadrature using Eqn. 6.113.

$f(x) = a + b \cdot x$ is a polynomial of order 1, so that Gauss-Laguerre quadrature is exact up to $2 \cdot n - 1$ or $n = 1$. One point is required to do this integral. Begin with Eqn. 5.115

$$L_n(x) = e^x \cdot \frac{d^n}{dx^n}\left(e^{-x} \cdot x^n\right) = e^x\left(-e^{-x} \cdot x - e^{-x} \cdot 1\right) = 1 - x \tag{5.128}$$

$$L_1(x) = 0 \qquad @ \qquad x_1 = 1 \tag{5.129}$$

$$w_i = \frac{(n!)^2}{x_i \cdot [L_n'(x_i)]^2} = \frac{1}{1(-1)^2} = 1 \tag{5.130}$$

$$I = w_i \cdot (a + b \cdot x_1) = a + b \tag{5.131}$$

From Eqn. 5.121

$$E_1 = \frac{(n!)^2 \cdot y^{(2n)}(\zeta)}{(2n)!} = \frac{(1!)^2 \cdot y^{(2)}(\zeta)}{(2)!} = \frac{1}{2} \cdot 0 = 0 \tag{5.132}$$

2. Apply the Gauss-Laguerre quadrature to solve

$$\int_0^\infty e^{-x} \cdot \sin(x)\, dx = \frac{1}{2} \tag{5.133}$$

Estimate the error for a $n = 6$ Gauss-Laguerre computation using Eqn. 6.113. Compute the integral and compare the real error to the predicted error.

Estimate the error for a $n = 14$ Gauss-Laguerre computation using Eqn. 6.113. Compute the integral and compare the real error to the predicted error.

$$\begin{aligned} y(x) &= \sin(x) && (5.134)\\ y^1(x) &= \cos(x) && (5.135)\\ y^2(x) &= -\sin(x) && (5.136)\\ y^3(x) &= -\cos(x) && (5.137)\\ y^4(x) &= \sin(x) && (5.138)\\ y^{12}(x) &= \sin(x) && (5.139)\\ y^{28}(x) &= \sin(x) && (5.140) \end{aligned}$$

$$E_6 = \frac{(6!)^2 \cdot y^{(12)}(\zeta)}{(12)!} = \frac{720}{12!} \cdot \sin(\zeta) = 1.08 \cdot 10^{-3} \cdot \sin(\zeta) \leq 1.08 \cdot 10^{-3} \tag{5.141}$$

$$E_{14} = \frac{(14!)^2 \cdot y^{(28)}(\zeta)}{(28)!} = -3.0 \cdot 10^{-8} \cdot \sin(\zeta) \leq -3.0 \cdot 10^{-8} \tag{5.142}$$

$$I_n = \sum_{i=1}^{n} w_i \cdot \sin(x_i) \tag{5.143}$$

using the weights, w_i, and positions, x_i, from the table in the notes we get (DOUBLE precision in IDL):

n	I_n	error
6	0.5000494607	$4.95 \cdot 10^{-5}$
14	0.4999999918	$-8.15 \cdot 10^{-9}$

3. Apply the Gauss-Chebyshev quadrature to solve

$$\int_{-1}^{1} \frac{x^4}{\sqrt{1-x^2}} dx = \frac{3\pi}{8} \tag{5.144}$$

show that the $n = 3$ quadrature rule will result in an exact solution (since $n = 3$ results in a $2n-1 = 5^{\text{th}}$ order polynomial). For Eqn. 5.123

$$T_3(x) = \cos\left[3 \cdot \cos^{-1}(x)\right] \tag{5.145}$$

which has the zeroes given by

$$x_i = \cos\left[\frac{(2i-1)\pi}{6}\right] = \frac{\sqrt{3}}{2} \tag{5.146}$$

$$x_1 = \cos\left[\frac{\pi}{6}\right] \tag{5.147}$$

$$x_2 = \cos\left[\frac{3\pi}{6}\right] = 0 \tag{5.148}$$

$$x_3 = \cos\left[\frac{5\pi}{6}\right] = \cos\left[\frac{-1\pi}{6}\right] = \cos\left[\frac{pi}{6}\right] \tag{5.149}$$

and all the coefficients, w_i, are equal and given by

$$w_i = \frac{\pi}{3} \tag{5.150}$$

$$y(x) = x^4 \tag{5.151}$$

$$I = \sum_{i=1}^{3} w_i \cdot x_i = \frac{\pi}{3}\left[2 \cdot \cos^4\left(\frac{\pi}{6}\right)\right] = \frac{\pi}{3}\left[2 \cdot \left(\frac{\sqrt{3}}{2}\right)^4\right] = \frac{\pi}{3} \cdot 2 \cdot \frac{9}{16} = \frac{\pi \cdot 3}{8} \tag{5.152}$$

$$E_3 = \frac{2\pi \cdot y^{(6)}(\zeta)}{2^6 \cdot (6)!} = \frac{2\pi}{2^6 \cdot (6)!} \cdot y^{(6)}(\zeta) = 0 \tag{5.153}$$

5.10 Problem: Peak Oil Production

World Oil Peak Production

(an example of Differentiation and Quadrature)

I acquired an excel spreadsheet from Gary Long (an author of ref # 1 below) and extracted his estimate of the world's production rate, $p(t)$, of oil from 1857 through 2000. The table is in the file called peakprod.prn and contains the year, and 3 columns of oil production in millions of barrels per year (MBl/y) for the world, USA, and western europe, respectively. Here is a sample of that file

We will be using this data as well as estimates of the total amount of oil from a number of sources to estimate the date of peak production, t_p, of oil. The date of world peak production is extremely important to society and to climate scientists because it indicates when the price of oil will become unbounded (demand exceeds supply) and, consequently, the shift from oil to other fuels, such as coal, nuclear, and solar.

Table 5.5: Oil Production, a sample of the data file used in this section

t (year)	$p(y)$ (MBl/year)	t (year)	$p(t)$ (MBl/year)
1857	0.002	1950	3797.400
1858	0.004	1960	7673.694
1859	0.006	1970	16677.887
1860	0.509	1980	21927.712
1861	2.131	1990	22223.908
1862	3.092	1994	22592.785
1870	5.799	1995	22774.192
1890	76.633	1996	23331.462
1910	327.743	1997	24244.284
1920	688.864	1998	24875.544
1930	1409.997	1999	24248.777
1940	2150.684	2000	25016.649

Estimating the total available oil is a difficult task because countries and companies have economic and political motivation to jiggle the numbers. Campbell and Laherrere, 1998 discuss these problems. Every year *The World Almanac* publishes the estimates published by *Oil and Gas Journal* and *World Oil* for known reserves of oil in units of billions of barrels. The last few years the estimates were:

Table 5.6: Known reserves of oil

World Almanac	date of estimate	World O& GJ	World WO	USA O& GJ	USA WO
2002	1/1/2000	1016.8	981.4	21.8	21.8
2001	1/1/2000	1016.0	967.5	21.0	21.0
2000	1/1/1998	1020.1	975.0	22.5	22.5
1999	1/1/1997	1018.5	1160.1	22.0	22.0
1998	1/1/1996	1007.4	1107.3	22.4	22.4

The amount of known reserves; however, does not take into account oil discovery or improvements that might be economically feasible if the price of oil dramatically rises (current price is quite low at ≈ 18.00/barrel). The USGC (Ref. # 4) has recently published their assessment of reserve growth and undiscovered oil. Their mean value of ultimate recovery is $O_u = 3.012 \cdot 10^{12}$ barrels of oil, $3.62 \cdot 10^{11}$ of which is in the United States.

References:

- http://www.eia.doe.gov/analysis/1999anal18.html
- Campbell, C.J. & J.H. Laherrere 1998. "The End of Cheap Oil.", Scientific American **3**, pg. 78-83.
- World Almanac, 1998,1999,2000,2001,2002
- http://greenwood.cr.usgs.gov/energy/WorldEnergy/DDS-60/
- http://pubs.usgs.gov/factsheet/fs119-00/fs119-00.pdf
- http://www.hubbertpeak.com/laherrere/usgs2000/johnwood.com

4. Compute the total production of oil from 1857 through 2000. In the data provided $p(t)$ is given as the average production for an entire year, therefore, one quadrature method should appear most appropriate.

$$O(2000) = \int_{1857}^{2000} p(t')dt' \tag{5.154}$$

Use midpoint rule, since the number reflects the total production for that year. Since a year is Jan. to Dec. the mid-point of that year is most logical.

```
openr, 10, 'peakprod.prn'
  cx = ' '
  readf, 10, cx
  readf, 10, cx
  Nyr = 2000-1857+1
  ix = 0L
  rx = FLTARR(3)
  year = LONARR(Nyr)
  oil_world = FLTARR(Nyr)
  oil_us = FLTARR(Nyr)
  oil_europe = FLTARR(Nyr)
  for i = 0,  Nyr-1 do begin
    readf, 10, ix, rx
    year(i) = ix
    oil_world(i)  = rx(0)/1000.0  ; convert to billion
    oil_us(i)     = rx(1)
    oil_europe(i) = rx(2)
  endfor
close,10
oil_consumed = TOTAL(oil_world)
```

$O(2000) = 887.8$ billion barrels

5. Compute the derivative of the world production, $dp(t)/dt$. Identify the reasons for certain recent kinks in the derivative production rate. Derive a reasonable rate of increase for future production rate from your plot. Your analysis should result in an simple predictive equation, such as,

$$p(t+1) = p(t) \cdot (1.0 + C) \quad \text{for} \quad t \geq 2000 \tag{5.155}$$

```
oil_der = FLTARR(Nyr)
for i = 1, Nyr-2 do begin
   oil_der(i) = (oil_world(i+1)-oil_world(i-1))/2.0
endfor
oil_rate = 100.0*oil_der/oil_world
```

The the change in production (derivative) and the % change in products (derivative divided by production, see Fig. 5.6) has, in recent times, fallen to a very small number in the 1-4% range.

My point about the peaks are that there is little correlation with events such as weather, war, or oil production issues. To first order, the price is inversely proportional to production rate (supply versus demand); however, the price and/or production is artificially altered by OPEC, Iran, Saudi Arabia to effect the profit margins. Here are some events that are relevent.

6. Assuming that the world production will increase at the rate derived in question # 5, compute the date, t_f, when the oil will completely vanish.

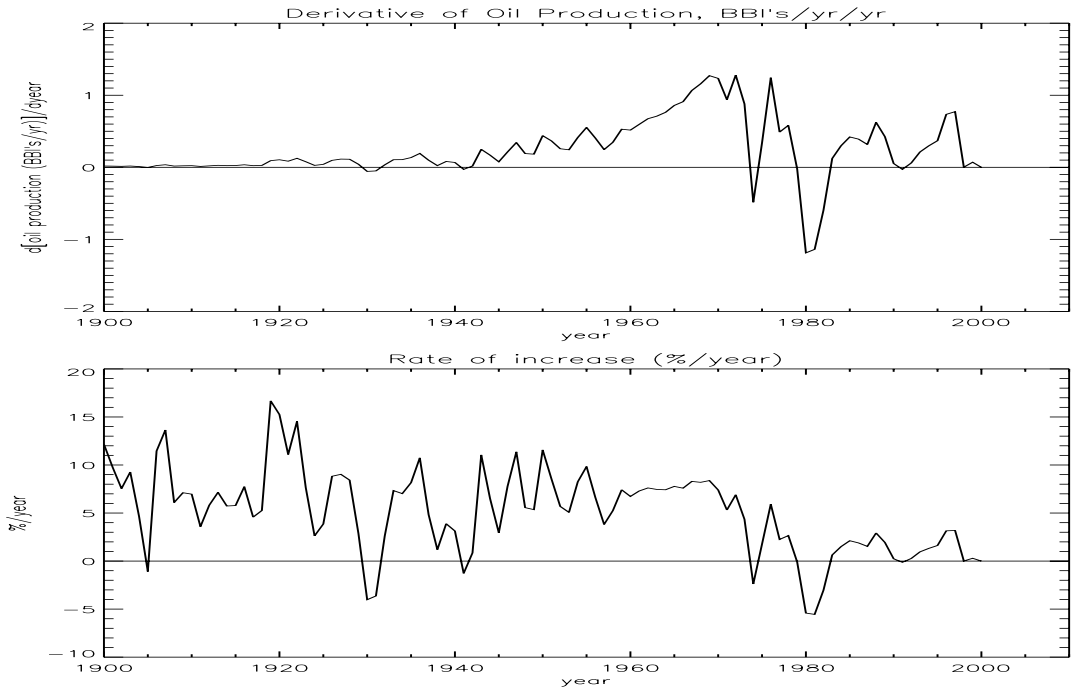


Figure 5.6: Derivative of Oil Production

```
Nyr2 = Nyr+300
year_gone = LONARR(Nyr2)
year_gone(0:Nyr-1) = year
oil_gone = FLTARR(Nyr2)
oil_gone(0:Nyr-1) = oil_world
igone = 0L
for i = Nyr, Nyr2-1 do begin
  year_gone(i) = year_gone(i-1) + 1L
  oilx = oil_gone(i-1)*pg
  if(TOTAL(oil_gone) lt oil_ultimate) then begin
     igone = i-1
     oil_gone(i) = oilx
  endif
endfor
```

Table 5.7: Example of historical events that may, or may not correlate with production of oil

date	event
6/20/77	Prudhoe Bay Alaska pipeline completed
4/1/79	OPEC hikes price 15%
11/4/79	90 hostages taken at US embassy in Iran
8/1/79	OPEC hikes price another 15%
1980	Saudi Arabia hikes price from \$19 to \$34/barrel
1996	Cold winter in US and Europe
3/24/89	Exxon Valdez spills $10.08 \cdot 10^6$ gallons
8/2/90	Iraq invades Kuwait
1/18/91	Desert Storm begins
1998-1999	OPEC cuts production to raise profits

$$O_u = \int_{1857}^{t_f} p(t')dt' \tag{5.156}$$

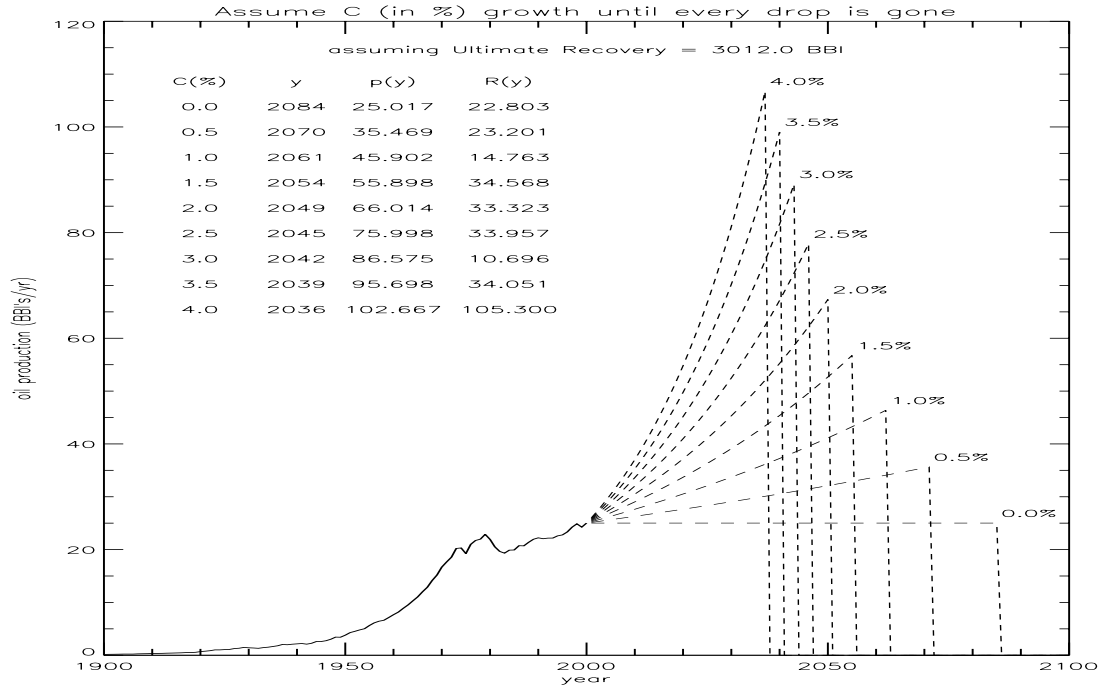


Figure 5.7: Prediction of peak-production if growth rate is maintained until every drop of oil is gone.

This can also be determined analytically by noting that the future production rate is modelled as

$$p(t) = p(0) \cdot (1+C)^t \tag{5.157}$$

where $t = 0$ is the beginning of our model, Jan. 1, 2001.

$$R(0) = \int_0^{t_p} p(0)\,(1+C)^t \cdot dt = \frac{p(0)}{\log_e(1+C)} \cdot (1+C)^t \Big|_0^{t_p} \tag{5.158}$$

noting that

$$\int b^x \cdot dx = \frac{b^x}{\log_e(b)} \qquad \text{CRC \#13} \tag{5.159}$$

so our integral becomes

$$\frac{R(0) \cdot \log_e(1+C)}{p(0)} = (1+C)^t \quad - \quad 1 \tag{5.160}$$

$$t_p = \frac{\log_e\left(\frac{R(0)\cdot\log_e(1+C)}{p(0)} + 1\right)}{\log_e(1+C)} \tag{5.161}$$

since C is a small number we can simplify the expression (NOTE: this really isn't necessary, but makes the equation cleaner)

$$\log_e(1+C) = C - \frac{C^2}{2} + \frac{C^3}{3} + \ldots \quad \simeq \quad C \qquad \text{for } |C| \ll 1 \tag{5.162}$$

$$t_p \simeq \frac{\log_e\left(\frac{R(0)\cdot C}{p(0)} + 1\right)}{\log_e(1+C)} \simeq \frac{\log_e\left(\frac{R(0)\cdot C}{p(0)} + 1\right)}{C} \tag{5.163}$$

7. M. King Hubbert published a paper in 1956 in which he predicted the US peak production in 1970. He used the idea that economics will cause the production rate to follow a bell curve. Compute the date of peak-production, t_0, assuming that the production rate will decrease at the same rate that it increases. That is, if you assume that the rate of increase is $C\%$ make a curve where $p(t+1) = p(t)\cdot(1+C)$ before t_0 and $p(t+1) = p(t)\cdot(1-C)$ after t_0. The resulting world production of oil as a function of time is shown in Fig. 5.8 along with the date of peak production using this model.

```
peak1_idx = Nyr + LONG((year_gone(igone) - 2000)/2)

lp1:
  year_peak = year_gone
  oil_peak1 = FLTARR(Nyr2)
  oil_peak1(0:Nyr-1) = oil_world
  for i = Nyr, Nyr2-1 do begin
    case 1 of
    (year_peak(i) lt year_peak(peak1_idx)): oil_peak1(i) = oil_peak1(i-1)*pg
    (year_peak(i) eq year_peak(peak1_idx)): oil_peak1(i) = oil_peak1(i-1)
    else: oil_peak1(i) = oil_peak1(i-1)/pg
    endcase
  endfor
  resid = TOTAL(oil_peak1)-oil_ultimate
  if(ABS(resid) gt oil_peak1(peak1_idx)) then begin
;   print, year_peak(peak1_idx), resid
    if(resid lt 0) then peak1_idx = peak1_idx + 1
    if(resid gt 0) then peak1_idx = peak1_idx - 1
    if(peak1_idx lt Nyr) then goto, cn1
    if(peak1_idx ge Nyr2) then stop
    goto, lp1
  endif

cn1:
```

Again, this can also be determined analytically by noting that the future production rate is modelled as

$$p(t) = p(0)\cdot(1+C)^t \qquad \text{for } t \le t_p \tag{5.164}$$

$$p(t) = p(t_p)\cdot(1-C)^{(t-t_p)} \qquad \text{for } t \ge t_p \tag{5.165}$$

where $t = 0$ is the beginning of our model, Jan. 1, 2001.

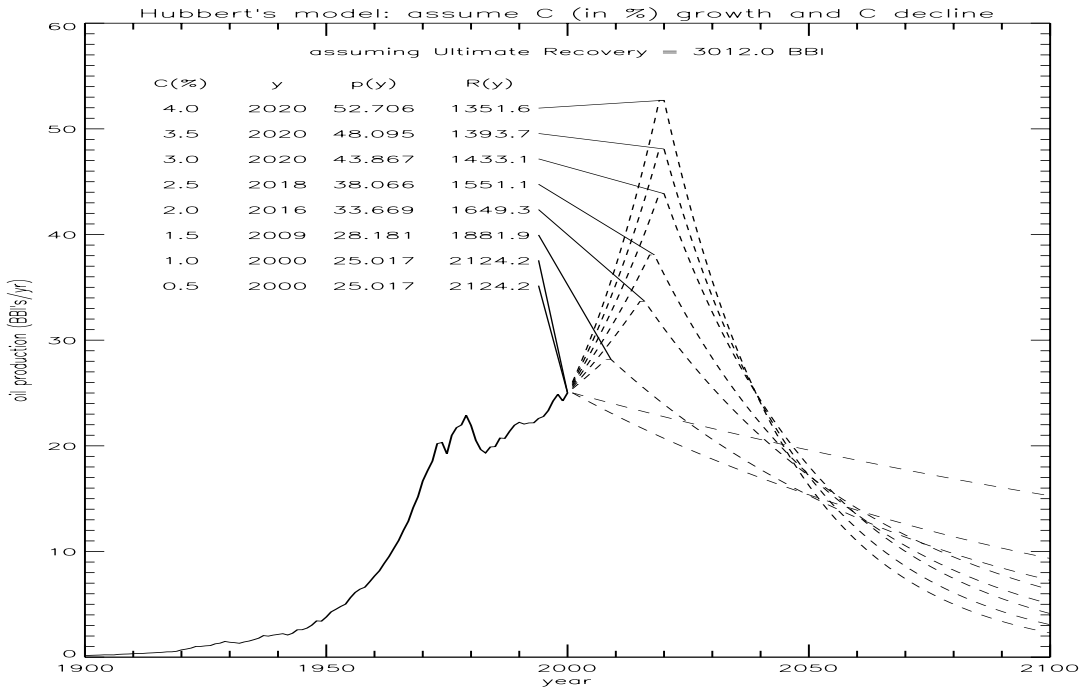


Figure 5.8: Prediction of peak-production if decay rate is set equal to growth rate

$$R(0) = \int_0^{t_p} p(0)\,(1+C)^t \cdot dt \quad + \quad \int_{t_p}^{\infty} p(t_p)\,(1-C)^t \cdot dt \tag{5.166}$$

$$R(0) = p(0) \cdot \int_0^{t_p} (1+C)^t \cdot dt \quad + \quad p(0) \cdot (1+C)^{t_p} \cdot \int_{t_p}^{\infty} (1-C)^{t-t_p} \cdot dt \tag{5.167}$$

$$= \frac{p(0)}{\log_e(1+C)} \cdot (1+C)^t \Big|_0^{t_p} \quad + \quad \frac{p(0)\cdot(1+C)^{t_p}}{\log_e(1-C)} \cdot (1-C)^{t-t_p} \Big|_{t_p}^{\infty} \tag{5.168}$$

$$= \frac{p(0)}{\log_e(1+C)} \left[\cdot (1+C)^{t_p} - 1\right] \quad - \quad \frac{p(0)\cdot(1+C)^{t_p}}{\log_e(1-C)} \cdot (1-C)^0 \tag{5.169}$$

$$= p(0) \cdot (1+C)^{t_p} \cdot \left[\frac{1}{\log_e(1+C)} - \frac{1}{\log_e(1-C)}\right] \quad - \quad \frac{p(0)}{\log_e(1+C)} \tag{5.170}$$

$$\frac{R(0)\cdot\log_e(1+C)}{p(0)} = (1+C)^{t_p} \cdot \left[1 \quad - \quad \frac{\log_e(1+C)}{\log_e(1-C)}\right] \quad - \quad 1 \tag{5.171}$$

$$(1+C)^{t_p} = \frac{\left[\frac{R(0)\cdot\log_e(1+C)}{p(0)} \quad + \quad 1\right]}{\left[1 \quad - \quad \frac{\log_e(1+C)}{\log_e(1-C)}\right]} \simeq \frac{\left[\frac{R(0)\cdot C}{p(0)} \quad + \quad 1\right]}{2} \tag{5.172}$$

8. Wood and Long derive a curve based on the idea that production rate will increase by $C\%$ per year until the ratio of reserves to annual production, $R(t)/p(t)$, reaches 10, and afterwards $p(t) = R(t)/10$.

Currently, $R(t)/p(t) \simeq 50$. For this analysis you can assume that $R(t) = O_u - \int_0^t p(t')dt'$; however, in reality it is a non-linear relationship involving many factors, including the price of oil. The resulting world production of oil as a function of time using the Wood and Long model is shown in Fig. 5.9 and compute the date of peak production using this model.

```
year_peak = year_gone
oil_peak2 = FLTARR(Nyr2)
oil_peak2(0:Nyr-1) = oil_world
for i = Nyr, Nyr2-1 do begin
   R = oil_ultimate - TOTAL(oil_peak2)
   if(R/oil_peak2(i-1) gt 10.0) then begin
     oil_peak2(i) = oil_peak2(i-1)*pg
     peak2_idx = i
   endif else begin
     oil_peak2(i) = R/10.0
   endelse
endfor
```

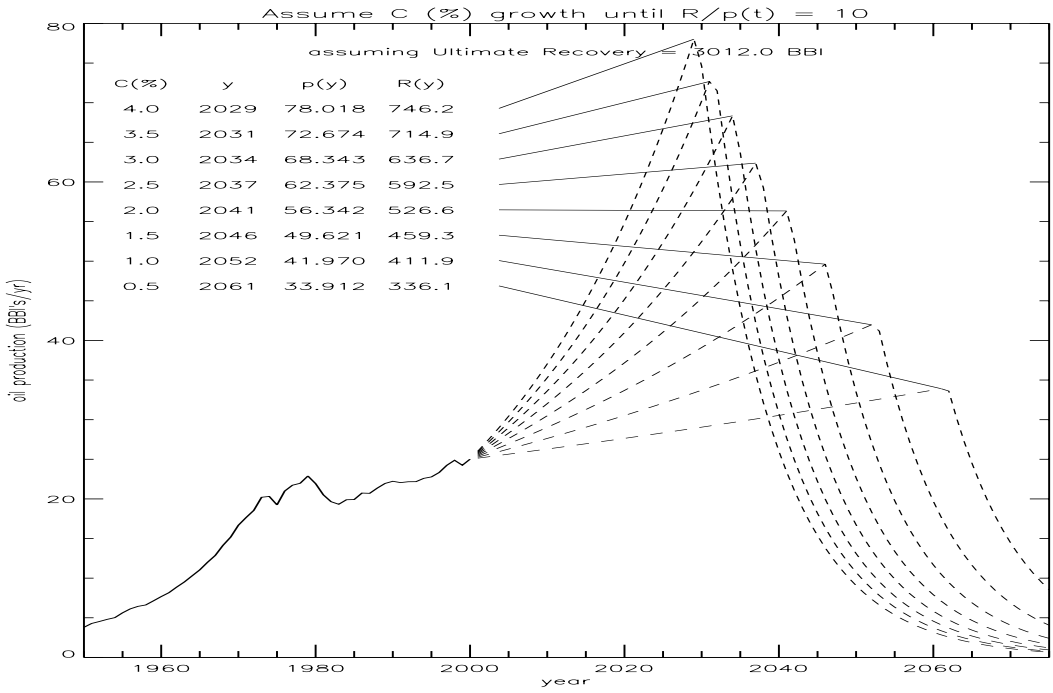


Figure 5.9: Peak production of oil using the Wood and Long assumption that the rate will fall off as the reserves/10

9. Campbell and Laherrere assume that the ultimate recovery of oil will be $O_u = 1.8 \cdot 10^{12}$ barrels. Re-compute t_0 in steps #7 and #8 with this value of ultimate recovery.

For completeness I also re-computed the date at which oil will be depleted (question # 6

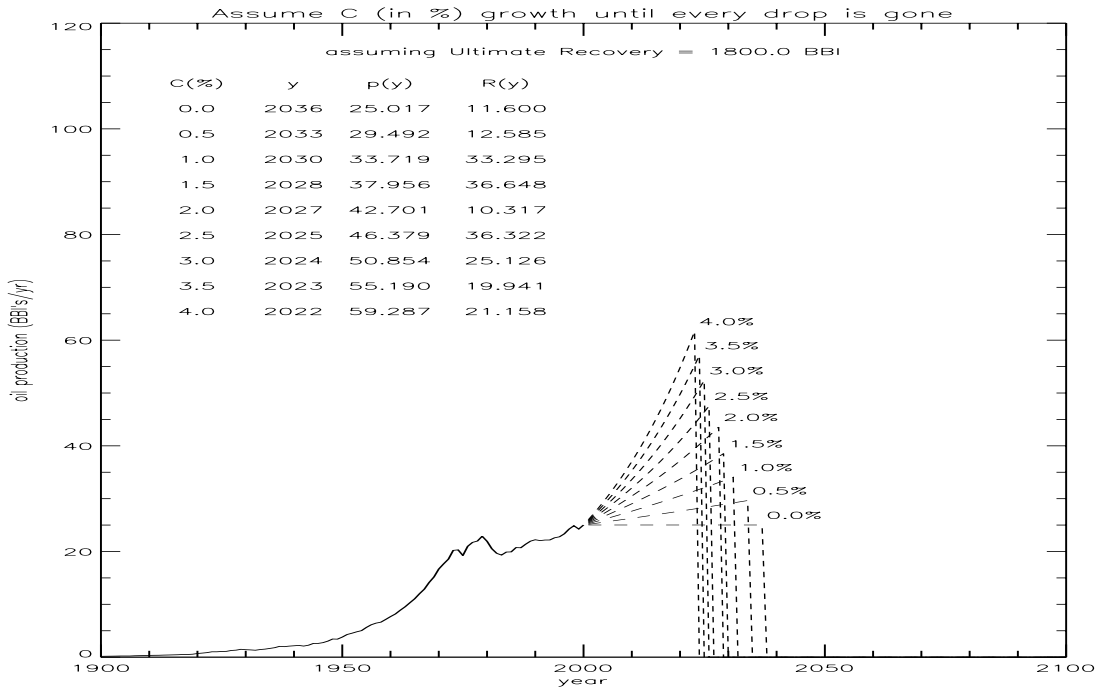


Figure 5.10: "Good to the last drop model assuming ultimate recovery of 1800 billion barrels

Assume rate of growth, r, until all oil is consumed								
	Ultimate Production in Billion Barrels (BBl)							
	1800.0		2500.0		3012.0		3500.0	
r(%)	year	BBl/y	year	BBl/y	year	BBl/y	year	BBl/y
0.5	2033	29.6	2055	33.1	2070	35.6	2083	38.0
1.0	2030	34.1	2049	41.1	2061	46.4	2071	51.2
1.5	2028	38.5	2044	48.9	2054	56.7	2062	63.9
2.0	2027	43.6	2041	57.5	2049	67.3	2056	77.3
2.5	2025	47.5	2038	65.5	2045	77.9	2051	90.3
3.0	2024	52.4	2035	72.5	2042	89.2	2047	103.4
3.5	2023	57.1	2033	80.6	2039	99.0	2043	113.7
4.0	2022	61.7	2031	87.8	2036	106.8	2041	129.9

Assume rate of growth, r, will equal decline rate								
	Ultimate Production in Billion Barrels (BBl)							
	1800.0		2500.0		3012.0		3500.0	
r(%)	year	BBl/y	year	BBl/y	year	BBl/y	year	BBl/y
0.5	2000	25.0	2000	25.0	2000	25.0	2000	25.0
1.0	2000	25.0	2000	25.0	2000	25.0	2005	26.0
1.5	2000	25.0	2000	25.0	2009	28.2	2018	32.2
2.0	2000	25.0	2007	28.2	2016	33.7	2022	37.9
2.5	2000	25.0	2011	32.0	2018	38.1	2024	44.1
3.0	2002	25.8	2013	35.7	2020	43.9	2025	50.9
3.5	2004	27.7	2014	39.1	2020	48.1	2025	57.1
4.0	2006	30.4	2015	43.3	2020	52.7	2024	61.7

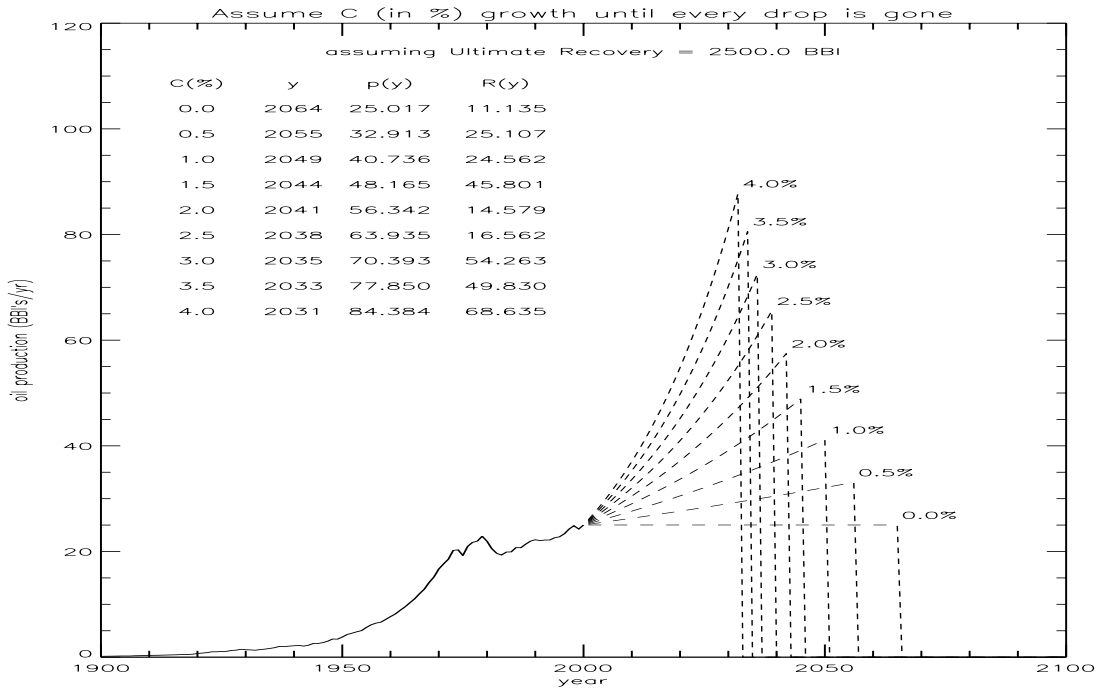


Figure 5.11: "Good to the last drop model assuming ultimate recovery of 2500 billion barrels

Assume rate of growth, r, until R/p = 10								
	Ultimate Production in Billion Barrels (BBl)							
	1800.0		2500.0		3012.0		3500.0	
r(%)	year	BBl/y	year	BBl/y	year	BBl/y	year	BBl/y
0.5	2024	28.2	2047	31.6	2061	33.9	2075	36.4
1.0	2022	31.1	2041	37.6	2052	42.0	2062	46.4
1.5	2020	33.7	2036	42.8	2046	49.6	2054	55.9
2.0	2019	36.4	2033	48.1	2041	56.3	2048	64.7
2.5	2017	38.1	2030	52.5	2037	62.4	2043	72.3
3.0	2016	40.1	2028	57.2	2034	68.3	2039	79.2
3.5	2015	41.9	2026	61.2	2031	72.7	2036	86.3
4.0	2015	45.1	2024	64.1	2029	78.0	2033	91.3

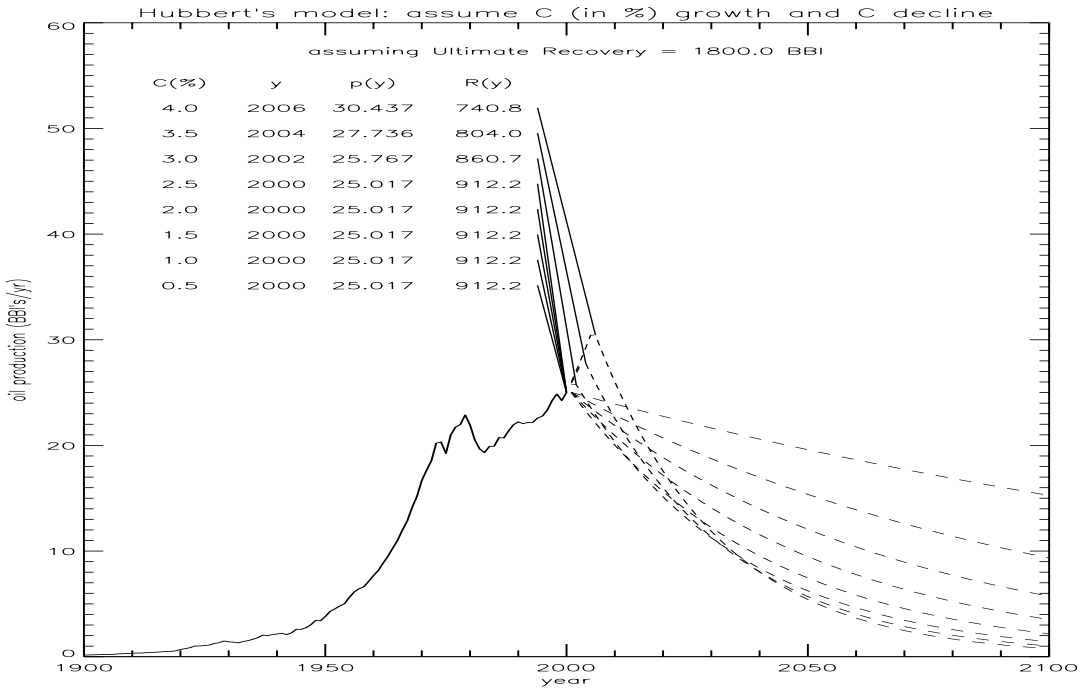


Figure 5.12: Hubbert model assuming ultimate recovery of 1800 billion barrels

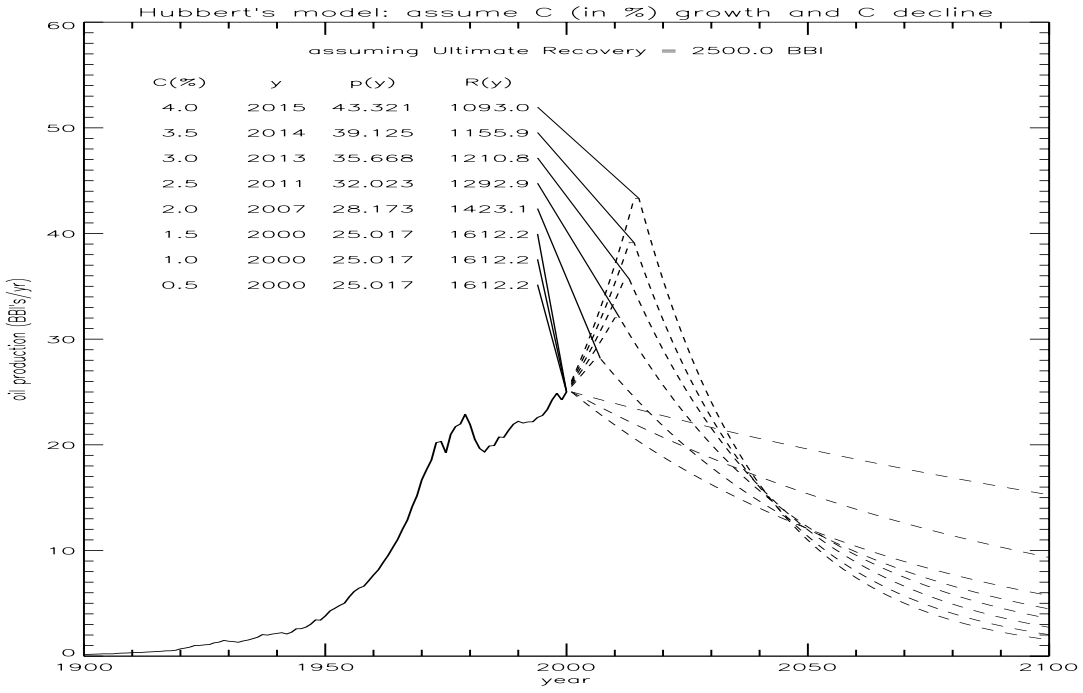


Figure 5.13: Hubbert model assuming ultimate recovery of 2500 billion barrels

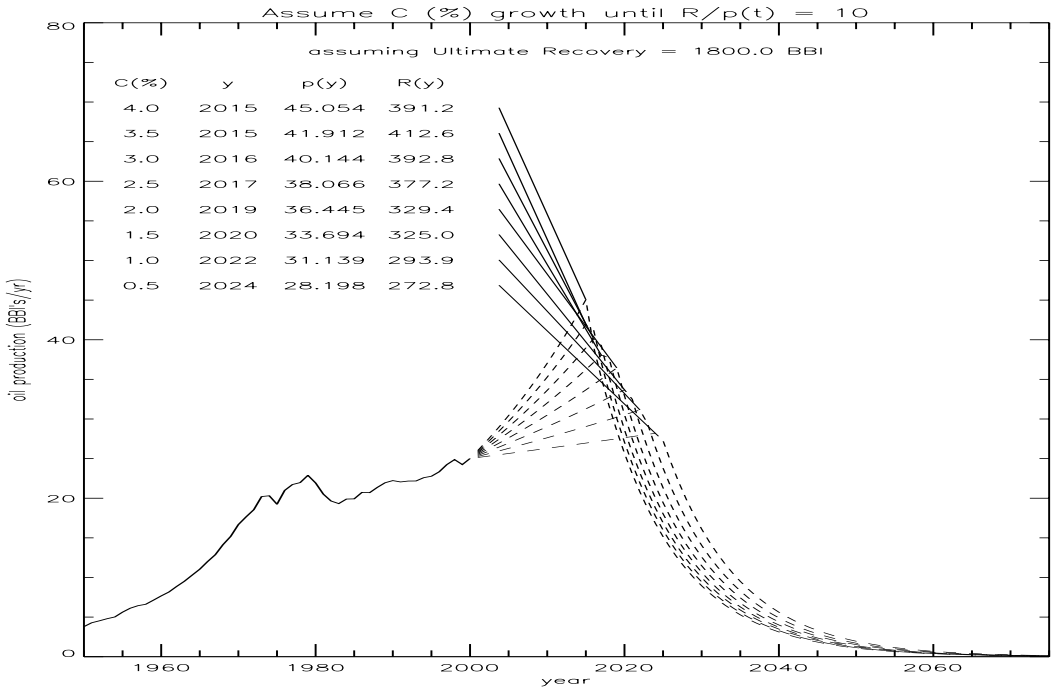


Figure 5.14: Wood and Long model assuming ultimate recovery of 1800 billion barrels

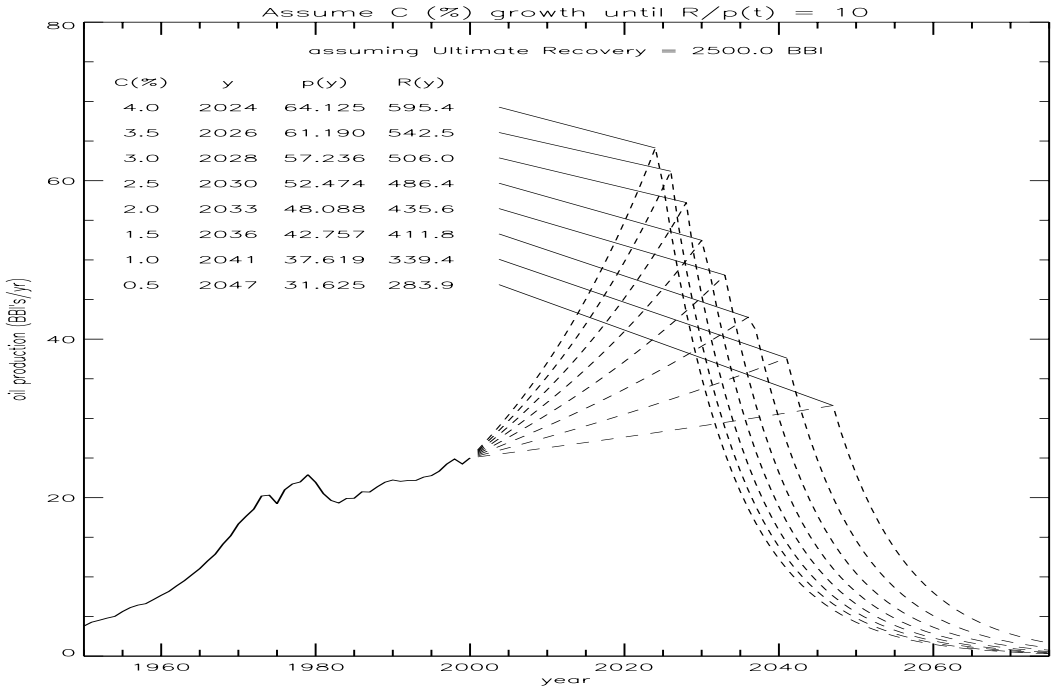


Figure 5.15: Wood and Long model assuming ultimate recovery of 2500 billion barrels

Chapter 6

Monte Carlo Methods

Statistical or stochastic methods are coined Monte Carlo methods after the sovergn province in Monacco. These method are employed when

- a physical model does not exist to sufficient accurary,
- a physical model is too complex or requires too much computational resource,
- Additional statistical information can be used to constrain a problem when the measurements do not contain enough information to formally solve the problem.

6.1 Computing Random Numbers with Arbitrary distributions

Monte Carlo problems usual depend on computing random sequences with specific distributions, $P_x(x) \cdot dx$, which is numerically constrained over the range $a \leq x < b$. In Section 2.4 we discussed uniform deviates and how to compute a Gaussian distribution. A simple way to compute a random number with an arbitrary distribution is the Monte-Carlo method of *Acceptance-Rejection*. We first select a value P_x^{max} which satisfies the condition

$$P_x^{max} \geq P_x(x) \quad \text{for all } x \tag{6.1}$$

1. Pick a trial value of x using a uniform deviate random number, r_1, (*e.g.*, see section B.15)

$$x_{try} = a + (b - a) \cdot r_1 \tag{6.2}$$

2. Then compute $P_x(x_{try})$.
3. Using a second random value, r_2, if

$$P_x(x_{try}) \geq r_2 \cdot P_x^{max} \tag{6.3}$$

   is true then accept x_{try}, otherwise, goto step #1.

This method relies on the fact that we are more likely to accept a try when $P_x(x)$ is large. We random select pairs of points, (x, y), within the rectangle bounded by $a \leq x < b$ and $0 \leq y < P_x^{max}$. If it falls within $P_x(x)$ we accept it, otherwise we reject it. This method is simple to implement and will produce good results if the uniform deviate has a good random distribution. Another advantage is this method will work with a discrete distribution.

6.1.1 Example: Monte Carlo accept/reject method for non-uniform random deviates

For example, suppose we want to compute the probability distribution $P(x) \cdot dx = \exp(x)$. The code would look like

```
100 x = 3.0*RAN(seed)   ! compute over the range of 0 <= x <= 3
    ytest = EXP(-x)
    if(ytest.le.RAN(seed)) goto 100
```

and a histogram of the distributuion of x would look like

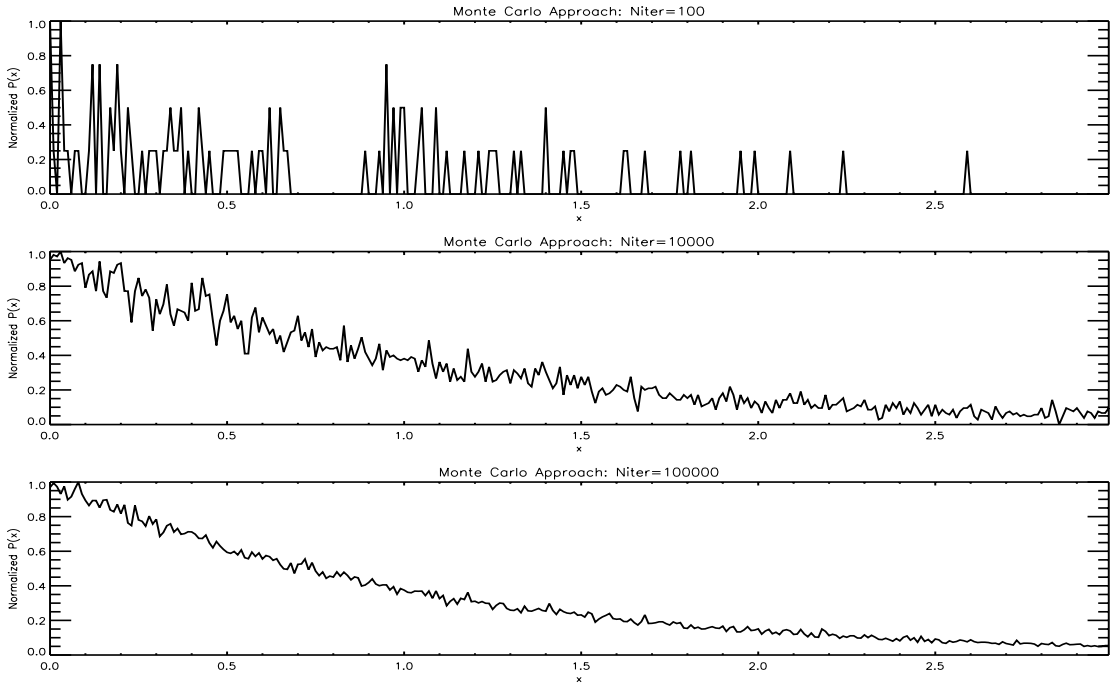


Figure 6.1: Illustration of Monte Carlo "accept/reject method to product a desired probability density function, compare to Fig. 2.2

Compare this to the result in Section 2.4.1 in which we used the exponential of a uniform deviate to compute this distribution.

6.2 Evaluation of Definite Integrals

Imagine a pond in the middle of a field of known area, A. If we throw rocks *randomly* into the field and hear a splash then we know that we hit the pond, otherwise we assume the rock landed on land. If we count the total "spashes", N_s, relative to the total number of throws, N_t, it should be close to the area of the lake, A_l.

$$A_l = A \cdot \frac{N_s}{N_t} \tag{6.4}$$

To evaluate an integral we will use this process. The "field" is given by the rectangle defined by the finite interval $a \le x \le b$ and a finite range such that $-H \le y(x) \le H$ for all values of x. We can compute random values of x_i, y_i and if $y_i \le y(x_i)$ then we count that as a "splash". Therefore, the integral is equal to

$$\int_a^b f(x)dx \approx 2H \cdot (b-a) \cdot \frac{N_s}{N_t} \tag{6.5}$$

However, since we need to compute $y(x_i)$ in order to evaluate whether or not a splash occured we can use a uniform deviate for x_i to compute the integral with the mean value theorem.

$$\int_a^b f(x)dx \approx \frac{(b-a)}{N_t} \sum_{i=1}^{N_t} f(x_i) \tag{6.6}$$

This can be compared to the trapezoidal quadrature where the values of x_i are chosed to be uniformly sampled, $x_i = a + (b-a) * i/N_t$. For the same number of points, Eqn. 6.6 will work better than Eqn. 5.10 if the function $y(x)$ is of multiple dimensions. That is, complex multiple integrals will be evaluated more quickly with this method.

See Gould and Tobochnik, 1996, pg.351 for more details on computing integrals with random sequences.

6.3 Random Walk Processes: Simulation of complex systems

One of the first applications of stochastic methods was the kinetic theory of dilute gases. A dilute gas of N particles in a volume V has the property that the diameter of the particles, d, is much less than the mean free path between particles. Note that this technique can be used to simulate small objects (*e.g.*, molecules) or large objects (*e.g.*, galaxies) if the following condition is met.

$$d \ll (V/N)^{\frac{1}{3}} \tag{6.7}$$

In molecular dynamics problems a dilute gas at standard temperature (273.15) and pressure (1013.25 mb) has $2.687 \cdot 10^{19}$ particles per cubic centimeter (Loschmidt's number). A computer simulation solving the dynamics equation

$$\frac{d^2}{dt^2}\vec{r_i} = \frac{1}{m} \sum_{j \neq i} F_{i,j} \tag{6.8}$$

however, even for a machine with 1 GB of memory we would be hard pressed to compute a reasonable ensemble. 10^9 particles is still only 37 cubic microns of gas. Even on a super-computer evolving the system a single step (on the order of femto-seconds) would take $\approx 10^{12}$ MEGA-FLOPS.

Statistical mechanics computes probabilities for ensembles of particles. The velocity distribution of these particles can be determined for the canonical ensemble, where the particals are in thermodynamic equilibrium and are ideal (all the energy is kinetic). This velocity distribution is known as the Maxwell-Boltzmann distribution and is given by

$$P_v(v) = 4\pi \cdot \left(\frac{m}{2\pi kT}\right)^{\frac{3}{2}} \cdot v^2 \cdot \exp\left(-\frac{mv^2}{2kT}\right) \tag{6.9}$$

which is the probability that a particles velocity is between v and $v+dv$. A simulation of complex systems can be performed by keeping track of the state vector, $\vec{r_i}$, v_i, for N particles but applying acceptance/rejection tests in a random way (*e.g.*, see Garcia 2000, pg. 340-350). Collisions are modelled simply using random numbers and statistical probabilities to replace molecular dynamic computations. For example, the collision probability for a hard-sphere model is given by

$$P_{coll}(i,j) = \frac{|v_i - v_j|}{\sum_{m=1}^{N} \sum_{n=1}^{m} |v_m - v_n|} \tag{6.10}$$

but is computed as follows in a Monte-Carlo scheme

1. pick a pair, i, j at random.
2. compute their relative speed, $v_r = |v_i - v_j|$.
3. accept a collision if $v_r > v_r^{max} \cdot r$, where r is a uniform deviate, $0 \leq r < 1$.
4. if the collison is accepted compute the new velocities (constrained by molecular statistics).

In this way, complex systems, much too complex for traditional statistical methods, can be simulated and studied. See Gould and Tobochink, 1996; Garcia, 2000; and Frenkel and Smit, 2001 for more examples.

6.3.1 Example: using Monte-Carlo fit X- and γ-ray spectrum

In Sobol 1994, an example is given on pg. 53 of using Monte-Carlo techniques to model a spectrum of hard X-rays from the nucleus of a Seyfert galaxy, NGC-4151. The flux is proportional to the probability density of photons escaping from a cloud. A low-frequency photon source is assumed to have a black-body spectrum with radiation temperature, T_r. The hot electrons in the cloud are Maxwellian with temperature, T_e and the cloud has a radius of τ. The hot-electrons increase the photon's energy in multiple Compton scattering events. Monte-Carlo techniques were used to compute the probability distribution of photons exiting the cloud and determine the physical conditions ($kT_r = 0.5$ eV, $kT_e = 2$ MeV) that best fit the observations. See Pozdnyakov *et al.*, 1983.

6.4 Random Walk Processes: Scattering Application

In light or particle scattering processing the scattering event can be described in terms of a single scattering albedo, ω_0, a scattering phase function, $\Psi(\theta)$, and a distribution of scatterers, $n(r)$. A random walk process can be made through a specified media where the encounters result in random operations. For example, the phase function with scatter the particle in a random direction with a random distributuion function $P_x(x) = \Psi(\theta)$. Other parameters can be functions of random distributions to make the atmosphere more realistic. The ineraction of radiation with a complex media can be studied as a function of the media's properties.

The angle between the incident flux and the local normal is given by α and $\mu_0 \equiv \cos(\alpha)$. The angle between the local normal and the observer is given by ϵ and $\mu \equiv \cos(\epsilon)$. The phase angle, θ, is the angle between the incident and emission through the origin. The azimuthal angle, $\Delta\psi$, is the projection of the solar incidence and emission directions onto the local horizon. Using spherical trigonometry cosine laws (e.g., see CRC pg. 146) we can easily obtain the value of $\Delta\psi$) given μ_0, μ, and θ. For solar scattering to an observer above the atmosphere the angles are related by:

$$\cos(\theta) = \sqrt{((1-\mu_0^2)(1-\mu^2))} \cdot \cos(\Delta\psi) \mp \mu\mu_0 \quad (\mathrm{S} = +,\ \mathrm{T} = -) \tag{6.11}$$

This is the method employed by Chardrasekar, Liou. and others. We could also write the equations in terms of the scattering angles. The scattering angle between incidence and emission, Θ, which is related to the phase function, is given by $\Theta = \pi - \theta$. The scattering angle in the horizontal plane, $\Delta\phi$ is related to the azimuthal angle, and is given by $\Delta\phi = \pi - \Delta\psi$. This definition is used by Hansen, Tomasko, and Danielson. It is the method employed within the VIAMP programs. Note the sign change between the two methods.

$$\cos(\Theta) = \sqrt{((1-\mu_0^2)(1-\mu^2))} \cdot \cos(\Delta\phi) \pm \mu\mu_0 \quad (\mathrm{T} = +,\ \mathrm{S} = -) \tag{6.12}$$

The Rayleigh phase function is given in terms of scattering angles or phase angles by

$$P_{ray}(\Theta) = \frac{3}{4}(1 + \cos^2(\Theta)) = \frac{3}{4}(1 + \cos^2(\theta)) = P_{ray}(\theta) \tag{6.13}$$

The $\cos^2$ term can be expressed in terms of μ, μ_0, and $\Delta\phi$ as

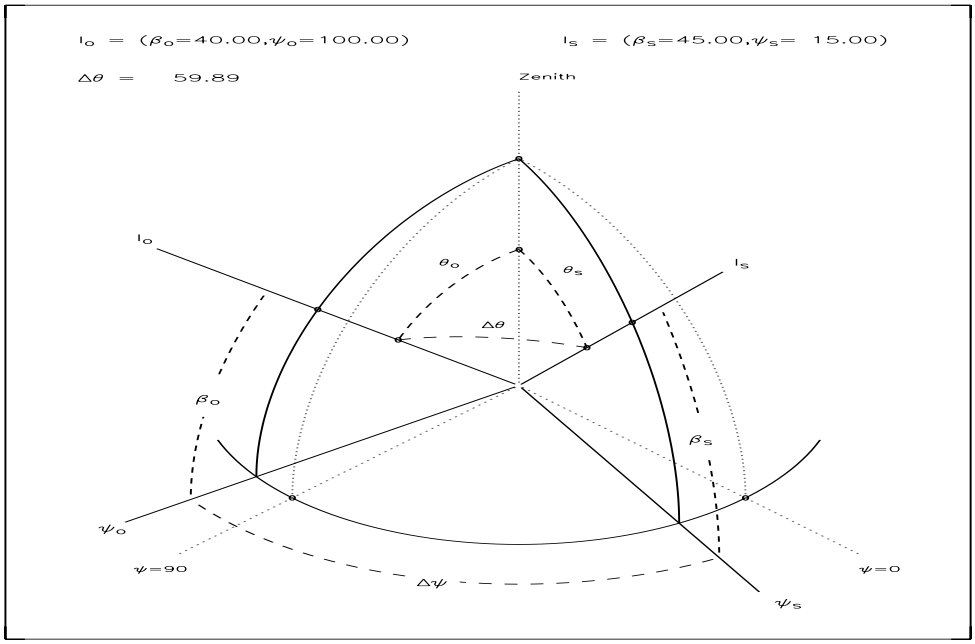


Figure 6.2: Definition of angles for Monte Carlo simulation of scattering

$$\cos^2(\Theta) = ((1-\mu_0^2)(1-\mu^2)) \cdot \cos^2(\Delta\phi) \pm 2\mu\mu_0\sqrt{((1-\mu_0^2)(1-\mu^2))} \cdot \cos(\Delta\phi) + \mu^2\mu_0^2 \tag{6.14}$$

Note that $\cos(2\Delta\phi) = 2\cos^2(\phi) - 1$ or, $\cos^2(\phi) = \frac{1}{2}\cos(2\Delta\phi) + \frac{1}{2}$

$$\cos^2(\Theta) = \frac{1}{2}((1-\mu_0^2)(1-\mu^2)) \cdot \cos(2\Delta\phi) \pm 2\mu\mu_0\sqrt{((1-\mu_0^2)(1-\mu^2))} \cdot \cos(\Delta\phi) + \mu^2\mu_0^2 + \frac{1}{2}((1-\mu_0^2)(1-\mu^2)) \tag{6.15}$$

So the Rayleigh phase function can be written in terms of 3 Fourier moments: $P_{ray}(\Theta) = P_1 + P_2\cos(\Delta\phi) + P_3\cos(2\Delta\phi)$
where the 3 coefficients are equal to

$P_0 = \frac{3}{4}(1 + \mu^2\mu_0^2 + \frac{1}{2}((1-\mu_0^2)(1-\mu^2))) = \frac{3}{8}(3 + 3\mu^2\mu_0^2 - \mu_0^2 - \mu^2)$
$P_1 = \pm\frac{3}{2}\mu\mu_0\sqrt{((1-\mu_0^2)(1-\mu^2))}$ (T = +, S = -)
$P_2 = \frac{3}{8}((1-\mu_0^2)(1-\mu^2))$

6.5 Simulated Annealing methods

Simulated annealing has many applications in optimization problems and might be more properly treated in non-linear methods discussed in section 14.2.

6.5.1 Example of an optimization problem

A interesting example of simulated annealing is given in *Numerical Recipes, pg. 328* (Press *et al.* 1986). It involves the optimization of a traveling salesman's itinerary given constraints, such as fear of crossing a river, etc. The saleperson visits N cities at locations x_i, y_i, returning to the city of origin. The itinerary can be

reorganized by (a) replacing a segment of the path with its reverse order or (b) a segment of the path is replaced by another random segment. This total distance is computed as

$$L = \sum_{i=1}^{N} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \tag{6.16}$$

with the point $(x_{N+1}, y_{N+1}) \equiv (x_1, y_1)$. The method is quite flexible to adding other constraints. For example, if the salesman has an irrational fear of crossing a river, then the total distance can be computed with an additional parameter, u_i, equal to +1 on one side of the river and -1 on the other side. A parameter λ can control the effect of crossing the river.

$$L = \sum_{i=1}^{N} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} \quad + \quad \lambda \cdot (u_i - u_{i-1})^2 \tag{6.17}$$

and a penalty of 4λ is assigned to a river crossing. Figures 10.9.1 in *Numerical Recipes* illustrate the solution for the itinerary for various values of λ. For $\lambda = 0$ the shortest path is found; however there are 4 river crossings in the example. For λ equal to a large positive value the path is slightly longer, but there are only 2 river crossings. For λ equal to a large negative values the simulation makes the itinerary cross the river many time, simulating a saleperson who crosses the river on the flimsiest excuse (*e.g.*, a smuggler).

6.5.2 Example of an inversion problem

Here we will consider using simulated annealing to solve equations that are too complex for traditional non-linear techniques. The following discussion is taken from Rayner, P.J., I.G. Enting, and C.M. Trudinger 1996. "Optimizing the CO_2 observing network for constraining sources and sinks." *Tellus* **48B**, p.33-444. These authors used simulated annealing techniques to solve for CO_2 sources (*e.g.*, fossil fuel emissions, biomass burning, soil respiration) and sinks (*e.g.*, ocean and land photosynthesis) using weekly CO_2 measurements made at the surface from the global network of 56 stations. This is a highly complex process that is under-sampled.

Simulated annealing is a technique for optimizating highly structured non-linear functionals of many variables. It is an analogy with annealing of a crystal lattice in which quasi-steady state conditions while slowly cooling allow the attainment of a minimum energy state in contrast to more rapid methods that are analygous to quenching. The general procedure for minimizing $f(x)$ proceeds as follows:

- At any step i we have a state x_i.
- We perturb this by $\delta x(T)$ proportional to a pseudo-temperature, T.
- If $f(x_i + \delta x(T)) < f(x_i)$ then we accept $x_i + \delta x(T)$ as x_{i+1}. If $f(x_i + \delta x(T)) \geq f(x_i)$, then we may still accept the new state based on a Boltzmann probability distribution. We calculate the probability of the new state as

  $$P(x_i + \delta x(T)) = \exp\left(-\frac{f(x_i + \delta x(T)) - f(x_i)}{k \cdot T}\right) \tag{6.18}$$

  where T is a pseudo-temperature and k is the scaling factor between temperature and energy, We accept the new state if P is greater than a pseudo-random number between 0 and 1.
- We carry our a series of trials for a given temperature and then reduce the temperature. Reducing the temperature
  - reduces the size of the perturbations, and
  - makes the acceptance of $f(x_i + \delta x(T))$ less likely.

The hope is that by the time the simulated annealing is complete we are in the global minimum and we will gradually settle towards the bottom. There are three parameters one can tune for the simulated annealing algorithm. They are:

1. Cooling rate, ΔT: this is a computation trade-off. The slower the cooling rate the less likely we are to freeze the system in a local, rather than global, minimum.

2. Perturbation size, $\delta x(T)$: This is selected to vary over a reasonable parameter space.

3. The Boltzmann constant, k: This parameter related the pseudo- temperature with the residuals.

Chapter 7

Differentiation

We will be discussing differentiation of a tabulated set of data and of image data. This is an introduction to building and solving differential equations, which will be covered in more detail later.

First, lets consider differentiation of a vector of data. For example, we will consider the following 3 points within a larger sample of data.

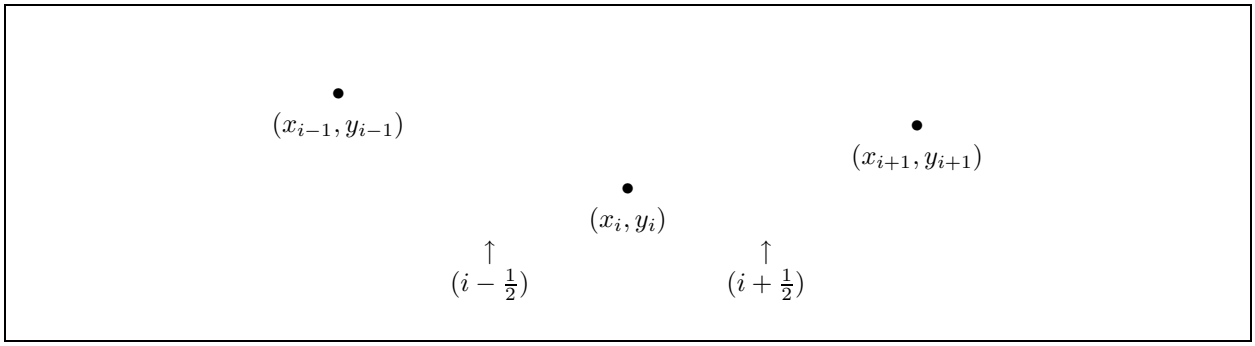


Figure 7.1: Illustration of finite differences in 2-dimensions

Numerically we can compute a derivative by *finite differencing* as follows

$$\left.\frac{dy(x)}{dx}\right|_{i+\frac{1}{2}} \simeq \frac{y(i+1) - y(i)}{x(i+1) - x(i)} \tag{7.1}$$

and if x is on a equally spaced grid, then

$$\left.\frac{dy(x)}{dx}\right|_{i+\frac{1}{2}} \simeq \frac{y(i+1) - y(i)}{\Delta x} \tag{7.2}$$

or we can compute a *centered* derivative as follows

$$\left.\frac{dy(x)}{dx}\right|_{i} \simeq \frac{y(i+1) - y(i-1)}{2 \cdot \Delta x} \tag{7.3}$$

A second derivative on a uniform gird can be computed by *finite differencing* by taking the derivative of the first derivatives on each side of $(x(i), y(i))$.

$$\left.\frac{d^2y(x)}{dx^2}\right|_i \simeq \frac{\left.\frac{dy(x)}{dx}\right|_{i+\frac{1}{2}} - \left.\frac{dy(x)}{dx}\right|_{i-\frac{1}{2}}}{\Delta x} = \frac{y(i+1) - 2\cdot y(i) + y(i-1)}{(\Delta x)^2} \tag{7.4}$$

which can be written as a matrix operator as follows:

$$\left.\frac{d^2y(x)}{dx^2}\right|_i = \frac{1}{(\Delta x)^2} \cdot \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} y(1) \\ y(2) \\ y(3) \\ y(4) \\ \dots \\ y(N-3) \\ y(N-2) \\ y(N-1) \\ y(N) \end{pmatrix} \tag{7.5}$$

Derivative of time-varying 2-dimensional spatial derivatives on an evenly spaced grid

Now consider a function 3-dimensional function, $\Psi(x, y, t)$. At each time slice the spatial function looks like

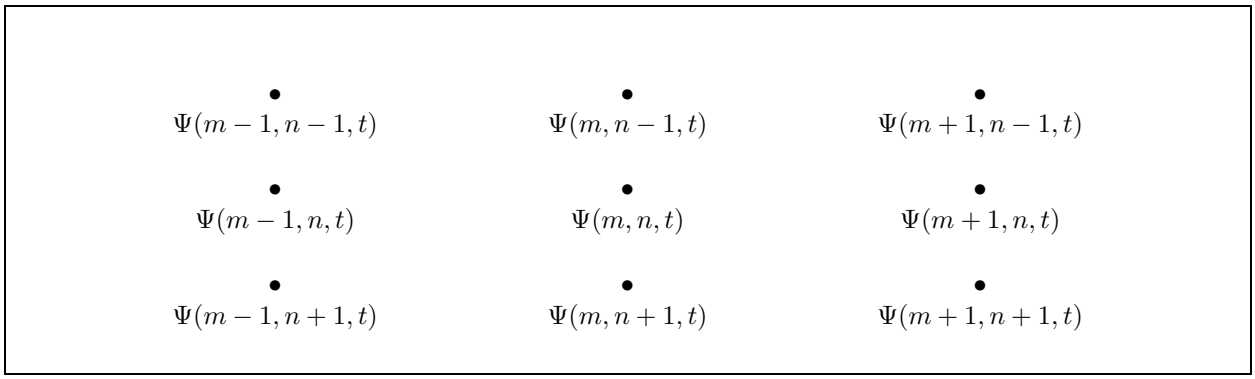


Figure 7.2: Illustration of finite differences in 2-dimensions

7.1 Finite Differencing of 2-d data

7.1.1 Centered First Derivative

$$\left(\frac{\partial\Psi}{\partial x}\right)_{(m,n,t)} \approx \frac{\Psi(m+1,n,t) - \Psi(m-1,n,t)}{2\cdot\Delta x} \tag{7.6}$$

$$\left(\frac{\partial\Psi}{\partial y}\right)_{(m,n,t)} \approx \frac{\Psi(m,n+1,t) - \Psi(m,n-1,t)}{2\cdot\Delta y} \tag{7.7}$$

7.1.2 Finite Second Derivatives

If we take the difference between first derivatives, evaluated at half grid points and half grid spacing then we will have calculated the second derivative on the grid boundaries.

$$\left(\frac{\partial \Psi}{\partial x}\right)_{(m,n,t)} \approx \frac{\Psi(m+1/2,n,t) - \Psi(m-1/2,n,t)}{\Delta x} \tag{7.8}$$

$$\left(\frac{\partial^2 \Psi}{\partial x^2}\right)_{(m,n,t)} \approx \frac{\left(\frac{\partial \Psi}{\partial x}\right)_{(m+1/2,n,t)} - \left(\frac{\partial \Psi}{\partial x}\right)_{(m-1/2,n,t)}}{\Delta x} \tag{7.9}$$

$$\left(\frac{\partial^2 \Psi}{\partial x^2}\right)_{(m,n,t)} \approx \frac{\Psi(m+1,n,t) - 2\cdot\Psi(m,n,t) + \Psi(m-1,n,t)}{(\Delta x)^2} \tag{7.10}$$

Similarly, for the second derivative w.r.t. y

$$\left(\frac{\partial \Psi}{\partial y}\right)_{(m,n,t)} \approx \frac{\Psi(m,n+1/2,t) - \Psi(m,n-1/2,t)}{\Delta y} \tag{7.11}$$

$$\left(\frac{\partial^2 \Psi}{\partial y^2}\right)_{(m,n,t)} \approx \frac{\Psi(m,n+1,t) - 2\cdot\Psi(m,n,t) + \Psi(m,n-1,t)}{(\Delta y)^2} \tag{7.12}$$

7.1.3 The Laplacian

$$\nabla \Psi = \hat{m}\frac{\partial \Psi}{\partial x} + \hat{n}\frac{\partial \Psi}{\partial y} \tag{7.13}$$

$$\nabla^2 \Psi = \left(\frac{\partial^2 \Psi}{\partial x^2}\right)_{(m,n,t)} + \left(\frac{\partial^2 \Psi}{\partial y^2}\right)_{(m,n,t)} \tag{7.14}$$

If $d \equiv \Delta x = \Delta y$ then

$$\nabla^2 \Psi \approx \frac{\Psi(m,n+1,t) + \Psi(m,n-1,t) + \Psi(m+1,n,t) + \Psi(m-1,n,t) - 4\cdot\Psi(m,n,t)}{d^2} \tag{7.15}$$

7.1.4 Convolution as an operator

In general, the first and second derivatives can be written as a convolution of a operator matrix and the data as follows:

$$op(\Psi(m,n,t)) = \sum_{i=1}^{3}\sum_{j=1}^{3} \Psi(m+i-2, n+j-2, t)\cdot op(i,j) \tag{7.16}$$

For example, the Laplacian operator is given by

$$op(i,j) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{7.17}$$

7.1.5 Example: Representation of Barotropic Vorticity Equation

In atmospheric dynamics ($\vec{F} = m \cdot \vec{a}$, 2^{nd} law of thermodynamics, and continuity equation), a system of equations can be written where the motions are constrained horizontally.

$$\frac{\partial}{\partial t}\nabla^2\Psi - \vec{V}_\Phi \cdot \nabla\left(\nabla^2\Psi + f\right) \tag{7.18}$$

where,

$$\vec{V}_\Phi \equiv \hat{m}\frac{-\partial\Psi}{\partial y} + \hat{n}\frac{\partial\Psi}{\partial x} \tag{7.19}$$

and $f = 2\Omega\cos(\theta)$, θ = latitude, and $\Omega = 2\pi/24$ hours. Remember that, ∇^2 is the Laplacian and can be written in terms of Eqn. 7.15

$$\begin{aligned}\frac{\partial\Psi}{\partial y}\left(\frac{-\partial\Psi}{\partial x}\nabla^2\Psi\right) &= \frac{1}{2d}\left[\frac{\Psi_{m+1,n+1,t} - \Psi_{m-1,n+1,t}}{2d}\nabla^2\Psi_{m,n+1,t}\right] \\ &+ \frac{1}{2d}\left[\frac{\Psi_{m+1,n-1,t} - \Psi_{m-1,n-1,t}}{2d}\nabla^2\Psi_{m,n-1,t}\right]\end{aligned} \tag{7.20}$$

etc. We can then integrate the equation with respect to t.

Chapter 8

Digital Image Processing

Image processing techniques are useful for a variety of applications including visualization of models and measurements.

For this topic I will show some examples on a test image. This is an image were $m = 600$ and $n = 400$ and is a Hubble Space Telescope Wide-Field/Planetary Camera image of Saturn during a rare storm in the equatorial region.

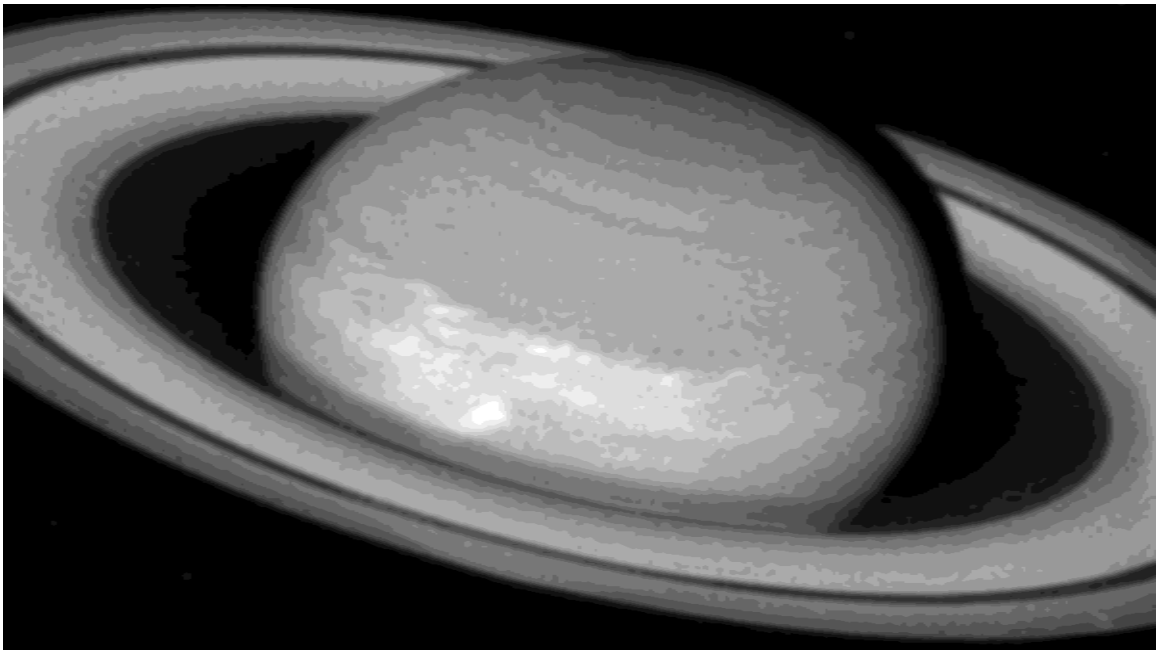


Figure 8.1: Original image of Saturn to be used to illustrate image processing tools

8.1 Image Enhancement

For image sharpening we can subtract the local gradient from the image value

$$DN' = DN - \nabla^2 DN \tag{8.1}$$

or

$$op(i,j) = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} \tag{8.2}$$

$$DN'(i_0,j_0) = \sum_{i=-N}^{N} \left(\sum_{j=-N}^{N} [DN(i+i_0, j+j_0) \cdot op(i+N, j+N)] \right) \qquad N = 1 \tag{8.3}$$

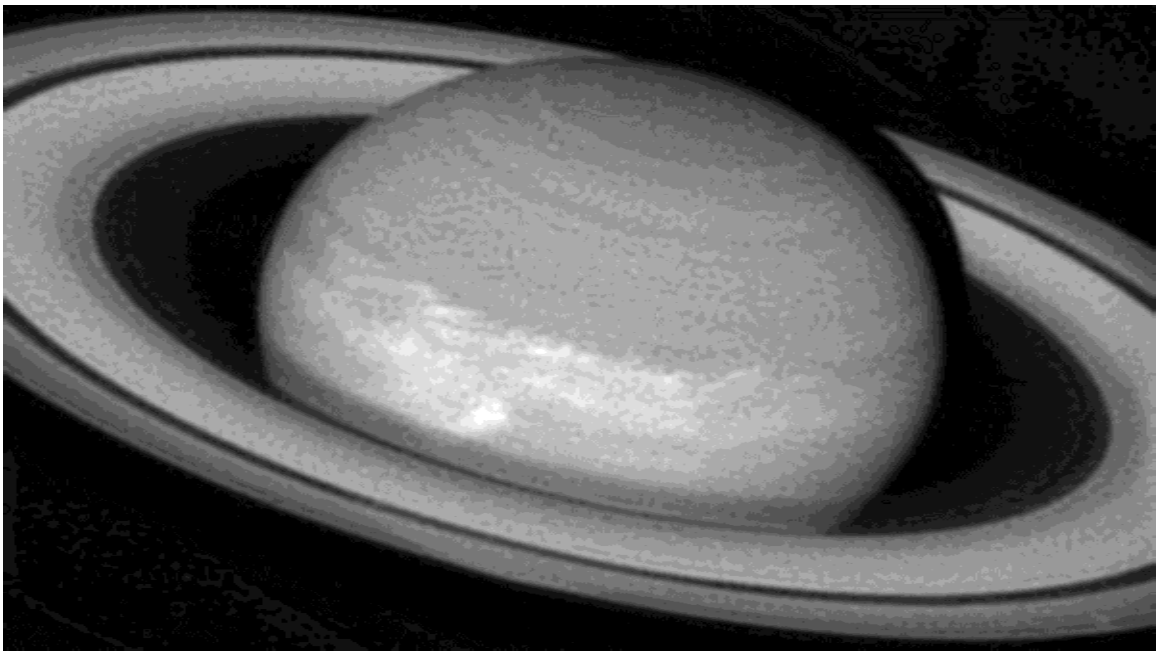


Figure 8.2: Derivative of Test Image

First, lets define an average value in the local neighborhood of an image. To do this we should weight the points relative to the center of the average. Many methods of weighting exist. One example, the average $\overline{DN}$ is calculated using a normalized weighting by $\cos(2d/(2N+1))$, where $2N+1$ is the size of the averaging box (usually $N >> 1$) and $d = \sqrt{(\Delta i)^2 + (\Delta j)^2}$. For $N = 1$ the averaging box looks like

$$weight(i,j) = \frac{1}{6.494} \cdot \begin{pmatrix} 0.588 & 0.786 & 0.588 \\ 0.786 & 1.000 & 0.786 \\ 0.588 & 0.786 & 0.588 \end{pmatrix} \tag{8.4}$$

$$\overline{DN}(i_0,j_0) = \sum_{i=-N}^{N} \left(\sum_{j=-N}^{N} [DN(i+i_0, j+j_0) \cdot weight(i+N+1, j+N+1)] \right) \tag{8.5}$$

8.2 Digital Filters

a. Low Pass

$$g_{lp}(DN) = \overline{DN} \tag{8.6}$$

For large N this function "blurs" the image and removes high frequency noise. It also softens edges.

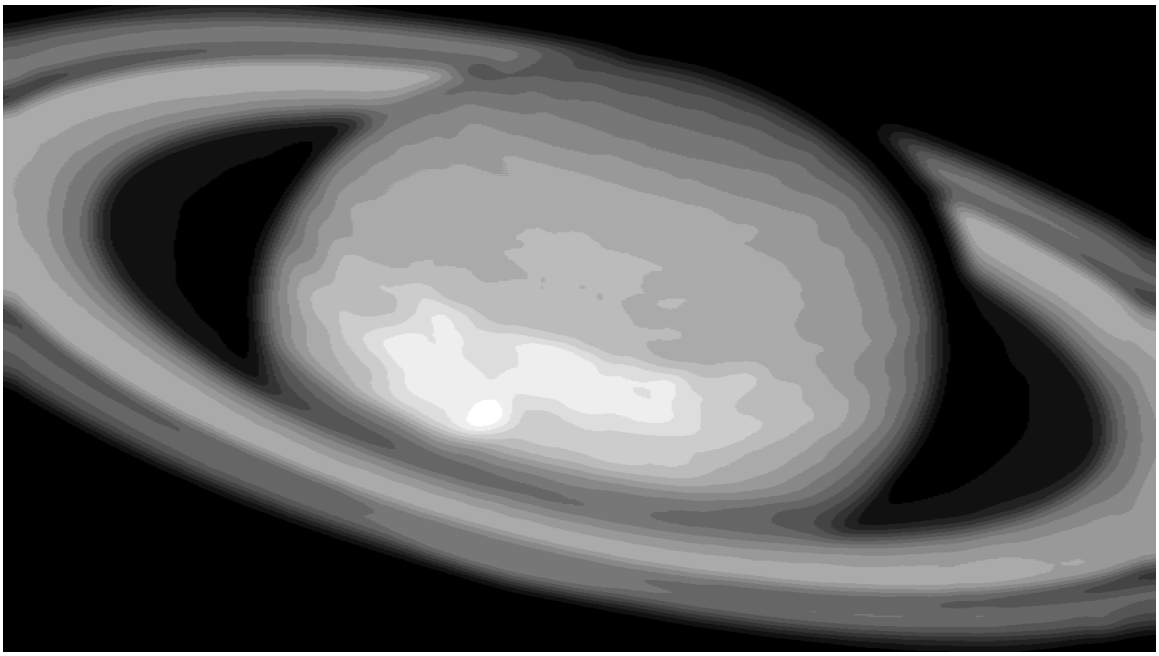


Figure 8.3: Low Pass Filter with N=7, $\Rightarrow$ $weight(15, 15)$

b. High Pass

$$g_{hp}(DN) = (DN - \overline{DN}) \tag{8.7}$$

In general, the original image is added to a fraction of the filtered image to produce a reasonable result.

$$DN' = a_0 + a_1 \left(f \cdot DN + (1 - f) \cdot g_{hp}(DN)\right) \tag{8.8}$$

For the Pic-du-midi "unsharp mask" filter set $a_0 = 0$, $a_1 = 3$, and $f = 1/3$. This is equivalent to adding two times the difference between the image and the local average to the original image. This has the effect of sharpening the features.

c. Scene Filter

$$g_s(DN) = \frac{(DN - \overline{DN})}{\overline{DN}} \tag{8.9}$$

This is a dramatic filtering technique. Usually f is very small and a_1 is very large, such that $a_1 \cdot f \approx 1$. This is a normalized high pass filter which accentuates small (*i.e,* on the scale of N) features in dark regions or near the limb (in typical visual images).

d. Divide Filter

$$g_d(DN) = \frac{DN}{\overline{DN}} \tag{8.10}$$

I've never figured out a use for this filter, but it is part of the standard Voyager processing set.

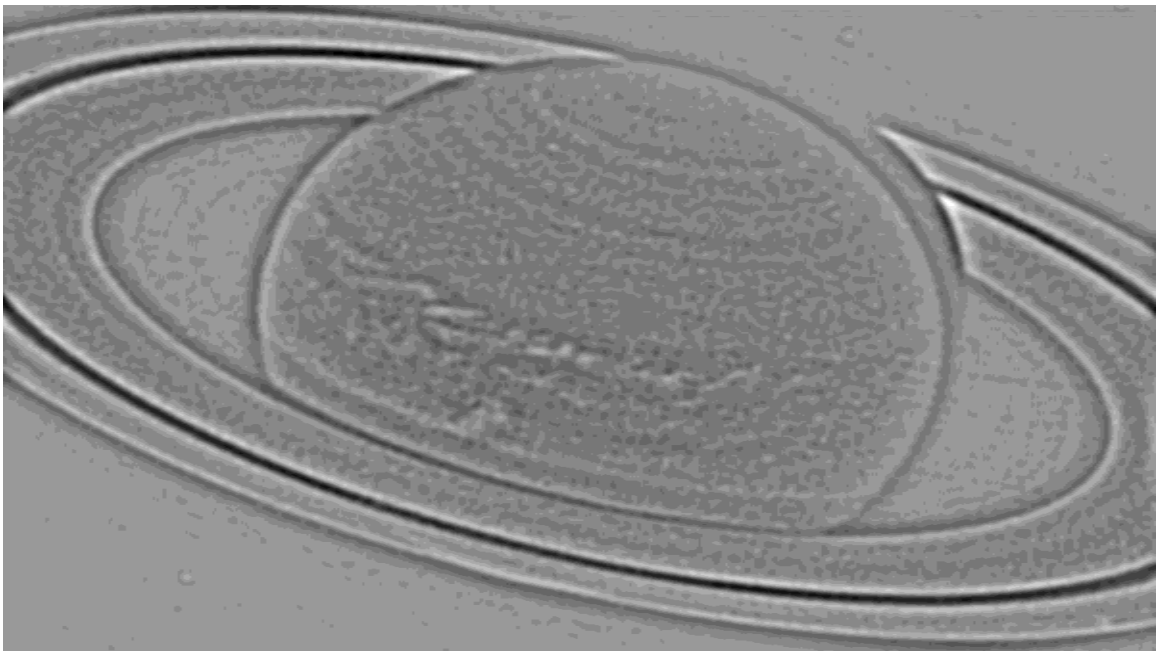


Figure 8.4: High Pass Filter

8.3 Edge Detection

Edge detection (*i.e.*, Activity Filter) uses local derivatives to identify the edges of features. This is necessary for many automated processes.

$$\begin{aligned} DN'(i,j) &= |DN(i-1,j-1) - DN(i+1,j+1)| \\ &+ |DN(i+1,j-1) - DN(i-1,j+1)| \end{aligned} \tag{8.11}$$

Figure 8.5: Pic-du-midi function: $DN' = 3\left(f \cdot DN + (1-f) \cdot g_{hp}(DN)\right), \quad f = 1/5$

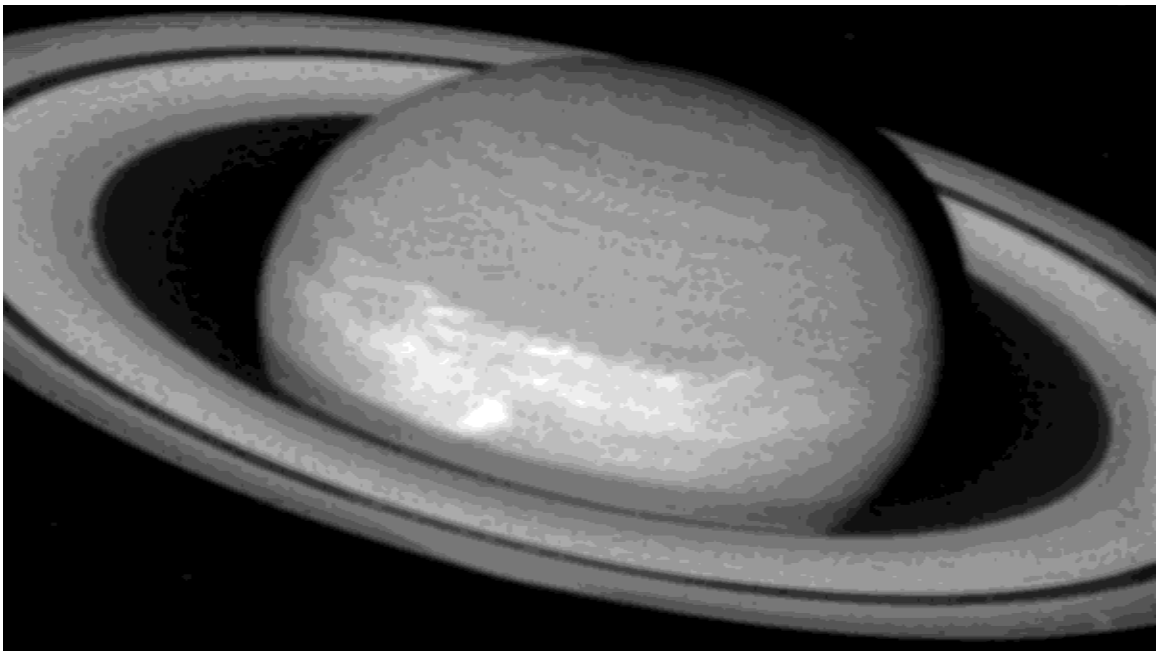


Figure 8.6: Scene Filter: $DN' = 10 * (f \cdot DN + (1-f) \cdot g_s(DN)), \quad f = 1/10$

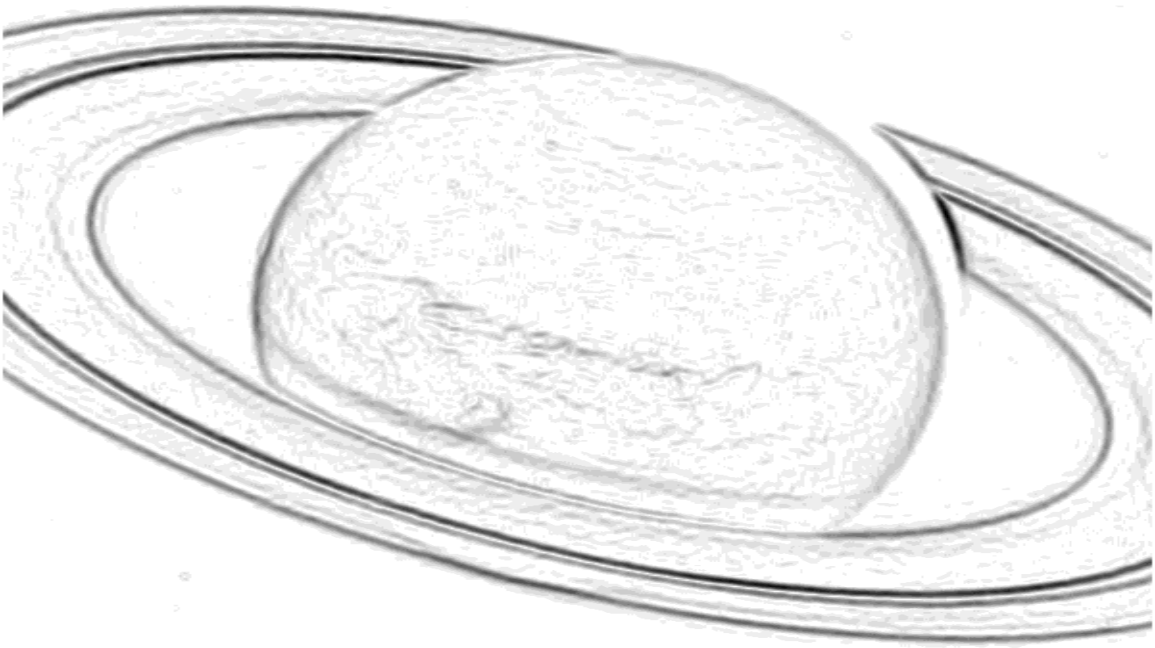


Figure 8.7: Edge Detection Filter

8.4 Image to planet coordinate transformations for mapping

8.4.1 Definition of Planet, Spacecraft, and Camera Coordinate Systems

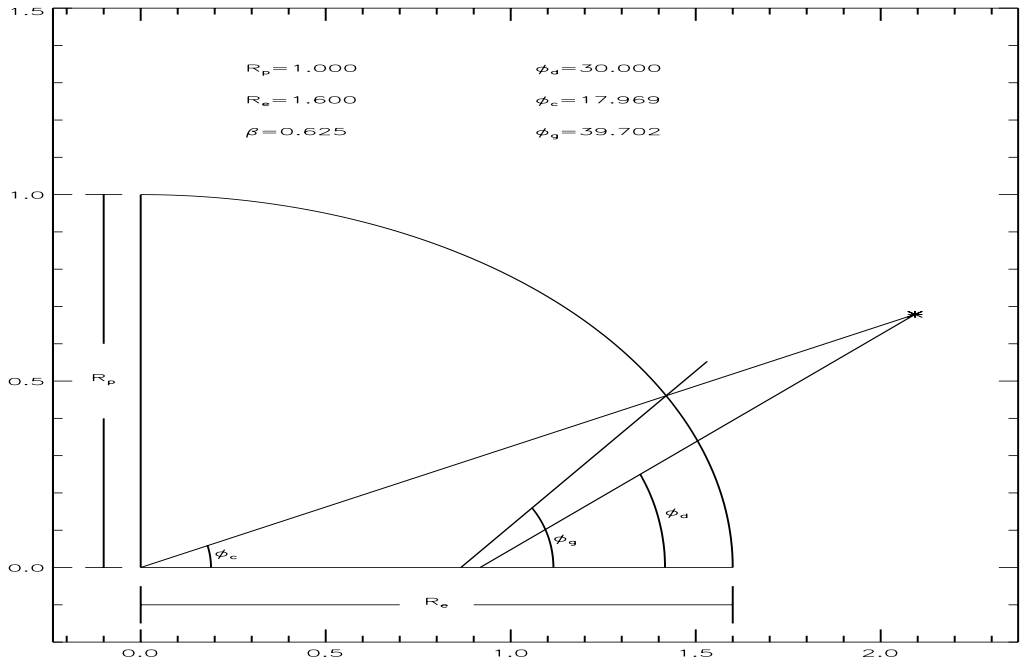


Figure 8.8: Illustration of planeto-centric, planeto-graphic, and planeto-detic latitude

The spacecraft and solar positions can be described in terms of latitude and longitude on the planet surface model, typically given by an oblate spheroid. The latitude system can be planeto-centric, φ_c, planeto-graphic (also known as planeto-detic), φ_g, and planeto-detic, φ_d

Planet Coordinate Systems

φ_c: planetocentric latitude, measured from equatorial plane

λ_w: west longitude, *i.e.* prime meridian increases with time

R_e: Equatorial radius of planet

R_p: Polar radius of planet

β: Polar to Equatorial Ratio,

$$\beta \equiv R_p/R_e \tag{8.12}$$

f: Flattening

$$f = \frac{R_{eq} - R_{pl}}{R_{eq}} \tag{8.13}$$

e: Eccentricity

$$e = \frac{\sqrt{R_{eq}^2 - R_{pl}^2}}{R_{eq}} = \sqrt{1 - \left(\frac{R_{pl}}{R_{eq}}\right)^2} \tag{8.14}$$

$$S = \text{total area} = 2\pi \cdot R_{eq}^2 + \pi \frac{R_{pl}^2}{e} \cdot \log_e \left(\frac{1+e}{1-e}\right) \tag{8.15}$$

For $e = 0$ this equation reduces to the volume of a sphere.

$$S = 4\pi \cdot R_{eq} \cdot R_{eq} \tag{8.16}$$

$$V = \text{total volume} = \frac{4}{3}\pi R_{eq}^2 \cdot R_{pl}^2 \tag{8.17}$$

CRC Standard Mathematical Tables 26^{th} edition, pg. 129. The volume of an oblate sphere is equal to a sphere with radius $\overline{R}$.

$$\overline{R} = \left(R_{eq}^2 \cdot R_{pl}\right)^{\frac{1}{3}} \tag{8.18}$$

Table 8.1: Oblate models for the planets

	R_{eq} Km	R_{pl} Km	R_{eq}/R_{pl}	f	e	$< R >$ Km
Venus	6051.0	6051.0	1.000000	0.000000	0.000000	6051.00
Earth	6378.5	6356.0	1.003540	0.003527	0.083920	6370.99
Mars	3393.0	3375.0	1.005333	0.005305	0.102869	3386.99
Jupiter	71492.0	66854.0	1.069375	0.064874	0.354316	69911.33
Saturn	60268.0	54364.0	1.108601	0.097962	0.431658	58232.01
Uranus	25559.0	24973.0	1.023465	0.022927	0.212906	25362.16
Neptune	24820.0	24274.0	1.022493	0.021998	0.208597	24636.66
Titan	2575.0	2575.0	1.000000	0.000000	0.000000	2575.00

The "surface" of an oblate spheroid model is given by

$$\frac{x^2}{R_e^2} + \frac{y^2}{R_p^2} = 1 \tag{8.19}$$

$$\frac{R^2 \cos^2(\phi_c)}{R_e^2} + \frac{R^2 \sin^2(\phi_c)}{R_p^2} = 1 \tag{8.20}$$

$$R(\varphi_c) = \frac{R_e R_p}{\sqrt{R_e^2 \cdot \sin^2(\varphi_c) + R_p^2 \cdot \cos^2(\varphi_c)}} = \frac{R_e}{\sqrt{\beta^2 \cdot \sin^2(\varphi_c) + \cos^2(\varphi_c)}} \tag{8.21}$$

and the vector representation of a point on the planet's "surface", (P), relative to the planet center, (O), is given by

$$\vec{OP} = R(\varphi_c) \cdot \begin{pmatrix} +\cos(\varphi_c)\cos(\lambda_w) \\ -\cos(\varphi_c)\sin(\lambda_w) \\ +\sin(\varphi_c) \end{pmatrix} \tag{8.22}$$

Conversion of planetographic to planetocentric latitude

Planetographic is the angle of the local normal relative to the local horizon. For an oblate sphere it is given by

$$\varphi_c = \tan^{-1}\left(\beta^2 \cdot \tan(\varphi_g)\right) \tag{8.23}$$

Conversion of planetocentric to planetographic latitude

$$\varphi_g = \tan^{-1}\left(\tan(\varphi_c)/\beta^2\right) \tag{8.24}$$

where, $\beta \equiv R_p/R_e$

Conversion of planetodetic to planetocentric latitude

Planetodetic latitude is the latitude at which the spacecraft is overhead (*i.e.*, at the local zenith)

We first define the absolute value of the planetodetic latitude as $\varphi'_d \equiv |\varphi_d|$ and then the conversion to planetocentric latitude is given by

$$\varphi'_c = \tan^{-1}\left(\frac{-b - \sqrt{b^2 - 4ac}}{2a}\right) \tag{8.25}$$

where, $a = R_s^2 - Z^2$, $b = -2R_s^2 \tan(\varphi'_d)$
$c = R_s^2 \tan^2(\varphi'_d) - Z^2$
$Z = \frac{R_e^2(1-\beta^2)\sin(\varphi'_d)}{\sqrt{R_e^2\cos^2(\varphi'_d)+R_p^2\sin^2(\varphi'_d)}}$
if $\varphi_d < 0$ then $\varphi_c = -\varphi'_c$, otherwise $\varphi_c = \varphi'_c$
as $R_s \to \infty$, $\varphi_d \to \varphi_c$
as $R_s \to R(\varphi)$, $\varphi_d \to \varphi_g$
where, $R(\varphi) = \frac{R_e R_p}{\sqrt{R_e^2\sin^2(\varphi'_d)+R_p^2\cos^2(\varphi'_d)}}$

Spacecraft Coordinate System

φ_s: planetocentric latitude, measured from equatorial plane to the spacecraft

λ_s: west longitude of spacecraft.

R_s: distance from planet center to spacecraft

$$\vec{OC} = R_s \cdot \begin{pmatrix} +\cos(\varphi_s)\cos(\lambda_s) \\ -\cos(\varphi_s)\sin(\lambda_s) \\ +\sin(\varphi_s) \end{pmatrix} \tag{8.26}$$

Camera Coordinate System

L: line number of point in image (1 at top). The lines increase in the $+y$ direction.

S: sample number of point in image (1 at left). The samples increase in the $+x$ direction.

f: camera focal length in mm's

T: camera scale factor in pixel/mm.

L_0: line number of optical axis (usually center of frame)

S_0: sample number of optical axis (usually center of frame)

L_c: line number of planet center which also corresponds to sub-spacecraft point.

S_c: sample number of planet center which also corresponds to sub-spacecraft point.

θ_n: north angle: angle between "up" (*i.e.*, $-y$ direction) and the spin axis of the object. This angle is measured clockwise.

8.4.2 Transformation from Camera Coordinates to Planet Coordinates

We desire a transformation from the camera coordinate system, (S, L), to the target coordinate system (planet, moons, rings - herein "planet"). We know the vector between planet center and the camera, $\vec{OC}$, and desire to know the vector from the planet center to the point of interest on the planet, $\vec{OP}$. The vector from the camera to the point on the planet, $\vec{CP}$ can be determined, relative to the camera axis, S_0, L_0, using the spacecraft (*i.e.*, camera) position, (φ_s, λ_s), and the camera orientation angles, θ_n, α, β. The angles α and β will be determined in the derivation and are related to the other parameters and the location of the planet center within the image coordinate system, (S_c, L_c). The planet center is the least well known parameter and, therefore, dominates the errors in the navigation process. In general, we determine $\vec{CP}$ by rotating the camera coordinate system, $\vec{CP''}$, by a matrix called the OM matrix. The form is:

$$\vec{OP} = \vec{OC} + \vec{CP} = \vec{OC} + r \cdot \vec{CP'} = \vec{OC} + r \cdot OM \cdot \vec{CP''} \tag{8.27}$$

$$\vec{CP''} = \begin{pmatrix} S - S_0 \\ L - L_0 \\ f \cdot T \end{pmatrix} \tag{8.28}$$

Where, the scale factor, r, is constrained by the planetary model (*i.e.,* the planetary model determines where $\vec{CP}$ intersects the surface and, therefore, the scale factor r). The OM matrix is determined from a series of rotations and has the form:

$$OM = M_z(\lambda_s) \cdot M_i \cdot M_y(\beta) \cdot M_x(\alpha) \cdot M_z(\theta_n) \tag{8.29}$$

We will discuss each rotation of the OM matrix separately and then discuss how they are used to transform from camera-to-planet and planet-to-camera coordinates.

1. $M_z(\theta_n)$

   Rotate the camera about the z-axis (pointing out of the image) until the north pole of the planet towards "up" in the image plane. This is illustrated in Fig. 8.9. This is denoted as M_z and the angle is the north angle, which is given in the Voyager and Galileo navigation data. For HST the orientation to celestial north is given in the FITS header (*i.e.,* the header supplied with the image) and the angle between celestial north and the north pole of the object is given in the *Astronomical Almanac*.

$$M_z(\theta_n) = \begin{pmatrix} \cos(\theta_n) & \sin(\theta_n) & 0 \\ -\sin(\theta_n) & \cos(\theta_n) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{8.30}$$

2. $M_x(\alpha)$

   Rotate about x-axis (image sample number) to position planet rotation axis parallel to the y-axis (image line number). This is illustrated in Fig. 8.10. If planet is at the optical center, then $\alpha = \varphi_s$, otherwise we must calculate the angle.

$$M_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \tag{8.31}$$

   We need to determine the value of α. If we know where the planet center, (S_c, L_c) is located then the vector from the optical axis, (S_0, L_0), to the planet center can be transformed as follows:

$$\begin{pmatrix} c'_x \\ c'_y \\ c'_z \end{pmatrix} = \begin{pmatrix} \cos(\theta_n) & \sin(\theta_n) & 0 \\ -\sin(\theta_n) & \cos(\theta_n) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} (S_c - S_0)/D \\ (L_c - L_0)/D \\ (f \cdot T)/D \end{pmatrix} \tag{8.32}$$

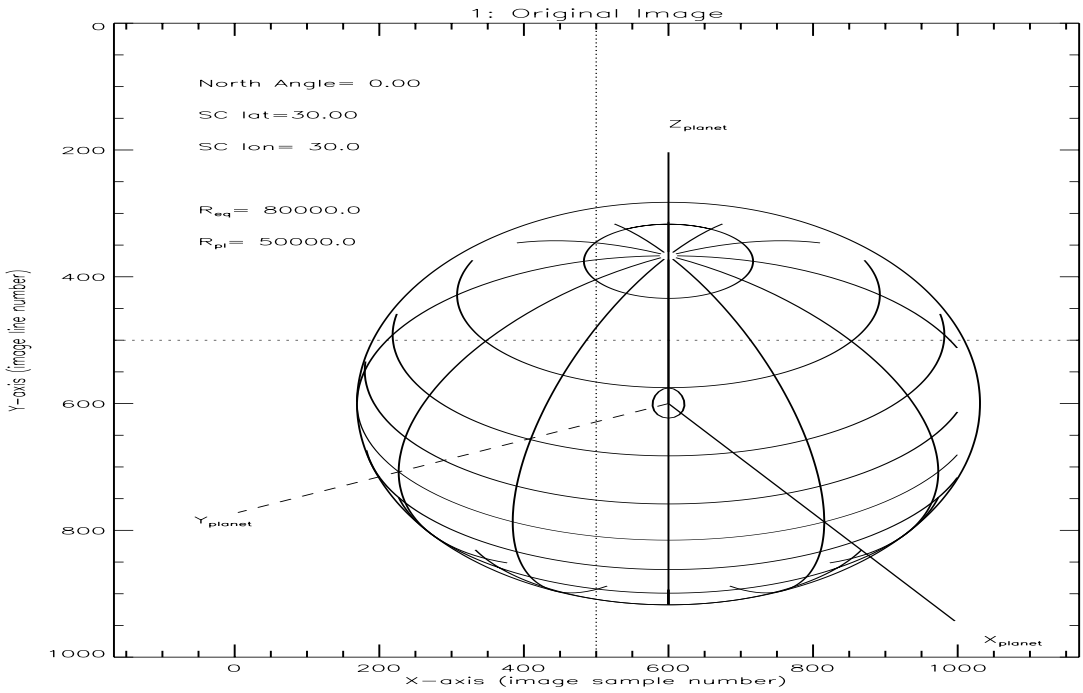


Figure 8.9: Rotation of the camera about the z axis

$$\text{where,} \quad D = \sqrt{(S_c - S_0)^2 + (L_c - L_0)^2 + (f \cdot T)^2} \tag{8.33}$$

and

$$\begin{pmatrix} c_x'' \\ c_y'' \\ c_z'' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} c_x' \\ c_y' \\ c_z' \end{pmatrix} \tag{8.34}$$

The projected image, after the rotation by α should have a projected normalized height of $\sin(\varphi_s)$, therefore,

$$c_y'' \equiv \sin(\varphi_s) = c_y' \cos(\alpha) + c_z' \sin(\alpha) \tag{8.35}$$

which, after some manipulation (Barrey, pg. 257/258) can be reduced to:

$$\sin(\alpha) = \frac{c_z' \sin(\varphi_s) - c_y' \sqrt{(c_y'^2 + c_z'^2) - \sin^2(\varphi_s)}}{c_y'^2 + c_z'^2} \tag{8.36}$$

The optical axis is now perpendicular to the planet's spin axis and the planet center, as seen from the camera, and subtends a normalized projected distance of $\sin(\varphi_s)$ from the original planet center.

3. $M_y(\beta)$

   At this point the planet's rotational axis is parallel to the -y axis and the planet center is projected at an angle α from the y axis. The planet center may also have a x-axis component which can be removed by a rotation about the y-axis. The angle of rotation, β is found by setting the result of the rotation of the planet center, c_x''', to zero. This is illustrated in Fig. 8.11.

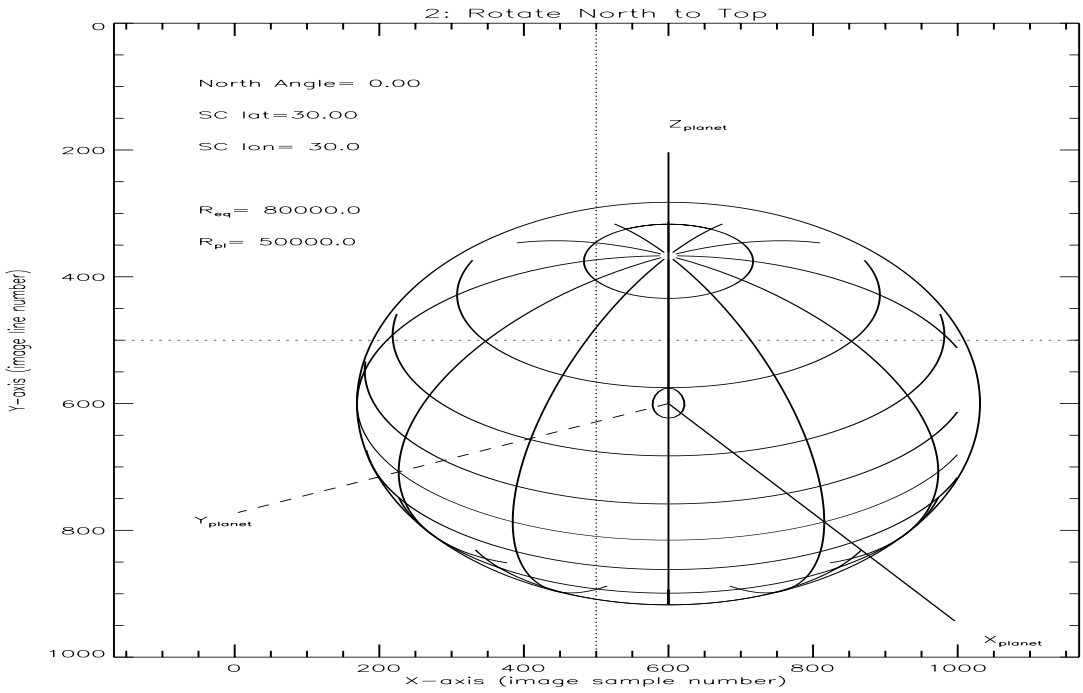


Figure 8.10: Rotation of the camera about the x axis

$$M_y(\beta) = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix} \tag{8.37}$$

and

$$\begin{pmatrix} c_x''' \\ c_y''' \\ c_z''' \end{pmatrix} = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{pmatrix} \cdot \begin{pmatrix} c_x'' \\ c_y'' \\ c_z'' \end{pmatrix} \tag{8.38}$$

The condition $c_x''' = 0$ is met when,

$$\beta = \tan^{-1}\left(\frac{c_x''}{c_x''}\right) \tag{8.39}$$

4. M_i

The rotation axis is now pointing along the -y axis. We would like to rename this axis the z axis. The x-axis and z-axis are along the equator with the z-axis pointing towards the spacecraft. Therefore, we would like to interchange the axis as

$$x' = -z \tag{8.40}$$

$$y' = +x \tag{8.41}$$

$$z' = -y \tag{8.42}$$

which can be accomplished with the following matrix multiplication:

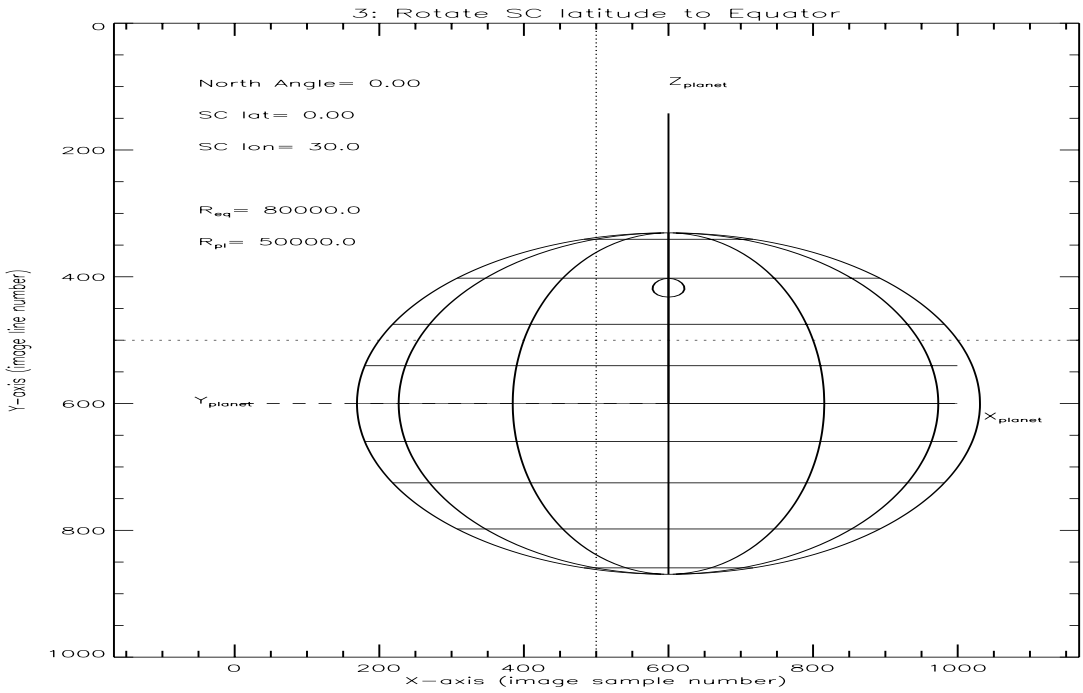


Figure 8.11: Rotation of the camera about the y axis

$$M_i = \begin{pmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix} \tag{8.43}$$

5. $M_z(\varphi_s)$

   The final transformation is to reference the longitude system to the prime meridian (*i.e.*, the longitude zero point). This is accomplished by a rotation about the new z-axis by the sub-spacecraft longitude and is illustrated in Fig. 8.13.

$$M_z(\lambda_s) = \begin{pmatrix} \cos(\lambda_s) & \sin(\lambda_s) & 0 \\ -\sin(\lambda_s) & \cos(\lambda_s) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{8.44}$$

8.4.3 Conversion from Camera to Planet Coordinates

Given the camera coordinate, (S, L) we can now specify the exact position on the planet if the OM matrix and the vector from the planet center to the spacecraft, $\vec{OC}$, are known. This process is useful in re-mapping an image into a gridded map (rectilinear, mercator, polar, etc.). The position on the planet can be written as a vector:

$$\vec{OP} = \vec{OC} + \vec{CP} = \vec{OC} + r \cdot \vec{CP'} \tag{8.45}$$

$$\vec{OC} = \begin{pmatrix} +R_s \cos(\varphi_s) \cos(\lambda_s) \\ -R_s \cos(\varphi_s) \sin(\lambda_s) \\ +R_s \sin(\varphi_s) \end{pmatrix} \tag{8.46}$$

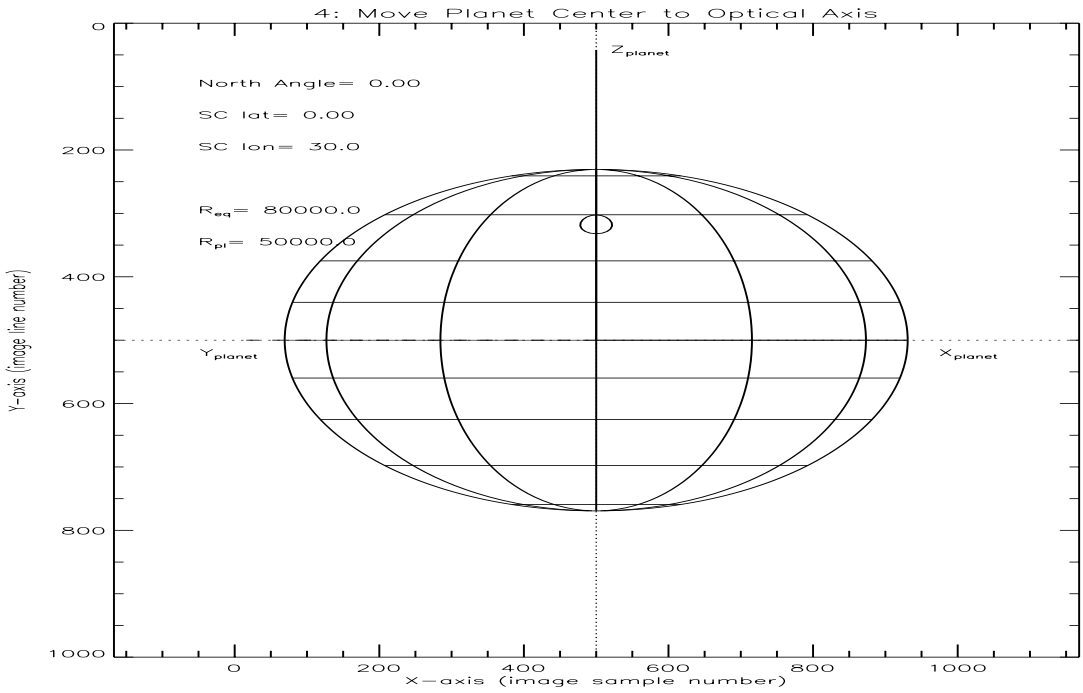


Figure 8.12: Displace planet center to center of optical axis.

$$C\vec{P}' = \begin{pmatrix} OM_{11} & OM_{21} & OM_{31} \\ OM_{12} & OM_{22} & OM_{32} \\ OM_{13} & OM_{23} & OM_{33} \end{pmatrix} \cdot \begin{pmatrix} S_c - S_0 \\ L_c - L_0 \\ f \cdot T \end{pmatrix} \tag{8.47}$$

The scaling parameter, r, is constrained by the planet model. In the outer planets we use an oblate spheroid defined by:

$$\frac{(x^2 + y^2)}{R_e^2} + \frac{z^2}{R_p^2} = 1 \tag{8.48}$$

so that

$$OP_x^2 + OP_y^2 + OP_z^2 \frac{R_e^2}{R_p^2} = R_e^2 \tag{8.49}$$

or

$$(OC_x + r \cdot CP'_x)^2 + (OC_y + r \cdot CP'_y)^2 + (OC_z + r \cdot CP'_z)^2 \frac{R_e^2}{R_p^2} = R_e^2 \tag{8.50}$$

Everything is known in this equation except r. After some manipulation the solution for r can be given as a quadratic equation, where

$$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{8.51}$$

For $b^2 - 4ac < 0$ there is no intersection of the line of sight with the planet surface. The negative root is the near side of the planet.

$$\gamma \equiv \frac{R_e^2}{R_p^2} \tag{8.52}$$

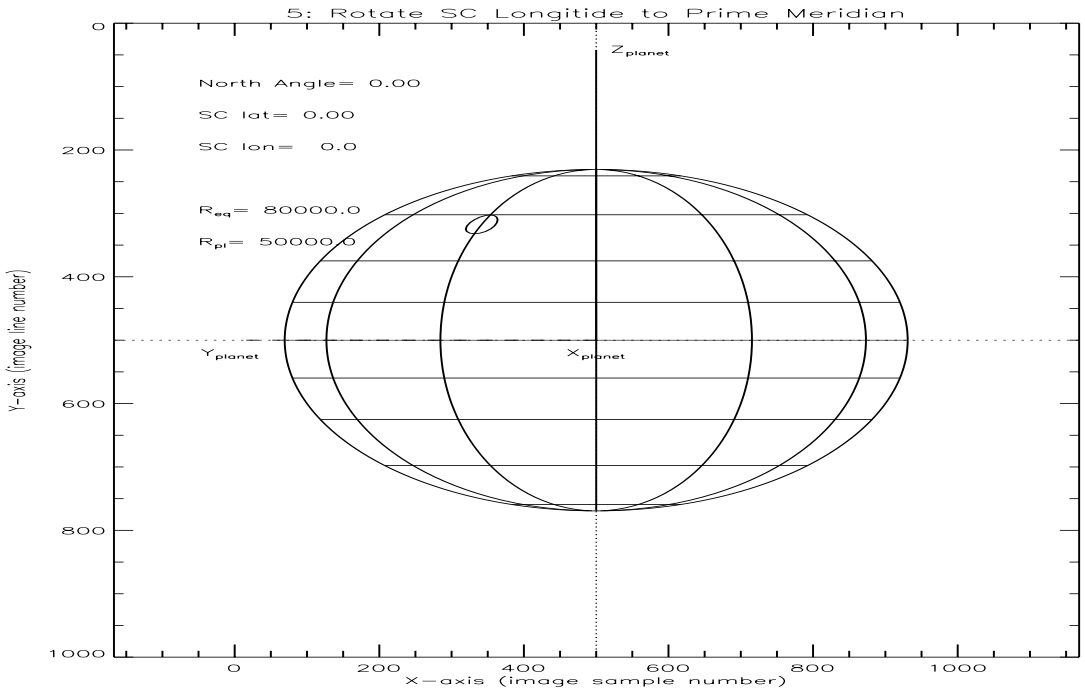


Figure 8.13: Rotate about the new z axis

$$a = CP_x'^2 + CP_y'^2 + \gamma \cdot CP_z'^2 \tag{8.53}$$

$$b = OC_x \cdot CP_x' + OC_y \cdot CP_y' + \gamma \cdot OC_z \cdot CP_y' \tag{8.54}$$

$$c = OC_x^2 + OC_y^2 + \gamma \cdot OC_z^2 - R_e^2 \tag{8.55}$$

and the west longitude, planetocentric latitude, and radius are calculated as follows:

The OM matrix has the property of being orthogonal because it is composed of simple rotation matrices. Therefore, the inverse matrix is simply the transpose of the OM matrix.

$$OM^{-1} = OM^T \tag{8.56}$$

In practice, this requires a simple interchange of the indices of a 2-d array.

$$\lambda_w = \tan^{-1}\left(\frac{-OP_y}{OP_x}\right) \tag{8.57}$$

$$\varphi_c = \tan^{-1}\left(\frac{OP_z}{\sqrt{(OP_x^2 + OP_y^2)}}\right) \tag{8.58}$$

$$\varphi_g = \tan^{-1}\left(\frac{\gamma \cdot OP_z}{\sqrt{(OP_x^2 + OP_y^2)}}\right) \tag{8.59}$$

$$R(\varphi_c) = \frac{R_e R_p}{\sqrt{R_e^2 \cdot \sin^2(\varphi_c) + R_p^2 \cdot \cos^2(\varphi_c)}} \tag{8.60}$$

8.4.4 Conversion from planet to camera coordinates

$$\vec{CP} = \vec{OP} + \vec{OC} \tag{8.61}$$

Where both $\vec{OP}$ and $\vec{OC}$ are known (see page 1).

$$\vec{CP}'' = \begin{pmatrix} (S - S_0) \\ (L - L_0) \\ f \cdot T \end{pmatrix} = c \cdot \begin{pmatrix} OM_{11} & OM_{12} & OM_{13} \\ OM_{21} & OM_{22} & OM_{23} \\ OM_{31} & OM_{32} & OM_{33} \end{pmatrix} \cdot \begin{pmatrix} CP_x \\ CP_y \\ CP_z \end{pmatrix} \tag{8.62}$$

Since all the terms within the z equation are known, the scaling constant, c, can be found as follows.

$$c = \frac{f \cdot T}{OM_{31} \cdot CP_x + OM_{32} \cdot CP_y + OM_{33} \cdot CP_z} \tag{8.63}$$

therefore,

$$S = S_0 + c \cdot (OM_{11} \cdot CP_x + OM_{12} \cdot CP_y + OM_{13} \cdot CP_z) \tag{8.64}$$

$$L = L_0 + c \cdot (OM_{21} \cdot CP_x + OM_{22} \cdot CP_y + OM_{23} \cdot CP_z) \tag{8.65}$$

In this case, the line and sample are real values, and will require interpolation into an image with integer line and sample indices. See Section 4.7 and Eqn. 4.38 for conversion of the DN value at L and S.

8.4.5 Determination of Insolation and Emission Angles

The angles between the local normal and the spacecraft or sun can be used to understand the radiative properties of an atmosphere. These properties can be used to enhance features within the image. For example, with the planet Uranus the hazy atmosphere acts like a Lambertian scatterer. A Lambertian surface can be subtracted from the image of Uranus to enhance faint features within the atmospheric. This is particular useful when attempting to compute wind vectors from cloud-tracers in the atmosphere.

The local normal can be defined from the planetographic latitude:

$$\vec{N} = \begin{pmatrix} \cos(\varphi_g)\cos(\lambda_w) \\ -\cos(\varphi_g)\sin(\lambda_w) \\ \sin(\varphi_g) \end{pmatrix} \tag{8.66}$$

cosine of insolation angle

The incident angle (insolation angle) can be obtained by taking the dot product of the local normal and the direction to the Sun.

$$\begin{aligned} OS_x &= +R_\odot \cos(\varphi_{sun})\cos(\lambda_{sun}) \\ OS_y &= -R_\odot \cos(\varphi_{sun})\sin(\lambda_{sun}) \\ OS_z &= +R_\odot \sin(\varphi_{sun}) \end{aligned}$$

$$\mu_0 = (OS_x \cdot N_x + OS_y \cdot N_y + OS_z \cdot N_z)/R_\odot \tag{8.67}$$

cosine of emission angle

For the far encounter mode the emission angle can be obtained by taking the dot product of the local normal and the direction to the spacecraft.

$$\begin{aligned} OC_x &= +R_s \cos(\varphi_s)\cos(\lambda_s) \\ OC_y &= -R_s \cos(\varphi_s)\sin(\lambda_s) \\ OC_z &= +R_s \sin(\varphi_s) \end{aligned}$$

$$\mu = \vec{OC} \cdot \vec{N} = (OC_x \cdot N_x + OC_y \cdot N_y + OC_z \cdot N_z)/R_s) \tag{8.68}$$

For near encounter it is necessary to calculate μ as follows:

$$\mu = \frac{-\vec{CP} \cdot \vec{N}}{|\vec{CP}|} \tag{8.69}$$

$$\vec{CP} = \vec{OP} - \vec{OC} \tag{8.70}$$

To develop mosaics, "snakeskins", or produce certain types of contrast enhancement (*e.g.*, Uranus) it is necessary to produce a simple model of the variation of intensity as a function of the insolation angle, emission angle, and phase function. If the image is divided by this function each pixel is an approximation of the intensity viewed at 0° phase angle from overhead.
Simple photometric functions are:
Lambert

$$DN(l,s) = A_0 \cdot \mu_0(\varphi(l,s), \lambda(l,s)) \tag{8.71}$$

Minnaert

$$DN(l,s) = A_0 \cdot \mu_0^k(\varphi(l,s), \lambda(l,s)) \cdot \mu^{(k-1)(\varphi(l,s),\lambda(l,s))} \tag{8.72}$$

$k = 1$ is Lambert (*i.e.*, isotropic scatterer)
Veverka

$$DN(l,s) = A_0 \cdot \frac{\mu(\varphi(l,s), \lambda(l,s))}{\mu_0(\varphi(l,s), \lambda(l,s)) + \mu(\varphi(l,s), \lambda(l,s))} \tag{8.73}$$

Hapke-Irvine

$$DN(l,s) = A_0 \cdot \frac{\mu_0(\varphi(l,s), \lambda(l,s))}{\mu_0(\varphi(l,s), \lambda(l,s)) + \mu(\varphi(l,s), \lambda(l,s))} \tag{8.74}$$

For example, in the Voyager encounter with Uranus the south pole was facing the Sun. Uranus was believed to have a significant haze layer and as the spacecraft approached every image had a "cue ball" appearance. To search for atmospheric features a Minnaert function was fit to the planet and then this model was subtracted from the image. The result was mapped into a coordinate system and velocities were computed. Two small features were seen circling the planet and a crude zonal wind map could be inferred.

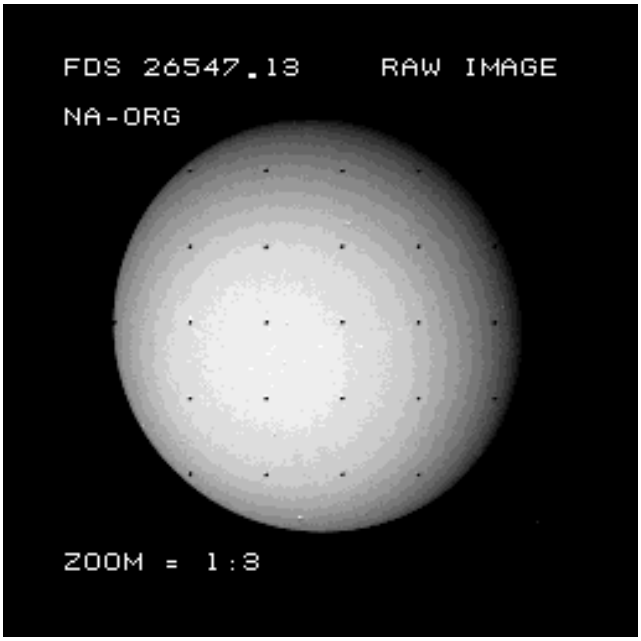
Table 8.2: Planetary encounters of the Voyager spacecraft

Event	Voyager 1	Voyager 2	T_{eff}	AU
Launch	9/5/77	8/20/77		
Jupiter	3/5/79	7/9/79	124 K	5.2
Saturn	11/12/80	8/26/81	95 K	9.55
Uranus	n/a	1/24/86	59 K	19.3
Neptune	n/a	8/25/89	59 K	30.2

The flight data system (FDS) of the Voyager spacecraft was a computer with 8192K words of memory. There were two systems. At Saturn, Voyager 2 lost 256 words of one FDS system. The Voyager's has two camera's each. The wide angle (WA) has a focal length of 201 mm and the narrow angle (NA) has a focal length of 1499 mm. At 30 Astronomical Units (1 AU = 93 million miles) the Voyager had 1:900 the amount of sunlight a camera on Earth has to illuminate a scene. The long exposure times needed "spacecraft nodding" to keep the images sharp. Light travel time is about 8 minutes per AU. At Neptune it took 8 hours for the spacecraft to acknowledge a command. Since the complex close encounter required many uploads, the communication sequence was to send 7 identical uploads and have on-board software "vote" as to what the correct command load should be.

The Voyager's closest encounter with Uranus occured on Jan. 24, 1986, 81,540 km above the cloud tops ($R_s = R_e + 81540$) at which time the spacecraft was 19.2 AU from the Earth and the light travel time was 2h 45m. In the months before and after Voyager took 6538 images.

The Voyager's closest encounter with Neptune was on Aug. 29, 1989, 4,850 km above the polar cloud tops ($R_s = R_p + 4850$) at which time the satellite was 30.1 AU from the Earth and the light travel time was 4h 6m. In the months before and after Voyager took $\approx$ 10000 images.



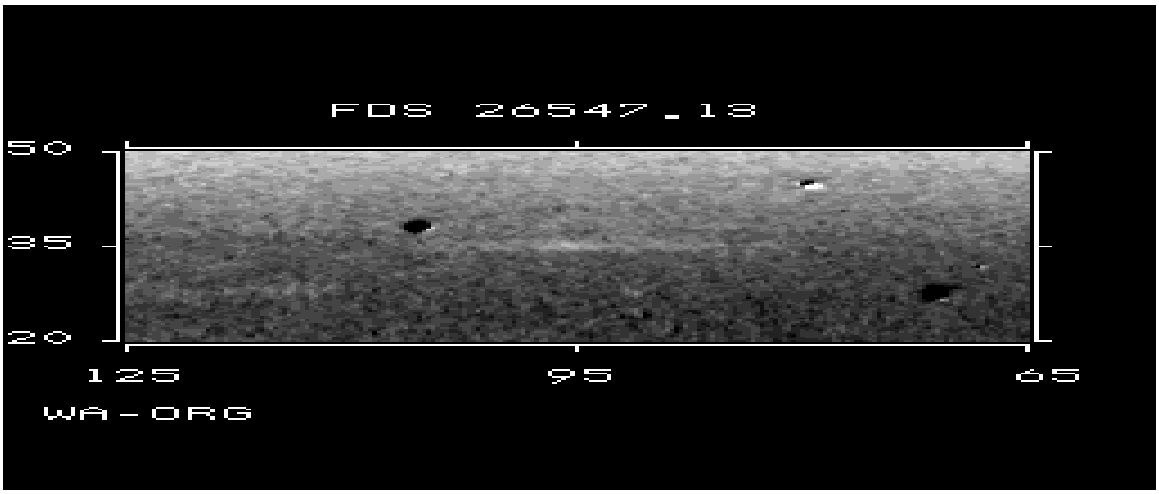
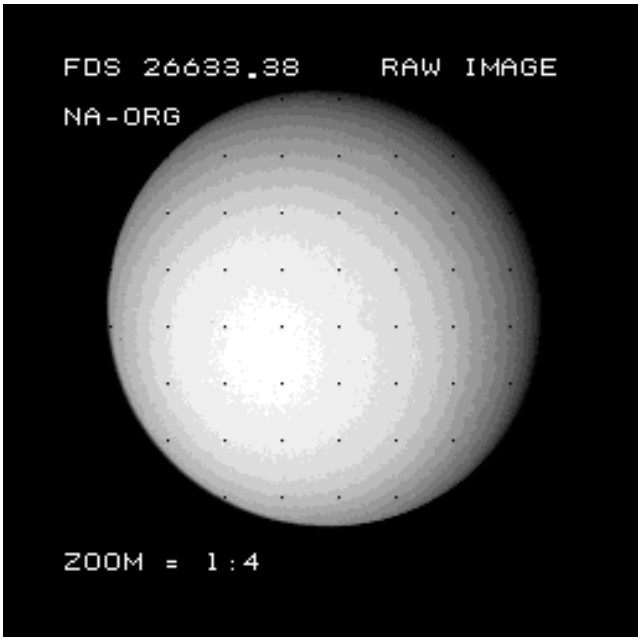


Figure 8.14: Original and re-mapping of image of Uranus, FDS=26547.13



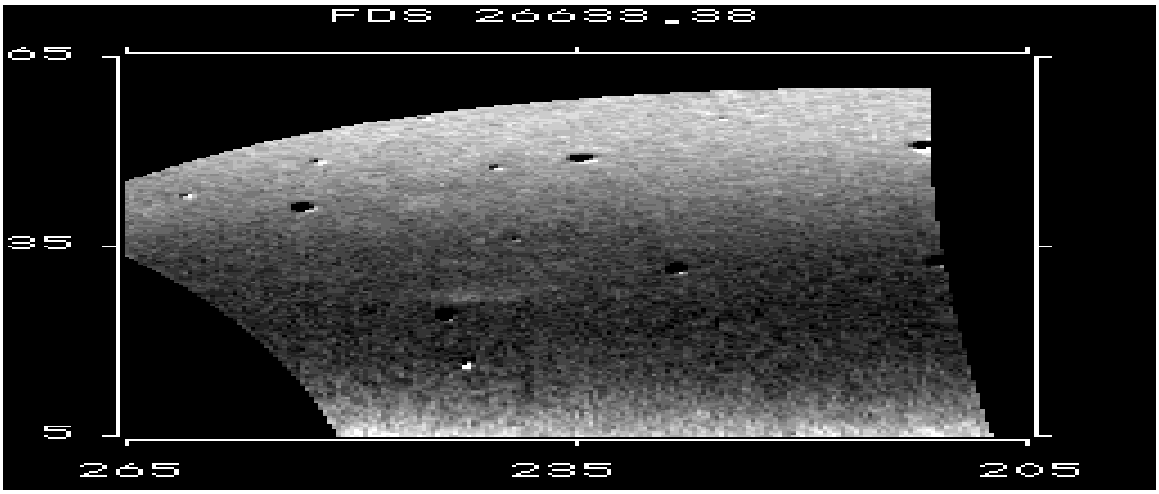


Figure 8.15: Original and re-mapping of image of Uranus, FDS=26633.38

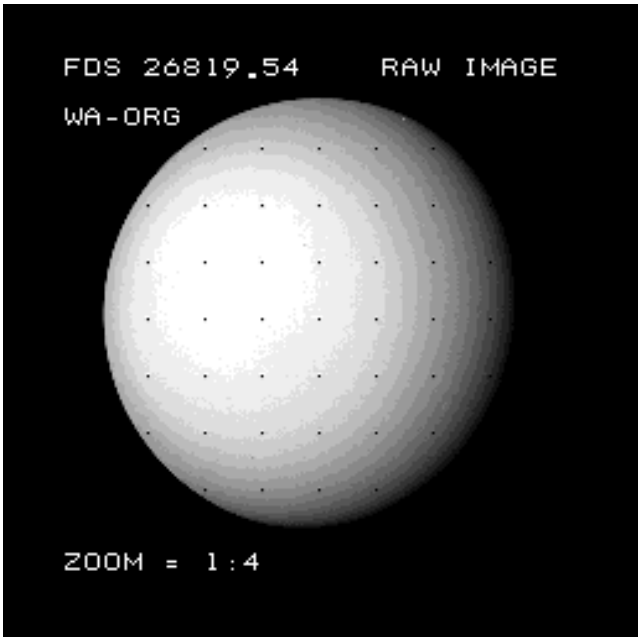


Figure 8.16: Original and re-mapping of image of Uranus, FDS=26819.54

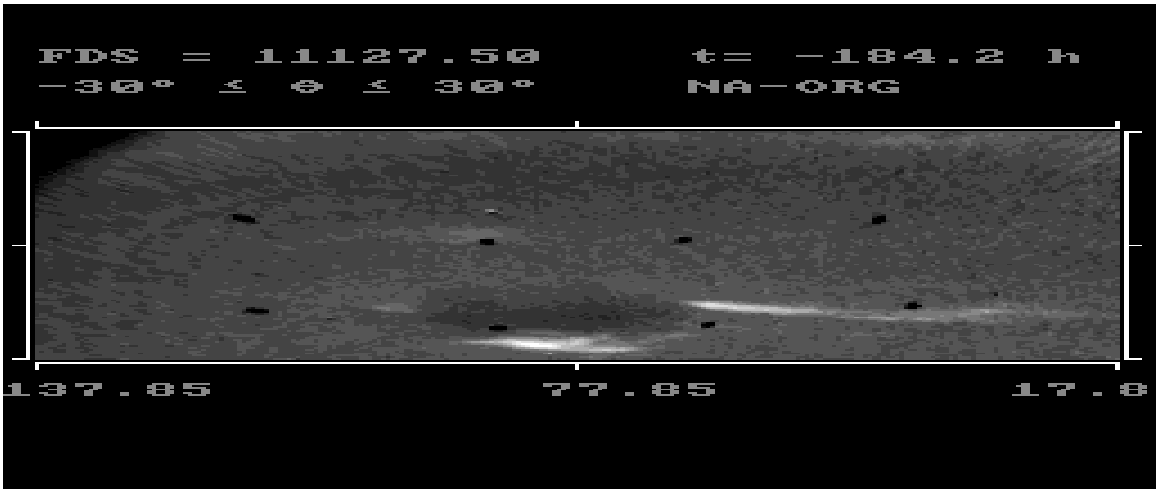


Figure 8.17: Map project of Neptune's Great Dark Spot, FDS=11127.50

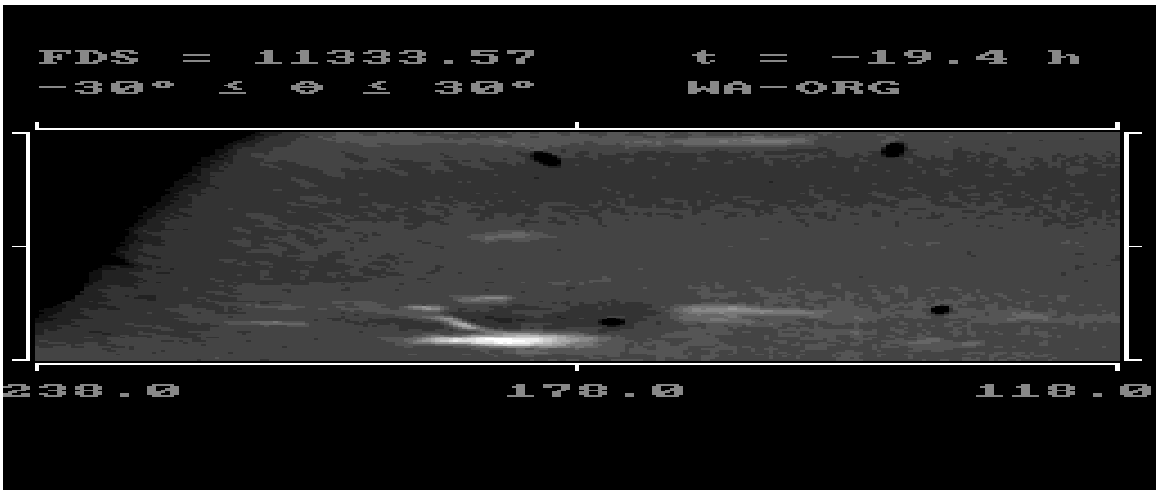


Figure 8.18: Map project of Neptune's Great Dark Spot, FDS=11333.57

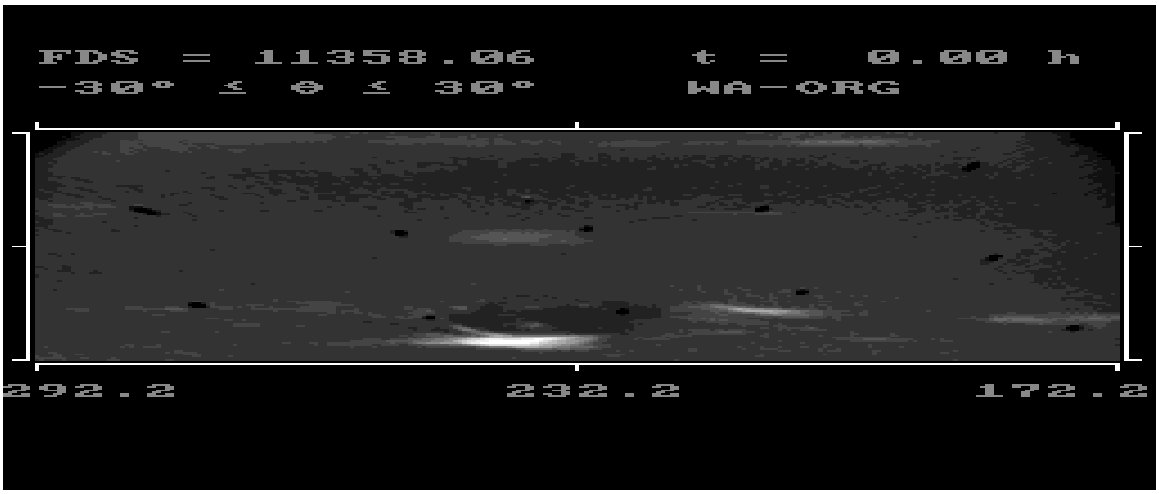


Figure 8.19: Map project of Neptune's Great Dark Spot, FDS=11358.06

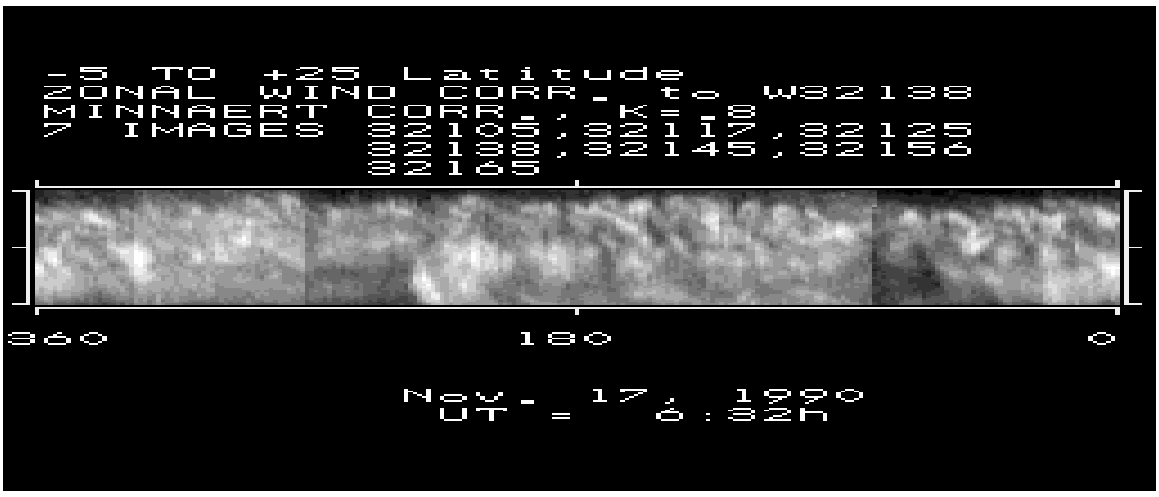


Figure 8.20: A mosaic of multiple map projected images of Saturn with Minnaert limb darkening correction, Nov. 17, 1990

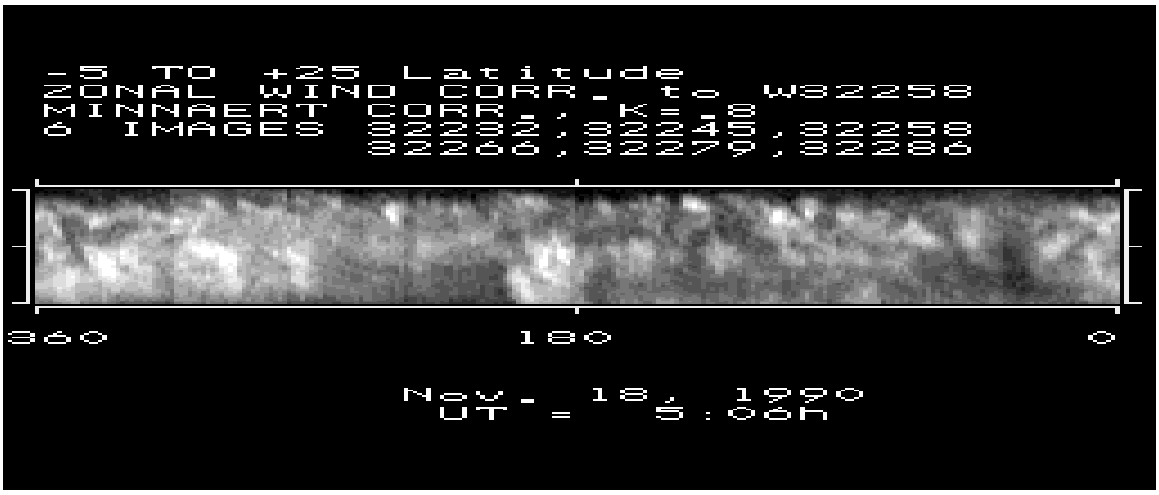


Figure 8.21: A mosaic of multiple map projected images of Saturn with Minnaert limb darkening correction, Nov. 18, 1990

Chapter 9

Solution of Differential Equations

Applications

- Vibration, sound analysis:

$$\nabla^2 u = \frac{1}{v^2}\frac{\partial^2 u}{\partial t^2} \tag{9.1}$$

- E&M: Charged particle acceleration, Antenna Propagation

$$\nabla^2 u = f(x, y, t) \tag{9.2}$$

- Orbital Dynamics, multi-body problem.
- Fluid Dynamics (Navier-Stokes Equations)
- Thermal diffusion

$$\nabla^2 u = \frac{1}{\alpha^2}\frac{\partial u}{\partial t} \tag{9.3}$$

- Quantum Mechanics
- Radiative transfer.
- Atmospheric dynamics
  - global circulation models
  - shallow water models (buoyancy models)
  - wave solutions (Rossby, Kelvin, etc.)
- Chemical rate equations, *i.e.,* simultaneous equations of chemical equilibrium.
- Electronic Circuits

$$C = \frac{\partial I}{\partial t} \quad rmor \quad C = \int V \cdot dt \tag{9.4}$$

$$L = \frac{\partial V}{\partial t} \quad rmor \quad L = \int I \cdot dt \tag{9.5}$$

- ...

9.1 Ordinary Differential Equations

9.1.1 Euler's Method

Euler's method is based on a simple estimate of the derivative at the current position,

$$f'(x_n, y_n) \equiv \left.\frac{dy}{dx}\right|_{x_n} \tag{9.6}$$

$$y_{n+1} = y_n + h \cdot f'(x_n, y_n) \quad + \quad O(h^2) \tag{9.7}$$

$$h = x_{n+1} - x_n \tag{9.8}$$

While this method is the most direct and simple, it is not as stable or accurate as other methods of similar computational burden because the solution at $x_n + h$ is derived solely on the derivative at x_n. This is a symmetrical and for large time steps becomes unstable.

As an example, a simple program to integrate the inner solar system can be envisioned with this method. We begin the simulation with the approximate positions and velocities of the steady state solar system (with the Sun positioned at the center). This system can be run for many years before errors begin to propagate. For fun we can introduce a renegade black hole into the system to see what will happen. Here are the characteristics and starting point of the objects.

	mass (gm)	Position (AU) $\hat{i}$	$\hat{j}$	$\hat{k}$	Velocity (km/s) $\hat{i}$	$\hat{j}$	$\hat{k}$
Sun	$0.19890 \cdot 10^{34}$	0	0	0	0	0	0
Earth	$0.59760 \cdot 10^{28}$	1	0	0	0	29.79	0
Venus	$0.48824 \cdot 10^{28}$	0.7233	0	0	0	35.0	0
Black Hole	$0.19890 \cdot 10^{34}$	8.0	0.5	0	-3.0	0	0

The gravitational acceleration on object n, $\vec{a}_n^i$, is computed from the positions at iteration i. The distance between two objects m and n in two dimensions is given by

$$\Delta\vec{r}_{m,n} = \hat{i} \cdot (x_m - x_n) + \hat{j} \cdot (y_m - y_n) \tag{9.9}$$

and the total acceleration on object n is given by

$$\vec{a}_n^i = \sum_{m \neq n} \frac{G \cdot m_m}{(x_m - x_n)^2 + (y_m - y_n)^2} \cdot \left[\hat{i} \cdot \cos(\theta_{m,n}) + \hat{j} \cdot \sin(\theta_{m,n})\right] \tag{9.10}$$

where the angle is given by

$$\theta_{m,n} = \tan^{-1}(\frac{(y_m - y_n)}{(x_m - x_n)} \tag{9.11}$$

the new velocity of object n at iteration i is given by

$$\vec{v}_n^i = \vec{a}_n^i \cdot \Delta t^i = \hat{i} v_{x,n}^i + \hat{j} v_{y,n}^i \tag{9.12}$$

If the velocity exceeded an upper limits (10^5 cm/s) then the time step of the next iteration, Δt^{i+1}, would be reduced. If the velocity was under a lower threshold (10^3 cm/s) then the time step of the next iteration is increased. After all the velocities are computed, then the new positions are computed

$$\begin{aligned} x_n^{i+1} &= x_n^i + v_{x,n}^i \cdot \Delta t^i \\ y_n^{i+1} &= y_n^i + v_{y,n}^i \cdot \Delta t^i \end{aligned} \tag{9.13}$$

An IDL code to perform the solar system simulation is given below.

```
    for itme = 0L, Ntme-1L do begin  ; big time step
    Nstep = Nstepnew
    DT = DT0/float(Nstep)            ; delta time (sec) for this sub-step
    dx = FLTARR(3)
    for istep = 0, Nstep-1 do begin  ; little time step
      for I=0,np-1 do begin

;       compute acceleration on object = I
;       we will assume 2-dimensional for now, to make it run faster
        AX=0.0d00
        AY=0.0d00
        AZ=0.0d00
        for J=0,NP-1 do begin
          if(I ne J) then begin
            for k=0,2 do begin
              dx(k) = POS(I,k) - POS(J,k)
            endfor
            ang = atan(dx(1),dx(0))
            force = mass(j)/(dx(0)*dx(0)+dx(1)*dx(1))
            AX = AX - force*COS(ang)          ; cm/s^2
            AY = AY - force*SIN(ang)
          endif
        endfor

;       compute Delta velocity on object=I
        DVX = AX*DT
        DVY = AY*DT

;       compute new velocity
        VEL(I,0) = VEL(I,0)+DVX
        VEL(I,1) = VEL(I,1)+DVY
        VEL(I,2) = 0.0d00

;       compute a new time step, if necessary, but DO NOT USE IT YET
        DVX = ABS(DVX)
        DVY = ABS(DVY)
        case 1 of
          (DVX gt 0.1d06 or DVY gt 0.1d06): begin
            Nstepnew = Nstep + 1L
            if(Nstepnew gt 10000L) then Nstepnew = 10000L
            end
          (Nstep eq Nstepnew and DVX lt 1.0d03 and DVY lt 1.0d03): begin
            Nstepnew = Nstep - 1L
            if(Nstepnew lt 1L) then Nstepnew = 1L
            end
        else:
        endcase
      endfor

;     now all planet's velocities are computed, determine new position
      TME = TME + DT                         ; new time for this step
      for I=0,NP-1 do begin
```

```
      for j = 0,2 do begin
        POS(I,J) = POS(I,J) + VEL(I,J)*DT   ; position, cm
      endfor
    endfor
  endfor  ; Nstep
```

Starting with the initial conditions above the Sun and Black Hole will collide at 2.5 years, at which time they will shoot out of the solar system in opposite directions (elliptic orbits with extremely high eccentricity). The Earth and Venus tumble for a couple of orbits and are also ejected at low velocity. Ooops.

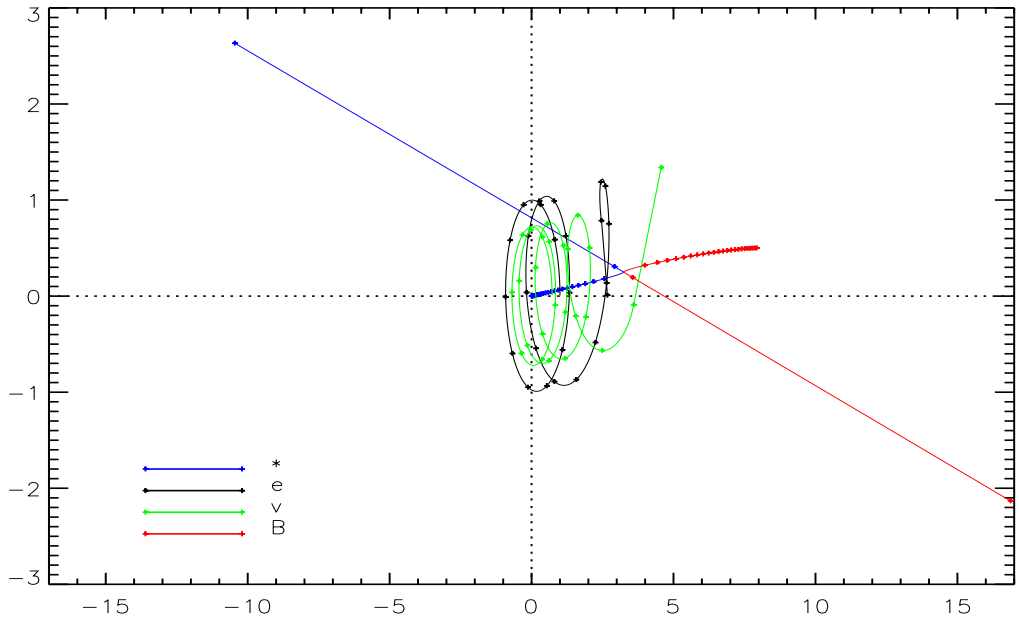


Figure 9.1: Illustration of a solar system simulator using Euler's ODE where an renegade black hole ejects the Earth

9.1.2 Modified Eulerian Forms

In general, if we want to solve for the positions of objects (Newton's Laws of Motion), $\vec{r}_j(t)$, of mass m_j, given the equation of motion in a force field for the many body problem we would have

$$\vec{a}_j(t) = \sum_{i \neq j} \frac{F(\vec{r}_i, \vec{v}_i, m_i, t)}{m_j} = \frac{1}{m_j} \cdot \frac{d\vec{v}_j(t)}{dt} = \frac{1}{m_j} \cdot \frac{d^2\vec{r}_j(t)}{dt^2} \tag{9.14}$$

In the Euler method we would solve

$$\vec{v}(n+1) = \vec{v}(n) + \vec{a}(n) \cdot \Delta t \tag{9.15}$$
$$\vec{r}(n+1) = \vec{r}(n) + \vec{v}(n) \cdot \Delta t \tag{9.16}$$

The **Euler-Cramer** is a modified Euler approach in which the predicted velocity is used to compute the new position

$$\vec{v}(n+1) = \vec{v}(n) + \vec{a}(n) \cdot \Delta t \quad (9.17)$$
$$\vec{r}(n+1) = \vec{r}(n) + \vec{v}(n+1) \cdot \Delta t \quad (9.18)$$

This leads to the conclusion that it might be better to use the average velocity to evaluate the interval. This leads to another modified Euler form called **Euler-Richardson**

$$\vec{v}(n+\frac{1}{2}) = \vec{v}(n) + \frac{1}{2} \cdot \vec{a}(n) \cdot \Delta t \quad (9.19)$$
$$\vec{r}(n+\frac{1}{2}) = \vec{r}(n) + \frac{1}{2} \cdot \vec{v}(n+1) \cdot \Delta t \quad (9.20)$$
$$\vec{a}(n+\frac{1}{2}\Delta t) = \frac{1}{m} \cdot F(\vec{r}(n+\frac{1}{2}), \vec{v}(n+\frac{1}{2}), t+\frac{1}{2}\Delta t) \quad (9.21)$$
$$\vec{v}(n+1) = \vec{v}(n) + \vec{a}(n+\frac{1}{2}\Delta t) \cdot \Delta t \quad (9.22)$$
$$\vec{r}(n+1) = \vec{r}(n) + \vec{v}(n+\frac{1}{2}\Delta t) \cdot \Delta t \quad (9.23)$$

We will see in the next section that this is an application of the 2^{nd}-order Runge Kutta applied to a second order equation.

9.1.3 Runge Kutta

The Runge Kutta method uses the midpoint rule of quadrature to compute a better estimate of the next value of y. The method is based on two papers one by Runge in 1895 (Runge, C. 1895. Über die numerische Auflösung von Differentialgleichungen, *Math. Ann.*, **46**, 167-178.) and one by Kutta in 1901 (Kutta, W. 1901. Beitrag zur näherungsweisen Integration totalier Differentialgleichungen *Z. Math. Phys.*,**46**, 435-453). In general, a N^{th} order Runge Kutta has an error of order, $O(h^{n+1})$.

2^{nd} order Runge Kutta

The derivative is evaluated at the current step, $f'(x_n, y_n)$, and used to compute an estimate of the endpoint, $(x_{n+1}, y_n + k_1)$ (Euler's estimate). An estimate of the value at the midpoint, $(x_n + h/2, y_n + k_1/2)$ is used to compute a new derivative and this derivative is used to improve the estimate of the value at the endpoint $(x_{n+1}, y_n + k_2)$. This method is discussed in Press *et al.* [1986].

$$k_1 = h \cdot f'(x_n, y_n) \quad (9.24)$$
$$k_2 = h \cdot f'(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \quad (9.25)$$
$$y_{n+1} = y_n + k_2 \quad + \quad O(h^3) \quad (9.26)$$

The 2nd order Runge-Kutta on an interval h is NOT equal to the Euler equation on an interval of $h/2$. The reason is that the Runge-Kutta derivative, computed at $h/2$ is applied to the entire interval from the last determined point (x_n, y_n). This is why it is more accurate, analygous to the improvement in accuracy of the midpoint quadrature rule with respect to the trapezoidal quadrature rule.

For example, the Runge-Kutta method, computed from the origin $(x_n = 0, y_n)$ over an interval of h would give

$$k_1 = h \cdot y'(0,0) \quad (9.27)$$
$$k_2 = h \cdot y'\left(\frac{h}{2}, \frac{k_1}{2}\right) \quad (9.28)$$
$$y(h) = y(0) + k_2 \quad (9.29)$$

while the Euler method on an interval of $\frac{h}{2}$ would give

$$y\left(\frac{h}{2}\right) = y(0) + \left(\frac{h}{2}\right) \cdot y'(0,0) = y(0) + \frac{k_1}{2} \tag{9.30}$$

and the Euler method on an interval of h would give

$$\begin{aligned} y(h) &= y(0) + \frac{k_1}{2} + \left(\frac{h}{2}\right) \cdot y'\left(\frac{h}{2}, \frac{k_1}{2}\right) && (9.31)\\ &= y(0) + \frac{k_1}{2} + \frac{k_2}{2} && (9.32)\\ &= y(0) + 0.5 \cdot (k_1 + k_2) && (9.33) \end{aligned}$$

If we compute the Taylor expansion of a derivative about the origin

$$y(h) = y(0) + h \cdot y'(0,0) + \left(\frac{h^2}{2!}\right) \cdot y''(0,0) + \left(\frac{h^3}{3!}\right) \cdot y'''(0,0) + \ldots \tag{9.34}$$

we can see that Euler's method is equal to the first two terms of the Taylor expansion

$$y(h) \simeq y(0) + h \cdot y'(0,0) + O(h^2) \tag{9.35}$$

A plausibility argument for the acccuracy of the 2^{nd}-order Runge Kutta is to note that computing the derivative about the midpoint

$$\begin{aligned} y(h) &= y(0) + h \cdot y'(\frac{h}{2}, \frac{h}{2} \cdot y'(0,0))\\ &\approx y(0) + h \cdot y'(0,0) + \left(\frac{h^2}{2!}\right) \cdot y''(0,0) + O(h^3) && (9.36) \end{aligned}$$

while this is not mathematically rigorous, it illustrates why the Runge Kutta methods tend to be more accurate. In general, one compares the ODE approximation to a Taylor expansion solution for a family of ODE's. See Section 9.1.4 for a comparison of the accuracy of Runge Kutta methods and the Euler method. Derivations of this kind (for 4th order) are given in Scheid [1989].

4^{th} order Runge Kutta

Press *et al.* [1986] discuss a 4^{th} order Runge-Kutta formula. Here the 2^{nd} order method is used to product an estimate of the endpoint, $y_n + k_2$ and then another derivative at the midpoint is computed and used again to compute a new estimate of the endpoint, $y_n + k_3$. The derivative is evaluated a fourth time at the endpoint and then the endpoint is estimated by the weighted sum of all the estimates of the endpoint from the left hard side, two at the center, and the right hand side. The weighting is chosen to minimize the errors (determined from a Taylor expansion)

$$\begin{aligned} k_1 &= h \cdot f'(x_n, y_n) && (9.37)\\ k_2 &= h \cdot f'(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) && (9.38)\\ k_3 &= h \cdot f'(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}) && (9.39)\\ k_4 &= h \cdot f'(x_n + h, y_n + k_3) && (9.40)\\ y_{n+1} &= y_n + \frac{1}{6} \cdot (k_1 + 2k_2 + 2k_3 + k_4) \quad + \quad O(h^5) && (9.41) \end{aligned}$$

The 4^{th}-order Runge-Kutta (RK4) algorithm is a generalization of Simpson's quadrature. If we consider a function of x only then we can write $f(x) = f(x, y)$. In this case and

$$y(x) - y(x_1) = \int_{x_1}^{x} f(x)\ dx \tag{9.42}$$

therefore, evaluating the term $y(x)$ with RK4 is the same as evaluating the integral. For a 3-point Simpsons's quadrature we see that for points separated by $h/2$ instead of h as in Eqn. 5.26 we would have

$$\int_{x_n}^{x_n+h} f(x) \cdot dx = \frac{h}{6}\left[y(x_n) + 4 \cdot y(x_{n+1}) + y(x_{n+2})\right] \tag{9.43}$$

and for $f(x) = f(x, y)$ we see that RK4 would have $k_3 = k_2$ and, therefore, represents the Simpson's quadrature.

5$^{\text{th}}$ order Runge Kutta

From Milne [1970]. Kutta and Nyström (Nyström, E.J. 1926. Über die numerische Auflösung von Differentialgleichungen, *Acta Soc. Sci. Fennicae*, **50**, 55pp.) obtained sets of formulas using six substitutions. The complexity of formulas increases rapidly as the order increases.

$$\begin{aligned}
k_1 &= h \cdot f'(x_n, y_n) && (9.44)\\
k_2 &= h \cdot f'(x_n + \frac{h}{3}, y_n + \frac{k_1}{3}) && (9.45)\\
k_3 &= h \cdot f'(x_n + \frac{2h}{5}, y_n + \frac{6k_2 + 4k_1}{25}) && (9.46)\\
k_4 &= h \cdot f'(x_n + h, y_n + \frac{15k_3 - 12k_2 + k_1}{4}) && (9.47)\\
k_5 &= h \cdot f'(x_n + \frac{2h}{3}, y_n + \frac{8k_4 - 50k_3 + 90k_2 + 6k_1}{81}) && (9.48)\\
k_6 &= h \cdot f'(x_n + \frac{4h}{5}, y_n + \frac{8k_4 + 10k_3 + 36k_2 + 6k_1}{75}) && (9.49)\\
y_{n+1} &= y_n + \frac{1}{192} \cdot (23k_1 + 125k_3 - 81k_5 + 125k_6) \quad + \quad O(h^6) && (9.50)
\end{aligned}$$

see Heath, pg 287-292.

9.1.4 A numerical experiment to test ODE integration methods

The function we will test is

$$y' = x \cdot y^{\frac{1}{3}} \tag{9.51}$$

We will start the integration with the boundary condition of $x(0) = 1, y(0) = 1$. Here is the relevant section of the FORTRAN code used to compute the tables below:

```
      euler = y
      rk2 = y
      rk4 = y
      do iter = 1, Nstep
        x2 = x + h/two

c       Euler's Method
        euler = euler + h*yp(x , euler)
```

```
c       2nd order Runge-Kutta
        k1 = h*yp(x  , rk2         )
        k2 = h*yp(x2 , rk2 + k1/two)
        rk2 = rk2 + k2

c       4th order Runge-Kutta
        k1 = h*yp(x   , rk4       )
        k2 = h*yp(x2  , rk4+k1/two)
        k3 = h*yp(x2  , rk4+k2/two)
        k4 = h*yp(x+h , rk4+k3    )
        rk4 = rk4 + (k1 + two*k2 + two*k3 + k4)/6.0d00
```

For a step size of h=0.5 the results are

k	x(k)	euler(k)	rk2(k)	rk4(k)	y(k)	y'(k)
1	1.5000000	1.5000000	1.6732608	1.6860903	1.6861706	1.7853571
2	2.0000000	2.3585357	2.7970410	2.8282454	2.8284271	2.8284271
3	2.5000000	3.6896463	4.5053699	4.5600633	4.5603591	4.1457810
4	3.0000000	5.6211917	6.9380074	7.0207167	7.0211321	5.7445626
5	3.5000000	8.2882595	10.2371558	10.3518482	10.3523850	7.6280732
6	4.0000000	11.8298024	14.5461170	14.6962805	14.6969385	9.7979590
7	4.5000000	16.3869087	20.0087186	20.1974438	20.1982220	12.2551010
8	5.0000000	22.1018906	26.7690524	26.9991028	27.0000000	15.0000000

Where "euler" was done using Euler's method, "rk2" was done with a 2^{nd}-order Runge-Kutta method and "rk4" was done with a 4^{th}-order Runge-Kutta method. The values of $y(k)$ and $y'(k)$ are the value and the derivative computed with a 4^{th}-order Runge-Kutta method and a step size equal to $h = 0.001$ and can be taken as the "truth" values in this example.

With a step size of $h = 0.1$ the results are

k	x(k)	euler(k)	rk2(k)	rk4(k)	y(k)	y'(k)
1	1.1000000	1.1000000	1.1067216	1.1068166	1.1068166	1.1378488
2	1.2000000	1.2135508	1.2276794	1.2278795	1.2278796	1.2849903
3	1.3000000	1.3415480	1.3638205	1.3641359	1.3641360	1.4417697
4	1.4000000	1.4849248	1.5161236	1.5165644	1.5165645	1.6084775
5	1.5000000	1.6446461	1.6855943	1.6861705	1.6861706	1.7853571
10	2.0000000	2.7238773	2.8270308	2.8284268	2.8284271	2.8284271
15	2.5000000	4.3680975	4.5579247	4.5603585	4.5603591	4.1457810
20	3.0000000	6.7147708	7.0174705	7.0211313	7.0211321	5.7445626
25	3.5000000	9.9045972	10.3473324	10.3523840	10.3523850	7.6280732
30	4.0000000	14.0798543	14.6903521	14.6969372	14.6969385	9.7979590
35	4.5000000	19.3836427	20.1899760	20.1982205	20.1982220	12.2551010
40	5.0000000	25.9595190	26.9899823	26.9999983	27.0000000	15.0000000

The Taylor expansion is defined by

$$y(x+h) = y(x) + h \cdot y'(x) + \frac{h^2}{2!} \cdot y''(x) + \frac{h^3}{3!} \cdot y'''(x) + \frac{h^4}{4!} \cdot y''''(x) + \ldots \tag{9.52}$$

and for our function ($y' = x \cdot y^{\frac{1}{3}}$),

$$y'(x) = x \cdot y^{\frac{1}{3}} \tag{9.53}$$

$$y''(x) = y^{\frac{1}{3}} + x \cdot \frac{1}{3} y^{\frac{-2}{3}} \cdot y' = y^{\frac{1}{3}} + \frac{1}{3} x^2 \cdot y^{\frac{-1}{3}} \tag{9.54}$$

$$y'''(x) = x \cdot y^{\frac{-1}{3}} - \frac{1}{9} x^3 \cdot y^{-1} \tag{9.55}$$

$$y''''(x) = y^{\frac{-1}{3}} - \frac{2}{3} x^2 \cdot y^{-1} + \frac{1}{9} x^4 \cdot y^{\frac{-5}{3}} \tag{9.56}$$

$$y^{(5)}(x) = \frac{5}{3} x \cdot y^{-1} - \frac{10}{9} x^3 \cdot y^{\frac{-5}{3}} - \frac{5}{27} x^5 \cdot y^{\frac{-7}{3}} \tag{9.57}$$

In Fig. 9.2 below shows the values $y(x)$ and the values from Euler's method for both the $h = 0.5$ and the $h = 0.1$ experiment for $1 \leq x \leq 5$.

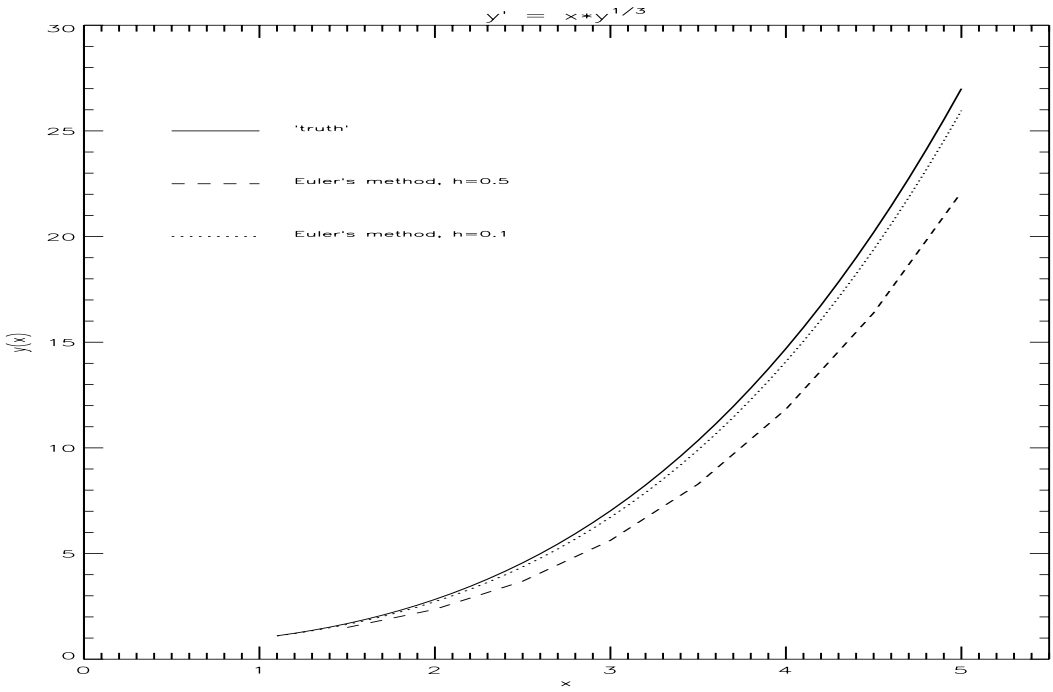


Figure 9.2: Comparison of Euler's method with different step sizes

1. **What can you say about the error of the Euler method?** HINT: Compare the Euler method to Eqn. 9.52.

   Euler's method always underestimates the value for this particular function. This is because we are effectively truncating the Taylor expansion, which for this function the quadratic term (Eqn. 9.54) is always positive.

2. **For other methods (*e.g.,* the 2$^{\text{nd}}$-order Runge-Kutta, and 4$^{\text{th}}$-order Runge-Kutta) do their errors exhibit the same characteristic as the Euler method (look at tables)?**

   Yes. The 2$^{\text{nd}}$-order Runge-Kutta also always underestimates the error as does the 4$^{\text{th}}$-order Runge-Kutta, albeit much smaller values.

3. **Is their something you could imagine to reduce this error?**

   We could:

- reduce step size
- use higher order Runge-Kutta.
- If we ran the integration backwards, the bias would become evident and the difference could be used to correct the biases.

4. **Estimate the 1st five terms of the Taylor expansion for $y' = x \cdot y^{\frac{1}{3}}$ at $x = 1, y = 1$.**

term	equation	Eqn. for $x \cdot y^{\frac{1}{3}}$	@h=0.5	@h=0.1
1	$h \cdot y'(x)$	h	0.500000	0.1000000
2	$h^2/2! \cdot y''(x)$	$2h^2/3$	0.166666	0.0066667
3	$h^3/3! \cdot y'''(x)$	$4h^3/27$	0.018519	0.0001481
4	$h^4/4! \cdot y''''(x)$	$h^4/54$	0.001157	0.00000185
5	$h^5/5! \cdot y^{(5)}(x)$	$h^5/324$	$9.6{\cdot}10^{-5}$	$3.1{\cdot}10^{-8}$

5. **Using the tables above estimate the error in the first iteration (*e.g.*, for $k = 1$, $euler(1) - y(1) = -0.0068166$) for all three methods (Euler, Runge-Kutta 2nd and 4th order) shown for both the $h = 0.5$ and $h = 0.1$ case. Then indicate which Taylor series expansion term that most closely resembles the error in the Euler, 2nd-order Runge-Kutta method, and 4th-order Runga-Kutta.**

	h=0.5	h=0.1	term
Euler	-0.1862	-0.006817	2
2nd-order Runge-Kutta	-0.0129	-0.000095	3
4th-order Runge-Kutta	-0.00008	$< 1.0 \cdot 10^{-7}$	5

6. **When computing an integral with each of these methods at what value of h should we consider using double precision (64-bit or real*8) arithmetic?**

   We should use double precision when the truncation error is less than $\approx 10^{-7}$. We can use the value of h determined by the function $f(h)$ in #1. This occurs at the following step sizes:

method	equation	@h=
Euler	$h^2 \leq 1.5 \cdot 10^{-7}$	0.00039
2nd-order Runge-Kutta	$h^3 \leq 7 \cdot 10^{-7}$	0.00888
4th-order Runge-Kutta	$h^5 \leq 3.2 \cdot 10^{-5}$	0.1265

9.1.5 Adaptive Step Size

The ordinary differential equation integrator should exert some adaptive control over its own progress, making frequent changes in its step-size if the Δy's become too large. There is a trade-off between step-size and

execution time. An adaptive approach will allow the code to "tiptoe" through treacherous terrain and make great strides through uninteresting countryside. The results in accuracy and speed can be many orders of magnitude. With 4^{th} order Runge-Kutta, the most natural adjustment is halving the step size.

One can run both a step by h and a step by $2h$ in parallel, as discussed in Press *et al.* [1986]; however, in practice one can simply monitor $y_{n+1} - y_n$ to determine if the step size should be adjusted up or down by a factor of two.

9.1.6 Stiff sets of equations

Systems in which different components operate on quite different time scales are called stiff systems and offer more than normal resistance to numerical solution (Scheid, 1989, page 226). For example,

$$y' = -100 \cdot y + 99 \cdot e^{-x} \tag{9.58}$$

If $y(0) = 0$ it has the solution

$$y = e^{-x} - e^{-100 \cdot x} \tag{9.59}$$

Both terms tend towards zero; however, the second term decays much faster than the first. At $x = 0.1$ the second term is already zero to four decimal places and can be considered a transient term relative to the first term. The step size for Runga-Kutta methods will blow up if the transient term is not treated properly. Thus, one would want a very small step size until the transient term vanishes and then the step size could be increased.

In many cases the ordinary differential equations are coupled or have components which are changing during the integration. For example, the harmonic pendulum equation in one dimension can be written as (Gould and Tobochnik [1996, pg. 96-100].

$$\frac{d^2x}{dt^2} = -\gamma \cdot \frac{dx}{dt} - \omega_0^2 \cdot x + \frac{A_0}{m} \cdot \cos(\omega_d \cdot t) \tag{9.60}$$

where ω_0 is the natural frequency of the pendulum, γ is a damping coefficient, ω_d is the driving frequency, A_0 is the driving amplitude, and m is the mass of the pendulum. This can be written as a pair of 1^{st} order differential equations; however, the value of x and dx/dt will be coupled.

In Andrews *et al.* 1987 and Holton [1979] methods of alternating the grid spacing is discussed such that the system can be stabilized. We will discuss this later under the Lax Wendroff time step.

9.1.7 Exampel of 4^{th} order Runge Kutta with the Lorenz Equations

Edward Lorenz determined that the dynamic equations of motion of the atmosphere were chaotic (see Lorenz, Edward 1963. Deterministic nonperiodic flow. J. Atmos. Sci. v.20 p.130-141), using a model of a convective cell with a warm surface and cooling at the top. Prior to his computations the understanding of the day was that turbulent flow resulted from nonperiodic or random forcing. His system of equations, while a simplification of nature, should have resulted in deterministic motions, but they show that certain equations are subject to extreme sensitivity to initial conditions.

The 1963 paper used a numerical model based on the same form of the convective equations that Richardson used to study turbulent flow. Afterwards, Lorenz built an analog computer (using capacitors and inductors to more quickly simulate the differential equations) to demonstate the effects. The equations he solved were a parameteritized and scaled form of the equation of motion ($\vec{F} = m \cdot \vec{a}$), the equation of continuity, and the 2^{nd} law of thermodynamics. The form he derived was

$$\frac{\partial x}{\partial t} = -\sigma \cdot x + \sigma \cdot y \tag{9.61}$$

$$\frac{\partial y}{\partial t} = -x \cdot z + r \cdot x - y \tag{9.62}$$

$$\frac{\partial z}{\partial t} = +x \cdot y - b \cdot z \tag{9.63}$$

Where x is proportional to the convective velocity (+ is clockwise, - is counterclockwise), y is proportional to the temperature difference between ascending and descending currents, and z is proportional to the distortion of the vertical temperature profile from linearity, a positive value indicating that the strongest gradients occur near the boundaries.

Where $\sigma = \nu/\kappa$ is the ratio of the kinematic viscosity to the thermal conductivity, $b = 4(1 + a^2)^{-1}$, and $r = R_a/R_c$, the *Prandtl number*. R_a is the *Rayleigh number*, given by $R_a = g \cdot \alpha \cdot H^3 \cdot \Delta T/(\nu \cdot \kappa)$ where H is the depth of the fluid, g is the acceleration of gravity, and α is the coefficient of thermal expansion. ΔT is the temperature difference between the top and bottom of the convective cell. For a wave of wavenumber a, the flow will be turbulent if $R_a \gg R_c$, where $R_c = \pi^4 \cdot a^{-2} \cdot (1 + a^2)^3$.

In 1977 Robert Shaw and Peter Scott showed that the Lorenz equations were chaotic with an analog computer (NOVA "The Strange New Science of Chaos", 1989)

With the values σ=10, b=8/3, and r=10 with $x(t)$=1, $y(t)$=1, and $z(t)$=20 at $t = 0$. The problem was integrated from $t = 0$ to $t = 20$ using a step size of 0.0025 seconds, the results are

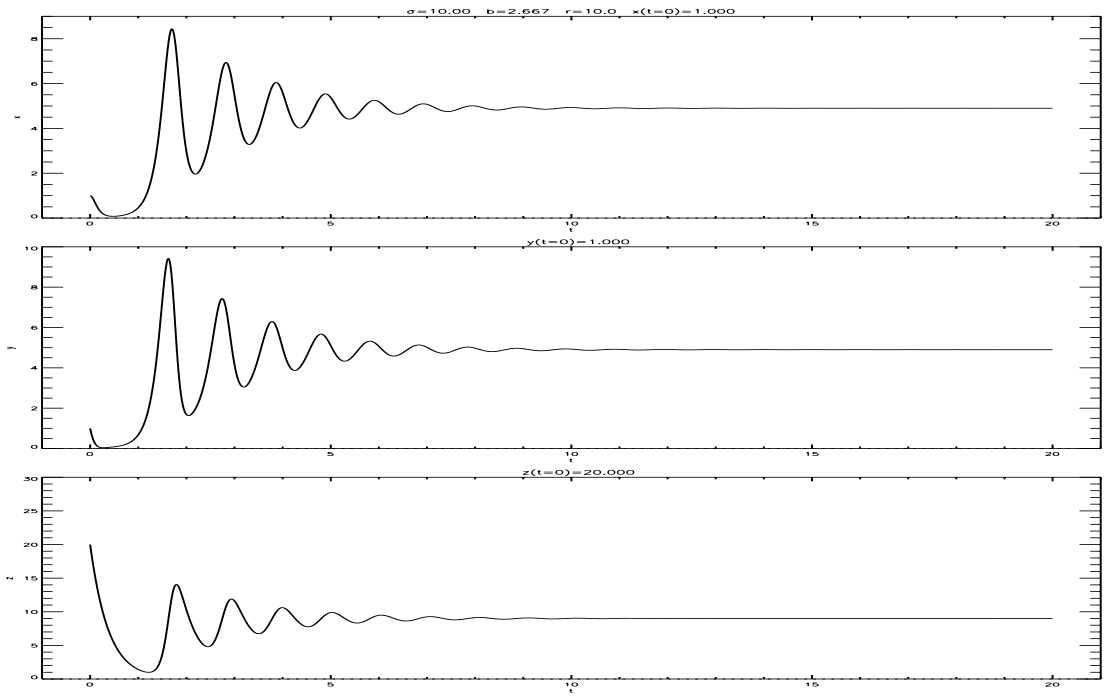


Figure 9.3: Integration of the Lorentz differential equations, (σ=10, b=8/3, and r=10 with $x(t)$=1): $x(t), y(t), z(t)$

$\sigma = 10, b = 2.667, r = 10$		
variable	@$t = 0$	@$t = 20$
x	1.000	4.899
y	1.000	4.899
z	20.000	9.000

The solution with $x(t)$=1.001, $y(t)$=0.999, and $z(t)$=20.001 at $t = 0$ is significantly different and looks like

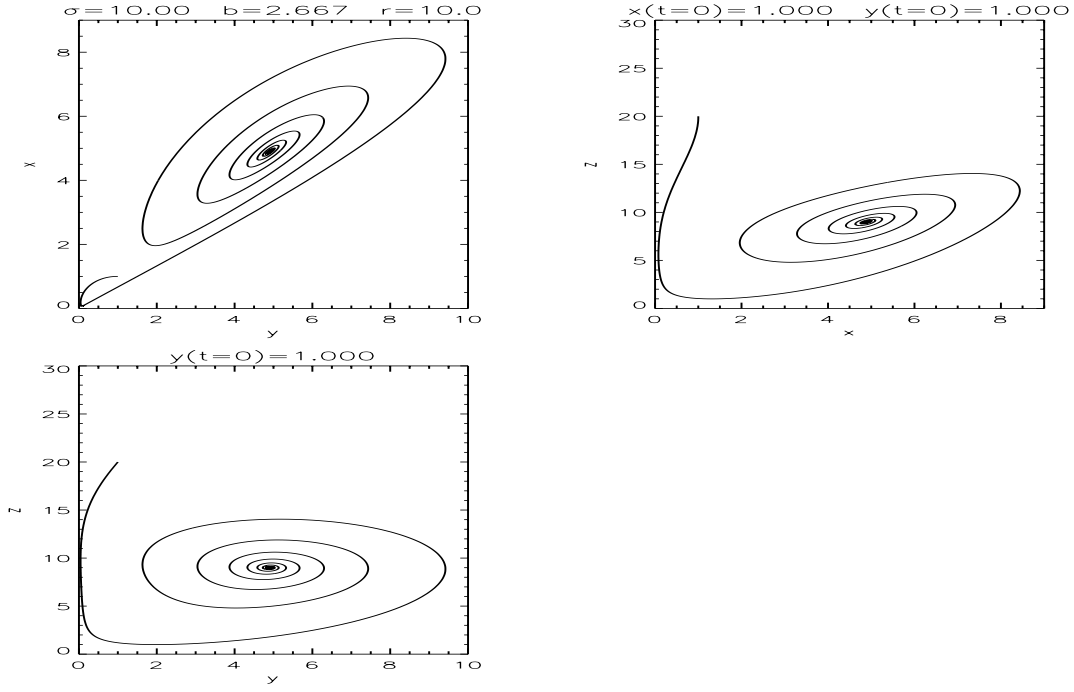


Figure 9.4: Integration of the Lorentz differential equations, (σ=10, b=8/3, and r=10 with $x(t)$=1): $x(y), z(x), z(y)$

$\sigma = 10, b = 2.667, r = 10$		
variable	@$t = 0$	@$t = 20$
x	1.001	4.899
y	0.999	4.899
z	20.001	9.000

With the values σ=10, b=8/3, and r=28, the results are In this example we started with the initial conditions, $x(t)$=1, $y(t)$=1, and $z(t)$=20 at $t = 0$

$\sigma = 10, b = 2.667, r = 10$		
variable	@$t = 0$	@$t = 20$
x	1.000	-0.077
y	1.000	0.731
z	20.000	18.948

The solution with $x(t)$=1.001, $y(t)$=0.999, and $z(t)$=20.001 at $t = 0$ is significantly different and looks like

$\sigma = 10, b = 2.667, r = 10$		
variable	@$t = 0$	@$t = 20$
x	1.001	11.943
y	0.999	12.114
z	20.001	31.340

9.2 Partial Differential Equations

There are a number of approaches to solve partial differential equations which are a function of spatial variables (x, y, z) and time, t. For periodic solutions, a Fourier series can be used to separate variables and solve for parts of the equations, simplifying the solution. For many physical problems, the spatial derivatives are computed on a grid over a finite region $x = x_0 + h \cdot i$ for $i = 1, I$ and $y = y_0 + h \cdot j$ for $j = 1, J$, etc. We

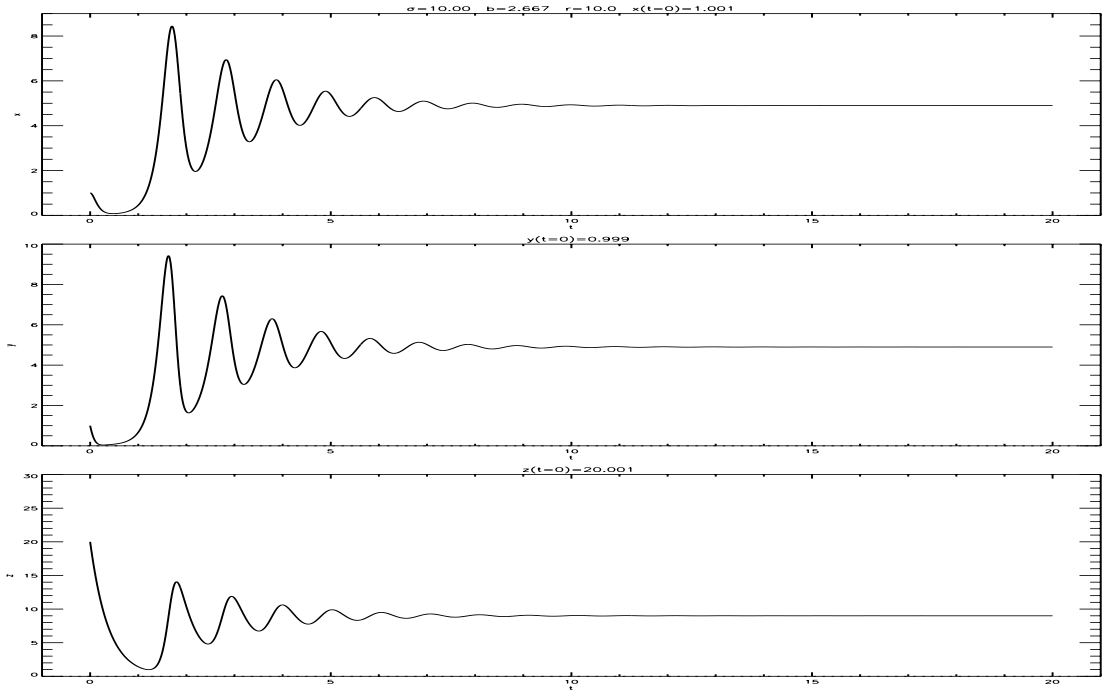


Figure 9.5: Integration of the Lorentz differential equations, (σ=10, b=8/3, and r=10 with $x(t)$=1) from a different starting point: $x(t), y(t), z(t)$

can compute derivatives as finite differences. For example a 2-dimensional Laplacian is computed at each grid point as

$$\nabla u^2_{i,j} = \frac{\partial^2 u_{i,j}}{\partial x^2} + \frac{\partial^2 u_{i,j}}{\partial y^2} \tag{9.64}$$

$$= \frac{u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2 \cdot u_{i,j} + u_{i,j-1}}{h^2} = 0 \tag{9.65}$$

9.2.1 Relaxation Methods

Relaxation method are used to find the static equilibrium solution of complex differential equations with boundary conditions.

The differential equation is placed on a grid of points that covers the range of the integration. A trial solution, which does not satisfy the differential equation, is used as a starting point. The residuals are as indicators of how to correct the trial solution into successively closer agreement with the finite difference equations.

For example, consider the Laplacian, $\nabla^2 U = 0$, where U is usually electrostatic or gravitational potential. In this case, the new value of U can be derived at iteration k from the residual, $U^{k+1} = \nabla^2 U$.

The boundary conditions must be set $(u_{i,1}, u_{i,J}, u_{1,j}, u_{I,j})$ and then we begin with an estimate of the solution, $u^k_{i,j}$, at iteration k.

$$u^{k+1}_{i,j} = \frac{u^k_{i-1,j} + u^k_{i,j-1} + u^k_{i+1,j} + u^k_{i,j+1}}{4} \tag{9.66}$$

For example, Poisson's equation, for the electrostatic potential is written

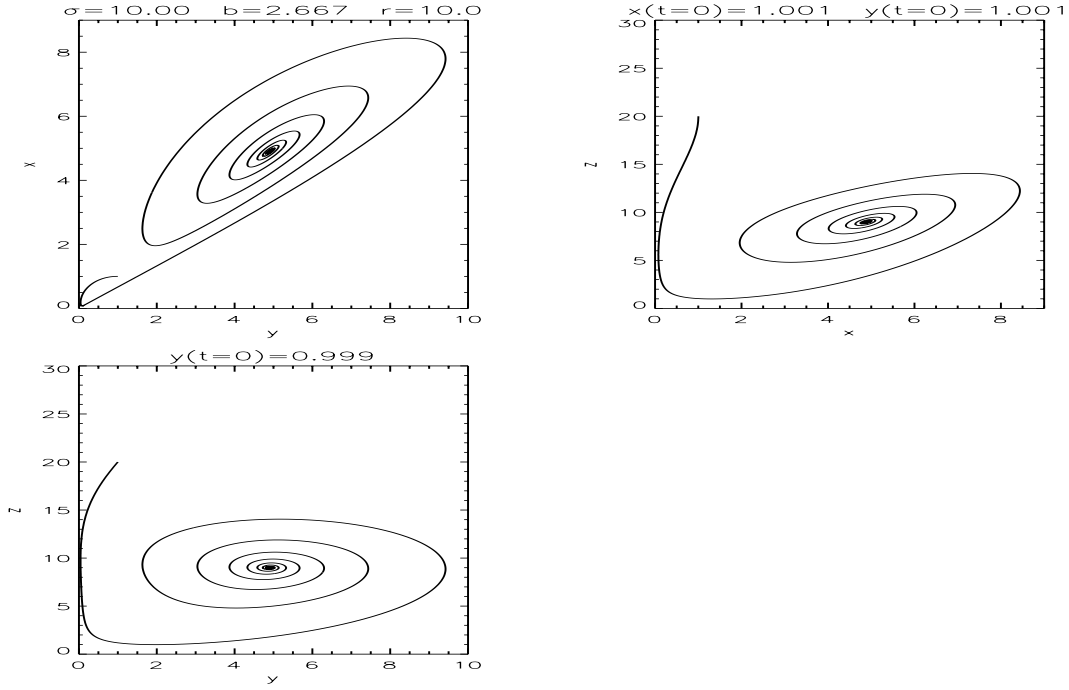


Figure 9.6: Integration of the Lorentz differential equations, (σ=10, b=8/3, and r=10 with $x(t)$=1) from a different starting point: $x(y), z(x), z(y)$

$$\nabla^2 U(r) = \frac{-\rho(\vec{r})}{\epsilon_0} \tag{9.67}$$

which can be written in finite difference form as

$$u^k_{i+1,j} + u^k_{i-1,j} + u^k_{i,j+1} + u^k_{i,j-1} - 4 \cdot u_{i,j} - \frac{\rho(x,y)}{4 \cdot \epsilon_0} = 0 \tag{9.68}$$

If $u^k_{i,j}$ is an estimate of the solution, the residuals (non-zeroness) at any given step can be used to create a new estimate of the solution

$$u^{k+1}_{i,j} = \frac{u^k_{i-1,j} + u^k_{i,j-1} + u^k_{i+1,j} + u^k_{i,j+1}}{4} + \frac{\rho(x,y)}{4 \cdot \epsilon_0} \tag{9.69}$$

In general, to solve the equation with both a Poisson's form and 1$^{\text{st}}$ derivatives we could write the equation

$$a_{i,j} \cdot u^k_{i+1,j} + b_{i,j} \cdot u^k_{i-1,j} + c_{i,j} \cdot u^k_{i,j+1} + d_{i,j} \cdot u^k_{i,j-1} + e_{i,j} \cdot u_{i,j} = f_{j,l} \tag{9.70}$$

which has the residual

$$r^k_{i,j} = a_{i,j} \cdot u^k_{i+1,j} + b_{i,j} \cdot u^k_{i-1,j} + c_{i,j} \cdot u^k_{i,j+1} + d_{i,j} \cdot u^k_{i,j-1} + e_{i,j} \cdot u^k_{i,j} - f_{i,j} \tag{9.71}$$

to achieve a zero residual, the value of $u^{k+1}_{i,j}$ should be set to

$$u^{k+1}_{i,j} = \frac{1}{e_{i,j}} \cdot \left(f_{i,j} - a_{i,j} \cdot u^k_{i+1,j} - b_{i,j} \cdot u^k_{i-1,j} - c_{i,j} \cdot u^k_{i,j+1} - d_{i,j} \cdot u^k_{i,j-1} \right) \tag{9.72}$$

but this can be related to the original value and the residual

$$u^{k+1}_{i,j} = \frac{1}{e_{i,j}} \cdot \left(e_{i,j} \cdot u^k_{i,j} - r^k_{i,j} \right) \tag{9.73}$$

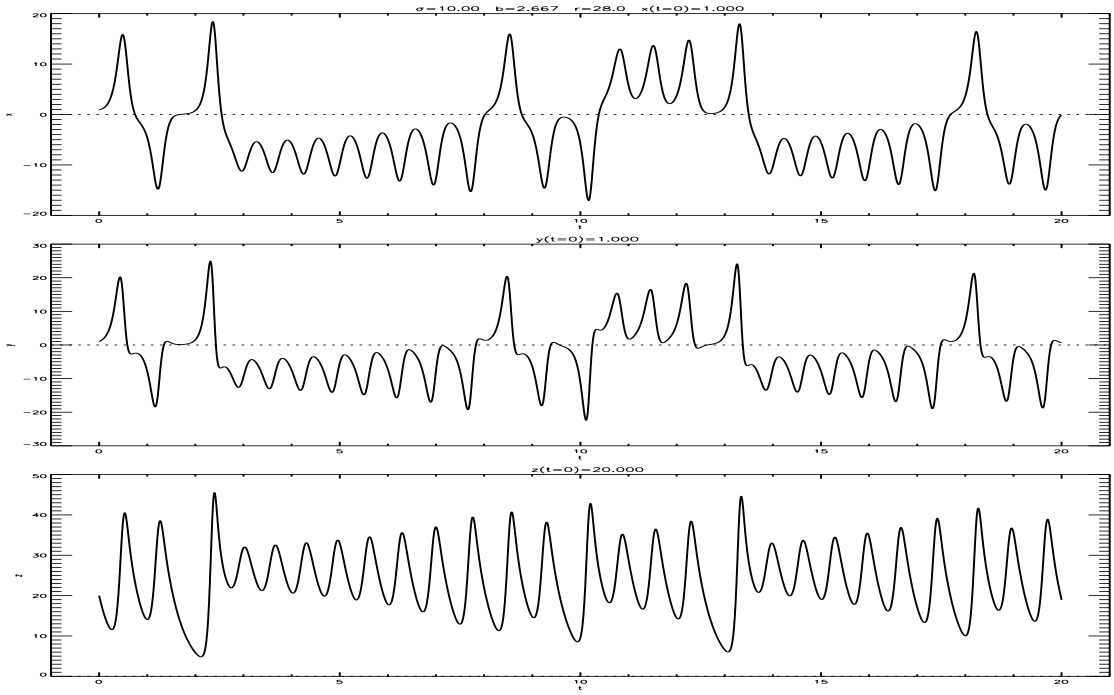


Figure 9.7: Integration of the Lorentz differential equations, (σ=10, b=8/3, and r=28 with $x(t)$=1): $x(t), y(t), z(t)$

or

$$u_{i,j}^{k+1} = u_{i,j}^{k} - \frac{r_{i,j}^{k}}{e_{i,j}} \tag{9.74}$$

9.2.2 Over-relaxation

Giving extra weight (anticipating future corrections) to the residuals can speed convergence. This is a tradeoff between stability of the solution and speed of convergence.

$$u_{i,j}^{k+1} = u_{i,j}^{k} \quad - \quad \omega \cdot \frac{r_{i,j}^{k}}{e_{i,j}} \tag{9.75}$$

where $1 \leq w \leq 2$. The value of w is determined during the iterations by the size of the residuals.

Another practical point concerns the order in which mesh points are processed. Separating the odd and even (like the black and white squares of a chess board) will stabilize the solutions. Inspection of Eqn. 9.71 will illustrate why this is so.

9.2.3 Example of Relaxation with the Thermal Diffusion Equation

In "Mathematical Methods in the Physical Sciences" by Mary Boas, (Wiley, 1966, pg. 631) an analytic solution to a rectangular plate of x dimension of 10 cm and y dimension of 30 cm. The solution is given by the steady state thermal diffusion equation

$$\nabla^2 T(x, y) = 0 \tag{9.76}$$

with the following boundary conditions:

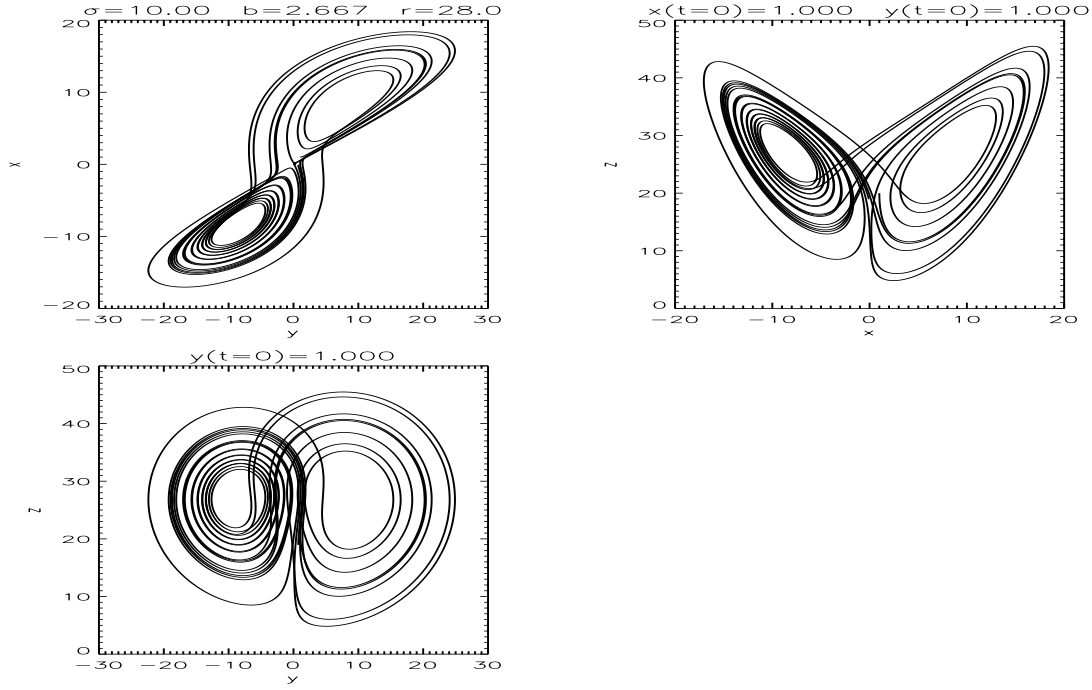


Figure 9.8: Integration of the Lorentz differential equations, (σ=10, b=8/3, and r=28 with $x(t)$=1): $x(y), z(x), z(y)$

analytic	numerical
$T(x, y = 0) = 100$	$T(i, 1) = 100$
$T(x, y = 30) = 0$	$T(i, Ny) = 0$
$T(x = 0, y) = 0$	$T(1, j) = 0$
$T(x = 10, y) = 0$	$T(Nx, j) = 0$

The analytic result was given in Boas as

$$T(x,y) = \sum_{oddn} \frac{400}{n\pi(1-e^{-6n\pi})} \left[e^{-n\pi y/10} - e^{-6n\pi} \cdot e^{n\pi y/10}\right] \cdot \sin(k\pi \cdot x/10) \tag{9.77}$$

The analytic solution must be computed with a large value of n. A computation with $n = 51$ is sufficient for single precision results. The function looks like

The relaxation method can be used to compute this function for a grid of $x = (i-1) \cdot 10/(N_x - 1)$ for $1 \le i < N_x$ and $y = (j-1) \cdot 30/(N_y - 1)$ for $1 \le j < N_y$. Note that the numeric equation will require $N_y/N_x = 3$ for an evenly spaced grid. Alternatively, one could weigh the derivatives in y differently from derivatives in x, but this is more complex of an algorithm.

A first guess of $T(i,j) = 100 \cdot e^{-j \cdot \pi/Ny}$ was used as a starting point with $N_x = 60$ and $N_y = 180$. The relaxation was computed on alternate (checkerboard pattern) grid points and used an over-relaxation parameter of $w = 1.1$ until iteration 100 and then used $w = 1.5$. The single precision calculation converged to within 1% within 100 iterations. A reasonable test for convergence is based on

$$\chi^2 = \frac{1}{N_x - 2} \cdot \frac{1}{N_y - 2} \cdot \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} r(i,j)^2 \tag{9.78}$$

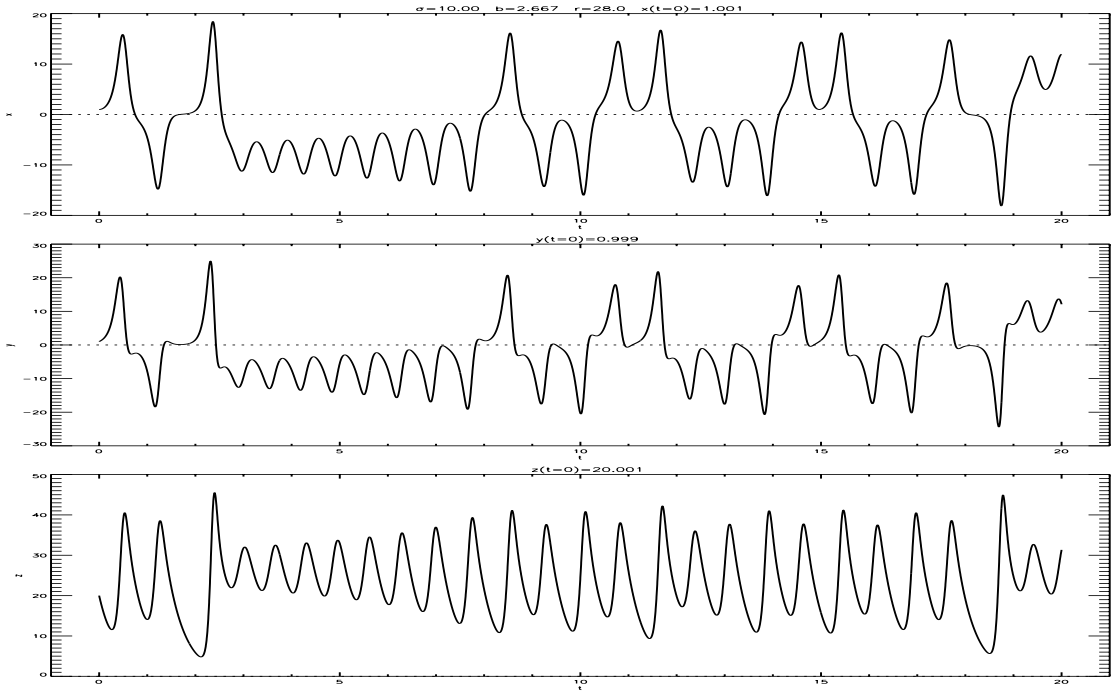


Figure 9.9: Integration of the Lorentz differential equations, (σ=10, b=8/3, and r=28 with $x(t)$=1) with a different starting point: $x(t), y(t), z(t)$

9.2.4 Example of Relaxation using Dynamics Model

As you will discover soon enough, there are only a few uses for a dissertation topic. I once heard a University Microfilm International (UMI) representative state that a "best seller" of a disertation was a few copies. Sometimes a dissertation sells 10's of copies, but that is most likely someone forcing their students to buy the dissertation for use as a textbook in a class. In this section I will use my dissertaion topic as an example of how very complex systems can be modelled via over-relaxation methods. In Barnet et al. [1992, 1990], a zonally averaged (averaged over longitude) seasonal model of Saturn was solved by simultaneous over-relaxation methods. The equilibrium temperature, T_e^n, are a set of N complex Fourier series coefficients related to the time dependent solar flux, including shadowing and scattering from the rings. In the first fiqure we show the geometry for the calculations. Sunlight is absorbed by the rings and reflected from the rings.

If Saturn has no rings, the solar insolation would vary due to the tilt and orbit of the planet. In Fig. 9.18 the solar insolation as a function of latitude and season is shown for a planet at the Saturnian distance (9.6 AU) and obliquity (27°).

The rings of Saturn are complex with transparent gaps and varying optical depth. A model of the insolation with seven distinct regions corresponding to the major ring elements is shown in Fig. 9.19. The difference of Fig. 9.18 and Fig. 9.19 is shown in Fig. 9.20

Finally, the ring scattering term is shown in Fig. 9.21. Notice that it is of similar magnitude as the ring shadowing affect but it is out of phase, seasonally. This term amplifies the insolation modulation by the rings.

Putting all the terms together yields a seasonal function that is quite complex. A Fourier series was employed at each latitude of Saturn. In Fig. 9.22 the equilibrium temperature, T_e, is shown for a Saturnian latitude of 35°. The equilibrium temperature is simply the solar radiance represented in terms of the temperature a Planck function would have at the distance of Saturn. Therefore, it is derived from the figures above and the solar insolation. The individual effects of the A, B, and C rings of Saturn can easily be seen in this figure.

Also shown is a 7 term Fourier series fit to $T_e(t)$. While the series does not exactly match the computation,

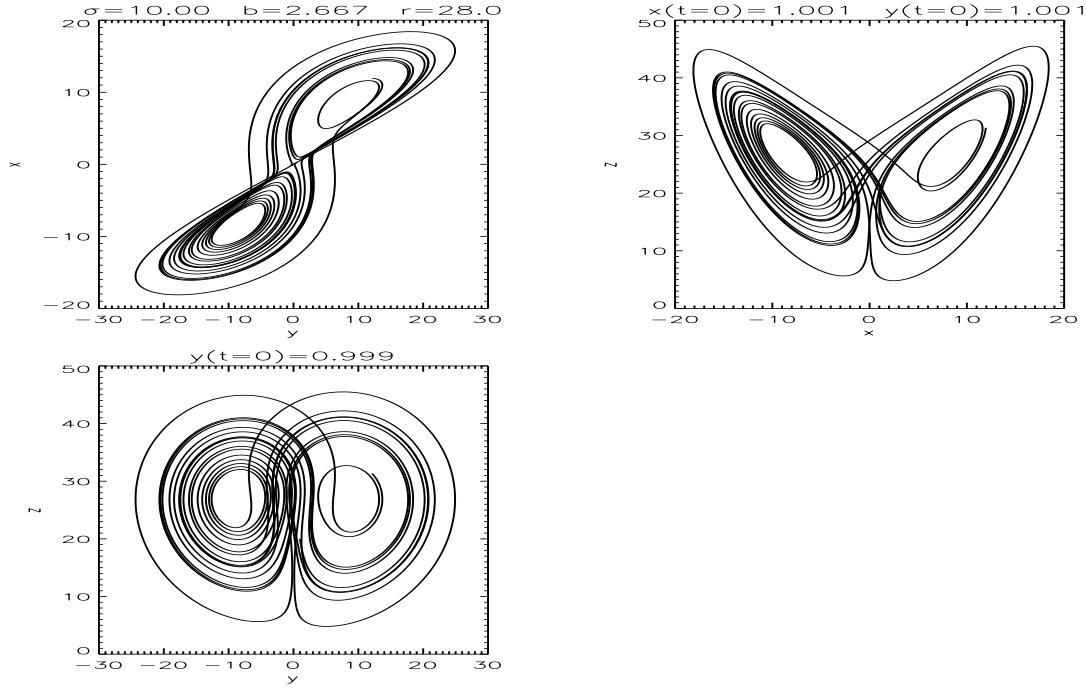


Figure 9.10: Integration of the Lorentz differential equations, (σ=10, b=8/3, and r=28 with $x(t)$=1) with a different starting point: $x(y), z(x), z(y)$

we will find that Saturn's atmosphere acts like a low pass filter and only the first few Fourier series terms are needed.

The equations of motion, conservation of mass, and conservation of energy can be written in terms of a stream function operator,

$$v_{i,j} \equiv \frac{\partial \psi}{\partial \varphi} \tag{9.79}$$

$$w_{i,j} \equiv \frac{\partial \psi}{\partial z} \tag{9.80}$$

and a single elliptic differential equation:

$$\begin{aligned} \frac{\cos(\varphi)}{\rho_0} \cdot \frac{\partial}{\partial \varphi}\left(\frac{1}{\cos(\varphi)}\frac{\partial \psi^n}{\partial \varphi}\right) \quad + \quad \frac{4\Omega^2 a^2 \sin^2(\varphi) D_r^n}{N^2}\frac{\partial}{\partial z}\left(\frac{1}{D_f^n \rho_0}\frac{\partial \psi^n}{\partial z}\right) \\ = \quad \frac{aR_g \cos(\varphi)}{N^2 \tau_r H} \cdot \frac{\partial T_e^n}{\partial \varphi} \quad + \quad \frac{2\Omega a^2 \sin(\varphi)\cos(\varphi) D_r^n}{N^2}\frac{\partial}{\partial z}\left(\frac{U_e^n}{\tau_f D_f^n}\right) \end{aligned} \tag{9.81}$$

A scaled normalized form of the equation is produced and a grid with 109 meridional points in $\sin(\varphi)$ and 80 height points in $\log(P/P_0)$ is constructed. The value of the stream function at the poles ($\varphi = \pm\pi$) and upper boundary ($P = 0.1$) is forced to zero, which is a statement that the meridional winds are zero at the poles and the vertical wind is zero at the upper boundary. The stream function at the lower boundary ($P_0 = 1000$) is specified using Eqn. 9.82 and is given by

$$\frac{\partial \psi^0(\varphi, P_0)}{\partial P} \quad = \quad \frac{\rho_0}{2\Omega a \tan(\varphi) \cdot \tau_f} \cdot \left[U^0(\varphi, P_0) - U_e^0(\varphi, P_0)\right] \qquad \textbf{and}$$

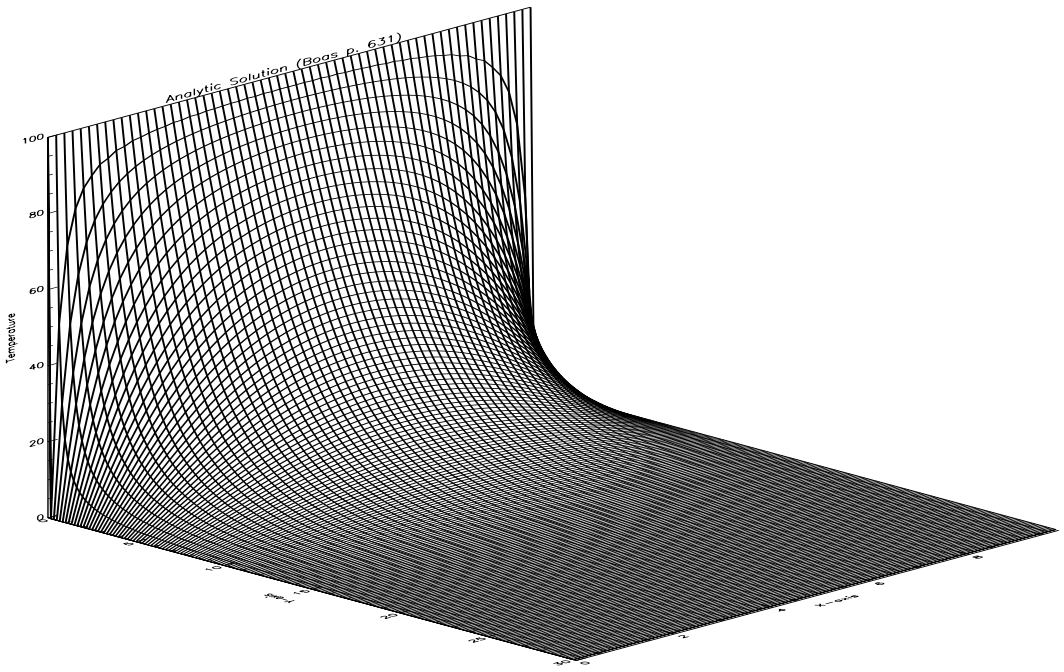


Figure 9.11: Analytic solution of a the steady state temperature of a plate

$$\frac{\partial \psi^n(\varphi, P_0)}{\partial P} = 0 \qquad \textbf{for} \qquad \text{for } n > 0. \tag{9.82}$$

The annually averaged zonal wind, $U^0(\varphi, P_0)$ and the equilibrium wind, $U_e^0(\varphi, P_0)$, must be specified at the lower boundary. This boundary condition is an assumption that the zonal winds are driven from below the lower boundary and are steady, which is supported by historical observations (see Fig. 10 of paper).

The right hand side of Eqn. 9.82 is forced to zero at $\varphi = 0$ to avoid a singularity. In the case where $U_e(\varphi = 0, P)$ is zero, the model cannot reproduce the equatorial wind at $U^0(\varphi = 0, P = P_0)$ and the model responds as though the wind goes to zero rapidly towards the equator, which is known to be false for Saturn. The equatorial wind is presumed to be due to a non-linear convergence that cannot be modeled by this linearized model.

Most of the arguments were precomputed into arrays so that the iterations would take place in less time. The heart of the FORTRAN code to solve Eqn. 9.81 looked like

```
      chi_old = 0.0
      nconverg = 0
      errcnt = 0

      do 1000 its=1,nit

c                            d {          d{psi} }
c         op1 = Ai(1,iy)* --- { Ai(2,iy) ------ }
c                          diy {           diy   }
c
c                             d {          d{psi} }
c         op2 = Bi(1,ip) * --- { Bi(2,ip) ------ }
c                           dip {           dip   }
```

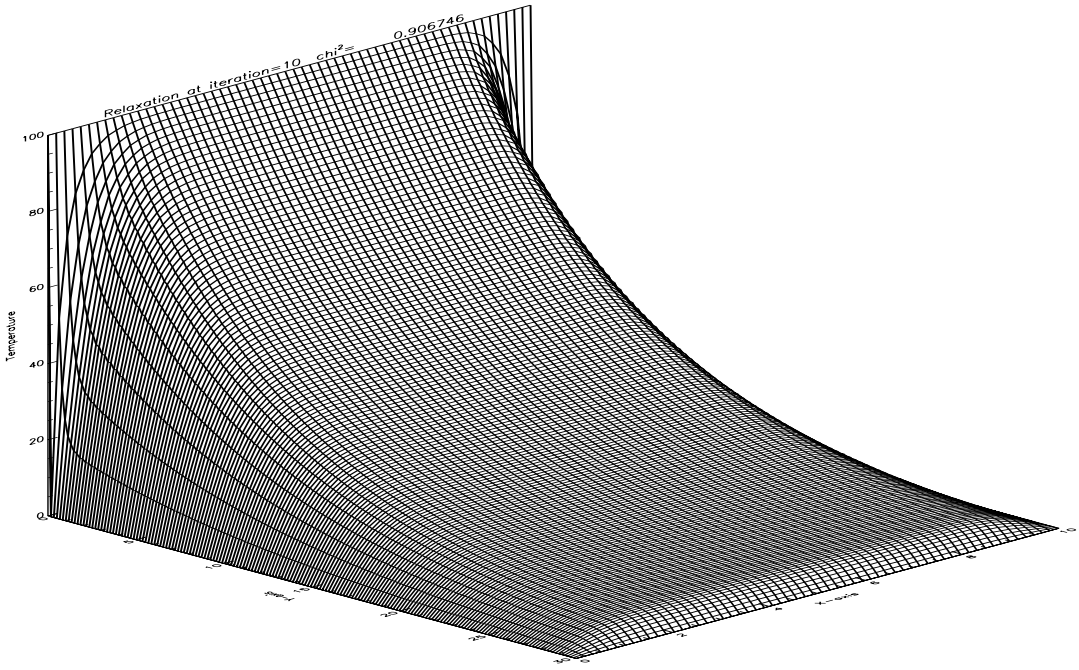


Figure 9.12: Temperature of rectangular plate using relaxation: iteration = 10

```
chi = 0.0
resmax = 0.0
nover = 0

if(mod(its,2).eq.0) then
   iy1 = 2
   iy2 = ny-1
  iy12 = 1
   ip1 = 2
   ip2 = np-1
  ip12 = 1
else
   iy1 = ny-1
   iy2 = 2
  iy12 = -1
   ip1 = np-1
   ip2 = 2
  ip12 = -1
endif

do 1500 isquare = 0,1
  do 2000 iy = iy1,iy2,iy12
    do 2100 ip = ip1,ip2,ip12

    if(mod(iy+ip,2).eq.isquare) then
      op11 = SAi(1,iy)*(psi(iy+1,ip) - psi(iy,ip))
      op12 = SAi(2,iy)*(psi(iy-1,ip) - psi(iy,ip))
      op1 = op11 + op12
```

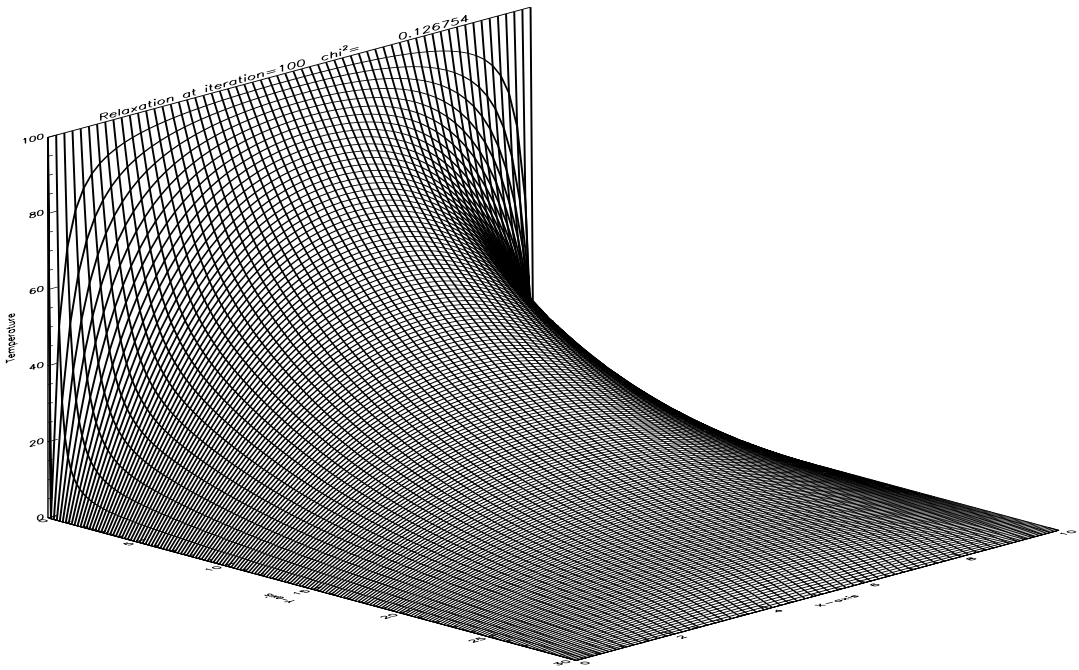


Figure 9.13: Temperature of rectangular plate using relaxation: iteration = 100

```
            if(bnd_flg.eq.2.and.ip.eq.np-1) then
              op11 = (SBi(1,ip)*dbase)*psi_bnd(iy)
            else
              op11 = SBi(1,ip)*(psi(iy,ip+1)-psi(iy,ip))
            endif

            op12 = SBi(2,ip)*(psi(iy,ip-1)-psi(iy,ip))
            op2 = sin2(iy)*(op11 + op12)

            dpsi2 = op1 + op2 + force(iy,ip)
            resid = dpsi2*Wrkblk(iy,ip)
            psi(iy,ip) = psi(iy,ip) + omcheb*resid

            x = cabs(resid)
            if(x.gt.resmax) resmax = x
            if(x.gt.1.e3) then
              chi=chi+1.e3
              nover=nover+1
            else
              chi=chi + x*x
            endif
          endif
2100      continue
          if(bnd_flg.eq.2) then
            psi(iy,np) = psi(iy,np-1) + dbase*psi_bnd(iy)
          endif
2000    continue
1500  continue
```

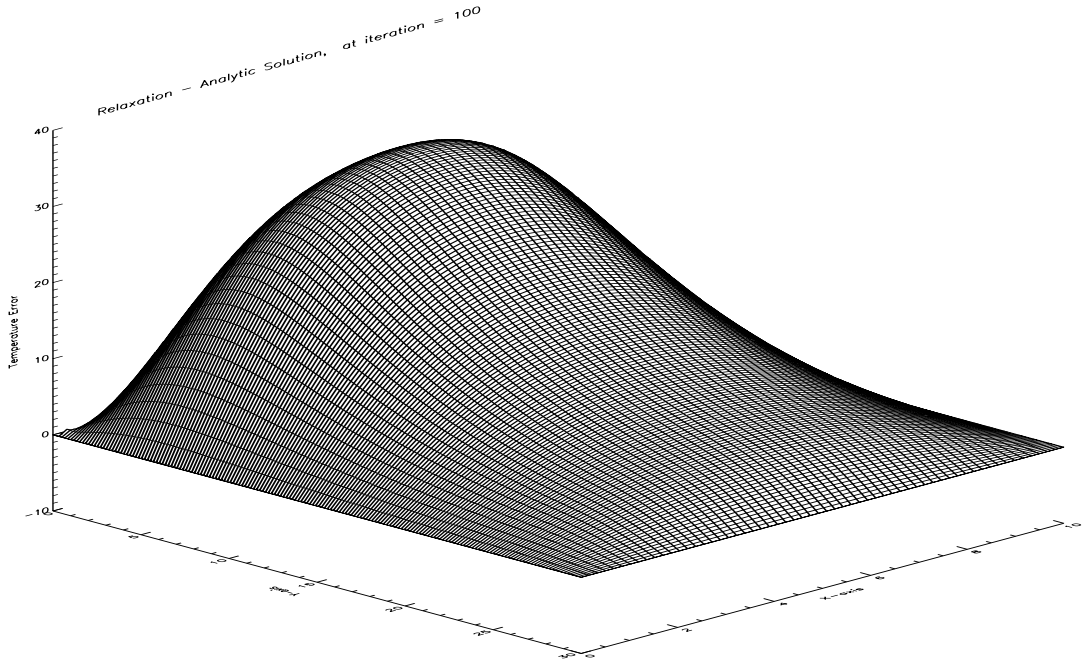


Figure 9.14: Error in Relaxation at iteration = 100

```
      chi=sqrt(chi/((ny-2)*(np-2)))

      if(nover.eq.0) then
        write(*,9010) ifr,its,chi,resmax,real(omcheb)
      else
        write(*,9020) ifr,its,chi,resmax,real(omcheb),nover
      endif

      if((mod(its,itm).eq.1).or.(its.lt.6)) then
        if(nover.eq.0) then
          write(9,9010) ifr,its,chi,resmax,real(omcheb)
        else
          write(9,9020) ifr,its,chi,resmax,real(omcheb),nover
        endif
      endif

c     adjust over-relaxation parameter
c     ================================

      if(its.eq.1) then
        x = worksor/(1.0-rho2/2.0)
        omcheb = cmplx(x,0.0)
      else
        x = worksor/(1.0-rho2*real(omcheb)/6.0)
        omcheb = cmplx(x,0.0)
      endif

      if(chi.le.threshold.and.its.gt.9) then
```

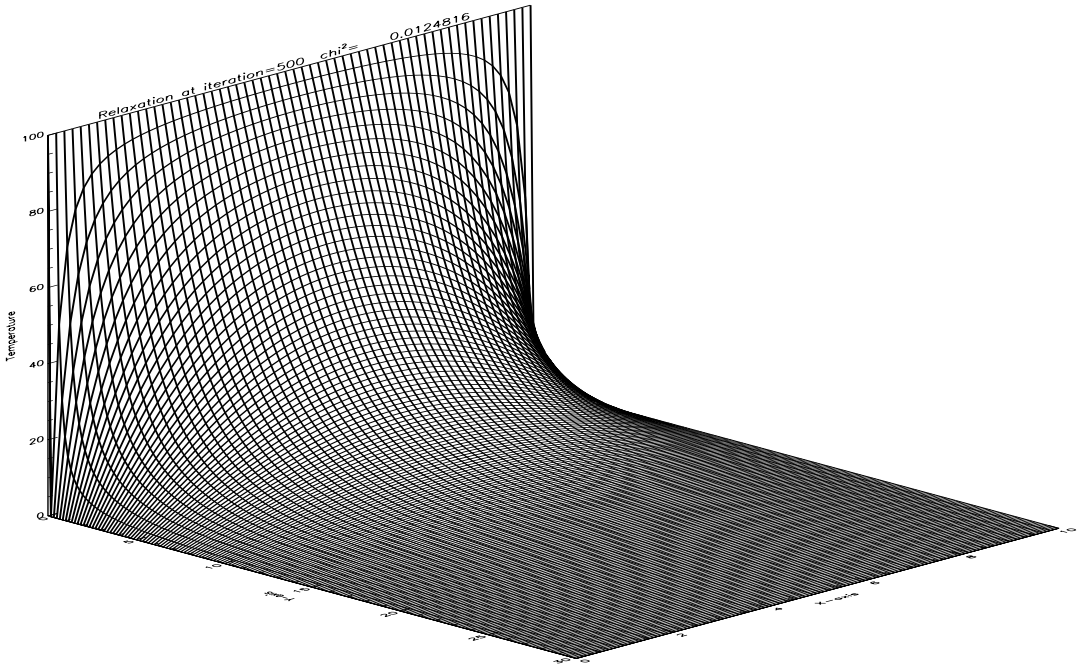


Figure 9.15: Temperature of rectangular plate using Relaxation at iteration = 500

```
  trm_mes = 'chi < threshold'
  goto 1200
endif
x = (chi-chi_old)/chi
chi_old = chi
if(x.gt.0.0.and.its.gt.2) then
  omcheb = cmplx(0.90,0.0)*omcheb
  worksor = 0.90*worksor
  errcnt = errcnt + 1
  nconverg = 0
elseif(x.gt.-0.05.and.worksor.le.omegasor) then
  worksor = 1.05*worksor
  omcheb = cmplx(1.05,0.0)*omcheb
  errcnt = 0
  nconverg = nconverg + 1
else
  errcnt = 0
  nconverg = 0
endif
if(errcnt.gt.8) then
  trm_mes = 'psi diverging'
  goto 1200
endif
if(x.gt.-0.01.and.x.lt.0.0.and.nconverg.gt.8) then
  trm_mes = 'psi is not converging'
  goto 1200
endif
```

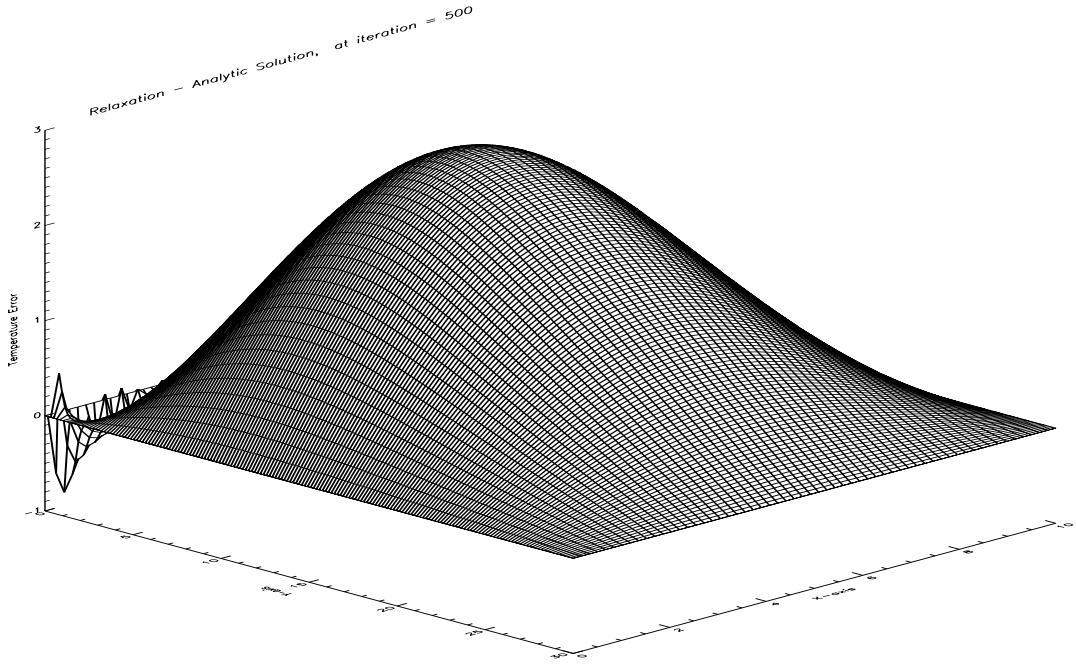


Figure 9.16: Error in Relaxation at iteration = 500

```
1000 continue
```

9.2.5 Lax-Wendroff two step time integration

Differential equations of the form

$$\frac{\partial \chi}{\partial t} + \frac{\partial F(x,y,t)}{\partial x} + \frac{\partial G(x,y,t)}{\partial y} = Q(x,y,t) \tag{9.83}$$

can be solved using the Lax-Wendroff two step time integration algorithm (Kasahara, 1966; Houghton et al., 1966).

$$\chi^{l+1} = \bar{\chi}^l - \Delta t \left(\frac{\partial F^l}{\partial x} + \frac{\partial G^l}{\partial y} \right) + \Delta t \left(\frac{Q^{l+1} + Q^l}{2} \right) \tag{9.84}$$

$$\chi^{l+2} = \chi^l - 2\Delta t \left(\frac{\partial F^{l+1}}{\partial x} + \frac{\partial G^{l+1}}{\partial y} \right) + 2\Delta t \left(Q^{l+1} \right) \tag{9.85}$$

For multi-dimensional problems the value of Q used in the equations will require averaging over the spatial dimensions. In addition, the values within the grid may need to be spatially filtered near the edges (reflected waves) and a low pass filter may need to be employed to remove high frequency instabilities. In Appendix C we discuss an detailed example of a shallow water model in detail.

The time step is calculated from the grid spacing Δx as follows:

$$\Delta t \leq \frac{\Delta x}{\sqrt{2} \cdot (U_m + V_m)} \tag{9.86}$$

where U_m is the maximum velocity in the x direction and V_m is the maximum velocity in the y direction.

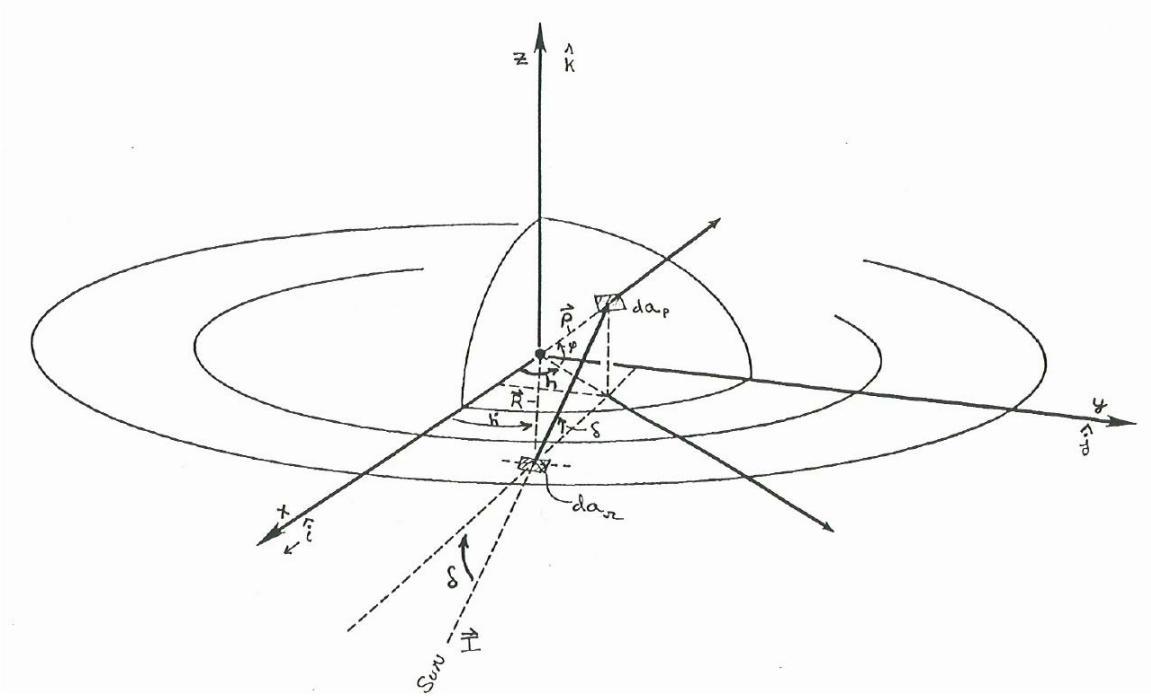


Figure 9.17: Geometry of Saturn ring computation

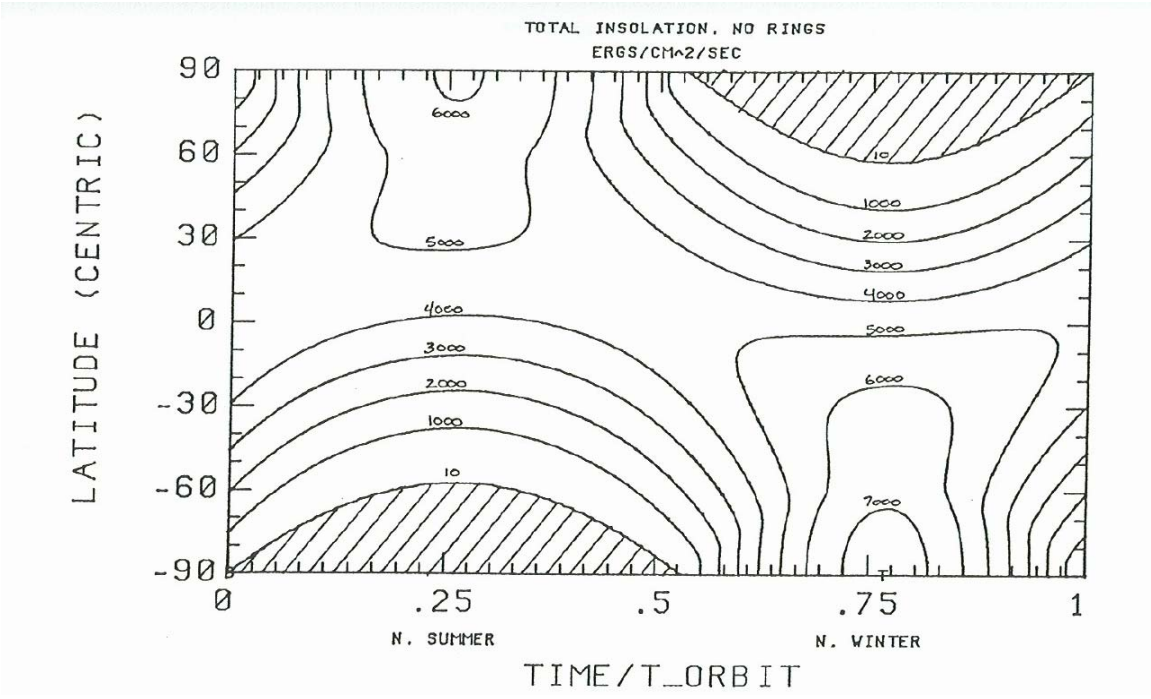


Figure 9.18: Example of the solar insolation at Saturn, if Saturn had no rings

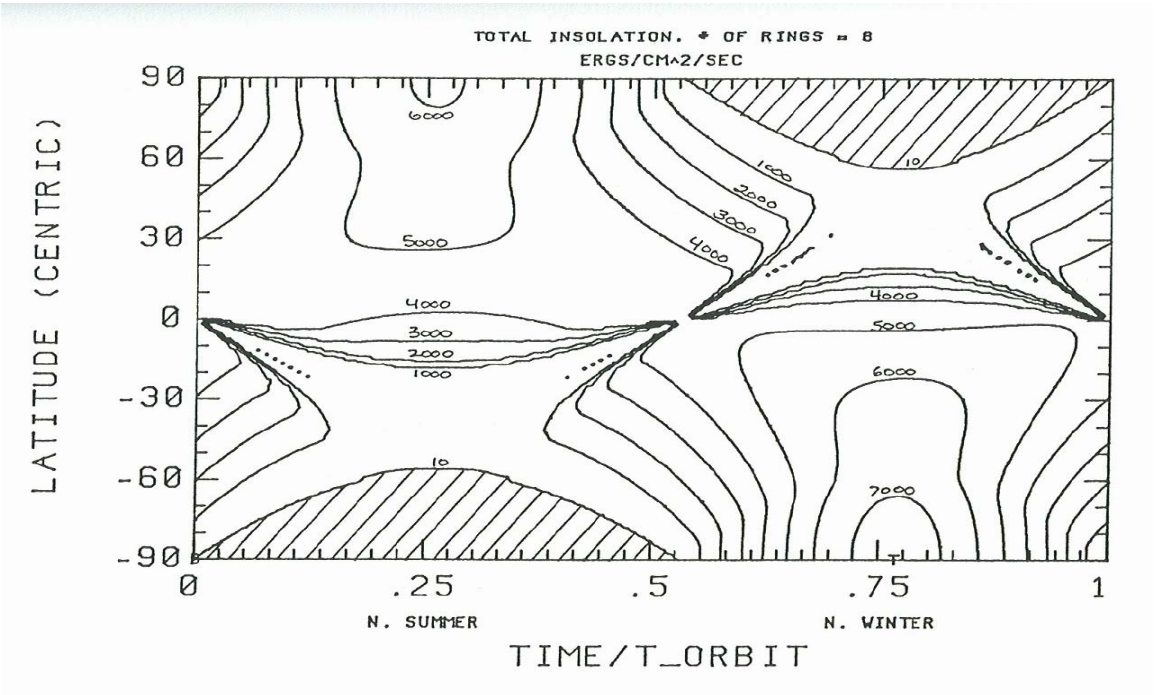


Figure 9.19: Insolation at Saturn taking the ring optical depth into account

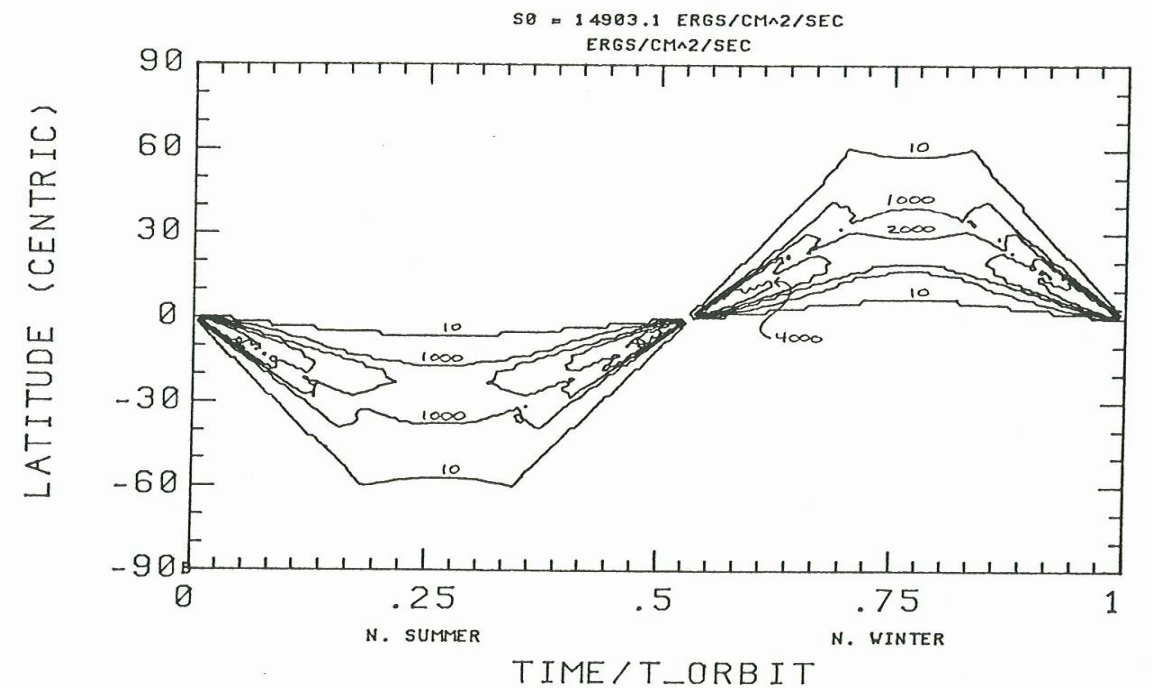


Figure 9.20: The difference in insolation due to ring absorption

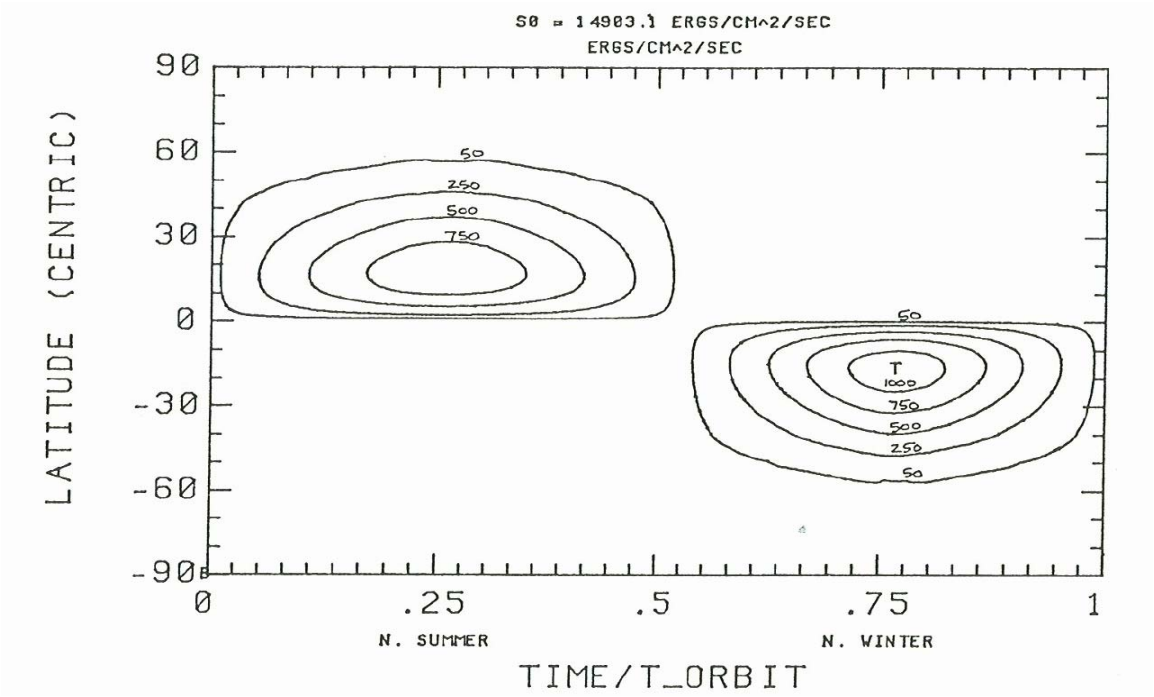


Figure 9.21: Scattering of insolation onto Saturn

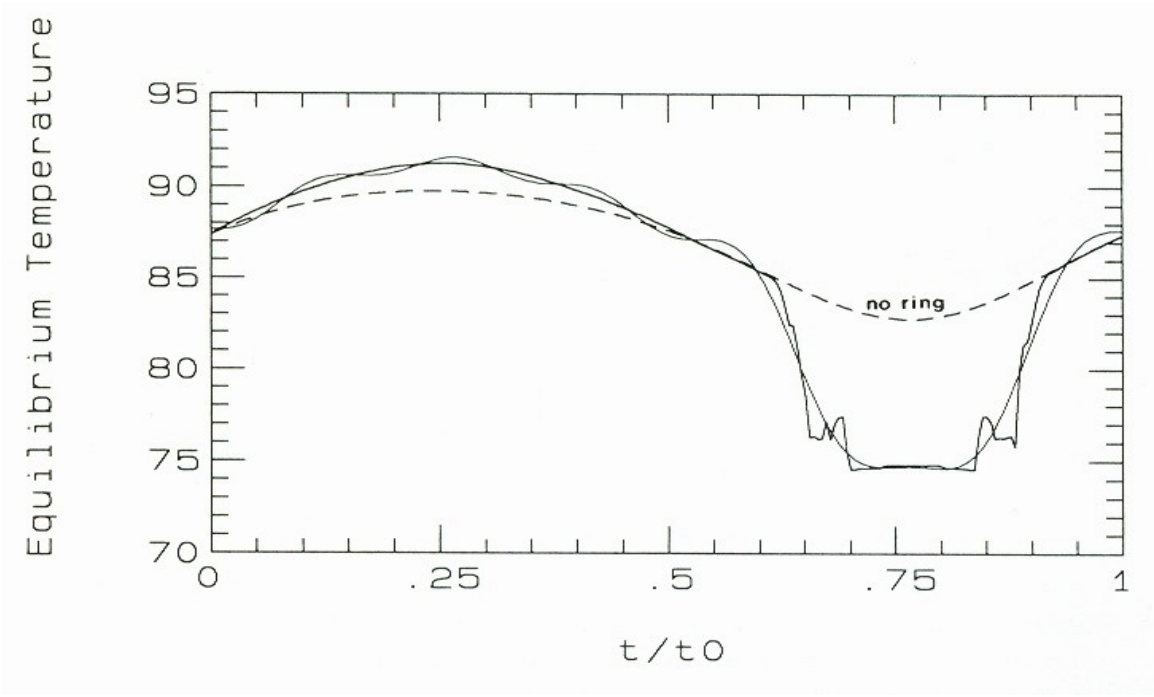


Figure 9.22: The combined effect of ring shadowing and scattering at 35° latitude.

Chapter 10

Fourier Series

10.1 Applications of Fourier Analysis

Fourier Series can be used to solve differential equations. For example:

1. Solutions of the wave equation. For example, displacement (*i.e.,* vibration) of a string of length L, where c^2 is the tension divided by the density (mass/L),

$$\frac{\partial^2 y}{\partial t^2} = c^2 \cdot \frac{\partial^2 y}{\partial x^2} \tag{10.1}$$

$$\begin{aligned} y(x,t) &= y_1(x) \cdot y_2(t) \\ &= \cos(m \cdot c \cdot t + \alpha) \cdot \sum_{k=1}^{K} a_k \cdot \cos(m \cdot x + \beta) \end{aligned} \tag{10.2}$$

   and the Fourier series coefficients are used to specify boundary conditions

   - $y(0,t) = 0 \quad \Rightarrow \quad \beta = 0$
   - $y(L,t) = 0 \quad \Rightarrow \quad m = k \cdot \pi / L$
   - $\partial y(x,0)/\partial x = 0 \quad \Rightarrow \quad \alpha = 0$
   - $y(x,0) = \sum_{k=0,K} a_k \cdot \sin(k \cdot \pi \cdot x/L)$

2. Newton's law of cooling for temperature $T(x,y,z,t)$ is derived from the first law of thermodynamics ($\delta Q = dU + \delta W$) and can be written as

$$\frac{dT(x,y,z,t)}{dt} = \frac{T(x,y,z,t) - T_e(x,y,z,t)}{\tau} \tag{10.3}$$

   where τ is a time constant and $T_e(x,y,z,t)$ is the equilibrium temperature, which can be expressed as a product of a spatial and a temporal function. The temporal function can then be expressed as a Fourier series.

$$T_e(x,y,z,t) = T_1(x,y,z) \cdot T_2(t) \tag{10.4}$$

In atmospheric thermodynamics, the total time derivative can be expanded into partials to account for advection of the temperature field.

$$\frac{dT}{dt} = \frac{\partial T}{\partial t} + \vec{U} \cdot \vec{\nabla} T = \frac{T - T_e}{\tau} \tag{10.5}$$

3. Thermal conductivity, with a thermal diffusivity, $\kappa = K/\sigma/\rho$, where K is the thermal conductivity, σ is the specific heat, and ρ is the density

$$\frac{\partial T}{\partial t} = \kappa \cdot \nabla^2 T \tag{10.6}$$

4. Holography

5. Structural analysis.

10.2 Review of Fourier Series

All functions can be represented by a sum of even and odd functions (*i.e.*, a Taylor expansion followed by separation of even and odd powers).

We will begin with a review of Fourier series, which is related to the Fourier transform. The Fourier series operates on real data. We will show in a later section that the Fourier series coefficients are equal to the Fourier transform for a real function. Therefore, we will use the Fourier series discussion to provide insight and build intuition for the Fourier transform discussions to follow. In 1807 Fourier proposed that an infinite series could represent a periodic function, $f(x)$, if:

- $f(x)$ is periodic over the interval $0 \le x \le P$, *i.e.*, $f(m \cdot P + x) = f(x)$ for any arbitrary value of m
- $f(x)$ and $f'(x)$ are piecewise continuous
- $f(x)$ is defined for each element and is single valued.

and is written as

$$\begin{aligned}
f(x) &\simeq \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cdot \cos\left(\frac{k \cdot 2\pi \cdot x}{P}\right) + \sum_{k=1}^{\infty} b_k \cdot \sin\left(\frac{k \cdot 2\pi \cdot x}{P}\right) && (10.7)\\
&= \frac{1}{2} \sum_{k=-\infty}^{\infty} c_k \cdot \exp\left(\frac{i \cdot k \cdot 2\pi \cdot x}{P}\right) && (10.8)\\
&= \frac{\Re(c_0)}{2} + \sum_{k=1}^{\infty} \Re(c_k) \cdot \cos\left(\frac{k \cdot 2\pi \cdot x}{P}\right) - \sum_{k=1}^{\infty} \Im(c_k) \cdot \sin\left(\frac{k \cdot 2\pi \cdot x}{P}\right) && (10.9)\\
&= \frac{\Re(c_0)}{2} + \sum_{k=1}^{\infty} \Re\left[c_k \cdot \exp\left(\frac{i \cdot k \cdot 2\pi \cdot x}{P}\right)\right] && (10.10)
\end{aligned}$$

where the coefficients a_k and b_k are real and the coefficients c_k are complex. The complex exponential is evaluated using Euler's identity

$$\exp(i \cdot \theta) = \cos(\theta) + i \cdot \sin(\theta) \tag{10.11}$$

The coefficients c_k are related to (a_k, b_k) as follows

$$\begin{aligned} c_k &= a_k - i \cdot b_k \quad \text{for} \quad k \geq 0 \\ &= a_k + i \cdot b_k \quad \text{for} \quad k < 0 \end{aligned} \tag{10.12}$$

therefore, and c_{-k} is the complex conjugate of c_k. The coefficients, a_n and b_n are computed using the orthogonal properties of the sine and cosine functions as follows:

$$a_k = \frac{2}{P} \int_{x=0}^{P} f(x) \cdot \cos\left(\frac{k \cdot 2\pi \cdot x}{P}\right) \cdot dx \tag{10.13}$$

$$b_k = \frac{2}{P} \int_{x=0}^{P} f(x) \cdot \sin\left(\frac{k \cdot 2\pi \cdot x}{P}\right) \cdot dx \tag{10.14}$$

and since $\cos(0) = 1$ then the first term becomes

$$a_0 = \frac{2}{P} \int_{x=0}^{P} f(x) \cdot \cos(0) \cdot dx = \frac{2}{P} \int_{x=0}^{P} f(x) \cdot dx = 2 \cdot \overline{f(x)} \tag{10.15}$$

10.3 Descretization of the Fourier Series

We can compute the trapezoidal integral of the intergral equations assuming that there are N points within the period (*i.e.*, $f(m \cdot N + i) = f(i)$, for all values, m). We will assume equally spaced sampling interval so that $x = n \cdot h$ and $P = N * h$. The endpoints of the trapezoidal integral are equal so that

$$\frac{f(0) + f(N)}{2} = f(0) \tag{10.16}$$

$$a_0 = \frac{2}{N} \sum_{n=0}^{N-1} f(n) = 2 \cdot \overline{f(n)} \tag{10.17}$$

$$\begin{aligned} a_k &= \frac{2}{N} \sum_{n=0}^{N-1} f(n) \cdot \cos\left(\frac{k \cdot 2\pi \cdot n}{N}\right) & (10.18) \\ &= \frac{2 \cdot f(0)}{N} + \frac{2}{N} \sum_{n=1}^{N-1} f(n) \cdot \cos\left(\frac{k \cdot 2\pi \cdot n}{N}\right) & (10.19) \end{aligned}$$

$$\begin{aligned} b_k &= \frac{2}{N} \sum_{n=0}^{N-1} f(n) \cdot \sin\left(\frac{k \cdot 2\pi \cdot n}{N}\right) & (10.20) \\ &= \frac{2}{N} \sum_{n=1}^{N-1} f(n) \cdot \sin\left(\frac{k \cdot 2\pi \cdot n}{N}\right) & (10.21) \end{aligned}$$

We will show in the next Chapter that the Fourier transform of a real function is simply can be used to compute the Fourier series coefficients. A Discrete Fourier Transform (DFT) or and Fast Fourier Transform (FFT) can also be used to compute the complex coefficients. For a real array, $f(n), n = 0, N-1$ the DFT equation is given by

$$c_k = 2 \cdot \mathrm{DFT}^{-1}\left(f(n)\right) = \frac{2}{N} \cdot \sum_{n=0}^{N-1} f(n) \cdot \exp\left(i \cdot 2\pi \cdot k \cdot n/N\right) \tag{10.22}$$

and the $K \leq N$ Fourier coefficients will be given by

$$a_k = \Re\left(c_k\right) \qquad \text{for} \quad k = 1, K \tag{10.23}$$

$$b_k = -\Im\left(c_k\right) \qquad \text{for} \quad k = 1, K \tag{10.24}$$

In Section B.4 an IDL program (*ftp*/four_fit.pro) can be used to compute Fourier Series coefficients by direct trapezoidal integration. Another program on the ftp site, four_cal.pro, will compute the Fourier series fit using the coefficients from four_fit.pro and compute the residual errors.

10.3.1 Example #1: Fourier series fit to a Square Wave

$$\begin{aligned} f(x) &= 1 \quad \text{for} \quad 0 < x < \frac{P}{2} \\ &= \frac{1}{2} \quad \text{for} \quad x = 0, \frac{P}{2}, P \\ &= 0 \quad \text{elsewhere} \end{aligned} \tag{10.25}$$

$$a_0 = \frac{2}{P} \int_0^P f(x) \cdot dx = \frac{2}{P} \int_0^{P/2} dx = \left. \frac{2 \cdot x}{P} \right|_0^{P/2} = 1 \tag{10.26}$$

Note that a_0 is equal to two times the average.

$$\begin{aligned} a_k &= \frac{2}{P} \int_{x=0}^P f(x) \cdot \cos\left(\frac{k \cdot 2\pi \cdot x}{P}\right) \cdot dx = \frac{2}{P} \int_{x=0}^{P/2} \cos\left(\frac{k \cdot 2\pi \cdot x}{P}\right) \cdot dx \quad + \quad 0 \\ &= \frac{2}{P} \left(\frac{P}{k \cdot 2\pi}\right) \cdot \sin\left(\frac{k \cdot 2\pi \cdot x}{P}\right) \Bigg|_0^{P/2} = \frac{1}{k \cdot \pi} \left[\sin\left(k \cdot \pi\right) - \sin\left(0\right)\right] \\ &= 0 \quad \text{for} \quad k = 1, 2, 3, 4, 5, \ldots \end{aligned} \tag{10.27}$$

Over the interval $0 \leq x < P$ the square wave function and the cosine functions are both <u>odd</u> and, therefore, the a_k terms are all zero.

$$\begin{aligned} b_k &= \frac{2}{P} \int_{x=0}^P f(x) \cdot \sin\left(\frac{k \cdot 2\pi \cdot x}{P}\right) \cdot dx = \frac{2}{P} \int_{x=0}^{P/2} \sin\left(\frac{k \cdot 2\pi \cdot x}{P}\right) \cdot dx \quad + \quad 0 \\ &= \frac{2}{P} \left(\frac{-P}{k \cdot 2\pi}\right) \cdot \cos\left(\frac{k \cdot 2\pi \cdot x}{P}\right) \Bigg|_0^{P/2} = \frac{-1}{k \cdot \pi} \left[\cos\left(k \cdot \pi\right) - \cos\left(0\right)\right] \\ &= \frac{1}{k \cdot \pi} \left[1 - \cos\left(k \cdot \pi\right)\right] \quad = \quad \frac{2}{k \cdot \pi} \quad \text{for} \quad k = 1, 3, 5, 7, \ldots \end{aligned} \tag{10.28}$$

Notice that in this example, a odd function will yield non-zero coefficients in the odd-component of the Fourier series (due to orthogonality of the sines and cosines).

We can scale the equations by an arbitrary constant and add an offset, $g(x) = d \cdot f(x) + e$ and the coefficients are easily adjusted

$$\begin{aligned} a'_0 &= d \cdot a_0 + 2 \cdot e \\ a'_k &= d \cdot a_k \\ B'_k &= d \cdot b_k \end{aligned} \tag{10.29}$$

In Fig. 10.1 the numeric routine was used with $N = 256$ points was used to determine the coefficients for the case of $g(x) = f(x) - \frac{1}{2}$. Therefore, $a'_k = 0$ and $b'_k = b_k$. In Fig. 10.2 we show the Fourier series fit to a square wave with 3, 9, and 31 terms.

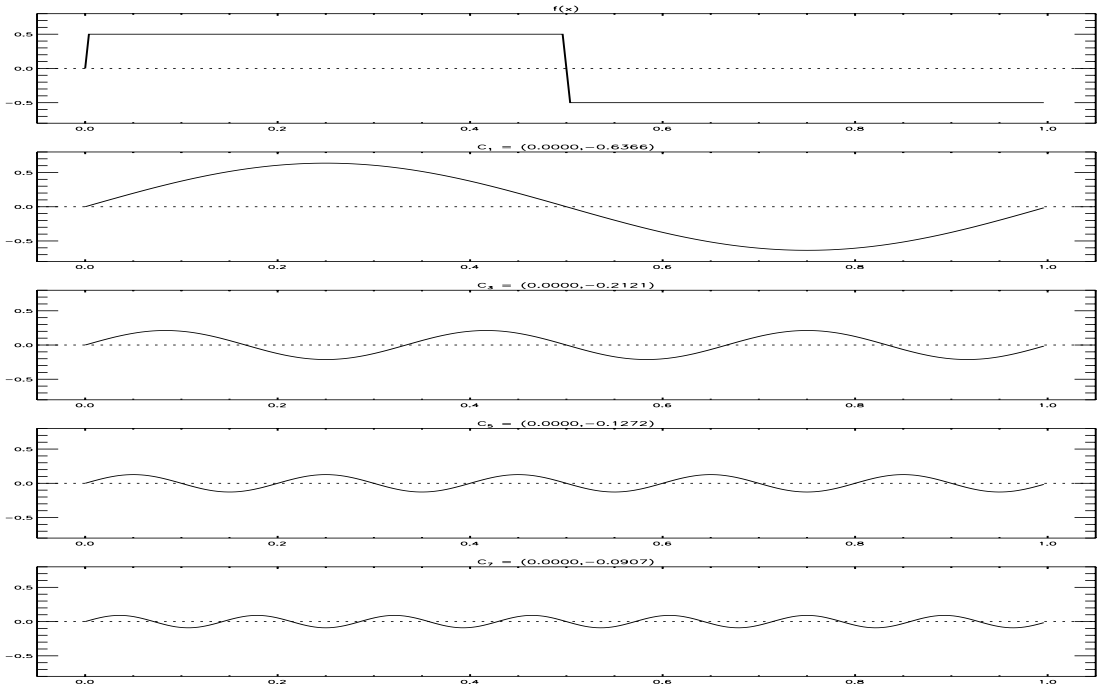


Figure 10.1: Illustration of the Fourier Series terms for a square wave

i	analytic b_k	numeric b_k
1	0.636620	0.636588
3	0.212207	0.212111
5	0.127324	0.127164
7	0.090946	0.090722

Using a square wave as an input to and amplifier or filter can quickly identify the low and high pass characteristics of a circuit.

10.3.2 Example #2: Fourier series fit to a Pulse Function

$$\begin{aligned} f(x) &= 1 \quad \text{for} \qquad \frac{P}{4} < x < \frac{P}{2} \\ &= \frac{1}{2} \quad \text{for} \quad x = \frac{P}{4}, \frac{P}{2}, P \end{aligned}$$

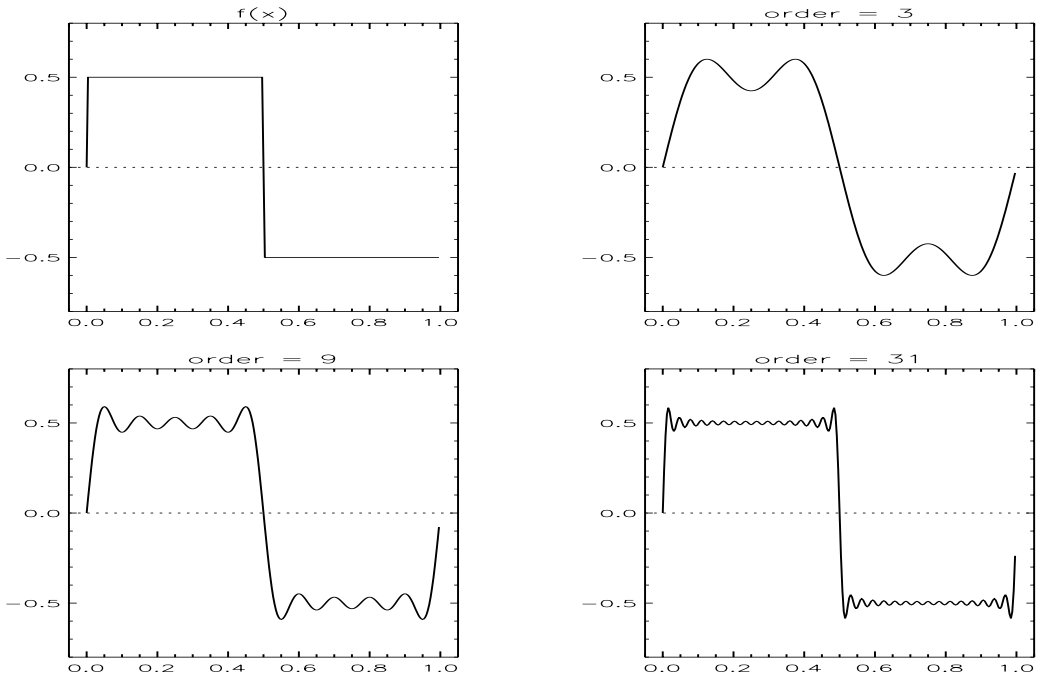


Figure 10.2: Example of a Fourier Series fit to a square wave with 3, 9, and 31 terms

$$= \quad 0 \quad \text{elsewhere} \tag{10.30}$$

$$a_0 = \frac{2}{P}\int_0^P f(x)\cdot dx \quad = \quad \frac{2}{P}\int_{P/4}^{P/2} dx = \frac{2\cdot x}{P}\Bigg|_{P/4}^{P/2} = \frac{1}{2} \tag{10.31}$$

$$\begin{aligned}
a_k &= \frac{2}{P}\int_{x=0}^{P} f(x)\cdot\cos\left(\frac{k\cdot 2\pi\cdot x}{P}\right)\cdot dx \\
&= \frac{2}{P}\int_{x=P/4}^{P/2}\cos\left(\frac{k\cdot 2\pi\cdot x}{P}\right)\cdot dx \quad + \quad 0 \\
&= \frac{2}{P}\left(\frac{P}{k\cdot 2\pi}\right)\cdot\sin\left(\frac{k\cdot 2\pi\cdot x}{P}\right)\Bigg|_{P/4}^{P/2} \\
&= \frac{1}{k\cdot\pi}\left[\sin(k\cdot\pi) - \sin\left(\frac{k\cdot\pi}{2}\right)\right] \quad = \quad \frac{-1}{k\cdot\pi}\cdot\sin\left(\frac{k\cdot\pi}{2}\right) \\
&= \frac{-1}{k\cdot\pi} \quad \text{for} \quad k = 1,5,9,13,\ldots \\
&= \frac{+1}{k\cdot\pi} \quad \text{for} \quad k = 3,7,11,15,\ldots \\
&= 0 \qquad \text{for} \quad k = 2,4,6,8,10,\ldots
\end{aligned} \tag{10.32}$$

$$
\begin{aligned}
b_k &= \frac{2}{P}\int_{x=0}^{P} f(x)\cdot\sin\left(\frac{k\cdot 2\pi\cdot x}{P}\right)\cdot dx = \frac{2}{P}\int_{x=P/4}^{P/2}\sin\left(\frac{k\cdot 2\pi\cdot x}{P}\right)\cdot dx \quad + \quad 0 \\
&= \frac{2}{P}\left(\frac{-P}{k\cdot 2\pi}\right)\cdot\cos\left(\frac{k\cdot 2\pi\cdot x}{P}\right)\Bigg|_{P/4}^{P/2} \\
&= \frac{-1}{k\cdot\pi}\left[\cos(k\cdot\pi)-\cos\left(\frac{k\cdot\pi}{2}\right)\right] \quad = \quad \frac{1}{k\cdot\pi}\left[1-\cos(k\cdot\pi)\right] \\
&= \frac{1}{k\cdot\pi} \quad \text{for} \quad k=1,3,5,7,\ldots \\
&= \frac{-2}{k\cdot\pi} \quad \text{for} \quad k=2,6,10,14,\ldots \\
&= 0 \qquad \text{for} \quad k=4,8,12,16,\ldots
\end{aligned}
\tag{10.33}
$$

	analytic		numeric	
i	a_k	b_k	a_k	b_k
0	-0.250000	n/a	-0.250000	n/a
1	0.318310	0.318310	-0.318294	0.318294
2	0.000000	-0.318310	0.000000	-0.318246
3	-0.212207	0.106103	0.106055	0.106055
4	0.000000	0.000000	0.000000	0.000000
5	0.063662	0.063662	-0.063582	0.063582
6	0.000000	-0.106103	0.000000	-0.105911
7	-0.090946	0.045473	0.045361	0.045361
8	0.000000	0.000000	0.000000	0.000000
9	0.035368	0.035368	-0.035224	0.035224
10	0.000000	-0.063662	0.000000	-0.063342
11	-0.057875	0.028937	0.028761	0.028761
12	0.000000	0.000000	0.000000	0.000000
13	0.024485	0.024485	-0.024277	0.024277
14	0.000000	-0.045473	0.000000	-0.045025
15	-0.042441	0.021221	0.020980	0.020980

10.3.3 Example #3: Fourier series fit to a Triangle Wave

We define $f(x)$ to be an even function over the periodic interval. For example, this could be the boundary condition for a guitar string plucked from the middle.

$$
\begin{aligned}
f(x) &= \frac{2\cdot x}{P} \quad \text{for} \quad 0\le x\le\frac{P}{2} \\
&= \frac{2}{P}(P-x) \quad \text{for} \quad \frac{P}{2}\le x\le P
\end{aligned}
\tag{10.34}
$$

$$
a_0 = \frac{2}{P}\int_0^P f(x)\cdot dx = \frac{4}{P^2}\left[\int_0^{P/2} x\cdot dx + \int_{P/2}^{P}(P-x)\cdot dx\right]
$$

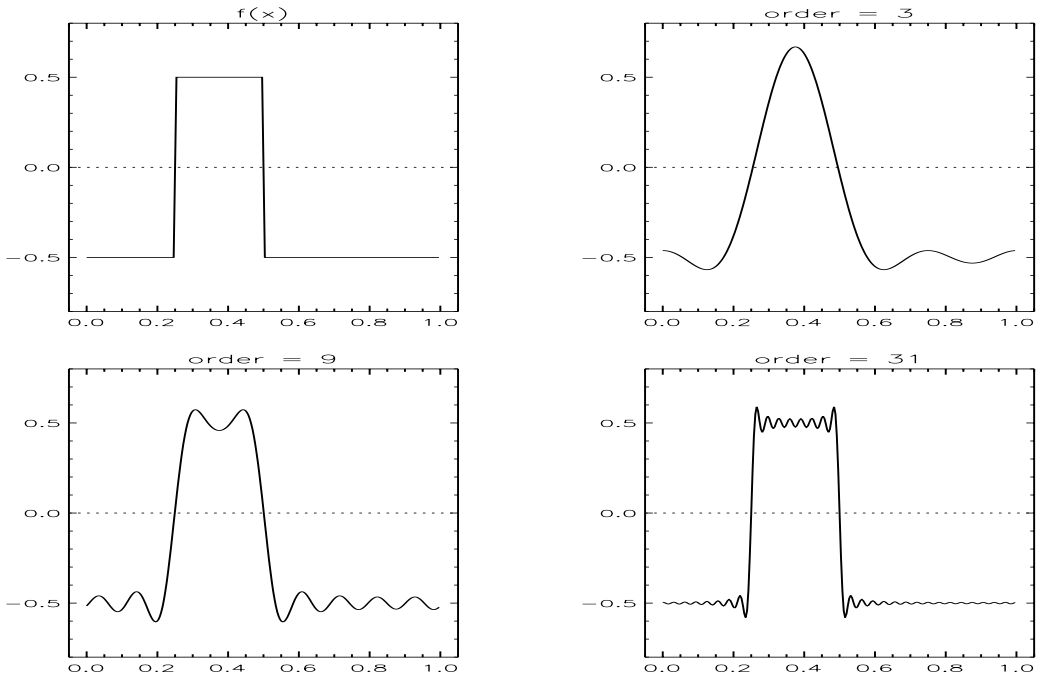


Figure 10.3: Example of a Fourier series fit to a pulse function with 3, 9, and 31 terms.

$$= \frac{4}{P^2}\left[\frac{x^2}{2}\bigg|_0^{P/2} + \frac{P^2}{2} - \frac{x^2}{2}\bigg|_{P/2}^{P}\right] = 1 \tag{10.35}$$

$$\begin{aligned} a_k &= \frac{2}{P}\int_{x=0}^{P} f(x)\cdot\cos\left(\frac{k\cdot 2\pi\cdot x}{P}\right)\cdot dx \\ &= \frac{4}{P^2}\int_{x=0}^{P/2} x\cdot\cos\left(\frac{k\cdot 2\pi\cdot x}{P}\right)\cdot dx \quad + \quad \frac{4}{P^2}\int_{x=P/2}^{P}(P-x)\cdot\cos\left(\frac{k\cdot 2\pi\cdot x}{P}\right)\cdot dx \\ &= \frac{-16}{4\pi^2\cdot k^2} \quad \text{for} \quad k = 1,3,5,7,\ldots \end{aligned} \tag{10.36}$$

$$b_k = \frac{2}{P}\int_{x=0}^{P} f(x)\cdot\sin\left(\frac{k\cdot 2\pi\cdot x}{P}\right)\cdot dx \quad = \quad 0 \tag{10.37}$$

which we can solve by observation. The integral of our even function, $f(x)$, with an odd function, sin(), over a full period is zero.

i	analytic a_k	numeric a_k
1	-0.40529	-0.4053
3	-0.04504	-0.0451
5	-0.01622	-0.0162
7	-0.00828	-0.0083

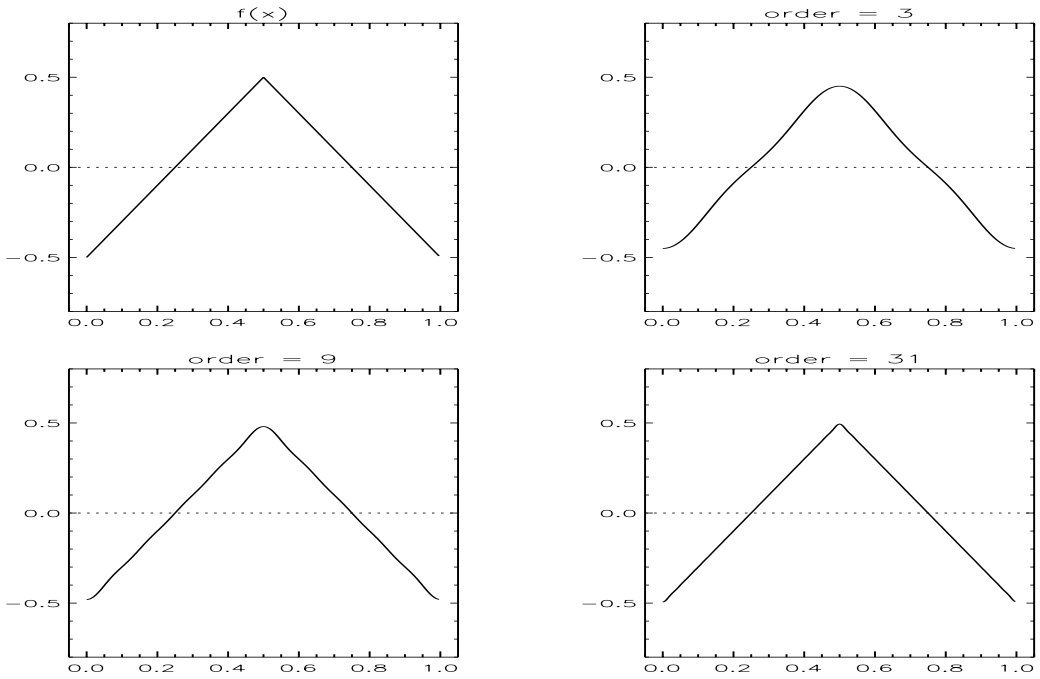


Figure 10.4: Example of a Fourier series fit to a triangular function with 3, 9, and 31 terms.

10.3.4 Example #4: Fourier series fit to a Clipped Sine Wave

We define $f(x)$ to be an odd function over the periodic interval. The function is defined as a sine wave; however, its value is not allowed to exceed a certain threshold. This is the "fuzz" tone one hears when a transistor amplifier distorts the signal (*i.e.*, the volume is too high).

$$f(x) = \text{MAX}\left[\text{MIN}\left(\sin\left(\frac{2\pi \cdot x}{P}\right), 0.6\right), -0.6\right] \tag{10.38}$$

i	numeric b_k
1	+0.715242
3	+0.130379
5	-0.003130
7	-0.020293
9	-0.006948

The 3rd *harmonic* is roughly 7% of the fundamental frequency, which the human ear interprets as a "fuzz" sound. Notice that there are no even harmonics (since this is an odd function over the period). The human ear distorts loud sounds in progressive harmonics, including even and odd harmonics. The old vacuum tube amplifiers also tend to distort with all harmonics which is why blues musicians like the "tone" of the old "6L6" tube amplifiers.

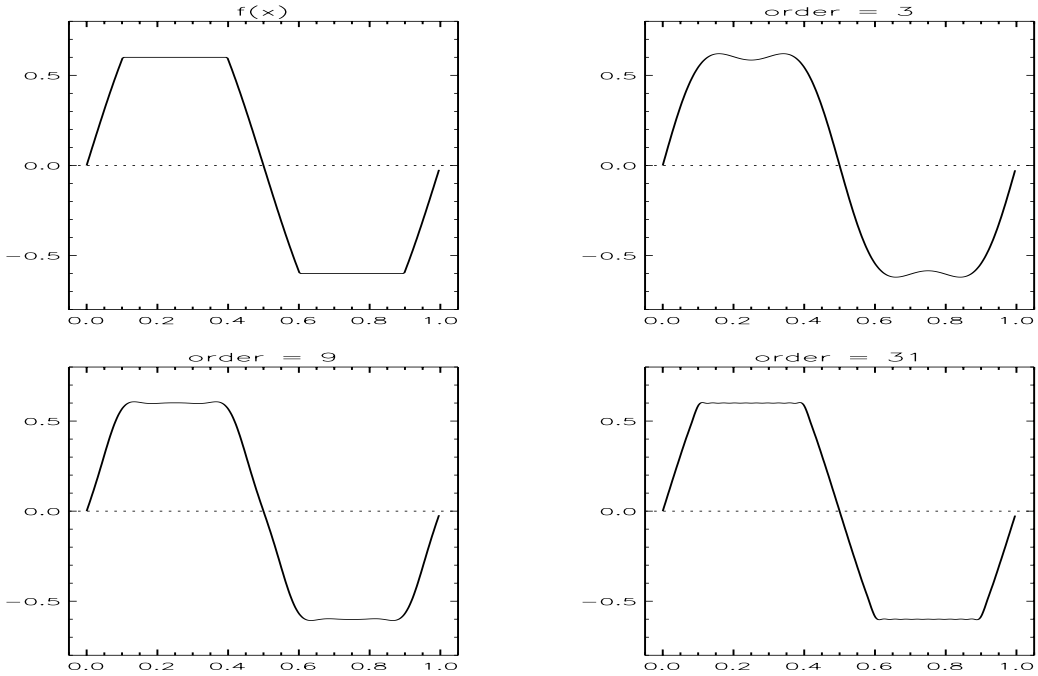


Figure 10.5: Example of Fourier series to a clipped (distorted) sine wave, the odd harmonics are excited by clipping

Chapter 11

Discrete and Fast Fourier Transforms

Applications of Fast Fourier Transforms:

- Signal and Image Processing: noise reduction
- Image Processing: deconvolution (*e.g.*, "Lucy" HST image deconvolution)
- Interferometers: cosine transform of interferogram
- Mass Spectrometers: Fourier transform of cyclotron frequencies.
- Boundary Value Problems, solving differential equations.
- Quantum Mechanics (time, position)
- Antenna beam patterns

11.1 The Definition of a Continuous Fourier Transform

The *forward* continous Fourier transform of a function of variable x (can be spatial or temporal) is given by

$$F_c(\nu) = \int_{-\infty}^{\infty} f_c(x) \cdot \exp\left(-i \cdot 2\pi \cdot x \cdot \nu\right) \cdot dx \equiv FT(f_c(x)) \tag{11.1}$$

and the *inverse* continous Fourier transform from the frequency domain, ν, back to the spatial or temporal domain, x, is given by

$$f_c(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F_c(\nu) \cdot \exp\left(i \cdot 2\pi \cdot \nu \cdot x\right) \cdot d\nu \equiv FT^{-1}(F_c(\nu)) \tag{11.2}$$

The attribute of "forward" and "inverse" is not standardized in the literature and is application dependent. Since most applications involve spectral analysis of temporal or spatial functions "forward" tends to reflect that application. A Fourier series is an inverse process in this notation.

Properties of Continuous Fourier Transforms	
x domain	ν domain
Real	Real even, Imaginary odd
Imaginary	Real odd, Imaginary even
Real even, Imaginary odd	Real
Real odd, Imaginary even	Imaginary
Real and even	Real and even
Real and odd	Imaginary and odd
Imaginary and even	Imaginary and even
Imaginary and odd	Real and odd
Complex and even	Complex and even
Complex and odd	Complex and odd

Properties of Continuous Fourier Transforms			
operator	x domain	ν domain	operator
Addition	$f(x) + g(x)$	$F(\nu) + G(\nu)$	Addition
Scaling	$h(sx)$	$\frac{1}{s} \cdot F(\frac{\nu}{s})$	Scaling
Scaling	$\frac{1}{s} \cdot f(\frac{x}{s})$	$H(s\nu)$	Scaling
Shifting	$f(x - x_0)$	$F(\nu) \exp(-i2\pi\nu \cdot x_0)$	Phase Shift
Modulation	$f(x) \exp(i \cdot 2\pi \cdot x\nu_0)$	$F(\nu - \nu_0)$	Freq. Shift

11.1.1 Windowing

The finite Fourier transform is equivalent to multiplying by a windowing function, $g(x)$, defined by

$$\begin{aligned} g_w(x) &= 1 \quad \text{for} \quad -L < x \leq L \\ &= \frac{1}{2} \quad \text{for} \quad |x| = L \\ &= 0 \quad \text{otherwise} \end{aligned} \tag{11.3}$$

$$F_w(\nu) = \int_{-\infty}^{\infty} g_w(x) \cdot f_c(x) \cdot \exp\left(-i \cdot 2\pi \cdot x \cdot \nu\right) \cdot dx = \int_{-L}^{L} f_c(x) \cdot \exp\left(-i \cdot 2\pi \cdot x \cdot \nu\right) \cdot dx \tag{11.4}$$

This will be discussed in more detail in the section on apodization (Chapter 12). In the next section we show that limiting the extent of the Fourier transform effectively limits the *resolution* of the transform result.

11.1.2 convolution theorem & the channel spectral response function (CSRF)

In general, the Fourier transform of the product of two functions is related to the Fourier transform of the separate functions by the convolution theorem. First we note that the functions under the integral in Eqn. 11.4 can be reproduced by the inverse transform of $F_w(\nu)$.

$$g_w(x) \cdot f_c(x) = FT^{-1}\left[F_w(\nu)\right] \tag{11.5}$$

But, they must also be reproduced by the respective inverse transforms of their Fourier transform counterparts, $F_c(\nu)$ and $G_w(\nu)$.

$$g_w(x) \cdot f_c(x) = FT^{-1}\left[F_c(\nu)\right] \cdot FT^{-1}\left[G_w(\nu)\right] \tag{11.6}$$

In spectral space it is instructive and useful to analyse the function, $G_w(\nu)$, which is convolved with our continuous Fourier transform, $F_c(\nu)$. If we use $\otimes$ to represent a convolution of two functions then we can write

$$F_c(\nu) \otimes G(\nu) \equiv \int_{-\infty}^{\infty} F_c(\nu - u) \cdot G(u) \cdot du \tag{11.7}$$

If we take the inverse Fourier Transform of both sides we have

$$\int_{-\infty}^{\infty} F_c(\nu) \otimes G(\nu) \exp(i \cdot 2\pi \cdot \nu \cdot x) \cdot d\nu \equiv \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F_c(\nu - u) \cdot G(u) \cdot du \cdot \exp(i \cdot 2\pi \cdot \nu \cdot x) \cdot d\nu \tag{11.8}$$

and since ν is not a function of u we can switch the order of integration

$$FT^{-1}[F_c(\nu) \otimes G(\nu)] \equiv \int_{-\infty}^{\infty} G(u) \int_{-\infty}^{\infty} F_c(\nu - u) \cdot \exp(i \cdot 2\pi \cdot \nu \cdot x) \cdot d\nu \cdot du \tag{11.9}$$

and we can separate u by frequency shifting $F_c()$ using the following substitution, for a fixed value of u

$$\begin{aligned} s &\equiv \nu - u & (11.10)\\ ds &\equiv d\nu & (11.11)\\ \exp(i \cdot 2\pi \cdot \nu \cdot x) &= \exp(i \cdot 2\pi \cdot s \cdot x) \cdot \exp(i \cdot 2\pi \cdot u \cdot x) & (11.12) \end{aligned}$$

so that

$$FT^{-1}[F_c(\nu) \otimes G(\nu)] \equiv \int_{-\infty}^{\infty} G(u) \int_{-\infty}^{\infty} F_c(s) \cdot \exp(i \cdot 2\pi \cdot s \cdot x) \cdot \exp(i \cdot 2\pi \cdot u \cdot x) \cdot ds \cdot du \tag{11.13}$$

and now we can separate the integrals

$$FT^{-1}[F_c(\nu) \otimes G(\nu)] \equiv \int_{-\infty}^{\infty} G(u) \cdot exp(i \cdot 2\pi \cdot u \cdot x) \cdot du \cdot \int_{-\infty}^{\infty} F_c(s) \cdot \exp(i \cdot 2\pi \cdot s \cdot x) \cdot ds \tag{11.14}$$

And, therefore, the convolution theorem, relates our Eqn. 11.6 to the convolution

$$\boxed{FT^{-1}[G_w(\nu) \otimes F(\nu)] = FT^{-1}[F_c(\nu)] \cdot FT^{-1}[G_w(\nu)]} \tag{11.15}$$

Thus, we can define a channel spectral response function (CSRF), which is sometimes called the Instrument Lineshape (ILS), in the frequency domain as the Fourier transform of the windowing function. This function represents the weighted sampling in the frequency domain.

$$\begin{aligned} G(\nu) &= \int_{-\infty}^{\infty} g_w(x) \cdot \exp(-i \cdot 2\pi \cdot x \cdot \nu) \cdot dx \\ &= \int_{-L}^{L} \exp(-i \cdot 2\pi \cdot x \cdot \nu) \cdot dx \\ &= \frac{\sin(2\pi \cdot L \cdot \nu)}{\pi \cdot \nu} \quad = \quad 2L \cdot \mathrm{sinc}(2\pi \cdot L \cdot \nu) \end{aligned} \tag{11.16}$$

In Section 12.2 the characteristics of the sinc() function are discussed. The function has a full width half maximum (FWHM) of 0.603355/L and zeros at $\nu = j/(2L)$. The function has large side-lobes that extend

to $\pm\infty$. The first side-lobe is -21.7% of the central lobe and they alternate in sign. See the table in Section 12.2.

Therefore, the consequence of windowing is to reduce the *resolution* in the transformed domain. The discrete convolution theorem for a periodic function of N points is given by

$$F_c(k) \otimes G(k) \equiv \sum_{i=-\infty}^{\infty} F_c(k-i) \cdot G(i) \tag{11.17}$$

11.1.3 Sampling

Most instruments are sampled over finite intervals. The sampling function, $S(x)$, looks like a series of Dirac delta functions (like a comb). When we apply both windowing and sampling our transform is

$$F_s(\nu) = \int_{-L}^{L} f_c(x) \cdot S(x) \cdot \exp(-i \cdot 2\pi \cdot x \cdot \nu) \cdot dx \tag{11.18}$$

$$S(x) \equiv \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} \delta(x - n \cdot \Delta x), \qquad \text{and} \qquad \Delta x = \frac{2L}{N} \tag{11.19}$$

$$F_s(\nu) = \int_{-L}^{L} f(x) \cdot \left[\sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} \delta(x - n \cdot \Delta x) \right] \cdot \exp(-i \cdot 2\pi \cdot x \cdot \nu) \cdot dx \tag{11.20}$$

For N data points the discrete trapezoidal integral can be written

$$F(\nu) = \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} f(n \cdot \Delta x) \cdot \exp(-i \cdot 2\pi \cdot n \cdot \Delta x \cdot \nu) \tag{11.21}$$

assuming $f(-N/2) = f(N/2)$. In some texts, (*e.g.*, Briggs & Hansen) it is written

$$F(\nu) = \sum_{n=-\frac{N}{2}-1}^{\frac{N}{2}} f(n \cdot \Delta x) \cdot \exp(-i \cdot 2\pi \cdot n \cdot \Delta x \cdot \nu) \tag{11.22}$$

For Nyquist sampling (see Section 4.1) the maximum frequency is given by $\nu_{max} = \frac{1}{2\Delta x}$ so that the frequency can be given by a discrete index, k defined by

$$\nu = k \cdot \Delta\nu = k \cdot \frac{2 \cdot \nu_{max}}{N} = k \cdot \frac{N/(4L)}{N/2} = \frac{k}{2L} \tag{11.23}$$

The consequence of sampling is that the result is a periodic function, therefore, *aliasing* can occur. That is, the interval $-L \le x < L$ has the information from the spectral window $-\nu_{max} \le \nu \le \nu_{max}$ and also from *aliased* (*i.e., out-of-band*) windows , for example the j^{th} spectral window, $j \cdot \nu_{max} \le |\nu| \le (j+1) \cdot \nu_{max}$ transforms to the same interferogram. To remove aliasing the sampling interval, Δx, can be reduced thereby increasing N and since $\nu_{max} \propto N$, the spectral window of interest can be selected.

$$F(k) = \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} f(n) \cdot \exp(-i \cdot 2\pi \cdot n \cdot k/N) \qquad \text{for} \quad \frac{-N}{2} \le k < \frac{N}{2} \tag{11.24}$$

$$= \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} f(n) \cdot \exp(-i \cdot 2\pi \cdot n \cdot (k + j \cdot N)/N) \tag{11.25}$$

Most DFT routines are written for positive variable indices. One can note that for any arbitrary integer value of j

$$\exp\left(-i\cdot 2\pi\cdot(-n)\cdot k/N\right) = \exp\left(-i\cdot 2\pi\cdot(N-n)\cdot k/N\right) \tag{11.26}$$

So that Eqn. 11.25 is equal to

$$\begin{aligned} F(k) = F(k\cdot\Delta\nu) &= \sum_{n=0}^{N-1} f(n)\cdot\exp\left(-i\cdot 2\pi\cdot n\cdot k/N\right) && (11.27)\\ &= \sum_{n=0}^{N-1} f(n)\cdot\exp\left(-i\cdot 2\pi\cdot n\cdot(k+j\cdot N)/N\right) && (11.28)\\ &= F\left((k+j\cdot N)\cdot\Delta\nu\right) && (11.29)\end{aligned}$$

however, we must always keep in mind the $f(N-1)$ in this indexing system is really $f(-1)$ and $f(N/2)$ is really $f(-N/2)$ and we have simply rearranged the summation indices for convenience.

A similar analysis of the inverse continuous transform, Eqn. 11.2, will yield

$$f(n) = \frac{1}{N}\sum_{k=0}^{N-1} F(k)\cdot\exp\left(\frac{i\cdot 2\pi\cdot n\cdot k}{N}\right) \tag{11.30}$$

Properties of Discrete Fourier Transforms				
x domain			ν domain	
windowing	$-L \le x < L$	$\Rightarrow$	$G(\nu) =$ $\mathrm{sinc}(2\pi L(\nu-\nu_0)$	finite spectral resolution
sampling	$\Delta x = \frac{2L}{N}$	$\Rightarrow$	$\pm\nu_{max} = \frac{1}{2\Delta x}$	aliasing
aliasing	$\pm L_{max} = \frac{1}{2\Delta\nu}$	$\Leftarrow$	$\Delta\nu = \frac{2\nu_{max}}{N}$	sampling
Addition	$f(n)+g(n)$		$F(k)+G(k)$	Addition
Scaling	$h(s\cdot n)$		$\frac{1}{s}\cdot F(\frac{k}{s})$	Scaling
Scaling	$\frac{1}{s}\cdot f(\frac{n}{s})$		$H(s\cdot k)$	Scaling
Shifting	$f(n-n_0)$		$F(k)\exp(-i2\pi\nu\cdot n_0/N)$	Phase Shift
Modulation	$f(n)\exp(i\cdot 2\pi n\cdot k_0/N)$		$F(k-k_0)$	Freq. Shift

11.2 The Standard Form of the DFT

The *forward* DFT

$$\begin{aligned} F(k) &= \sum_{n=0}^{N-1} f(n)\cdot\exp\left(\frac{-i\cdot 2\pi\cdot k\cdot n}{N}\right) && (11.31)\\ &= \sum_{n=0}^{N-1} f(n)\cdot W_N^{kn} && (11.32)\end{aligned}$$

and the *inverse* DFT

$$\begin{aligned} f(n) &= \frac{1}{N}\sum_{k=0}^{N-1} F(k)\cdot\exp\left(\frac{i\cdot 2\pi\cdot n\cdot k}{N}\right) && (11.33)\\ &= \frac{1}{N}\sum_{k=0}^{N-1} F(k)\cdot W_N^{-nk} && (11.34)\end{aligned}$$

where W_N is the N^{th} root of unity given by the complex operator

$$W_N^{kn} \equiv \exp\left(\frac{-i \cdot 2\pi \cdot k \cdot n}{N}\right) \tag{11.35}$$

The is no standard in the literature or software; however. Here are some examples:

Assume $W_N^{()} = \exp(-i \cdot 2\pi \cdot ()/N)$ unless otherwise noted		
	Forward	Inverse
ftp/cfft.F	$F(k) = \sum_{n=1}^{N} f(n) \cdot W_N^{(n-1)(k-1)}$ call cfft(N,+1,f)	$f(n) = \frac{1}{N} \sum_{k=1}^{N} F(k) \cdot W_N^{-(k-1)(n-1)}$ call cfft(N,-1,f)
MATLAB	$X(k) = \sum_{n=1}^{N} x(n) \cdot W_N^{(k-1)(n-1)}$ fft(x)	$x(n) = \frac{1}{N} \sum_{k=1}^{N} X(k) \cdot W_N^{-(n-1)(k-1)}$ ifft(X)
IDL	$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) \cdot W_N^{ux}$ FFT(f,-1)	$f(x) = \sum_{u=0}^{N-1} F(u) \cdot W_N^{-xu}$ FFT(F,+1)
Cochran *et al.*	$F_k = \sum_{n=1}^{N} f_n \cdot W_N^{(k-1)(n-1)}$	
Cooley *et al.*	$F(k) = \sum_{j=0}^{N} f(j) \cdot W_N^{-jk}$ $(W_N^{()} \equiv \exp(+i \cdot 2\pi \cdot ()/N))$	
Brigham	$G(n) = \sum_{k=0}^{N-1} g(k) \cdot W_N^{kn}$	$g(k) = \frac{1}{N} \sum_{n=0}^{N-1} G(n) \cdot W_N^{-nk}$
Briggs & Hensen (pg. 24)	$F(k) = \frac{1}{N} \sum_{n=0}^{N-1} f(n) \cdot W_N^{-kn}$ $(W_N^{()} \equiv \exp(+i \cdot 2\pi \cdot ()/N))$	

11.2.1 An example of the DFT with 32 points

A FORTRAN program to compute the DFT is shown below. Both the FORTRAN (dft.for) and IDL (dft.pro) versions are available on the ftp site.

```
      subroutine dft(Npts, fx, IDIR,  gx)
      implicit none

      integer*4 Npts   ! number of points for DFT
      complex*8 fx(*)  ! input function
      integer*4 IDIR   ! forward FFT, IDIR < 0
c                        reverse FFT, IDIR > 0
      complex*8 gx(*)  ! output function

c     local variables
c     ---------------

      integer*4 IDIRLOC
```

```
integer*4 k, n
real*4    pi
parameter (pi=3.14159265)
real*4    arg0, arg1, arg
complex*8 scl    ! scaling parameter
complex*8 sum    ! summation of inner loop
complex*8 Wn     ! Nth root of zero

IDIRLOC = IDIR
if(IDIR.le.0) IDIRLOC = -1   ! forward FFT
if(IDIR.gt.0) IDIRLOC = +1   ! inverse FFT

if(IDIRLOC.lt.0) then        ! FORWARD DFT
  arg0 = -2.0*pi/FLOAT(Npts)
  scl = CMPLX(1.0,0.0)
else
  arg0 =  2.0*pi/FLOAT(Npts)    ! INVERSE DFT
  scl = CMPLX(1.0/FLOAT(Npts),0.0)
endif
do k = 0, Npts-1
  arg1 = arg0*FLOAT(k)
  sum = fx(1)
  do n = 1, Npts-1
    arg = arg1*FLOAT(n)
    Wn = CMPLX(COS(arg),SIN(arg))  ! Nth root of zero
    sum = sum + Wn*fx(n+1)
  enddo
  gx(k+1) = scl*sum     ! DFT for point k
enddo

return
end
```

In the following examples the data provided to the DFT was properly folded so that $i = k = \frac{N}{2}$ represented $x = \nu = 0$.

The first column in Fig. 11.1 shows a delta function, $\delta(x - j \cdot \Delta x))$, for j = -1, 1, -7, -16, respectively. The sample spacing is given by $\Delta x = 2L/N$. The second column is the real portion of the inverse Fourier transform and the third column is the imaginary portion. A N =32 point transform was used, so the frequency axis in the transform domain is $\nu_{max} = \frac{1}{2\Delta x} = \frac{N}{4L} = \pm 8$ cycles/L. In all cases, a single point has both an imaginary and real component. If we were desired a real function we would need a symmetric pair of delta functions.

Also, notice that the transforms in the 3$^{\text{rd}}$ row do not reproduce the peaks of the cosine function because of the rational fraction in the argument of $\cos(2\pi \cdot 7 \cdot k/32)$. We are sampling a continuous cosine wave at these locations and the sampling is such that we can uniquely distinquish the phase and amplitude of 16 frequencies of $k/(2L)$ cycles/L.

In Fig. 11.2 there are three rows. In the first column is a real cosine function of ν= 0.5, 8, and 15.5 cycles/L. The second and third columns are the real and imaginary portions of the inverse transform, respectively. Notice that the 8 cycles/L figure has twice the weight because that frequency is represented by 1 point, rather than the pair of points. This is arises $f(0) = f(N)$. Also, notice that the $k = M - 1 \rightarrow$ 15.5 cycles/L transform is exactly equal to the lowest frequency transform. This cases is exactly alaised to a frequency of $16 - 15.5 = 0.5$ cycles/L. All transforms of frequencies greater than 8 cycles/L will suffer from alaising.

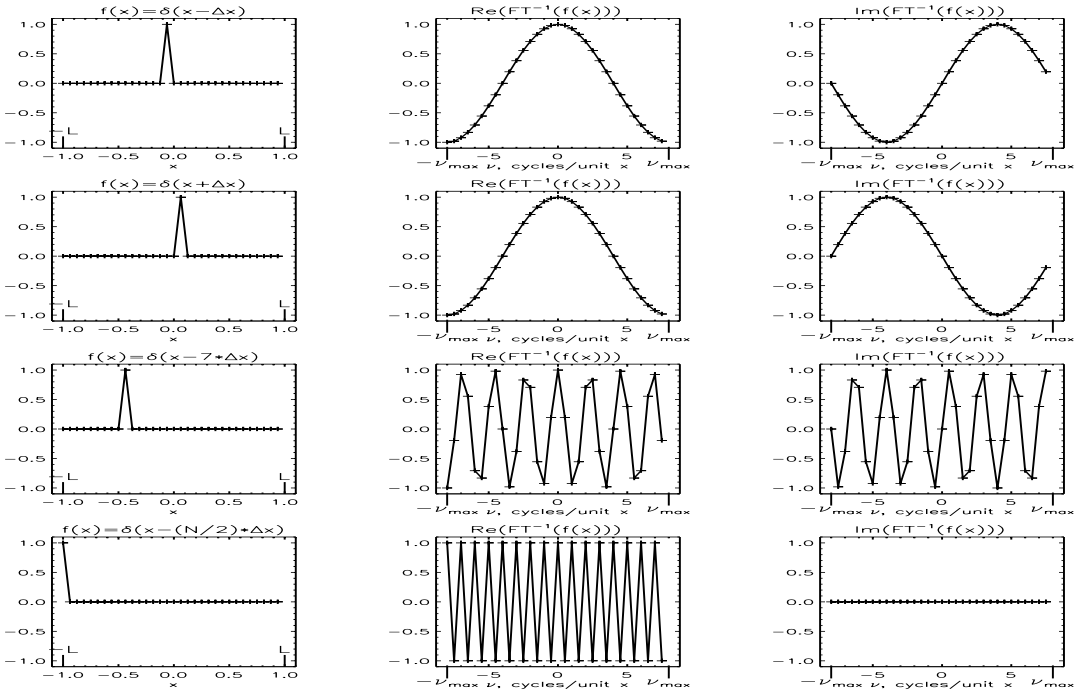


Figure 11.1: Example of a discrete Fourier transform (DFT) of delta functions

In the lower left of Fig. 11.3 we are plotting 15.5 cycles per L but only sampling 32 points and then drawing lines between the points. In reality, there is a full oscillation between each pair of points. Thus, this is a worst case example of aliasing.

11.2.2 Verify the inverse DFT works

Substituting the *forward* DFT (Eqn. 11.31) into a modifed form of the inverse DFT (substitute m for n in Eqn. 11.33) yields

$$\begin{aligned} f(m) &= \frac{1}{N}\sum_{k=0}^{N-1}\sum_{n=0}^{N-1} f(n)\cdot W_N^{kn}\cdot W_N^{-mk} \\ &= \sum_{k=0}^{N-1}\sum_{n=0}^{N-1}\frac{f(n)}{N}\cdot W_N^{k(n-m)} \\ &= f(n) \quad \text{for} \quad m=n \end{aligned} \tag{11.36}$$

which follows from the discrete orthogonality relationship

$$\begin{aligned} \sum_{k=0}^{N-1} W_N^{k(n-m)} &= N \quad \text{if} \quad m=n \\ &= 0 \quad \text{if} \quad m\neq n \end{aligned} \tag{11.37}$$

so that the inverse of the forward DFT reproduces the original array. The discrete orthogonality relationship is easily derived as follows:

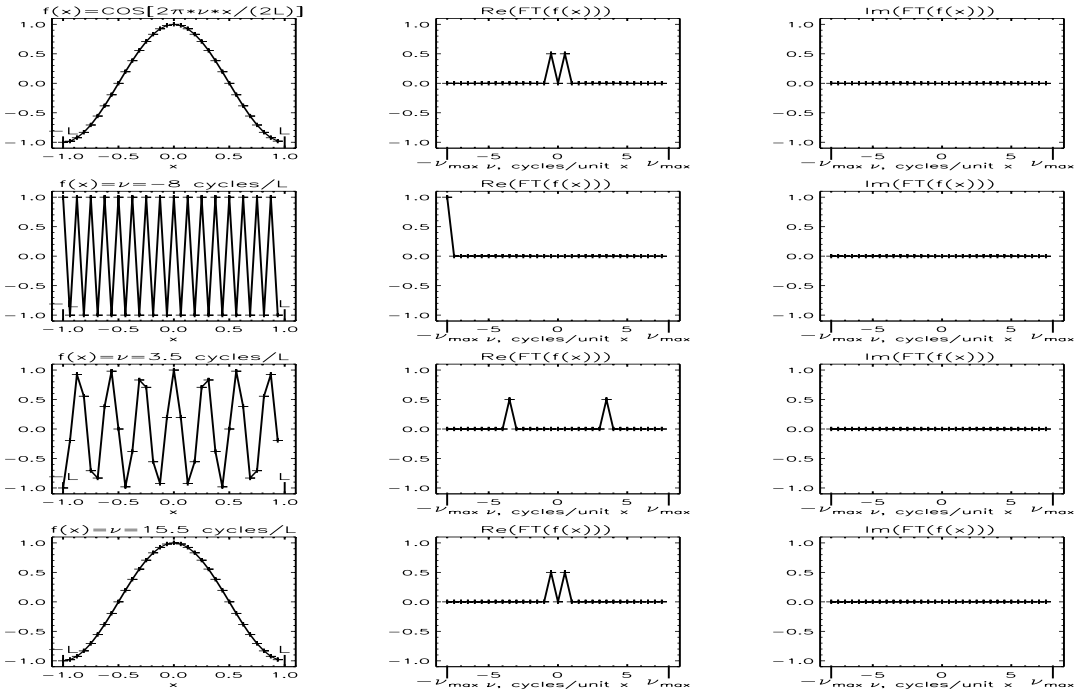


Figure 11.2: Symmetry of the DFT: Example of even and odd functions

$$
\begin{aligned}
\sum_{k=0}^{N-1} W_N^{k(n-m)} &= \sum_{k=0}^{N-1} \exp\left(-i \cdot 2\pi \cdot k \cdot (n-m)/N\right) \\
&= \sum_{k=0}^{N-1} \cos\left(2\pi \cdot k \cdot (n-m)/N\right) - i \cdot \sin\left(2\pi \cdot k \cdot (n-m)/N\right) \\
&= \sum_{k=0}^{N/2-1} \cos\left(2\pi \cdot k \cdot (n-m)/N\right) + \cos\left(2\pi \cdot k \cdot (n-m)/N + \pi \cdot (n-m)\right) \\
&\qquad -i \cdot \left[\sin\left(2\pi \cdot k \cdot (n-m)/N\right) + \sin\left(2\pi \cdot k \cdot (n-m)/N + \pi \cdot (n-m)\right)\right] \\
&= \sum_{k=0}^{N/2-1} \cos\left(\theta \cdot (n-m)\right) + \cos\left((\theta+\pi) \cdot (n-m)\right) \\
&\qquad -i \cdot \sin\left(\theta \cdot (n-m)\right) - i \cdot \sin\left((\theta+\pi) \cdot (n-m)\right) \qquad \theta \equiv 2\pi \cdot k/N \\
&= N \quad \text{if} \quad m = n \\
&= 0 \quad \text{if} \quad m \neq n
\end{aligned}
\tag{11.38}
$$

11.2.3 other quadratures in the DFT

We can show that the DFT is simply the trapezoidal quadrature formulation of the continuous Fourier transform for a finite window. See section on Fourier series, Eqn. 10.17 through Eqn. 10.20

Briggs and Hensen [1995] discuss different quadratures, including Simpson's quadrature. Given that most interferograms have large structure the higher order quadratures are not necessarily more accurate.

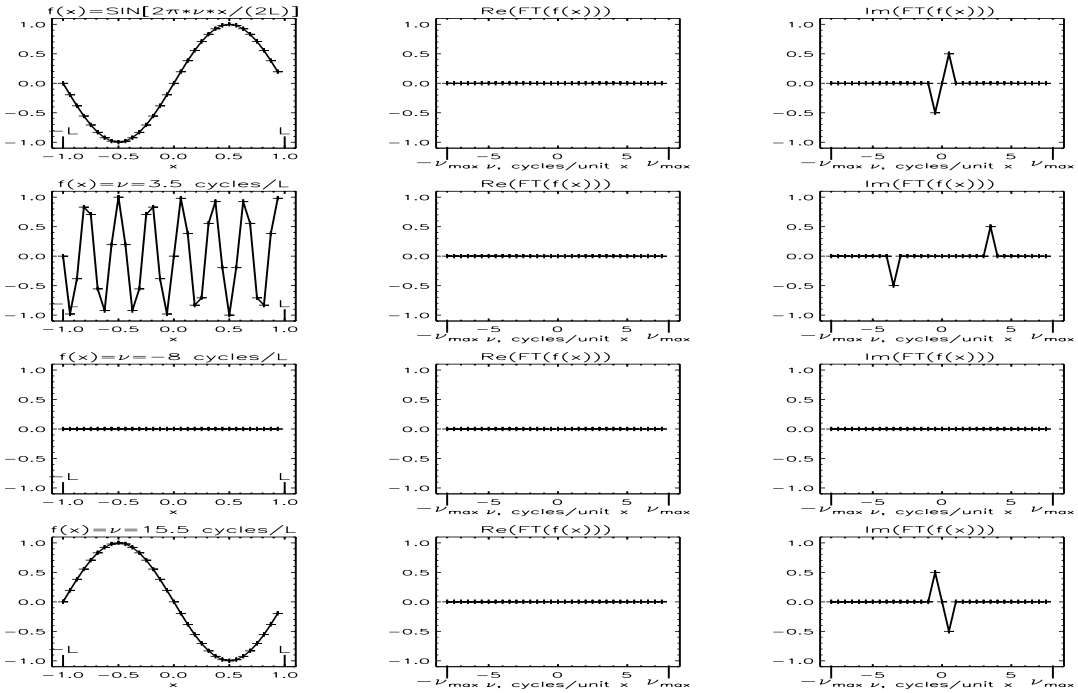


Figure 11.3: Aliasing of the DFT

11.2.4 The Fast Fourier Transform

There are many applications that use a Fourier Transform and the DFT is usually too slow to be practical. For example,

- spacecraft applications
- operational code, such as weather applications
- certain languages that are interpretive.

The Fast Fourier Transform (FFT) is an algorithm that computes the DFT quickly. Gauss derived a form of an FFT in an unpublished treatise on orbital interpolation methods, "Theoria interpolationis methodo nova tractata.", which appeared in Göttingen posthumously in 1866. This work was dated to 1805 and the derivation was lost until after Cooley and Tukey published thier derivation in 1965 (Briggs & Henson, 1995, pg. 5). The DFT arrives at exactly the same answer as the DFT, only faster and with less roundoff error. We begin our derivation with the equations for the DFT.

$$F(k) = \sum_{n=0}^{N-1} f(n) \cdot W_N^{kn} \tag{11.39}$$

$$W_N^{kn} \equiv \exp\left(\frac{-i \cdot 2\pi \cdot k \cdot n}{N}\right) \tag{11.40}$$

We can write the operations as a matrix. For a $N = 4$ DFT the matrix would be

$$\begin{bmatrix} F(0) \\ F(1) \\ F(2) \\ F(3) \end{bmatrix} = \begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^4 & W_4^6 \\ W_4^0 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} \cdot \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \end{bmatrix} \tag{11.41}$$

From this we can see that, in general, there are N^2 complex multiplications and $N \cdot (N-1)$ complex additions. We can also eliminate sines and cosines that are obvious by noting that $W_N^0 = e^0 = 1$, which results in

$$\begin{bmatrix} F(0) \\ F(1) \\ F(2) \\ F(2) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4^1 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} \cdot \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \end{bmatrix} \tag{11.42}$$

$$\begin{bmatrix} F(0) \\ F(1) \\ F(2) \\ F(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4^1 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^0 & W_4^2 \\ 1 & W_4^3 & W_4^2 & W_4^1 \end{bmatrix} \cdot \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \end{bmatrix} \tag{11.43}$$

Furthermore, $W_N^{N/2} = e^{-i\pi} = -1$, $W_N^{N/4} = e^{-i\pi/2} = -i$, $W_N^{3N/4} = e - i3\pi/2 = +i$.

$$\begin{bmatrix} F(0) \\ F(2) \\ F(1) \\ F(3) \end{bmatrix} = \begin{bmatrix} 1 & W_4^0 & 0 & 0 \\ 1 & W_4^2 & 0 & 0 \\ 0 & 0 & 1 & W_4^1 \\ 0 & 0 & 1 & W_4^3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & W_4^0 & 0 \\ 0 & 1 & 0 & W_4^0 \\ 1 & 0 & W_4^2 & 0 \\ 0 & 1 & 0 & W_4^2 \end{bmatrix} \cdot \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \end{bmatrix} \tag{11.44}$$

To see the matrix method of decimation see page 150 of Brigham [1974]

Decimation

We will use the definition of the forward DFT, Eqn. 11.31, which is rewritten here for convenience

$$F(k) = \sum_{n=0}^{N-1} f(n) \cdot W_N^{nk} \tag{11.45}$$

We can separate Eqn. 11.45 into even and odd components, each with $M = \frac{N}{2}$ points. $n = 2m$ is the even index and $n = 2m-1$ is odd so that

$$\begin{aligned} F(k) &= \sum_{m=0}^{N/2-1} f(2m) \cdot W_N^{2mk} + \sum_{m=0}^{N/2-1} f(2m+1) \cdot W_N^{(2m+1)k} \\ &= \sum_{m=0}^{N/2-1} f(2m) \cdot W_N^{2mk} + W_N^k \cdot \sum_{m=0}^{N/2-1} f(2m+1) \cdot W_N^{2mk} \qquad \text{for} \quad k = 1, N-1 \end{aligned} \tag{11.46}$$

This is equivalent to adding the result of two $\frac{N}{2}$ DFT's, each operating on the $M = \frac{N}{2}$ point arrays. Note that for $N = 2M$ the following simplifications can be made

$$W_N^{2mk} = e^{-i\cdot 2\pi \cdot 2mk/N} = e^{-i\cdot 2\pi \cdot mk/M} = W_M^{mk} \tag{11.47}$$

$$W_N^{M+k} = e^{-i\cdot 2\pi \cdot (M+k)/N} = e^{-i\cdot 2\pi/2} \cdot e^{-i\cdot 2\pi \cdot k/N} = -W_N^k \tag{11.48}$$

$$W_M^{m(M+k)} = e^{-i\cdot 2\pi \cdot m(M+k)/M} e^{-i\cdot 2\pi \cdot m} \cdot e^{-i\cdot 2\pi \cdot mk/M} = W_M^{mk} \tag{11.49}$$

so that Eqn. 11.46

$$\begin{aligned} F(k) &= \sum_{m=0}^{M-1} f(2m) \cdot W_M^{mk} + W_N^k \cdot \sum_{m=0}^{M-1} f(2m+1) \cdot W_M^{mk} \qquad \text{for} \quad k = 0, M-1 \quad \textbf{and} \\ F(M+k) &= \sum_{m=0}^{M-1} f(2m) \cdot W_M^{mk} - W_N^k \cdot \sum_{m=0}^{M-1} f(2m+1) \cdot W_M^{mk} \qquad \text{for} \quad k = 0, M-1 \end{aligned} \tag{11.50}$$

A DFT has N^2 complex multiplications and N(N-1) complex additions. Decimation allows $2 \cdot \left(\frac{N}{2}\right)^2 + N$ complex multiplications, therefore, it takes $\approx 1/2$ the time for large N. This process can be continued until we have pairs of points. For a sample size of $N = 2^\gamma$ there are γ decimation passes and ratio of multiplies is approximately $\frac{N\gamma}{N^2}$.

If we write the index as binary bits we can see that the process of γ even/odd sorting is equivalent to reversal of the bits. For our example with $N = 16$ there are 4 bits, which we can represent as (a, b, c, d). The first even $\rightarrow$ odd sort in binary looks like 0001 $\rightarrow$ 1000, 0010 $\rightarrow$ 0001, 0011 $\rightarrow$ 1001, etc., and in general abcd $\rightarrow$ dabc. After γ steps it we can see that it is equivalent to flipping the order of the bits from left to right.

$$\begin{aligned}
f(a,b,c,d) &\Rightarrow f(d,a,b,c) && \text{step } \#\gamma = 4 \\
f(d,a,b,c) &\Rightarrow f(d,c,a,b) && \text{step } \#3 \\
f(d,c,a,b) &\Rightarrow f(d,c,b,a) && \text{step } \#2 \\
f(d,c,b,a) &\Rightarrow f(d,c,b,a) && \text{step } \#1 \quad \text{degenerate pass}
\end{aligned} \tag{11.51}$$

The *radix-2 decimation* is represented graphically for a $N = 16$ DFT in Fig. 11.4.

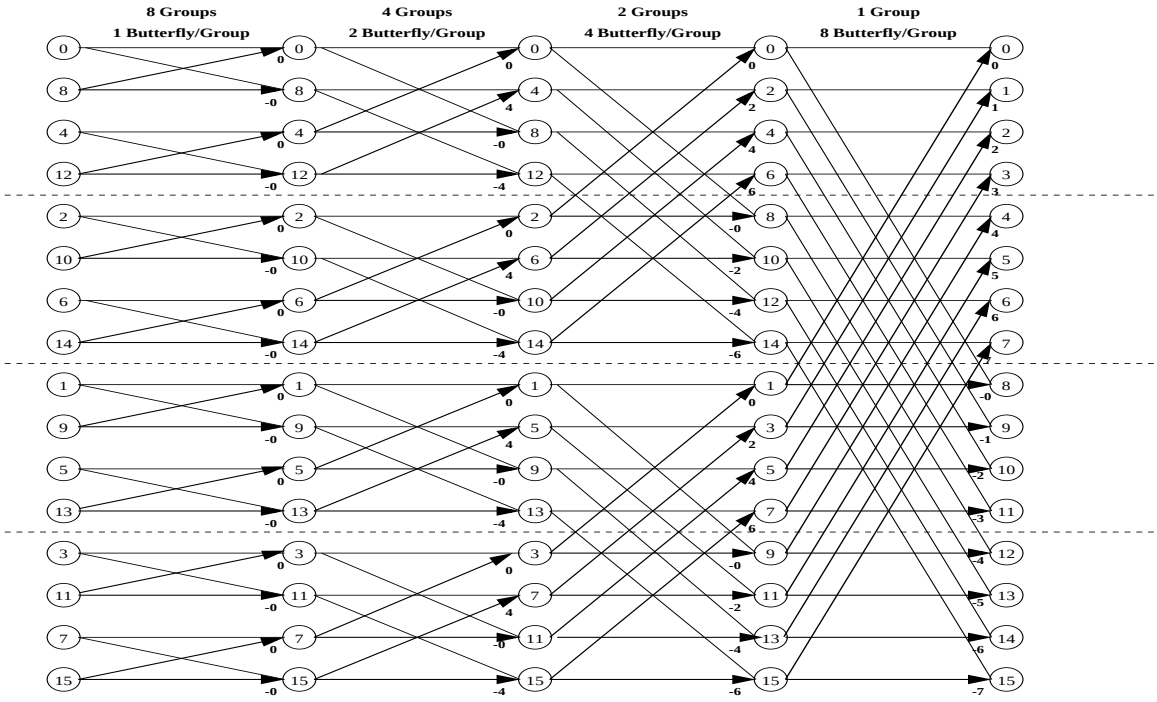


Figure 11.4: Illustration of the FFT radix-2 decimation

The basic operation can be done *in-place* where pairs of values are tranformed. This operation is called a *butterly* due to its appearance in Fig. 11.5.
and is computed as follows:

$$\begin{aligned}
Y_i &= X_i + W_N^p \cdot X_j \\
\Re(Y_i) + i \cdot \Im(Y_i) &= \Re(X_i) + i \cdot \Im(X_i) + (\cos(2\pi p/N) + i \cdot \sin(2\pi p/N)) \cdot \\
&\quad (\Re(X_j) + i \cdot \Im(X_j)) \\
&= \Re(X_i) + \Re(X_j) \cdot \cos(2\pi p/N) - \Im(X_j) \cdot \sin(2\pi p/N) \\
&\quad + i \cdot [\Im(X_i) + \Re(X_j) \sin(2\pi p/N) + \Im(X_j) \cdot \cos(2\pi p/N)]
\end{aligned} \tag{11.52}$$

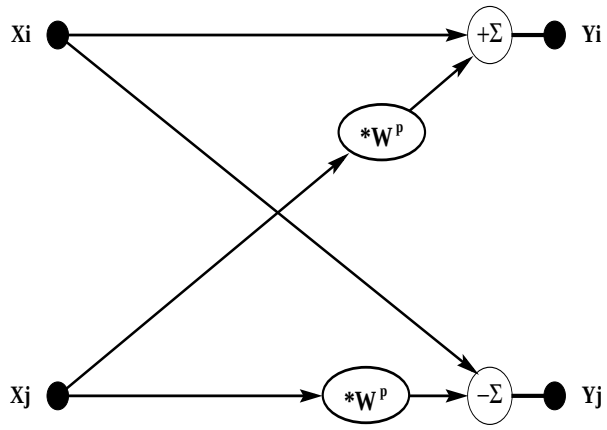


Figure 11.5: Illustration of a single FFT "butterfly" operator

$$\begin{aligned} Y_j &= X_j - W_N^p \cdot X_i \\ \Re(Y_j) + i \cdot \Im(Y_j) &= \Re(X_j) + i \cdot \Im(X_j) - (\cos(2\pi p/N) + i \cdot \sin(2\pi p/N)) \cdot \\ &\quad (\Re(X_i) + i \cdot \Im(X_i)) \\ &= \Re(X_j) - \Re(X_i) \cdot \cos(2\pi p/N) + \Im(X_i) \cdot \sin(2\pi p/N) \\ &\quad + i \cdot [\Im(X_i) - \Re(X_i) \cdot \sin(2\pi p/N) - \Im(X_i) \cdot \cos(2\pi p/N)] \end{aligned} \tag{11.53}$$

The first pass is called the *degenerate pass*, since the are only 2 points and the separation into even and odd is degenerate. This is shown in the figure. The sequence of γ even/odd separations is numerically equivalent to a *bit reversal* of original order of the data. Take the example in the figure with $N = 2^4 = 16$ points. We can represent the output index and the input index, determined graphically, in binary. We notice that the input is simply the output index with the binary bits reversed (from left to right). In general, the number of swaps is equal to

$$\#\text{ of swaps} = \text{INT}\left[\frac{N - Z}{2}\right] \qquad \text{where} \qquad Z = 2^{\text{INT}((log_2(N)+1)/2)} \tag{11.54}$$

and for our example of a $N = 16$ point FFT the swapping of indices is shown in binary

output index	input index	do we swap
0 = 0000	0000 = 0	no (equal)
1 = 0001	1000 = 8	YES
2 = 0010	0100 = 4	YES
3 = 0011	1100 = 12	YES
4 = 0100	0010 = 2	no (already swapped)
5 = 0101	1010 = 10	YES
6 = 0110	0110 = 6	no (equal)
7 = 0111	1110 = 14	YES
8 = 1000	0001 = 1	no (already swapped)
9 = 1001	1001 = 9	no (equal)
10 = 1010	0101 = 5	no (already swapped)
11 = 1011	1101 = 13	YES
12 = 1100	0011 = 3	no (already swapped)
13 = 1101	1011 = 11	no (already swapped)
14 = 1110	0111 = 7	no (already swapped)
15 = 1111	1111 = 15	no (equal)

N	$\log_2(N)$	# swaps	# equal	N^2	$N \cdot \log(N)$	$\frac{N \cdot \log(N)}{N^2}$
4	2	1	2	16	8.4	52.5%
8	3	2	4	64	24.8	38.8%
16	4	6	4	256	65.6	25.6 %
32	5	12	8	1,024	163.2	15.9%
64	6	28	8	4,096	390.4	9.53%
128	7	56	16	16,384	908.8	5.55%
256	8	120	16	65,536	2,073.6	3.16%
512	9	240	32	262,144	4,659.2	1.78%
1,024	10	496	32	$1.048 \cdot 10^6$	10,342.4	0.986%
2,048	11	992	64	$4.194 \cdot 10^6$	22,732.8	0.542%
4,096	12	2,016	64	$1.678 \cdot 10^7$	49561.6	0.295%
8,192	13	4,032	128	$6.711 \cdot 10^7$	107315	0.160%
16,384	14	8,128	128	$2.684 \cdot 10^8$	231014	0.086%
32,768	15	16,256	256	$1.074 \cdot 10^9$	494797	0.046%
65,536	16	32,640	256	$4.295 \cdot 10^9$	$1.055 \cdot 10^6$	0.0246%
131,072	17	65,280	512	$1.718 \cdot 10^{10}$	$2.241 \cdot 10^6$	0.0130%
262,144	18	130,816	512	$6.872 \cdot 10^{10}$	$4.745 \cdot 10^6$	0.00690%
524,288	19	261,632	1024	$2.749 \cdot 10^{11}$	$1.001 \cdot 10^7$	0.00364%

A simple FORTRAN code to perform the swapping can be seen in the beginning of the program cfft.for (also printed a few pages ahead).

The operation of bit-reversal can be done most efficiently in lower level lanquages. For example, in assembler you can load the index into register #1 and clear register #2. For N_b bits the following steps are performed

1. Logical shift register #2 left by 1 bit.
2. Logical shift register #1 right by 1 bit.
3. If the status register carry bit is set then increment register #2.

In FORTRAN the same operation can be done with integer math (and hope the compiler does a divide by 2 as a shift operation).

11.2.5 Other topics for the DFT

- Fast Fourier Transform algorithm can be done on any size array. If memory is exceeded the relationship between bit-reversal and even/odd decimation can be used to perform the FFT on a specified block size and then perform remaining passes by operating on pairs of blocks on a disk.

  See Buijs (1969) and Singleton (1967) for a description of how to transform arrays of data larger than available memory.

- In general, any decimation is allowed, 2, 3, 4, prime, 2^m, etc. IDL and MATLAB FFT functions do prime number decimation and does the DFT if there is not an efficient decimation. Therefore, the execution time is a strong function of the number of points for these functions; however, you are not limits to only $N = 2^j$ points. For example, Winograd and Nawab (1995) or Hayes (1999) for examples.

- Build a sine table lookup. This minimizes the calling of the sine and cosine functions, which are repetitive. For an N point FFT there needs to be enough memory of $N/2$ sine values. For indexes $k = 0, N-1$ and $n = 0, N-1$ the cosine starts at $N/4$ so the FFT computes and index (i = 1 +

MOD(k·n,N/2)) for sine and (i = 1 + MOD(N/4 + k*n, N/2)) for cosine. The tabulated values are sine(i) = $\sin(2\pi \cdot (i-1)/N)$.

- Compute and store the sine and cosine values so that complex mathematics can be used without additional indexing (*i.e.*, without adding the $\frac{\pi}{2}$ offset). This takes N memory words to store the data. The cosine table is easily built from the sine table and minimizing indexing when the FFT is computed.
- Build the bit reversal in hardware and build in the capability to quickly compute $2^i + a$ for any address a.
- Build the *butterfly* as a dedicated floating point component.
  - addition is more difficult than multiplication due to range of exponents
  - serial adders and multipliers make the hardware simple.

Here is an example FORTRAN program (bitinv.for on the ftp site) to do the bit-inversion

```
      function bitinv(nu,k)
      implicit none
      integer*4  bitinv,k,nu

c     local variables
      integer*4  k1,k2,ks,i
      integer*4  zero,one,two
      parameter  (zero=0, one=1, two=2)

        k1 = k-one  ! convert index starting point from 1 to 0
        ks = zero   ! clear output register
        do i = one,nu
          k2 = k1/two
          ks = two*(ks-k2) + k1  ! 2*ks + (k1-2*k2)
          k1 = k2
        enddo
        bitinv = ks + one  ! return bit reversal+1 (FORTRAN indexing)
      return
      end
```

A FORTRAN version of the FFT is shown in Section B.2. It is available on the ftp site (ftp/cfft.for).

11.3 Signal Processing Methods

11.3.1 Zero filling as a method of interpolation

As discussed earlier, when the Fourier transform of $F(\nu) = FT(f(x))$ is sampled and truncated to $-L \le x < L$ or $2L = N \cdot \Delta x$. As we saw with Eqn. 11.16 $F(\nu)$ has a finite resolution in ν. We will show later (see Eqn. 12.15) that the full width half maximum (FWHM) of the sinc function in Eqn. 11.16 is given by

$$\text{FWHM} = \frac{1.2}{2L} \qquad \textbf{FWHM of the SINC function} \tag{11.55}$$

The Nyquist sampling criteria for the DFT is

$$\Delta\nu = \frac{N}{2L} = \frac{1}{2\Delta x} \qquad \textbf{sampling in } \nu \textbf{ space} \tag{11.56}$$

We write the equation as $F(k) \cdot \mathrm{DFT}(f(n))$ for $0 \le n < N$ and $x = n \cdot \Delta x$ and $\nu = k \cdot \Delta\nu$. If we increase N then the resolution in ν will be increased accordingly. We can zero fill $f(n)$ from $n = N$ to $n = N'$ and the new transform $F(k') = \mathrm{DFT}(f(n')$ for $0 \le x' < L'$ will be sampled at $\Delta\nu' = \frac{1}{2L'}$. The sinc function have the same FWHM since we have not added any new information (*i.e.*, the extra zeroes do not contribute to the integral, so the limits of the integral are still effectively zero to L, NOT zero to L'). Also, notice that the bandpass of the transform has not changed, that is, the resulting transform is still representative of $\nu_1 \le \nu \le \nu_2$ where $\nu_2 = \nu_1 + N \cdot \Delta\nu = \nu_1 + \frac{1}{2\Delta x}$.

Since the original function, $f(n)$, $0 \le n < N$, was Nyquist sampled the original function is being represented in $F(k')$. Therefore, the process of zero filling can be used to re-sample a Nyquist sampled function to a different set of ν's, effectively interpolating a function.

For example, suppose you needed to interpolate a set of data such that the sampling would be increated by 4 (*i.e.,* N' = 4*N, $\Delta x' = \Delta x/4$, then you could

- For a given array of data f(n), associated with $x = n \cdot \Delta x$, take the DFT, Y(k) = DFT(f(n)) for N points.
- Zero fill the result, Y(k)=0, for k=N+1 to 4N·ν
- Take the inverse g(n)=4*DFT^{-1}(Y) where the factor of 4 to account for the normalization in the inverse DFT.
- Now the points in the new array, g(n), are interpolated to be associated with $x = n \cdot \Delta x/4$ and g(0),g(3),g(7),...,g(4N-1) are identical to f(0),f(1), f(2),...,f(N); however, the points in-between are interpolated by a fit to a N^{th} order Fourier Series to the original function, f(n).

11.3.2 Oversampling to *filter* out-of-band signal

The Fourier transform of a sampled function, $f(n)$, is periodic. The bandpass of the result is given by $\nu_1 \le \nu \le \nu_2$ where $\nu_2 = \nu_1 + N \cdot \Delta\nu = \nu_1 + \frac{1}{2\Delta x}$ where ν_1 can be 0 or any integer multiple of $\frac{1}{2\Delta x}$. All other bandpasses, measured in the frequency domain are alaised into the spatial or temporal domain. Thus,

$$J = \frac{\nu_2}{\nu_2 - \nu_1} = \nu_2 \cdot 2 \cdot \Delta x \tag{11.57}$$

is an integer that is the alias order of the measurement.

If we sampled $f(n)$ at a finer sampling then the band pass would be increased. For example, if we increased the sampling by a factor of J'/J (*i.e.,* the values of $f(n)$ must be computed or measured at the finer interval, $\Delta x'$, and have $N' = J' \cdot N/J$ samples) then the function $f(n')$ would be still be associated with $-L \le x < L$; however, we would have extended the frequency domain by $(\nu_2 - \nu_1)) \cdot J'/J$. The sampling and resolution of ν is unchanged. Therefore, $\Delta\nu = \frac{1}{2L} = \frac{1}{2 \cdot N \cdot \Delta x} = \frac{1}{2 \cdot N' \cdot \Delta x'}$; however, the DFT of $f(n)$ would be $F(\nu)$ from $\nu_1 \le \nu < \nu_3 = \nu_1 + N' \cdot \Delta\nu$.

In interferometer applications (will be discussed in next section), the instrument has a a finite spectral bandpass, defined by electronics, detectors and optics. If, for example, we desired measurements from ν_1 to $3 \cdot \nu_1$ we could perform the sampling in $f(n)$ with a sampling of $\Delta x' = \frac{1}{8 \cdot \nu_1}$, which is 2 times the sampling needed for our desired interval of $2 \cdot \nu_1$. The signal in $F(\nu)$ from $0 \le \nu < \nu_1$ and from $3 \cdot \nu_1 \le \nu < 4 \cdot \nu_1$ would contain the *out-of-band* signal and, therefore, is removed from the interval $\nu_1 \le \nu < 3 \cdot \nu_1$. In typical applications the amount of oversampling is 8 or 16. The Fourier filtering of the signal is equivalent to optical or electronic filters; however, the characteristics of the filters has very sharp edges. In space applications, the DFT can be performed on the space-craft and only the desired bandpass needs to be brought down (since down-link is typically a critical specification).

11.4 Symmetry Relationships of the DFT

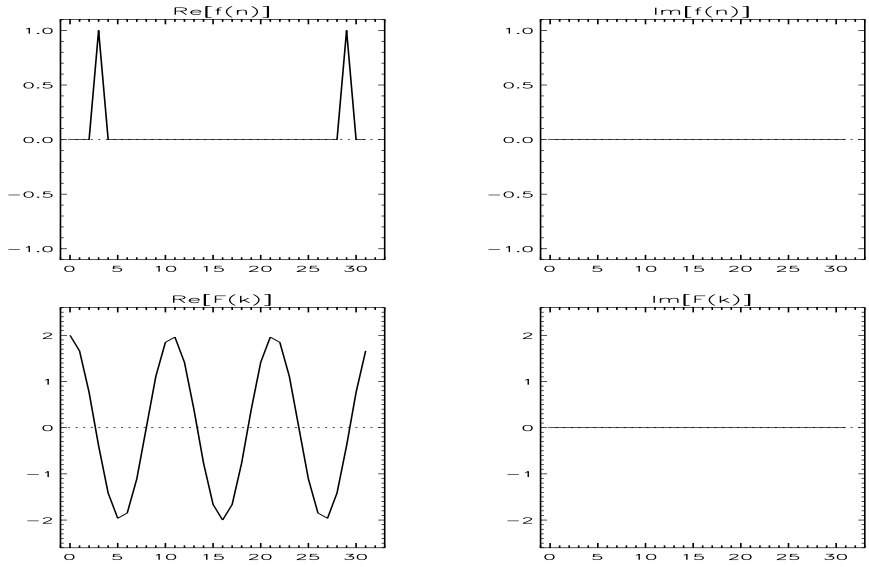


Figure 11.6: For an even real function the Fourier transform is real and even.

Therefore, the symmetry relationships can be summarized as follows.

Table 11.1: Summary of symmetric of the Fourier transform

$f(n)$	$F(k)$
Real, Even	Real, Even
Real, Odd	Imag., Odd
Imag., Even	Imag., Even
Imag., Odd	Real, Odd

If $f(n)$ is a real function with even and odd symmetry,

$$f(n) = h^e(n) + h^o(n) + i \cdot (g^e(n) + g^o(n)) \tag{11.58}$$

then the orthogonality of the Fourier transform, $FT[f] = R + i \cdot I$, relationships are

$$\begin{aligned} FT[h^e(n)] &= R^e(k) && (11.59)\\ FT[h^o(n)] &= I^o(k) && (11.60)\\ FT[i \cdot g^e(n)] &= I^e(k) && (11.61)\\ FT[i \cdot g^o(n)] &= R^o(k) && (11.62) \end{aligned}$$

11.4.1 Composite transform of two real functions

When performing transforms of real data we can utilize the symmetry properties of pure real and pure imaginary FFT's to utilize the available memory. The symmtery relationships of the Fourier transform allows a transform of two real functions, $h(n)$, and $g(n)$ can be done in-place with one complex Fourier transform, $FT[(h(n) + i \cdot g(n)]$.

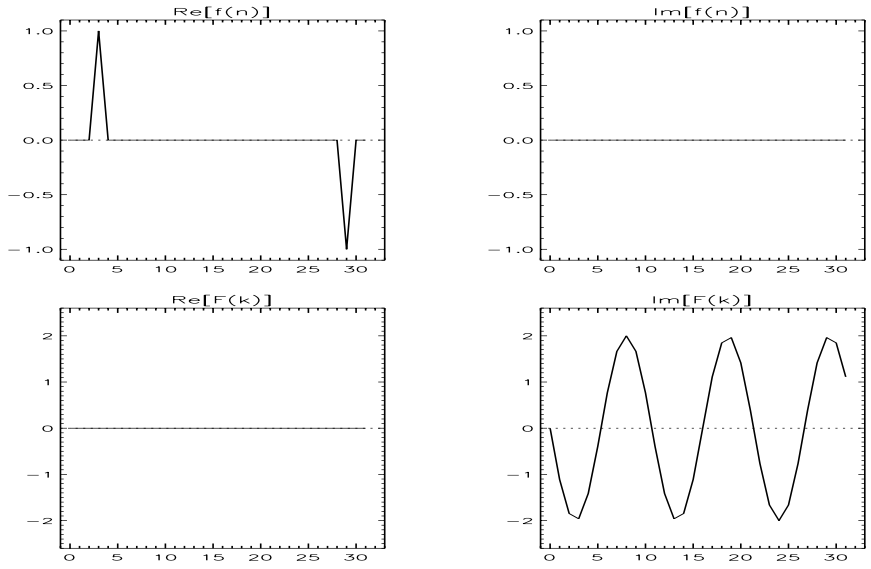


Figure 11.7: For an odd real function the Fourier transform is imaginary and odd.

$$H(k) \equiv FT[h(n)] \tag{11.63}$$

$$G(k) \equiv FT[g(n)] \tag{11.64}$$

We begin by assuming that the result of the FFT of the two functions is equal to the FFT of the individual function as follows

$$FT\left[h(n)\right] + i \cdot FT\left[g(n)\right] = FT\left[h(n) + i \cdot g(n)\right] \equiv R(k) + i \cdot I(k) \tag{11.65}$$

so that our original assumption can be written

$$\begin{aligned} R(k) + i \cdot I(k) &= \Re(H(k)) + i \cdot \Im(H(k)) + i \cdot (\Re(G(k)) + i \cdot \Im(G(k))) && (11.66) \\ &= [\Re(H(k)) - \Im(G(k))] + i \cdot [\Im(H(k)) + \Re(G(k))] && (11.67) \end{aligned}$$

Therefore, the symmetry relationships can be summarized as follows. If $h(n)$ and $g(n)$ are real and have even and odd symmetry, $h(n) = h^e(n) + h^o(n)$ and $g(n) = g^e(n) + g^o(n)$ then the orthogonality relationships discussed in the last section can be writtin

$$\begin{aligned} FT[h^e(n)] &= R^e(k) && (11.68) \\ FT[h^o(n)] &= I^o(k) && (11.69) \\ FT[i \cdot g^e(n)] &= I^e(k) && (11.70) \\ FT[i \cdot g^o(n)] &= R^o(k) && (11.71) \end{aligned}$$

Therefore, we can write and the FFT has symmetry properties such that

$$H(k) \equiv FT[h^e(n) + h^o(n)] = R^e(k) + i \cdot I^o(k) \tag{11.72}$$

and

$$\begin{aligned} G(k) \equiv FT[g^e(n) + g^o(k)] &= -i \cdot FT(i \cdot g^e + i \cdot g^o(n)] \\ = -i \cdot (R^o(k) + i \cdot I^e(k)) &= I^e(k) - i \cdot R^o(k) && (11.73) \end{aligned}$$

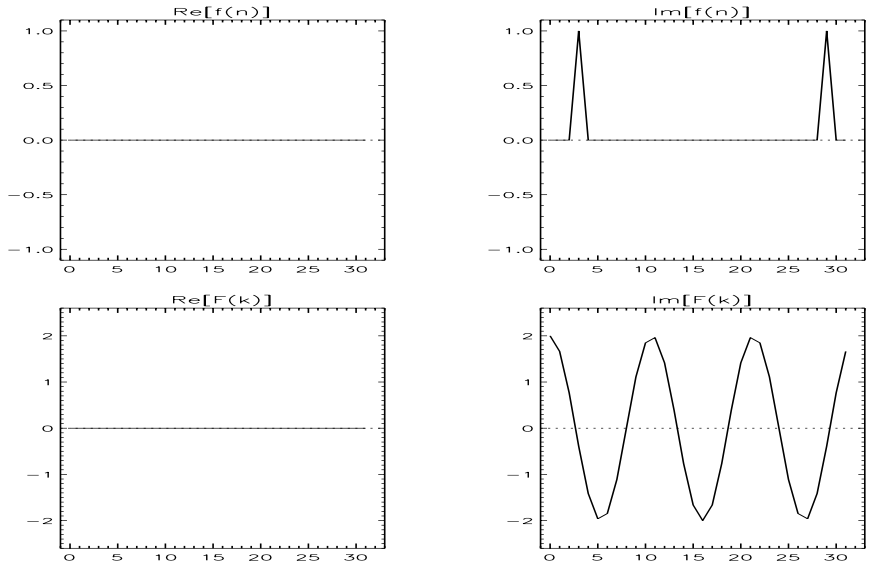


Figure 11.8: For an even imaginary real function the Fourier transform is imaginary and even.

therefore, the FFT of the composite complex function can be unraveled as follows

$$FT(h(n)) = H(k) = R^e(k) + i \cdot I^o(k) = \frac{R(k) + R(N-k)}{2} + i \cdot \frac{I(k) - I(N-k)}{2} \tag{11.74}$$

$$FT(g(n)) = G(k) = I^e(k) + i \cdot R^o(k) = \frac{I(k) + I(N-k)}{2} - i \cdot \frac{R(k) - R(N-k)}{2} \tag{11.75}$$

$$\textbf{NOTE}: \quad R(N) \equiv R(0) \qquad \textbf{and} \qquad I(N) \equiv I(0) \tag{11.76}$$

11.4.2 Transform of an N point real function with an $M = N/2$ complex FFT

NOTE: higher level lanquages, such as MATLAB, will do this automatically if the imaginary part of the array is all zeroes.

In this case we can split a real function into two blocks of even and odd functions and then perform the FFT on $N/2$ points. This save both memory and time. For a real function $x(n)$

$$X(k) = FT\left[x(n)\right] \qquad \text{for} \qquad k = 0, N-1 \qquad \text{and} \qquad n = 0, N-1 \tag{11.77}$$

and $\Re(X(k))$ will have even symmetry and $\Im(X(k))$ will have odd symmetry because $x(n)$ is a real function.. If we split up the even and odd components of the function

$$h(m) = x(2m) \qquad \text{where} \qquad 0 \le m \le \frac{N}{2} - 1, \tag{11.78}$$

$$g(m) = x(2m+1) \qquad \text{where} \qquad 0 \le m \le \frac{N}{2} - 1, \tag{11.79}$$

so that,

$$\begin{aligned} X(k) &= FT\left[x(n)\right] = H(k) + W_N^k \cdot G(k) \\ X(M+k) &= H(k) - W_N^{M+k} G(k) \\ &= H(k) - W_N^k G(k) \qquad \text{for} \quad k = 0, M-1 \end{aligned} \tag{11.80}$$

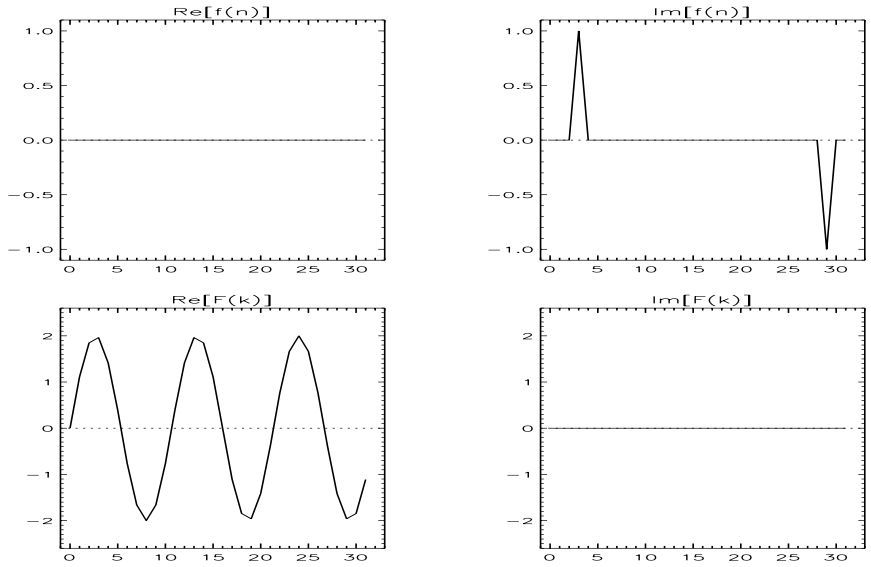


Figure 11.9: For an odd imaginary real function the Fourier transform is real and odd.

where the factor W_N^k arises due to the index $2n+1$ (see decimation introduction, Eqn. 11.50, ..or.. expand the $FT[]$ function with the definition of the DFT). So far, we have simply written the final pass of the FFT explictly. We can now utilize the symmetry of the FT to reduce the memory and execution time of the final pass.

$$H(k) = FT\left[h(m)\right] \qquad \text{and} \qquad G(k) = FT\left[g(m)\right] \tag{11.81}$$

We can expand the *real* transform for N points as follows.

$$w \equiv \frac{2\pi}{N} = \frac{\pi}{M} \tag{11.82}$$

$$\begin{aligned}
X(k) &= H(k) + W_N^k \cdot G(k) && (11.83)\\
&= \Re(H(k)) + i\cdot\Im(H(k) + (\cos(k\cdot w) - i\cdot\sin(k\cdot w))\cdot \\
&\quad (\Re(G(k)) + i\cdot\Im(G(k))) && (11.84)\\
&= \Re(H(k)) + \cos(k\cdot w)\cdot\Re(G(k)) + \sin(k\cdot w)\cdot\Im(G(k)) \\
&\quad + i\cdot[\Im(H(k)) + \cos(k\cdot w)\cdot\Im(G(k)) - \sin(k\cdot w)\cdot\Re(G(k))] && (11.85)\\
&= \Re(X(k)) + i\cdot\Im(X(k)) && (11.86)
\end{aligned}$$

so that, the $k = 0, M-1$ elements final pass are given by

$$\begin{aligned}
\Re(X(k)) &= \Re(H(k)) + \cos(k\cdot\nu)\cdot\Re(G(k)) + \sin(k\cdot\nu)\cdot\Im(G(k)), && (11.87)\\
\Im(X(k)) &= \Im(H(k)) + \cos(k\cdot\nu)\cdot\Im(G(k)) - \sin(k\cdot\nu)\cdot\Re(G(k)), && (11.88)\\
\Re(X(M+k)) &= \Re(H(k)) - \cos(k\cdot\nu)\cdot\Re(G(k)) - \sin(k\cdot\nu)\cdot\Im(G(k)), && (11.89)\\
\Im(X(M+k)) &= \Im(H(k)) - \cos(k\cdot\nu)\cdot\Im(G(k)) + \sin(k\cdot\nu)\cdot\Re(G(k)) && (11.90)
\end{aligned}$$

So that if we split the real function, $x(n)$, into even and odd components, $h(m)$ and $g(m)$, and then perform a complex Fourier transform of $(h(m), i\cdot g(m)$. The functions $h(m)$ and $g(m)$ have the even and odd symmetries discussed in the previous section of performing a composite FFT. The output from a complex FFT of $h(m) + i\cdot g(m)$ is written as

$$Y(k) \equiv FT\left(h(m), i \cdot g(m)\right) = R(k) + i \cdot I(k) \qquad \text{for} \quad k = 0, M-1 \tag{11.91}$$

and utilizing the even and odd symmetry of the composite functions

$$\begin{aligned} H(k) &= h^e(k) + i \cdot h^o(k) && (11.92)\\ &= \frac{R(k) + R(M-k)}{2} + i \cdot \frac{I(k) - I(M-k)}{2} && (11.93) \end{aligned}$$

where

$$R(M) = R(0) \qquad \textbf{and} \qquad I(M) = I(0) \tag{11.94}$$

and

$$\begin{aligned} G(k) &= g^e(k) - i \cdot g^o(k) && (11.95)\\ &= \frac{I(k) + I(M-k)}{2} - i \cdot \frac{R(k) - R(M-k)}{2} && (11.96) \end{aligned}$$

so we can expand Eqn 11.87 as

$$\begin{aligned} \Re(X(k)) &= \Re(H(k)) + \cos(k \cdot \nu) \cdot \Re(G(k)) + \sin(k \cdot \nu) \cdot \Im(G(k)) && (11.97)\\ &= \frac{R(k) + R(M-k)}{2} + \cos(k \cdot w) \cdot \frac{I(k) + I(M-k)}{2} \\ &\quad - \sin(k \cdot w) \cdot \frac{R(k) - R(M-k)}{2} && (11.98)\\ &= \frac{R(M-k) + R(k)}{2} + \cos(k \cdot w) \cdot \frac{I(M-k) + I(k)}{2} \\ &\quad + \sin(k \cdot w) \cdot \frac{R(M-k) - R(k)}{2} && (11.99) \end{aligned}$$

similarly Eqn. 11.88 becomes

$$\begin{aligned} \Im(X(k)) &= \Im(H(k)) + \cos(k \cdot \nu) \cdot \Im(G(k)) - \sin(k \cdot \nu) \cdot \Re(G(k)) && (11.100)\\ &= \frac{I(k) - I(M-k)}{2} - \cos(k \cdot w) \cdot \frac{R(k) - R(M-k)}{2} \\ &\quad - \sin(k \cdot w) \cdot \frac{I(k) - I(M-k)}{2} && (11.101)\\ &= \cos(k \cdot w) \cdot \frac{R(M-k) + R(k)}{2} + \sin(k \cdot w) \cdot \frac{I(M-k) - I(k)}{2} \\ &\quad - \frac{I(M-k) + I(k)}{2} && (11.102) \end{aligned}$$

and the equations for Eqn. 11.89 and Eqn. 11.90 are similar with the signs of the cosine and sine terms switched. Also, the inverse process can be shown to be the same except that the argument of W_N^k is replaced with W_N^{-k}.

To avoid computing $R(M)$ and $I(M)$ the $k = 0$ and $k = M$ points can be determined separately noting that $W_N^0 = 1$ and $W_N^M = -1$. Also, the real transform can be done "in-place" (*i.e.*, use output array, X, for intermediate array Y) by noting that

$$\begin{aligned} \cos\left(\frac{\pi \cdot k}{M}\right) &= -\cos\left(\frac{\pi \cdot (M-k)}{M}\right) \quad \textbf{and} && (11.103)\\ \sin\left(\frac{\pi \cdot k}{M}\right) &= +\sin\left(\frac{\pi \cdot (M-k)}{M}\right) && (11.104) \end{aligned}$$

The FORTRAN program (*ftp*/real_fft.for) in Section B.16 computes the DFT of a real function, fx(n), n=1,Npts, and returns the complex result, gx(k),k=1,Npts.

11.5 Applications of the Fourier Transform

11.5.1 Image Deconvolution

When the Hubble space telescope (HST) was launched in 1990 the mirror was found to be the wrong focal length. HST is a Ritchey-Chretien f/24 primary with a 2.4 meter focal length and weights 821 kg. The error in focal length was 1.3 mm which equates to 4 μm in "flatness" of the mirror. Fortunately, the images were perfectly out of focus (spherical aberration) so that the measured images could be corrected (albeit, computationally expensive). Images of stars were used to define the point spread function (PSF) of the cameras. The perfect mirror would have had a Gaussian PSF, called $P_p(x,y)$ herein, with a half-width equal to 2 pixels (Nyquist sampled image) while the HST's PSF, called $P_h(x,y)$ spread over a larger area. Since the PSF is convolved with the infinite resolution "image" to produce the image measured by the CCR, the HST images were blurred.

We can use the convolution theorem to remove the blurring. This approach was used initially to correct the HST images (prior to fixing the optics). The measured image, $I_m(x,y)$ is given by

$$I_m(x,y) = \int_{x'} \int_{y'} P_m(x',y') \cdot I_0(x-x', y-y') dx'\ dy' = P_m(x,y) \otimes I_0(x,y) \tag{11.105}$$

Using Eq. 11.15 we can take the 2-dimension Fourier transform, $FT()$, of both the PSF, P_m, and the measured image I_m to find the theoretical image (at infinite spatial resolution).

$$FT(P_m(x,y)) \cdot FT(I_0(x,y)) = FT(I_m(x,y)) \tag{11.106}$$

$$FT(I_0(x,y)) = \frac{FT(I_m(x,y))}{FT(P_m(x,y))} \tag{11.107}$$

where the double $FT(U(x,y))$ can be computed with FFT's as follows

- For each row y_k, compute the FFT($G_k(x)$) where $G_k(x) = U(x,y_k)$ and put the result back into $U(x,y_k)$
- Then for each column compute the FFT($G_k(y)$) where $G_k(y) = U(x_k,y)$ and put the result back into $U(x_k,y)$

The we can compute the perfect image for the HST with the theoretically perfect PSF, $P_p(x,y)$, and inverse 2-d Fourier transform, $FT^{-1}[]$, of the result to obtain a de-blurred HST image.

$$I(x,y) = FT^{-1}\left[FT(P_p(x,y) \cdot \frac{FT(I_m(x,y))}{FT(P_m(x,y))} \right] \tag{11.108}$$

This method works well for high signal-to-noise images; however, when signal-to-noise is low the signal and noise are given equal weight in this method. The algorithm of Lucy [1974] iterates the processing of the image; however, it ensures that the image flux remains positive. It effectively removes the high frequency noise that is introduced in the Fourier devonvolution process. Unfortunately, many iterations are required with the Lucy algorithm which makes it very computationally expensive.

See the IDL program in the ftp site (phys640/lucy.pro) for an example of the Lucy algorithm as applied to the Hubble Space Telescope images.

11.5.2 Mass Spectrometer

The Fourier Transform ion-cyclotron mass spectrometer was developed in 1974 by Melvin Comisarow and Alan Marshall. Pre-stage ionizers are used and all cyclotron frequencies are measured simultaneously in a constant magentic field. A Fourier transform of the signal yields the mass spectrum.

$$\vec{F} = q \cdot \vec{v} \times \vec{B} = \frac{m \cdot v^2}{r} \tag{11.109}$$

$$f_i = \frac{v}{r} = \frac{B_0 \cdot z_i}{2\pi \cdot m_i} \tag{11.110}$$

11.5.3 Processing interferometer signals

An interferometer splits an incoming collimated beam of photons with a beamsplitter. This is illustrated in Fig. 11.10. Path "A" is reflected from a fixed mirror and path "B" is reflected from a moving mirror with a displacement, δ, from the zero path difference location, shown as a dotted line in the figure. Each beam returns to the beamsplitter and is either directed out of the instrument or into the detector.

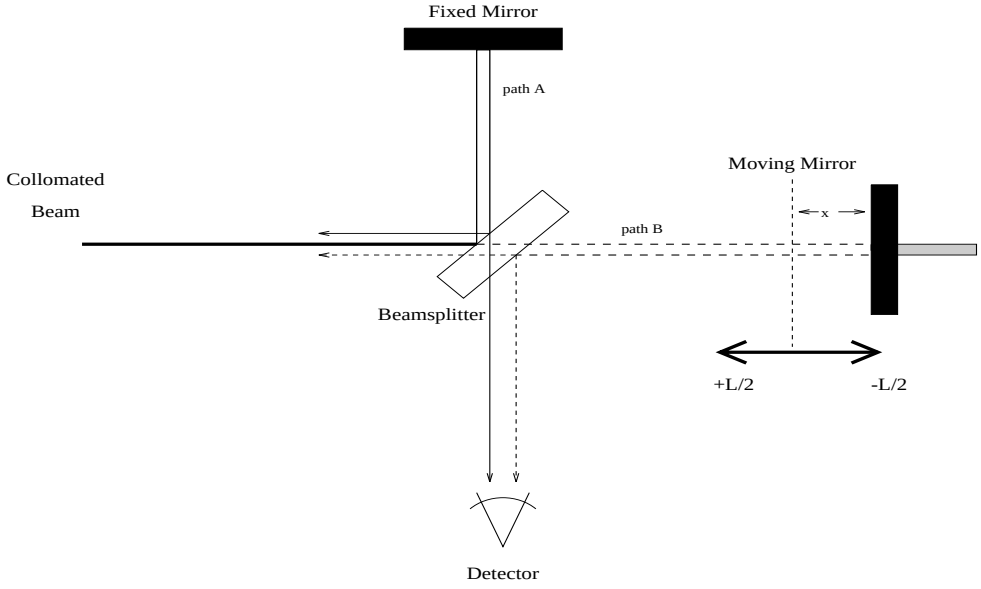


Figure 11.10: Illustration of an interferometer

If the optical path difference, $x = 2 \cdot \delta$, is zero, light from the two paths will constuctively interfere. If the optical path difference is equal to half a wavelength, $x = \lambda/2$, then the beams following the two paths will sum to zero, since path "A" is out of phase with path "B". In general, the interference between path "A" and "B" for light of wavenumber ν is given by

$$I(\nu, x) = N(\nu)[1 + \cos(2\pi\nu x)] \tag{11.111}$$

where $\nu = 1/\lambda$ is typically given in units of cm^{-1}. The detector is sensitive to photon flux, $N(\nu)$, which is calculated from the incoming Earth radiance, $R(\nu)$, typically given in units of milli-Watts $\cdot$ M$^{-2}\cdot$ steradian$^{-1}\cdot$ cm^{-1}

$$N(\nu) = \frac{1}{4} \cdot A \cdot \nu \cdot \frac{R(\nu)}{(h \cdot c) \cdot \nu} \cdot W_B(\nu) \tag{11.112}$$

where A is the detector area in meter^2, ν is the field of view in steradians, c is the speed of light ($3.0 \cdot 10^{10}$ cm/second), h is Planck's constant ($6.62620 \cdot 10^{-34}$ Joule $\cdot$ second), and $W_B(\nu)$ is the optical throughput resulting from transmittance of the beamsplitter, mirrors, windowing filter, and detector response function. In a Michelson Interferometer the factor of $\frac{1}{4}$ arises due to the incoming beam being divided into four paths. Two of the paths interfere while the other two paths exit the interferometer, as seen in Figure 1. Therefore, half the signal is always lost. Eqn. 11.111 can be integrated over all frequency to give

$$I(x) = \int_0^\infty N(\nu) d\nu + \int_0^\infty N(\nu) \cos(2\pi\nu x) d\nu. \tag{11.113}$$

The first term is not a function of the mirror displacement and represents a constant contribution at the detector for a given scene. This term contributes significantly to the noise and is scene dependent (*i.e.,* it is a function of the integrated incoming photon intensity). The component of Eqn. 11.113 varying with x can be written

$$I'(x) = \int_0^\infty N(\nu) \cos(2\pi\nu x) d\nu, \tag{11.114}$$

so that Eqn. 11.113 becomes

$$I(x) = I'(0) + I'(x) \tag{11.115}$$

Forward COSINE Transform: for use in transforming a measured interferogram into a radiance spectrum

$$I(x) \quad \text{is defined for} \quad -L < x \le L, \qquad L = \frac{N}{2} \cdot \Delta x \tag{11.116}$$

The definition of a continuous forward cosine transform over period, P, is given by

$$C(\nu) \equiv \frac{1}{P} \cdot \int_0^P f(x) \cdot \cos(\nu \cdot \frac{\pi x}{P}) \cdot dx \tag{11.117}$$

A discrete cosine transform is a special case of an FFT. The trapezoidal integration of Eqn. 11.117 is given by

$$C(k) \simeq \frac{f(0) + f(N) \cdot \cos(\pi k)}{2N} + \frac{1}{N} \sum_{n=1}^{N-1} f(n) \cdot \cos(k \cdot \frac{\pi i}{N}). \tag{11.118}$$

If we re-format our function $f(n)$ with $M = 2 \cdot N$ points, $g(m)$, from our data of $N + 1$ points, $f(n)$, as follows

- $g(n) = f(n) \quad \text{for } n = 0, N$
- $g(2N - n) = f(n) \quad \text{for } n = 1, N - 1.$

then we have built an even function from $f(n)$. The definition of a Forward FFT is given by

$$G(k) = \text{FFT}(\text{array}, -1) \equiv \frac{1}{M} \sum_{m=0}^{M-1} g(m) \cdot e^{-i \cdot k \cdot \frac{2\pi m}{M}} \quad \text{for } k = 0, M - 1 \tag{11.119}$$

In general, $G(k)$ has real and imaginary components; however, the even symmetry of the real function $g(m)$ results in a transformed function that is both real and even. We can substitute our even $g(m)$ function derived from $f(n)$ into Eqn. 11.119 to obtain

$$G(k) \equiv \text{FFT}(g(m), -1) \tag{11.120}$$

$$= \frac{1}{M}\sum_{m=0}^{M-1}\left[g(m)\cdot\exp\left(\frac{i\cdot 2\pi\cdot k\cdot m}{M}\right)\right] \tag{11.121}$$

$$= \frac{1}{2N}\sum_{m=1}^{2N-1}\left[g(m)\cdot\left(\cos\left(\frac{\pi\cdot k\cdot m}{N}\right) \quad + \quad i\cdot\sin\left(\frac{\pi\cdot k\cdot m}{N}\right)\right)\right] \tag{11.122}$$

$$= \frac{f(0)}{2N} + \frac{f(N)\cdot\cos(\pi k)}{2N} + \frac{1}{N}\sum_{n=1}^{N-1}\left(f(n)\cdot\cos\left(k\cdot\frac{\pi n}{N}\right)\right) \tag{11.123}$$

We can see that the lower $N+1$ points of the forward FFT is exactly equal to the discrete trapezoidal cosine transform. The data array, $f(n)$, should be created such that the number of points is related to a power of 2, $N = 2^j$ (NOTE, this is 2^j+1 data points), so that the radix-2 FFT operates on an even power of 2 number of points, $M = 2N = 2^{j+1}$. The first point in the array is the first point in the interferogram, $f(1) = I(x = 0)$, and the last point in the array is equal to the interferogram at the maximum OPD, $f(N) = I(x = L)$. Note that the full interferogram, $g(i)$, is assumed to be symmetric about $f(1)$ and that if the interferogram were not symmetric then the upper half of $g(2N - i) \neq f(i)$. This would introduce imaginary components to the radiances which are interpreted as phase errors in the interferogram.

The radiances produced by the forward cosine transform are spaced on a frequency grid of $\nu(0) = 0$ and $\nu(N) = \frac{1}{2\cdot\Delta x}$. Truncation of the interferogram is achieved by selecting $x(N') = L$ and solving the forward transform for N' points. The upper frequency is unchanged, since the interferogram spacing has not changed; however, the frequency spacing is now $\Delta\nu' = \frac{1}{2\cdot x(N')}$ instead of $\Delta\nu = \frac{1}{2\cdot x(N)}$.

It is possible to create an interferogram so that $\nu(0) \neq 0$; however, this is not recommended because it significantly alters the effective channel response function of the cosine transform to something that is not physical. The frequency limits resulting from a forward transform are always identical to those used to create the interferogram. A real interferogram operates on radiances from $0 \leq \nu \leq \infty$ and, therefore, has a frequency axis derived from the sampling, Δx, and number of points in the interferogram $N' + 1$.

Inverse COSINE Transform: for use in simulating an interferometer (*i.e.,* transforming monochromatic radiance spectrum into an interferogram).

In Eqn. 11.114 we showed than an interferometer, $f(x)$, is mathematically equivalent to an inverse cosine transform of the radiance spectrum, $F(\nu)$. In order to simulate an interferogram we begin by creating an even function from an array of monochromatic radiances. We make the radiance spectrum an even function which allows the use of an FFT algorithm to force an even interferogram without introducing phase errors. Since monochromatic radiance arrays are typically very large (especially with zero filling) the use of a FFT is an extreme savings in execution time of the transform. As before, the even function is equal to

- $G(k) = F(k) \quad \text{for } k = 0, N$
- $G(2N - k) = F(k) \quad \text{for } k = 1, N - 1$
- NOTE: typically $F(0) = 0 \quad \text{and} \quad F(N) = 0$

The definition of an Inverse FFT for $M = 2N - 1$ points is

$$g(x) = \text{FFT(G(k)}, +1) \equiv \sum_{k=0}^{M-1} G(k)\cdot e^{i\cdot x\cdot\frac{2\pi k}{M}} \quad \text{for } x = 0, M-1. \tag{11.124}$$

Substitution of the real even symmetric function $g(x)$ results in a real even symmetric transform which is identical to the definition of a discrete cosine transform.

$$f(n) = F(0) + F(N) \cdot \cos(\pi \cdot n) + 2 \cdot \sum_{k=1}^{N-1} \left(F(k) \cdot \cos(n \cdot \frac{\pi \cdot k}{N}) \right) \tag{11.125}$$

Typically, this transform is very large. Monochromatic radiances are defined from $(\nu(0), F(0))$ to $(\nu(k), F(k))$ and are transformed to an interferogram which goes from $x(n = 0) = 0$ to $x(n = N) = \frac{1}{2 \cdot \Delta\nu}$, where $\Delta\nu$ is the spacing of the monochromatic data. Notice that the initial frequency axis, $\nu(0)$ to $\nu(N)$, is implicitly used to create the interferogram.

The cosine transform assumes that the function is periodic within the interval $\nu(N) - \nu(0)$ and that the transform of another interval (*i.e.*, $\nu((L+1) \cdot n) - \nu(L \cdot n)$ where L is an integer), would result in exactly the same interferogram. It is recommended that the frequency axis start at zero, $\nu(0) \equiv 0$ to avoid complications with the effective channel response function. The $\mathrm{sinc}(z)$ term in the channel response function is implicit referenced to $\nu(0)$ and if this were not zero it would create strongly asymmetric channel response functions which are channel dependent over the interval chosen. For cases where $\nu(0) \neq 0$ the parameter z is equal to

$$z \equiv 2\pi L \cdot (\nu + \nu_i - \nu(0)), \tag{11.126}$$

11.5.4 Nuclear Magnetic Resonance

Chapter 12

Apodization

This chapter is a reference document that compares and contrasts apodization functions found in the literature. One should recognize that the process of apodization, to be defined and discussed in detail in this chapter, is usually a *reversible* process, and as such, does not alter the information content of the spectrum. Apodization, therefore, is a matter of convenience and the choice of an appropriate apodization function depends on the application. In remote sounding, for example, we may want to select measurements (*i.e.*, channels) that are optimized for sensitivity to the parameters of interest (*e.g.* temperature) and are also least sensitive to interfering parameters (*e.g.*, water, ozone, etc.). The "spectral purity" of a single channel is determined by apodization and we will see that there is a trade-off to spectral sensitivity and spectral interference. One may argue that if one could afford to use all the channels there is no advantage to apodization; however, it should be pointed out that if an algorithm is sensitive to the knowledge of the interfering parameters influence (*i.e.*, the need for an accurate error covariance) then the use of an apodized subset of channels could have an advantage over use of the entire spectrum. The computational advantage of using a sub-set of apodized channels is usually reason enough.

As an introduction to the subject I find the introductory paragraphs from a paper by Norton and Beer (1976) to be most appropriate:

Practitioners of Fourier spectrometry and, indeed, most individuals employing numerical Fourier transforms commonly employ apodization to improve the appearance of their outputs. All users are aware that such a damping of secondary maxima entails the inevitable smearing of the output. To the Fourier spectrometrist, the smearing manifests itself as a loss of spectral resolution, a loss that strikes him particularly hard because of the effort entailed in achieving a satisfactory spectral resolution in the first place. There is, of course, no real need to apodize at all: The unapodized, point-by-point output of a Fourier spectrometer system contains all the information derivable from the interferogram. Nonetheless, spectral analysts more familiar with the "smooth" output of scanning spectrometers dislike the "unsmooth" output intensely and demand that it be smoothed and interpolated. The apodization function is commonly used for this purpose, but with the foregoing in mind, Fourier spectrometrists are more conscious that most of the need to find and use the most suitable functions.

. . .

In 1967, Fellgett said "... the orthogonal properties of the sinc functiion ... are easily destroyed by apodization by apodization. This is way I believe that apodization should be done only by experts." Notwithstanding such structures, apodization is undertaken daily by experts and nonexperts alike. The fact that the destruction of the orthogonality of the since function is indisputable. In the present context, the consequence is that the output points following the Fourier transformation process no longer are statistically independent. Calculation of parameters such as signal-to-noise ratio must therefore proceed with caution.

In the previous chapter we introduced the concept of a discrete Fourier transform and discussed the symmetry properties. In this chapter we will be discussing the effect, in spectral space, on truncating an interferogram. While the application used in this chapter is remote sensing, the discussion in this chapter applies to any transform where the limits of the integral have been trucated. In general, the apodization function, $A(x)$, is a function that is multiplied by the argument of the Fourier transform to enhance the

physical interpretation of the measurement,

In real instruments, the optical path can induce its own apodization function. Imagine an interferometer in which the reflecting mirrors and beamsplitter does not fill the detector field of view. In that case, as the optical path delay becomes larger the signal is attenuated. We call this self-apodization and in the following discussions it will be assumed that this effect has been removed, that is the interferogram has been "de-apodized" and has the theoretical boxcar apodization or sinc() function. We will discuss the following apodization functions in the following sections

Section	Apodization Function
12.2	Boxcar
12.3	Triangular
12.4	von Hann
12.5	Hamming
12.6	Blackmann
12.7	Kaiser-Bessel
12.8	Dolph Chebyshev
12.9	Gaussian
12.10	Autoregressive spectral estimator (ASE)
12.11	Norton-Beer

First we will begin by comparing the effect of different apodizing functins in interferometer delay space on the spectral response in frequency space. In the lower panel of Fig. 12.1 we show the the boxcar function of width $\pm L$ as a dashed-dot line. Since these functions are symmetric, only the positive side is shown. The Hamming apodization function is equal to $0.54 + 0.46 \cdot \cos(\pi x/L)$ and is shown as a dashed line. A truncated Gaussian is shown as a solid line. Also shown is the cosine transform (*i.e.*, an interferogram) of a typical mid-latitude terrestrial spectrum for a bandpass of 600 to 1200 wavenumbers. The structures at x=0.65 and x =1.3 are due to the strong CO_2 absorption in this bandpass. Water vapor and ozone lines also contribute to the information content within this interferogram.

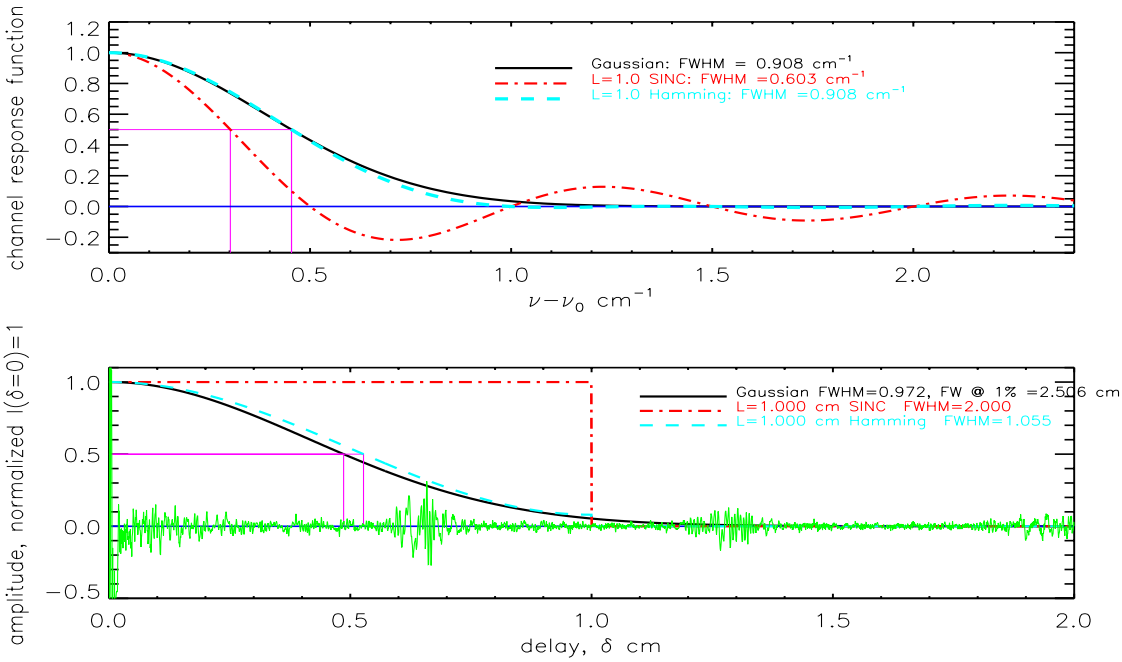


Figure 12.1: Example of an interferogram with the boxcar, Hamming, and truncated Gaussian apodization functions. The upper panel shows the channel response functions (see text)

The cosine transforms of these CSRF's are shown in the upper panel of the same figure. Again, since these functions are symmetric, only the positive side is shown. The boxcar apodization function function of width $\pm L$ transforms to a sinc(y) with a full-width-half-maximum (FWHM) of $0.6/L$. The Hamming apodization function transforms to well behaved function with very small side-lobes. The un-truncated Gaussian apodization function transforms to a Gaussian CSRF in spectral space.

Therefore, if a spectrally pure CSRF is desired and apodization function can be selected to minimize the side-lobes. This reduced side lobes do not come without a penalty; however, as can be seen in the upper panel of the previous figure. The FWHM of the Hamming function is $0.9/L$ or 50% larger than the FWHM of the unapodized CSRF. The un-truncated Gaussian has no side-lobes; however, we will find that it is not the most optimum function, for a fixed maximum optical path difference, that is the truncated-Gaussian will have large side lobes.

In Fig. 12.2 we show many of the apodization functions which will be discussed in this section. For each apodization function (or range of functions if it is a tunable apodization function) we show the FWHM as a function of the absolute value of the normalized height (peak of central lobe equals one) of the largest side-lobe.

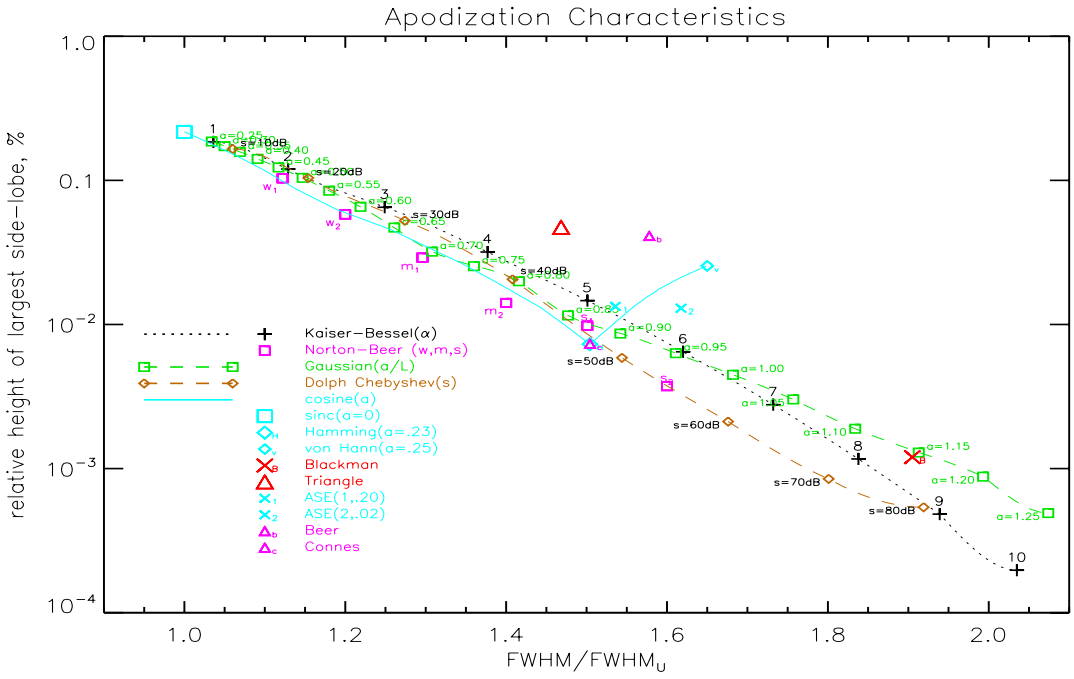


Figure 12.2: Tradeoff between FWHM and size of side lobes for common apodization functions

The ratio of the area of the central lobe to the total area of the CSRF is another useful measure of the trade-off between side-lobes and shown in Fig. 12.3 in percent as a function of the ratio of the FWHM to FWHM of an unapodized (FWHM$_U$) spectrum for some of the apodization functions discussed here. This is a measure of the effect of the side-lobes.

Transform Pairs

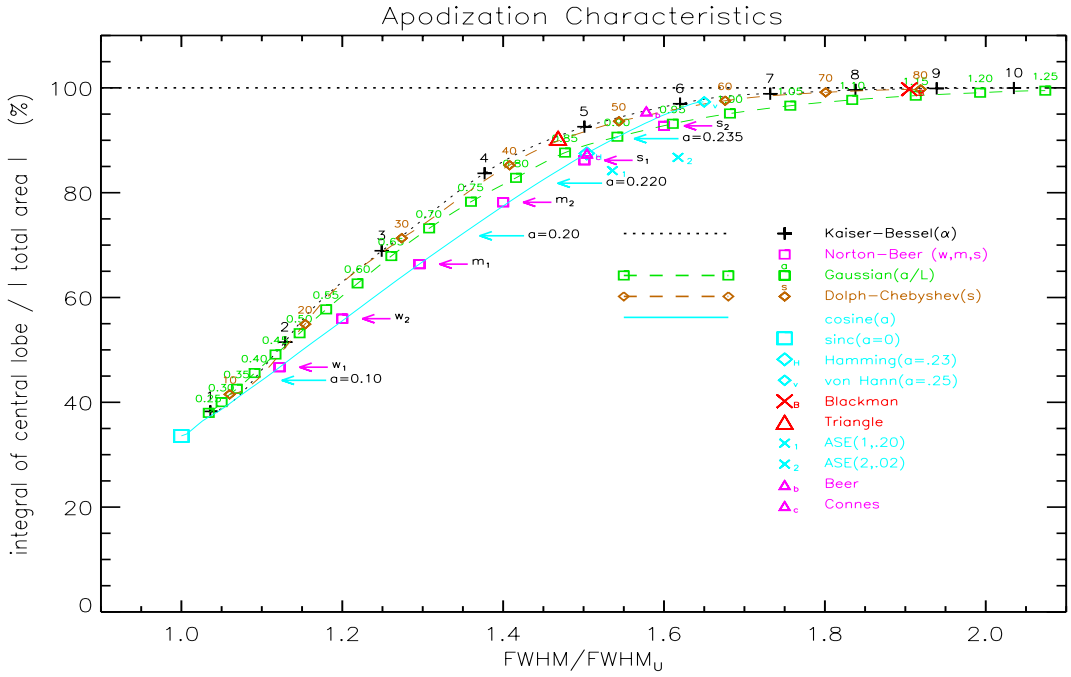


Figure 12.3: Tradeoff between FWHM and percent of signal in central lobe for common apodization functions

spectral(ν)	delay(x)
$f(\nu) = 2\int_0^\infty F(x)\cdot\cos(\pi\cdot\nu\cdot x)dx$	$F(x) = \int_{-\infty}^{\infty} f(\nu)\cdot\cos(\pi\cdot\nu\cdot x)d\nu$
$f(\nu) = \exp(-a\cdot\nu^2)$ $a = -4\log_e(0.5)/\mathrm{FWHM}_\nu^2$	$F(x) = B\cdot\exp(-b\cdot x^2)$ $B = \sqrt{a/\pi}$ $b = -4\log_e(0.5)/\mathrm{FWHM}_x^2 = \pi^2/a$ $\mathrm{FWHM}_x = -4\log_e(0.5)/\pi/\mathrm{FWHM}_\nu$
$f(\nu) = \frac{\sin(y)}{y}$ $y \equiv 2\pi\cdot L\cdot(\nu-\nu_i)$	$F(x) = \frac{1}{2\cdot L}$ $-L \le x \le L$
$f(\nu) = \frac{\sin(y)}{y}\left(1+\frac{0.46}{0.54}\cdot\frac{y^2}{\pi^2-y^2}\right)$	$F(x) = \frac{0.54+0.46\cdot\cos(\pi\cdot x/L)}{2\cdot L\cdot 0.54}$

12.1 Computing the Channel Spectral Response Function (CSRF)

To calculate channel averaged radiances for an interferometer, $R(i)$, from monochromatic radiances, $R(\nu)$, there are two methods which can be employed. Both methods begin with the monochromatic radiance and the instrument windowing function, $W_B(\nu)$, which truncates the Fourier transform to a finite definite integral.

- discrete cosine transform (DCT) method
  1. create an interferogram
  $$I(x) = \mathrm{DCT}^{-1}(R(\nu)\cdot W_B(\nu)) \tag{12.1}$$
  2. apply the apodization function, $A(x)$
  3. calculate the channel radiance from the cosine transform
  $$R(i) = \mathrm{DCT}(I(x)\cdot A(x)) \tag{12.2}$$

- convolution method
  1. calculate the channel response function, $\Phi_U(\nu - \nu_i)$, for the interferometer channel, ν_i
  2. convolve the radiance with the channel response function

$$R(i) \equiv \frac{\int_0^\infty \Phi(\nu - \nu_i) \cdot R(\nu) \cdot d\nu}{\int_0^\infty \Phi(\nu - \nu_i) \cdot d\nu} \tag{12.3}$$

Both of these methods have subtle numerical problems which require great care in simulating accurate channel radiances. In the case of the discrete cosine transform method the following problems arise:

1. a DCT, not FFT, must be used for both the inverse and forward transform. See final section for FFT's of even functions.
2. For the DCT the frequency range, (ν_1, ν_2), must be specified. Typically, for FFT's the number of points is a power of 2. Therefore, the DCT operates on $N = 1 + (\nu_2 - \nu_1)/\Delta\nu = 1 + 2^k$ points and the FFT will operate on $2 \cdot 2^k$ points.
3. The zero frequency point of the discrete transform should be set to $\nu_1 = 0$. It is possible to save memory by having a non-zero first frequency; however, this alters the effective channel response function of the instrument.
4. Trapezoidal integration errors can become significant for numerical calculations. The spacing in interferogram space is given by $\Delta x = \frac{1}{2\nu_2}$. By zero filling the monochromatic radiance, the interferogram spacing is smaller and the final DCT trapezoidal integration is more accurate.

Convolutions must also be done carefully. The following items apply:

1. The convolution must be done over the entire band. Truncation of the sinc function will produce significant errors.
2. The sinc function must be normalized over the full limits of the function and not simply within the band limits. Numerical normalizations must be truncated at the band limits and will introduce significant errors. The analytic normalization is strongly recommended

$$\int_{-\infty}^{\infty} \mathrm{sinc}(2\pi \cdot L \cdot \Delta\nu) d\Delta\nu = \frac{1}{2 \cdot L} \tag{12.4}$$

3. The sinc function must be calculated as accurately as possible. Interpolation of the sinc function or radiances will introduce error on the order of 0.05° NEΔT for Earth scenes. The most accurate method is to calculate the sinc function exactly at the frequency of the radiance data for the given channel center. If $\nu(j)$ is the frequency array associated with the radiance data, $R(j)$, evenly spaced by $\Delta\nu$, and ν_i is the frequency of the channel center, the numerical convolution is given by

$$R(i) = \frac{1}{2 \cdot L \cdot \Delta\nu} \left[\frac{R(1) \cdot \Phi(\nu(1), \nu_i)}{2} + \frac{R(J) \cdot \Phi(\nu(J), \nu_i)}{2} + \sum_{j=2}^{J-1} R(j) \cdot \Phi(\nu(j), \nu_i) \right] \tag{12.5}$$

4. For high accuracy the $\mathrm{sinc}(2\pi \cdot L \cdot (\nu + \nu_i))$ component must be included. Failure to include this term will introduce errors on the order of $NE\Delta T \approx 0.02°$ for an Earth scene in the longwave IR bands.

12.2 Truncation (the sinc() function)

An interferometer is operated over a finite range of optical delay, $-L \leq x \leq L$, where L is the maximum optical path difference (OPD) of the instrument. Eqn. 11.114 is an inverse cosine transform of the photon flux. To calculate radiances from an interferogram requires a forward cosine transform Eqn. 11.114 over the range $-\infty \leq x \leq \infty$. Therefore, in a real instrument the infinite cosine series is truncated. In general, this process is called apodization and the apodization function for "unapodized" radiances is given by

$$A(x) = 1 \quad \text{for } |x| \leq L \quad \text{else} \quad A(x) = 0 \tag{12.6}$$

The physical limitation of the OPD results in a channel spectral response function (CSRF). If we take the forward cosine transform of the apodization function we obtain the CSRF.

$$\begin{aligned} \Phi_U(\nu - \nu_i) &= \frac{2 \int_0^\infty A(x) \cdot \cos(2\pi\nu_i x) \cos(2\pi\nu x) dx}{\int_0^\infty A(x) dx} \\ &= \frac{2}{L} \int_0^L \cos(2\pi\nu_i x) \cos(2\pi\nu x) dx \\ &= \frac{\sin(2\pi L \cdot (\nu - \nu_i))}{2\pi L \cdot (\nu - \nu_i)} + \frac{\sin(2\pi L \cdot (\nu + \nu_i))}{2\pi L \cdot (\nu + \nu_i)} \quad \text{CRC\#317} \end{aligned} \tag{12.7}$$

If we define the difference and sum of frequencies as

$$y \equiv 2\pi L \cdot (\nu - \nu_i), \quad \text{and} \tag{12.8}$$

$$z \equiv 2\pi L \cdot (\nu + \nu_i), \tag{12.9}$$

then the channel response function can be written as

$$\Phi_U(\nu - \nu_i) = \text{sinc}(y) + \text{sinc}(z) \tag{12.10}$$

where the $\text{sinc}(x)$ function is defined as

$$\text{sinc}(x) \equiv \frac{\sin(x)}{x} \tag{12.11}$$

There is a indeterminate point at $y = 0$

$$\lim_{y \to 0} (\text{sinc}(y)) = 1 \tag{12.12}$$

$$A(x) = 1 \quad \text{for } \cdot x \leq L \quad \text{else} \quad A(x) = 0 \tag{12.13}$$

The approximate effective instrument profile is given as

$$\Phi(\Delta\nu) = \frac{\sin(2\pi L \cdot \Delta\nu)}{2\pi L \cdot \Delta\nu} = \frac{\sin(y)}{y} \equiv \text{sinc}(y) \tag{12.14}$$

The true instrument profile is given by

$$\frac{2}{L} \int_0^L \cos(2\pi\nu_0 x) \cos(2\pi\nu x) dx = \frac{\sin(2\pi L(\nu - \nu_0))}{2\pi L(\nu - \nu_0)} + \frac{\sin(2\pi L(\nu + \nu_0))}{2\pi L(\nu + \nu_0)} \quad \text{CRC\#317} \tag{12.15}$$

The Full-Width-at-Half-Maximum (FWHM) can be found by solving $\Phi(\Delta\nu) = 0.5$, FWHM $= 2 \cdot \Delta\nu$

$$\text{FWHM} = 2 \cdot \frac{0.30167725}{L} = \frac{0.603355}{L} \simeq \frac{1.2}{2L} \tag{12.16}$$

$$\sin(x) = \sum_{j=0}^{\infty} (-1)^j \cdot \frac{x^{2j+1}}{(2j+1)!} \simeq x - \frac{x^3}{(3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots = x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \ldots \tag{12.17}$$

$$\text{sinc}(x) = \sum_{j=0}^{\infty} (-1)^j \cdot \frac{x^{2j}}{(2j+1)!} \simeq 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!} + \ldots = 1 - \frac{x^2}{6} + \frac{x^4}{120} - \frac{x^6}{5040} + \ldots \tag{12.18}$$

$$\lim_{x \to 0} \frac{\sin(x)}{x} = 1 \tag{12.19}$$

$$\int \frac{\sin(x)}{x} dx = \sum_{j=0}^{\infty} (-1)^j \cdot \frac{x^{2j+1}}{(2j+1) \cdot (2j+1)!} \quad \text{CRC\#621} \tag{12.20}$$

$$\frac{1}{a} \cdot \int_{-\infty}^{\infty} \frac{\sin(ax)}{x} dx = \frac{\pi}{a} \quad \text{CRC\#621} \tag{12.21}$$

The normalization integral for evenly spaced points, $\Delta\nu = i \cdot \delta\nu$, is given by

$$\delta\nu \cdot \sum_{i \to -\infty}^{i \to \infty} \frac{\sin(2\pi L \cdot i \cdot \delta\nu)}{2\pi L \cdot i \cdot \delta\nu} = \frac{1}{2L} \tag{12.22}$$

Zeroes of the function are at $\Delta\nu \cdot L = n/2$
Maxima are found by taking the derivative of $\sin(y)/y$ w.r.t. y

$$y_{max} = \tan(y_{max}) \tag{12.23}$$

Maxima of $\sin(y)/y$

n	$\nu \cdot L$	%	n	$\nu \cdot L$	%	n	$\nu \cdot L$	%
2	0.715148	-21.72336	3	1.229512	12.83746	4	1.735445	-9.13252
5	2.238704	7.09135	6	2.740768	-5.79718	7	3.242193	4.90296
8	3.743237	-4.24796	9	4.244034	3.74745	10	4.744663	-3.35251
11	5.245173	3.03292	12	5.745593	-2.76897	13	6.245945	2.54731
14	6.746246	-2.35851	15	7.246505	2.19577	16	7.746730	-2.05405
17	8.246928	1.92951	18	8.747105	-1.81921	19	9.247260	1.72085
20	9.747401	-1.63258	21	10.247529	1.55292	22	10.747643	-1.48067
23	11.247747	1.41485	24	11.747844	-1.35463	25	12.247931	1.29933
26	12.748013	-1.24837	27	13.248088	1.20126	28	13.748158	-1.15757
29	14.248222	1.11695	30	14.748282	-1.07908	31	15.248338	1.04370

12.3 Triangle apodization

$$A(x) = \frac{L - |x|}{L} \quad \text{for } x \leq L \quad \text{else} \quad A(x) = 0 \tag{12.24}$$

The effective instrument profile is given as

$$\Phi(\Delta\nu) = \frac{\sin^2(\pi L \cdot \Delta\nu)}{(\pi L \cdot \Delta\nu)^2} \tag{12.25}$$

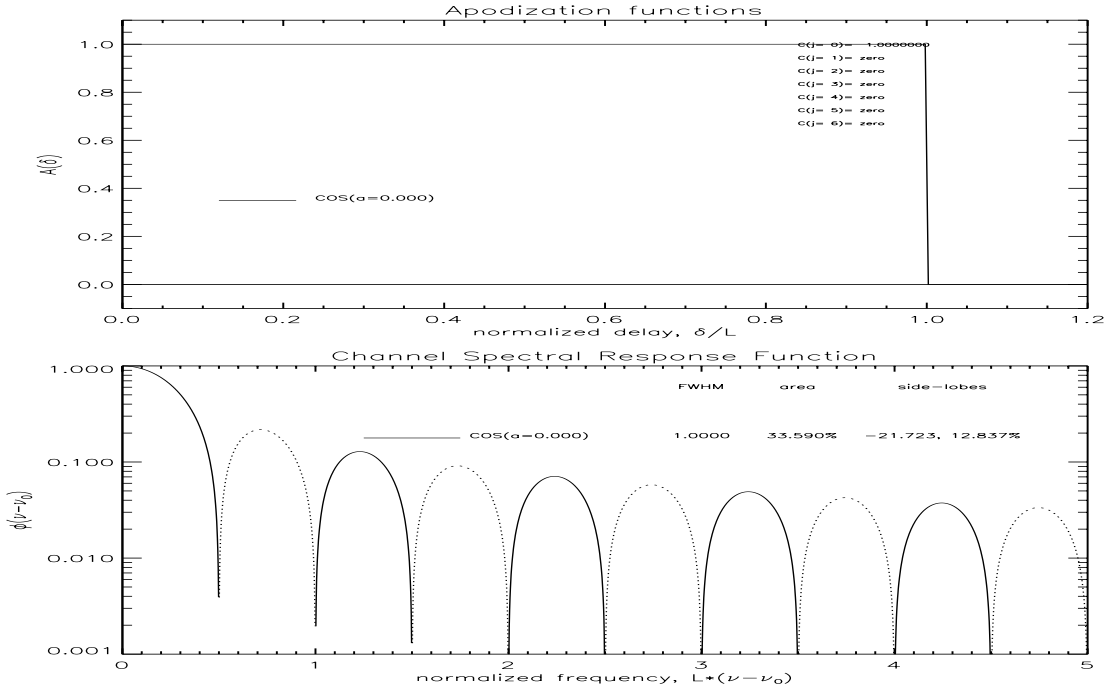


Figure 12.4: Channel response function for boxcar apodization

$$\int_0^\infty \frac{\sin^2(ax)}{x^2} dx = \frac{a\pi}{2} \quad \text{CRC\#630} \tag{12.26}$$

$$\text{FWHM} = 2 \cdot \Delta\nu = 2 \cdot 0.44294645836/L = 0.88589292/L = 1.468278 \cdot \sin(x)/x \tag{12.27}$$

Zeroes of the function are at $\Delta\nu \cdot L = n$
Maxima satisfy the condition:

$$\pi L \cdot \Delta\nu = \tan(\pi L \cdot \Delta\nu) \tag{12.28}$$

Maxima of $(\sin(y)/y)^2$

n	$\nu \cdot L$	%	n	$\nu \cdot L$	%
2	1.430297	4.71905	3	2.459024	1.64800
4	3.470890	0.83403	5	4.477408	0.50287
6	5.481536	0.33607	7	6.484387	0.24039
8	7.486474	0.18045	9	8.488069	0.14043
10	9.489326	0.11239	11	10.490345	0.09199
12	11.491185	0.07667	13	12.491890	0.06489
14	13.492492	0.05563	15	14.493011	0.04821
16	15.493461	0.04219	17	16.493856	0.03723
18	17.494209	0.03310	19	18.494520	0.02961
20	19.494802	0.02665	21	20.495058	0.02412

This tends to be an awful function to use because it rectifies the noise (all the side-lobes are positive), but it is a simple function to implement, therefore, it is used in many applications.

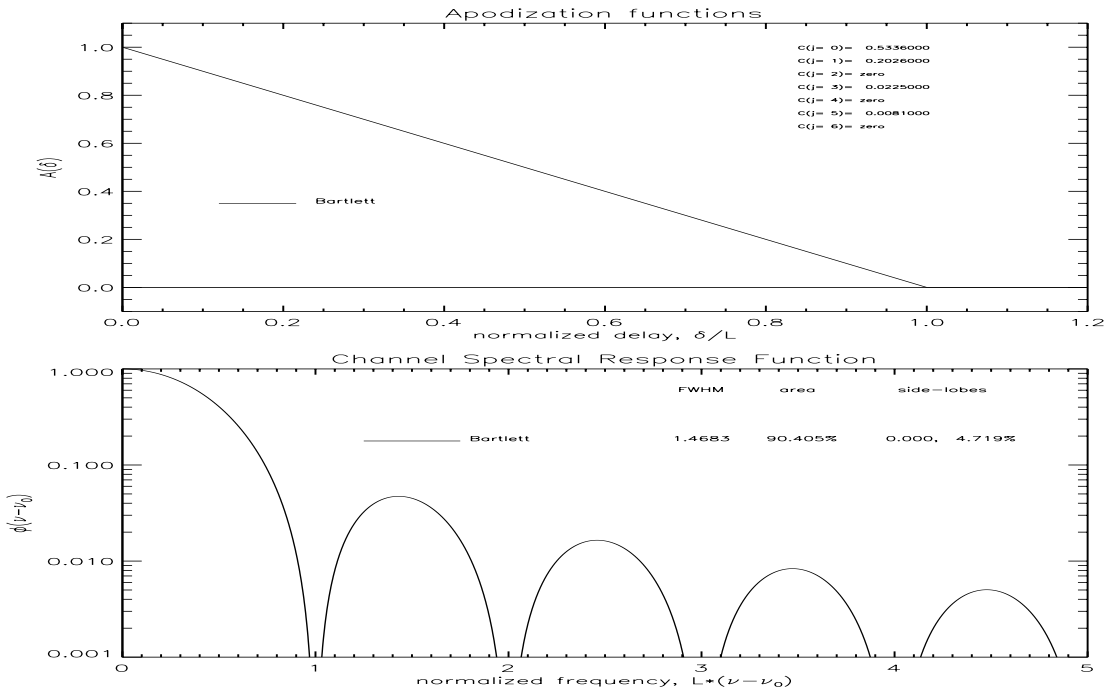


Figure 12.5: Channel response function for triangular apodization

12.4 von Hann apodization

A meteorologist, Julius von Hann, proposed the following

$$A(x) = 0.5 + 0.5 \cdot \cos\left(\pi \frac{x}{L}\right) \quad \text{if } |x| \le L \quad \text{else} \quad A(x) = 0 \tag{12.29}$$

The effective instrument profile is given as

$$\Phi(\Delta\nu) = \frac{\sin(y)}{y} \cdot \left[1 + \frac{y^2}{\pi^2 - y^2}\right], \quad y \equiv 2\pi L \cdot \Delta\nu \tag{12.30}$$

12.5 Hamming apodization

$$A(x) = 0.54 + 0.46 \cdot \cos\left(\pi \frac{x}{L}\right) \quad \text{if } |x| \le L \quad \text{else} \quad A(x) = 0 \tag{12.31}$$

The effective instrument profile is given as

$$\Phi(\Delta\nu) = \frac{\sin(2\pi L \cdot \Delta\nu)}{2\pi L \cdot \Delta\nu} + \frac{0.46}{0.54\pi} \cdot \frac{2L \cdot \Delta\nu \cdot \sin(2\pi L \cdot \Delta\nu)}{1 - 4 \cdot (L \cdot \Delta\nu)^2} \tag{12.32}$$

$$\Phi(\Delta\nu) = \frac{\sin(y)}{y} + \frac{0.46}{0.54} \cdot \frac{y \cdot \sin(y)}{\pi^2 - y^2} = \frac{\sin(y)}{y} \cdot \left[1 + \frac{0.46}{0.54} \cdot \frac{y^2}{\pi^2 - y^2}\right], \quad y \equiv 2\pi L \cdot \Delta\nu \tag{12.33}$$

$$\text{FWHM} = 2 \cdot 0.4538062/L = 0.907612/L = 1.50428 \cdot \sin(x)/x \tag{12.34}$$

For the Hamming apodization function the zeroes are at $L\Delta\nu = 1.0, 1.2990381, 1.5, 2.0, 2.5, \ldots$. Note that the largest side-lobe is side-lobe #4.

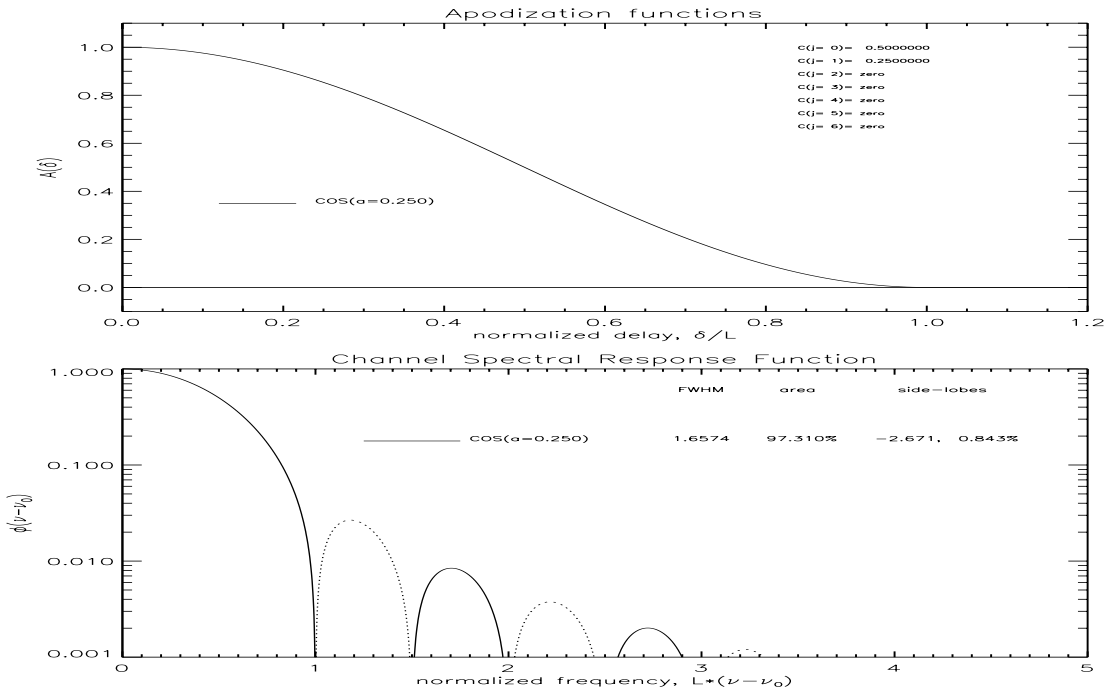


Figure 12.6: Channel response function for von Hann apodization

Note that there is a zero at $y = \pi\sqrt{0.54/0.08} = 2\pi\sqrt{0.54/0.32} = 8.1621$.
There is an in-determinant point when $y = \pi$ and the function at this point is equal to $\Phi(y = \pi) = (0.46/0.54)/2 = 0.425926$
The normalization integral for evenly spaced points, $\Delta\nu = i \cdot \delta\nu$, is given by

$$\delta\nu \cdot \sum_{i\rightarrow-\infty}^{i\rightarrow\infty} \Phi(i \cdot \delta\nu) = \left(1 + \frac{0.46}{0.54}\right) \cdot \frac{1}{2L} = \frac{1}{0.54} \cdot \frac{1}{2L} = \frac{1}{1.08L} \tag{12.35}$$

12.5.1 Matrix formulation of Apodization Functions

Both Hamming and von Hann apodization functions are part of a general class of cosine apodization functions in which the sinc() functions are co-added together to minimize the side-lobes. Cosine apodization functions can be written as a matrix operator for spectra spaced at the Nyquist sampling interval of $\Delta\nu = 0.5/L$. If R_U is the un-apodized spectrum, then the cosine apodized spectrum, R_C is given by

$$A(x) = (1 - 2a) + 2a\cos(\pi\frac{x}{L}) \tag{12.36}$$

$$\vec{R_C} = \mathbf{M} \cdot \vec{R_U} \tag{12.37}$$

$$\mathbf{M} = (1 - 2a) \cdot \begin{pmatrix} 1 & b & 0 & 0 & 0 & \dots \\ b & 1 & b & 0 & 0 & \dots \\ 0 & b & 1 & b & 0 & \dots \\ 0 & 0 & b & 1 & b & \dots \\ 0 & 0 & 0 & b & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \end{pmatrix} \tag{12.38}$$

Figure 12.7: Snapshot of Richard W. Hamming (Feb. 11, 1915 - Jan. 7, 1998) (from http://www-gap.dcs.st-and.ac.uk/ history/PictDisplay/Hamming.html)

and where b has the value $a/(1-2a)$.

$$c_0 = \sum_{k=0}^{\infty} \frac{(2k)!}{(k!)^2} b^{2k} \tag{12.39}$$

$$r = 1 - b^2 - b^4 - 2b^6 - 5b^8 - 14b^{10} - 42b^{12} - 132b^{14} - \ldots \tag{12.40}$$

In the Hamming case ($a = 0.23$), c_0 converges (with more than 45 channels) to within 10^{-12} to a value

$$c_0 = 1.909188309204\ldots \tag{12.41}$$

$$r = -.5590375815769\ldots \tag{12.42}$$

$$R_U(i) = \frac{c_0}{(1-2a)} \left[R_C(i) + \sum_{j=1}^{N} r^j \cdot (R_C(i+j) + R_C(i-j)) \right] \tag{12.43}$$

Hamming [1977] found an optimum value of a which minimized the first side-lobe of the CSRF. He also showed that the optimum value of a was a function of the number of points in the spectrum; however, the optimum value of a converged to 0.23 for more than 100 points. In Fig. 12.10 the height of the maximum side-lobe relative to the height of the central lobe is shown as a function of a for a large number of points. The number of the side-lobe with the maximum height is shown along the top of the figure, and at $a = 0.23$ it can be seen that all side-lobes are suppressed to less than 1%, with the fourth side-lobe being the largest.

The ratio of the FWHM of the CSRF with respect to FWHM_U increases with increasing values of the parameter a as shown in Fig. 12.11. When $a = 0.23$, corresponding to the Hamming function, the FWHM_H is 50.4% larger than FWHM_U.

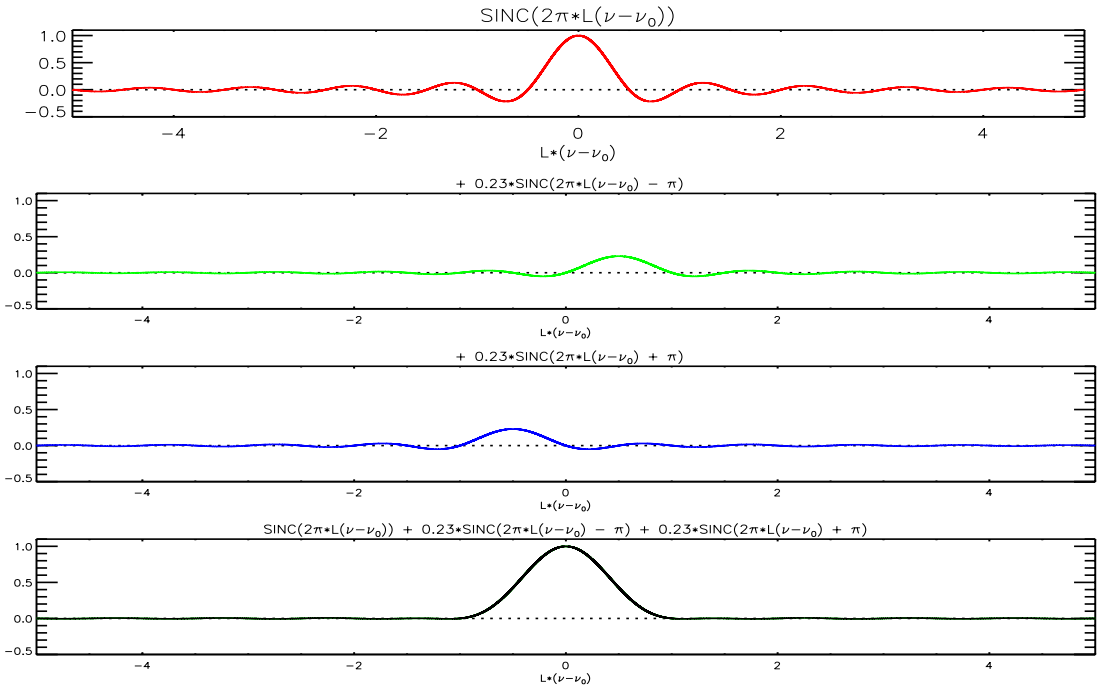


Figure 12.8: Illustration of co-adding 3 sinc() functions to produce the Hamming apodization function

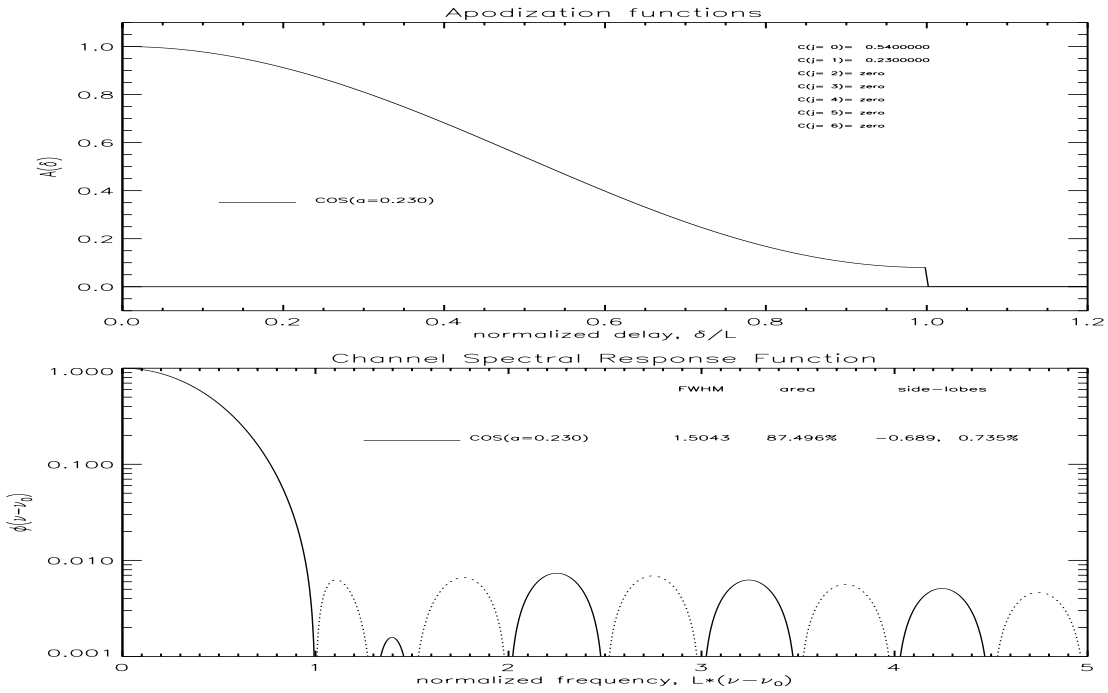


Figure 12.9: Channel response function for Hamming apodization

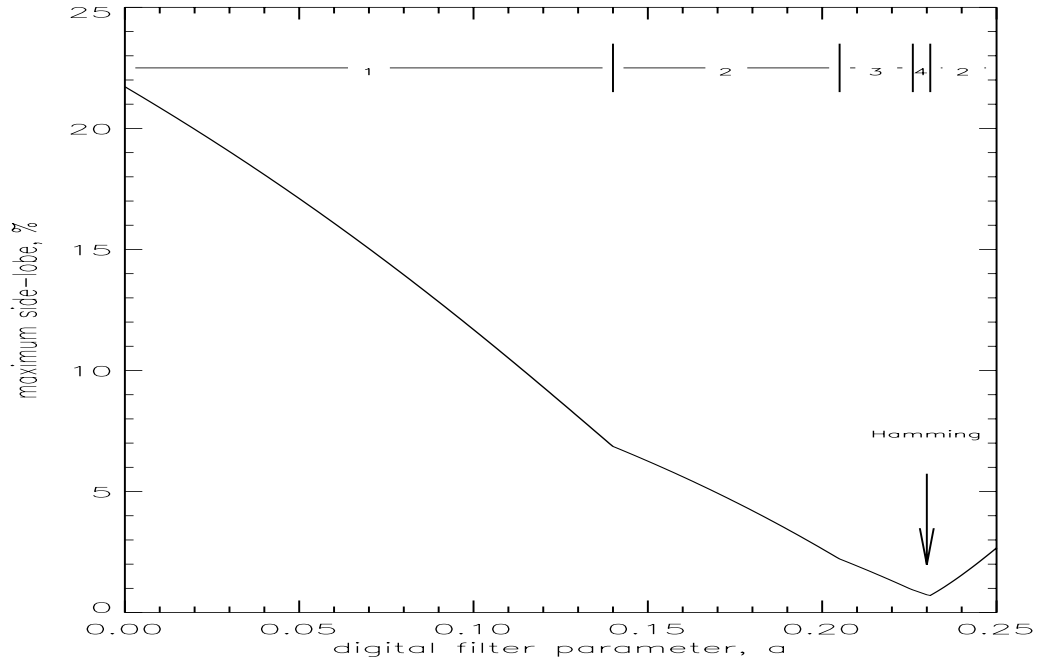


Figure 12.10: Size of side-lobes versus cosine apodization parameter a

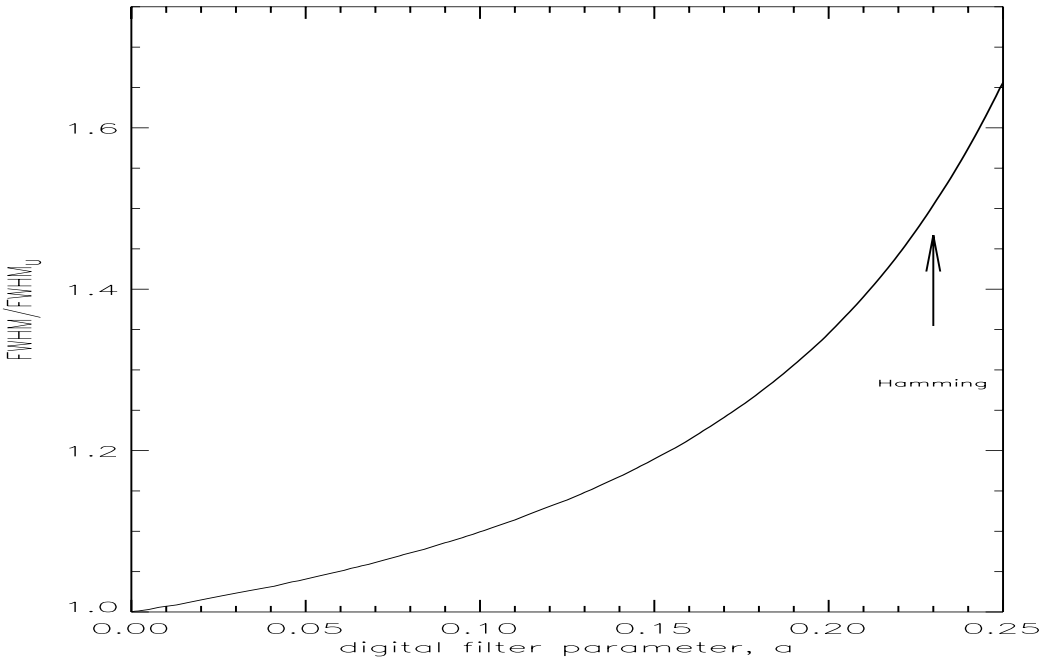


Figure 12.11: FWHM versus cosine apodization parameter a

12.6 Blackman Apodization

$$A(x) = a_0 + a_1 \cdot \cos\left(\pi \frac{x}{L}\right) + a_2 \cdot \cos\left(2\pi \frac{x}{L}\right) \tag{12.44}$$

and values of $a_0 = 0.42$, $a_1 = 0.50$, and $a_2 = 0.08$

$$\Phi(\Delta\nu) = \frac{\sin(y)}{y} \cdot \left[1 + \frac{0.50}{0.42} \cdot \frac{y^2}{\pi^2 - y^2} - \frac{0.08}{0.42} \cdot \frac{y^2}{4\pi^2 - y^2}\right], \quad y \equiv 2\pi L \cdot \Delta\nu \tag{12.45}$$

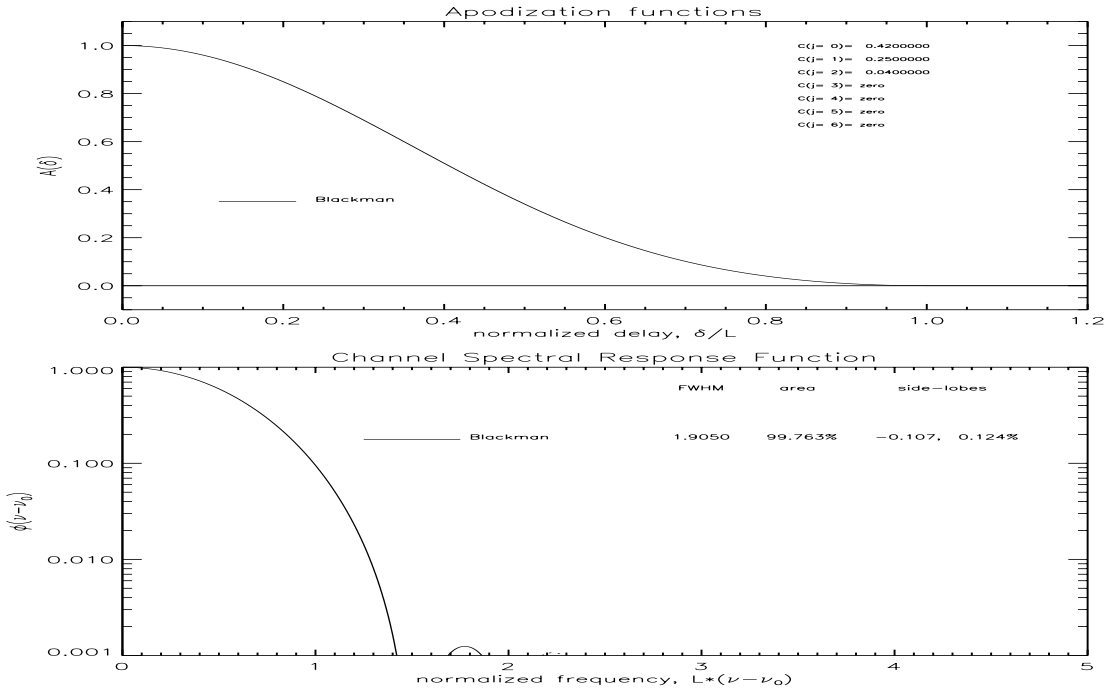


Figure 12.12: Channel response function for Blackmann apodization

$$\text{FWHM} = 2 \cdot 0.574700/L = 1.1494/L = 1.9050145 \cdot \sin(x)/x \tag{12.46}$$

zeroes are at $L\Delta\nu = 1.5, 1.527525, 2.0, 2.5, 3.0, ...$
Note that there is a zero at $y = 2\pi\sqrt{0.42/(5 \cdot .42 - 2 + 0.08)} = 2\pi\sqrt{0.42/0.18} = 9.59772$.
There are two indeterminate points at $y = \pi$ and $y = 2\pi$
equal to $\Phi(y = \pi) = (0.50/0.42)/2 = 0.595238$
and $\Phi(y = 2\pi) = (0.08/0.42)/2 = 0.0952381$

12.7 Kaiser-Bessel Apodization

Hamming (1977, pg. 171) discusses a filter using the modified Bessel function, $I_0(x)$.

$$A(x) = \frac{I_0\left(K \cdot \sqrt{1-(\frac{x}{L}^2)}\right)}{I_0(K)} \quad \text{for} \quad |x| \le L \tag{12.47}$$

$$I_0(x) = 1 + \sum_{j=1}^{\infty} \frac{1}{(j!)^2}\left(\frac{x}{2}\right)^{2j} \simeq 1 + \frac{x^2}{2^2\cdot(1!)^2} + \frac{x^4}{2^4\cdot(2!)^2} + \frac{x^6}{2^6\cdot(3!)^2} + ... \tag{12.48}$$

$$\Phi_K(f) = \frac{2L}{I_0(K)} \frac{\sinh\left(K\cdot\sqrt{1-\left(\frac{f}{f_a}\right)^2}\right)}{K\sqrt{1-\left(\frac{f}{f_a}\right)^2}} \tag{12.49}$$

where $f_a = K/L$ and $f = 2\pi\Delta\nu$.

$$\sinh(x) = \frac{e^x - e^{-x}}{2} \tag{12.50}$$

$$e^x = \sum_{j=0}^{\infty} \frac{x^j}{j!} \simeq 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... \tag{12.51}$$

$$\sinh(x) = \sum_{j=0}^{\infty} \frac{x^{2j+1}}{(2j+1)!} \simeq x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + ... \tag{12.52}$$

When $f > f_a$ the argument of the square root is complex. Noting that

$$\sinh(ix) = i\cdot\sin(x) \tag{12.53}$$

and

$$\sqrt{1-\left(\frac{f}{f_a}\right)^2} = i\cdot\sqrt{\left(\frac{f}{f_a}\right)^2 - 1} \tag{12.54}$$

$$\Phi(K,f) = \frac{2L}{I_0(K)} \frac{\sinh\left(K\cdot\sqrt{1-\left(\frac{f}{f_a}\right)^2}\right)}{K\sqrt{1-\left(\frac{f}{f_a}\right)^2}} \quad \text{for } |f| \le f_a \tag{12.55}$$

$$\Phi(K,f) = \frac{2L}{I_0(K)} \frac{\sin\left(K\cdot\sqrt{\left(\frac{f}{f_a}\right)^2 - 1}\right)}{K\sqrt{\left(\frac{f}{f_a}\right)^2 - 1}} \quad \text{for } |f| \ge f_a \tag{12.56}$$

We can substitute $f/f_a = y/K$ where $y = 2\pi\Delta\nu\cdot L$. To normalized this function we should divide by $\Phi(y=0)$ where

$$\Phi(y=0) = \frac{2L\sinh(K)}{K\cdot I_0(K)}, \quad y = 2\pi\Delta\nu\cdot L \tag{12.57}$$

$$\Phi(K,y) = \frac{\sinh\left(K \cdot \sqrt{1-\left(\frac{y}{K}\right)^2}\right)}{\sinh(K)\sqrt{1-\left(\frac{y}{K}\right)^2}} \quad \text{for } |y| \le K \tag{12.58}$$

$$\Phi(K,y) = \frac{\sin\left(K \cdot \sqrt{\left(\frac{y}{K}\right)^2-1}\right)}{\sinh(K)\sqrt{\left(\frac{y}{K}\right)^2-1}} \quad \text{for } |y| \ge K \tag{12.59}$$

The crossover point $y = K$ is usually within the central lobe. This point and the subsequent zeroes can be found when the argument of the sin() function equal to $n\pi$.

$$\sqrt{\left(\frac{y}{K}\right)^2-1} = n\pi \tag{12.60}$$

$$\Delta\nu = \frac{1}{2L}\sqrt{n^2+\frac{K^2}{\pi^2}} \tag{12.61}$$

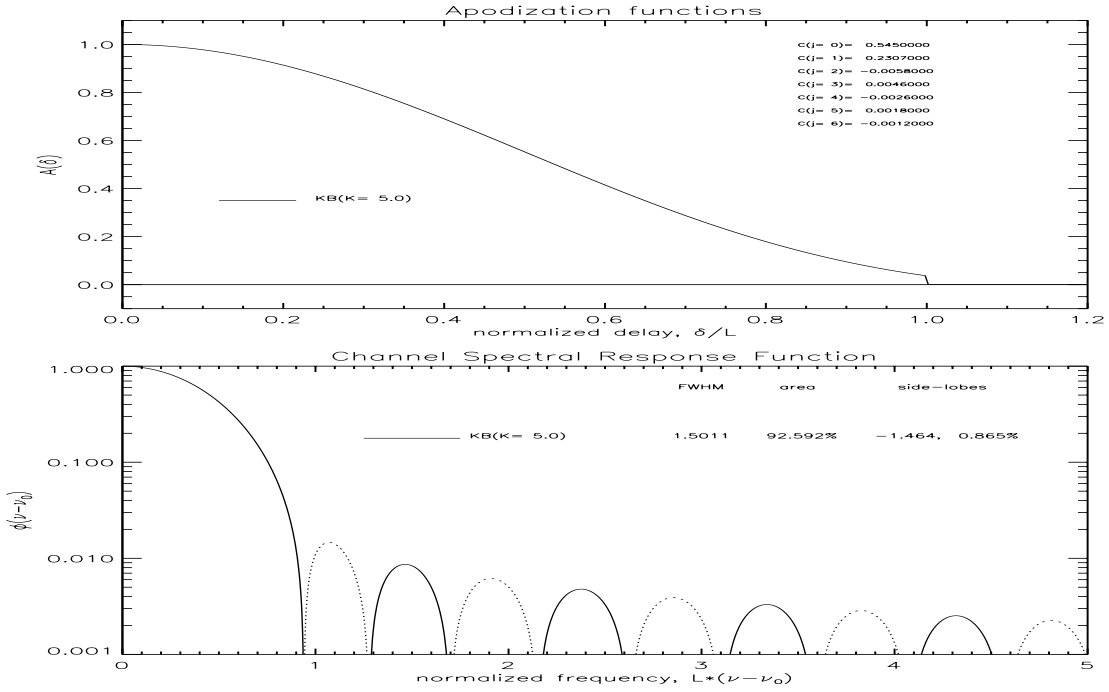


Figure 12.13: Channel response function for Kaiser-Bessel (K=5) apodization

For CIRRIS instrument they use $\alpha = K/\pi = 2.0, 2.5, 3.5$ and the first size lobe, given in decibels, is

$$\text{dB} = 10 \cdot \log_{10}(-\text{side}) \tag{12.62}$$

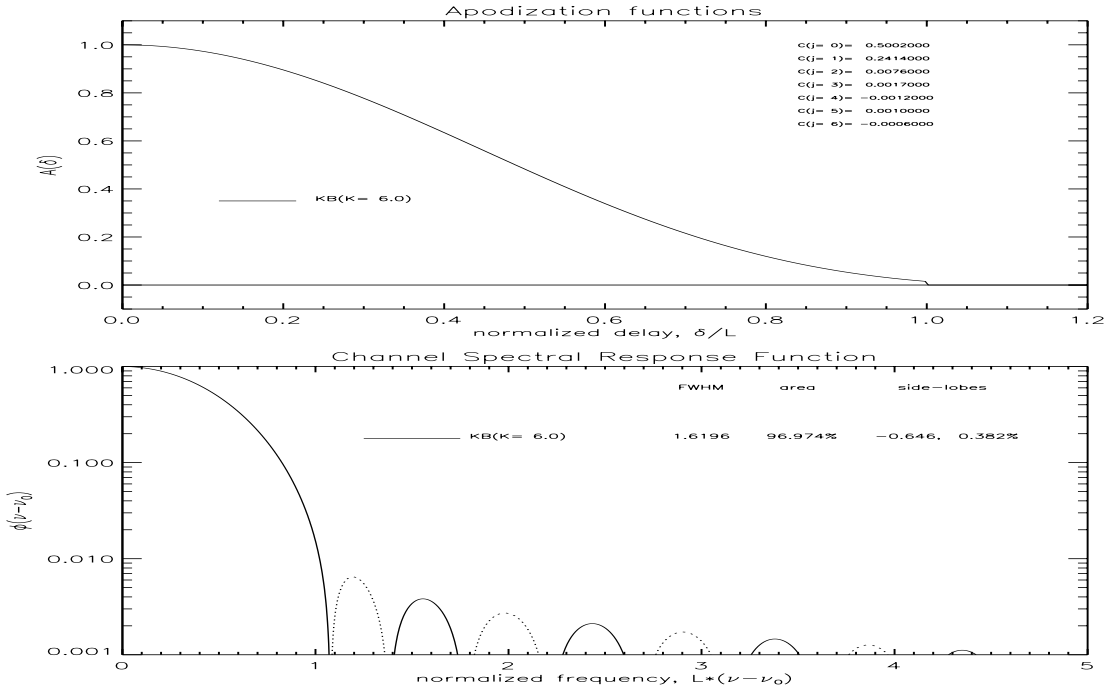


Figure 12.14: Channel response function for Kaiser-Bessel (K=6) apodization

Table 12.1: Location of zeroes of the Kaiser-Bessel apodization function
Kaiser Bessel 1'Zeros: L =1.00

K	n=0	n=1	n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=9
1	0.159	0.525	1.013	1.508	2.006	2.505	3.004	3.504	4.003	4.503
2	0.318	0.593	1.049	1.533	2.025	2.520	3.017	3.514	4.013	4.511
3	0.477	0.691	1.108	1.574	2.056	2.545	3.038	3.532	4.028	4.525
4	0.637	0.809	1.185	1.630	2.099	2.580	3.067	3.557	4.050	4.545
5	0.796	0.940	1.278	1.698	2.153	2.624	3.104	3.589	4.078	4.570
6	0.955	1.078	1.383	1.778	2.216	2.676	3.148	3.628	4.112	4.600
7	1.114	1.221	1.497	1.868	2.289	2.737	3.200	3.673	4.152	4.636
8	1.273	1.368	1.619	1.968	2.371	2.806	3.259	3.724	4.198	4.677
9	1.432	1.517	1.747	2.074	2.460	2.881	3.324	3.782	4.249	4.722
10	1.592	1.668	1.880	2.187	2.556	2.964	3.396	3.845	4.305	4.773

Table 12.2: Size of side-lobes and transition point for Kaiser-Bessel apodization functions

L	K	FWHM	+side	-side	transition	$I_0(K)$	sinh(K)	dB
1.00	1.00	1.0360	0.1092	0.1848	0.159	1.266	1.175	-7.33
1.00	2.00	1.1300	0.0708	0.1198	0.318	2.280	3.627	-9.22
1.00	3.00	1.2500	0.0384	0.0651	0.477	4.881	10.018	-11.87
1.00	4.00	1.3780	0.0188	0.0318	0.637	11.302	27.290	-14.97
1.00	5.00	1.5020	0.0087	0.0146	0.796	27.240	74.203	-18.35
1.00	5.44	1.5560	0.0061	0.0102	0.866	40.494	115.380	-19.90
1.00	6.00	1.6200	0.0038	0.0065	0.955	67.234	201.713	-21.90
1.00	8.00	1.8380	0.0007	0.0012	1.273	427.564	1490.479	-29.33
1.00	8.89	1.9280	0.0003	0.0005	1.414	981.277	3611.408	-32.72
1.00	10.00	2.0360	0.0001	0.0002	1.592	2815.717	11013.232	-37.06

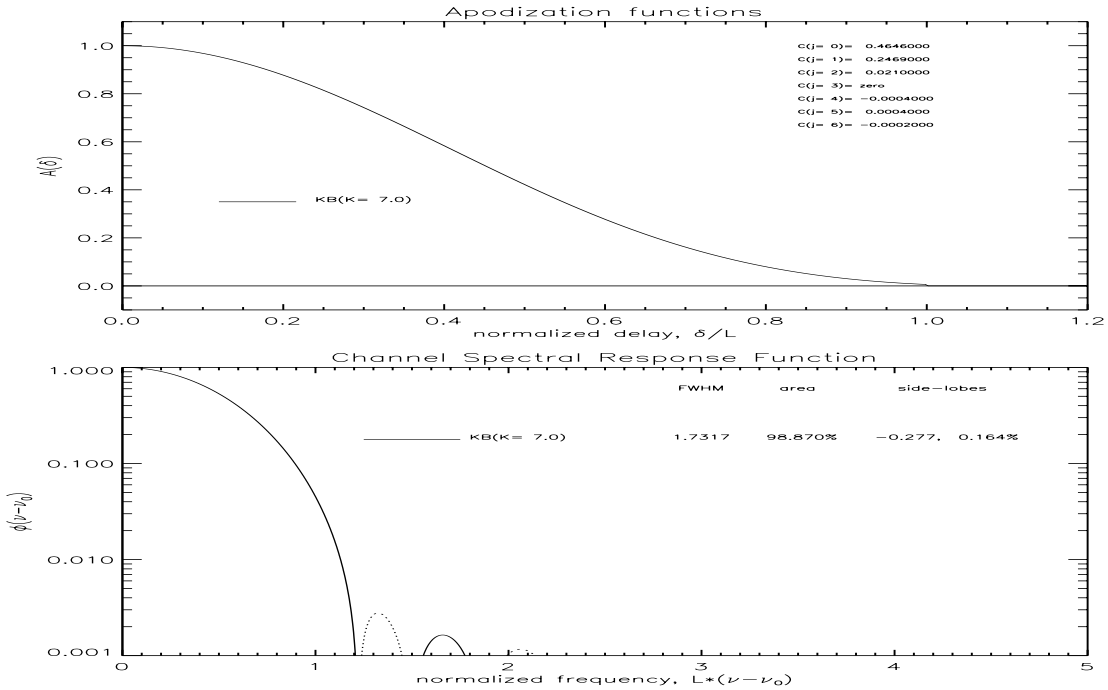


Figure 12.15: Channel response function for Kaiser-Bessel (K=7) apodization

12.8 Dolph-Chebyshev

The Dolph-Chebyshev apodization function [Brigham, 1974, pg. 183], denoted as "DC" in the table above and shown as small diamond symbols in Fig. 12.2 and Fig. 12.3, is a function of the parameter s, which is the side-lobe suppression in decibels. The form of the DC apodization function is given in Ward [1973]. This function is similar to the Kaiser-Bessel functions, in that the apodization is continuously tunable with the parameter s. The Dolph-Chebyshev apodization function has characteristics which are similar to the Kaiser-Bessel; the $s = 50$ dB function is shown in Fig. 12.16 and is similar to the Kaiser-Bessel $\alpha \simeq 5.2$ and the same conclusions hold for this function as well. Ward's [1973] equation for a inteferogram with N points from $x(i = 1) = -L$ to $x(N) = L$ is given as

$$\begin{aligned} A(i) &= \frac{N-1}{L-i} \sum_{k=0}^{M} \binom{i-2}{k} \cdot \binom{N-i}{k+1} \cdot \beta^{k+1} \qquad \text{for} i \neq 1 \text{ or } N \\ A(1) &= 1 \\ A(N) &= 1 \end{aligned} \tag{12.63}$$

and

$$\begin{aligned} M &\equiv i-2 \qquad i \leq (N+1)/2 \\ &\equiv N-i-1 \qquad i \geq (N+1)/2 \end{aligned} \tag{12.64}$$

$$\binom{n}{k} \equiv \frac{n!}{k! \cdot (n-k)!} \tag{12.65}$$

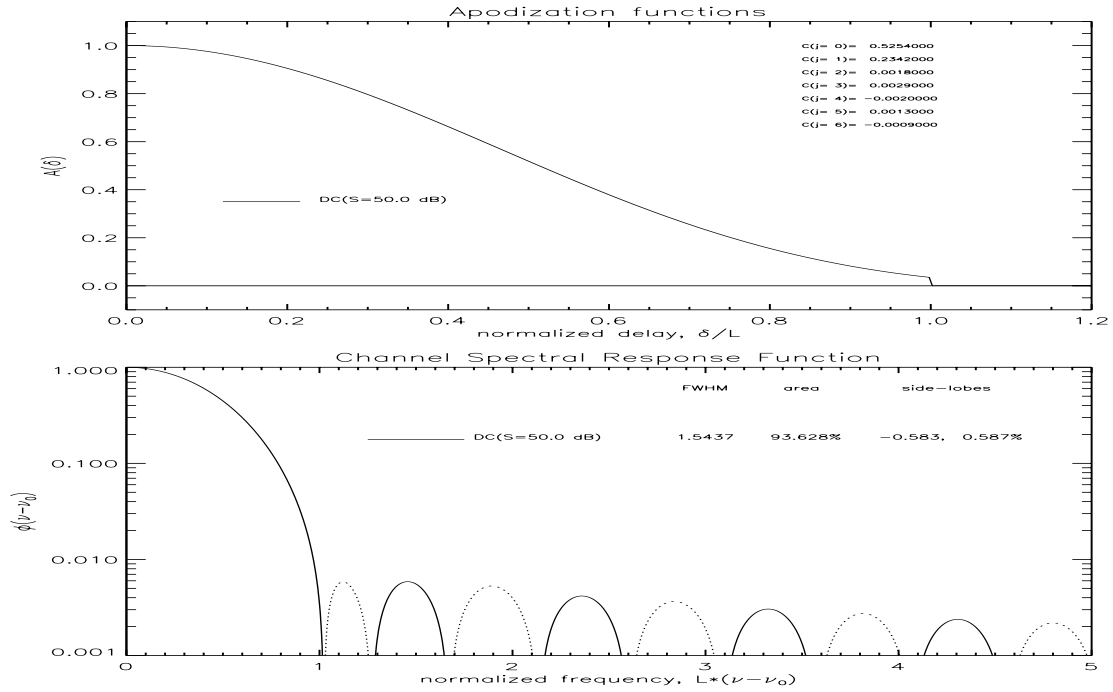


Figure 12.16: Channel response function for Dolph Chebyshev (s=50) apodization function

12.9 Gaussian

This apodization function is derived from the idea that a Fourier transform of a Gaussian is a Gaussian is spectral space. The cosine transform of a truncated Gaussian; however, has side-lobes. This function has the advantage that it is tunable, since we can set the FWHM of the Gaussian in delay space. If the FWHM of the Gaussian in spectral space is given by α/L, then the Gaussian in delay space is given by

$$A(x) = \exp\left(-a \cdot x^2\right) \tag{12.66}$$

where,

$$a = \frac{\pi^2 \alpha^2}{4L \cdot \log_e(0.5)} \tag{12.67}$$

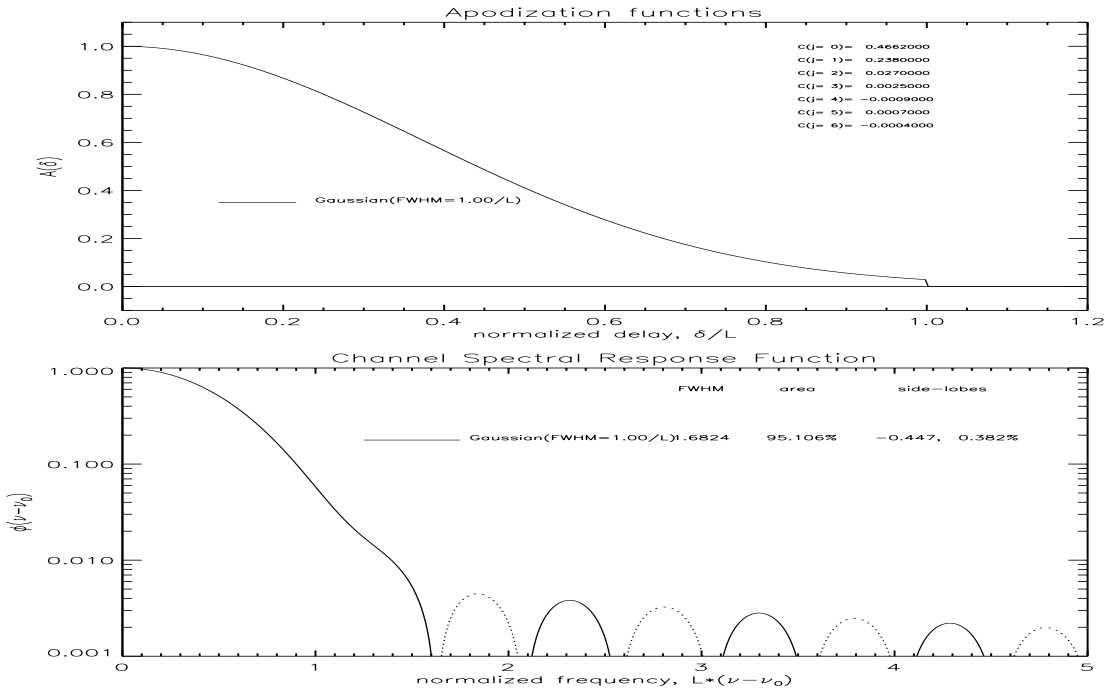


Figure 12.17: Channel response function for the truncated Gaussian for $\alpha = 1$

12.10 Autoregressive spectral estimator (ASE)

The autoregressive spectral estimator (ASE) apodization function of Amato *et al.* [1993a] is written in terms of two parameters, p and λ, which are found by minimizing radiance residuals with respect to a rectangular CSRF for an ensemble of spectra with a given signal-to-noise. The form of their equation is

$$A_{\text{ASE}}(x) = \frac{1}{1 + \lambda \cdot \left(\frac{2\pi x}{L}\right)^{2 \cdot p}}. \tag{12.68}$$

Results are shown for two sets of (p, λ). The first function, (1, 0.25), was deduced from the figures in [Amato *et al.*, 1993a] and the second function, (2, 0.02), was deduced from the figures in [Amato *et al.*, 1993b]. The ASE apodization functions (denoted as x_1 and x_2 in the tradeoff figure) fall to the right of the cosine apodization functions. This implies that they have no relative advantage in computation of the RTA. These

functions also require many terms of the cosine expansion and are difficult matrices to invert; however, the inverse matrix can be found numerically. Amato *et al.* [1993b] used their ASE function to demonstrate the equivalence between unapodized and apodized retrievals.

12.11 Norton-Beer Apodization Functions

Norton and Beer (1976) defined a set of apodization functions given by

$$A(x) = \sum_{i=0}^{N} C_i \cdot \left(1 - \frac{x^2}{L^2}\right)^i \tag{12.69}$$

$$\sum_{i=0}^{N} C_i \quad \equiv \quad 1 \tag{12.70}$$

with the Channel response functions given by

$$\Phi(y) = \sum_{i=0}^{N} C_i \cdot Q_i(y), \qquad y = 2\pi\Delta\nu L \tag{12.71}$$

where the Q functions can be found to be

$$Q_0(y) = \mathrm{sinc}(y) \tag{12.72}$$

$$Q_1(y) = \frac{3}{y^2} \cdot (\mathrm{sinc}(y) - \cos(y)) \tag{12.73}$$

$$Q_2(y) = \frac{-15}{y^2} \cdot \left[\left(1 - \frac{3}{y^2}\right) \cdot \mathrm{sinc}(y) + \left(\frac{3}{y^2}\right) \cdot \cos(y)\right] = \frac{15}{y^2} \cdot (Q_1 - Q_0) \tag{12.74}$$

$$Q_3(y) = \frac{105}{y^4} \cdot \left[\left(1 - \frac{15}{a^2}\right) \cdot \cos(y) + 6 \cdot \mathrm{sinc}(y) - \frac{15}{a^2} \cdot \mathrm{sinc}(y)\right] \tag{12.75}$$

$$Q_4(y) = \frac{945}{y^4} \cdot \left[\left(1 - \frac{45}{y^2} + \frac{105}{y^4}\right) \cdot \mathrm{sinc}(y) + \left(\frac{10}{y^2} - \frac{105}{y^4}\right) \cdot \cos(y)\right] \tag{12.76}$$

They used these functions to find the "best" coefficients and produced a tradeoff curve showing that they cannot cross an empirical line.

Table 12.3: Parameters for Connes, Beer, and Norton-Beer apodization functions

FWHM	function	C_0	C_1	C_2	C_3	C_4
1.00	sinc function	1.00000	- -	- -	- -	- -
1.22	Connes	0.23977	0.45806	0.22498	0.07719	- -
1.58	Beer	- -	- -	1.00000	- -	- -
1.12	F1, weak	0.54800	-0.0833	0.5353	- -	- -
1.20	F1, weak-2	0.384093	-0.08757	7 0.703484	- -	- -
1.30	F2, medium	0.26000	-0.15483	8 0.894838	- -	- -
1.40	F2, medium-2	0.152442	-0.13617	6 0.983734	- -	- -
1.50	F3, strong	0.09000	- -	0.5875	- -	0.32250
1.60	F3, strong-2	0.045335	- -	0.554883	- -	0.399782

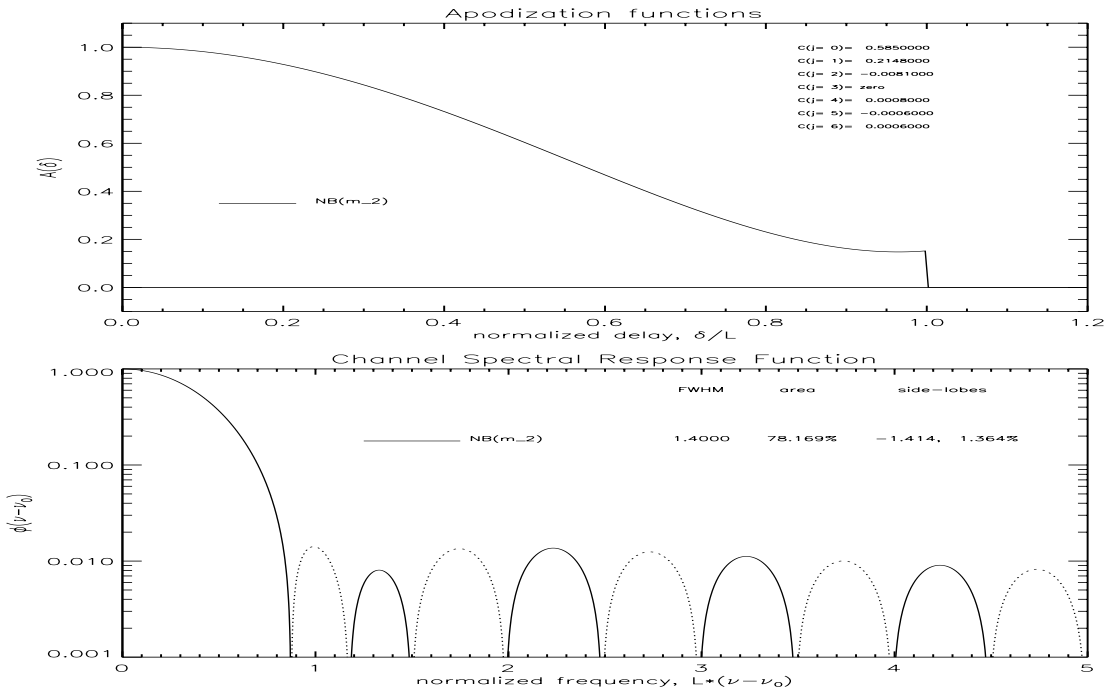


Figure 12.18: Channel response function for the Norton-Beer "medium" apodization function

$$\text{sinc}(x) = \sum_{j=0}^{\infty} (-1)^j \cdot \frac{x^{2j}}{(2j+1)!} \simeq 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!} + ... = 1 - \frac{x^2}{6} + \frac{x^4}{120} - \frac{x^6}{5040} + ... \tag{12.77}$$

$$\cos(x) = \sum_{j=0}^{\infty} (-1)^j \cdot \frac{x^{2j}}{(2j)!} \simeq 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + ... = 1 - \frac{x^2}{2} + \frac{x^4}{20} - \frac{x^6}{720} + ... \tag{12.78}$$

$$\Phi(\nu) = \frac{2}{L} \int_0^L A(x) \cos(2\pi\nu_0 x) \cos(2\pi\nu x) dx \tag{12.79}$$

$$\cos(mx)\cos(nx) = \frac{1}{2}[\cos((m-n)x) + \cos((m+n)x)] \tag{12.80}$$

$$\Phi(\nu) = \frac{1}{L} \int_0^L A(x) \cos(2\pi\Delta\nu x) dx \quad + \quad \frac{1}{L} \int_0^L A(x) \cos(2\pi(\nu+\nu_0)x) dx \tag{12.81}$$

$$a \equiv 2\pi\Delta\nu \quad \text{and} \quad y \equiv a \cdot L = 2\pi\Delta\nu L \tag{12.82}$$

12.11.1 Q_0 Function

$$\frac{1}{L} \int_0^L \cos(ax) dx = \frac{\sin(aL)}{aL} = \text{sinc}(y) \quad \text{CRC\#291} \tag{12.83}$$

$$Q_0(y) = \text{sinc}(y) = \sum_{j=0}^{\infty} (-1)^j \cdot \frac{y^{2j}}{(2j+1)!} \simeq 1 - \frac{y^2}{6} + \frac{y^4}{120} - \frac{y^6}{5,040} + \frac{y^8}{362,880} - ... \tag{12.84}$$

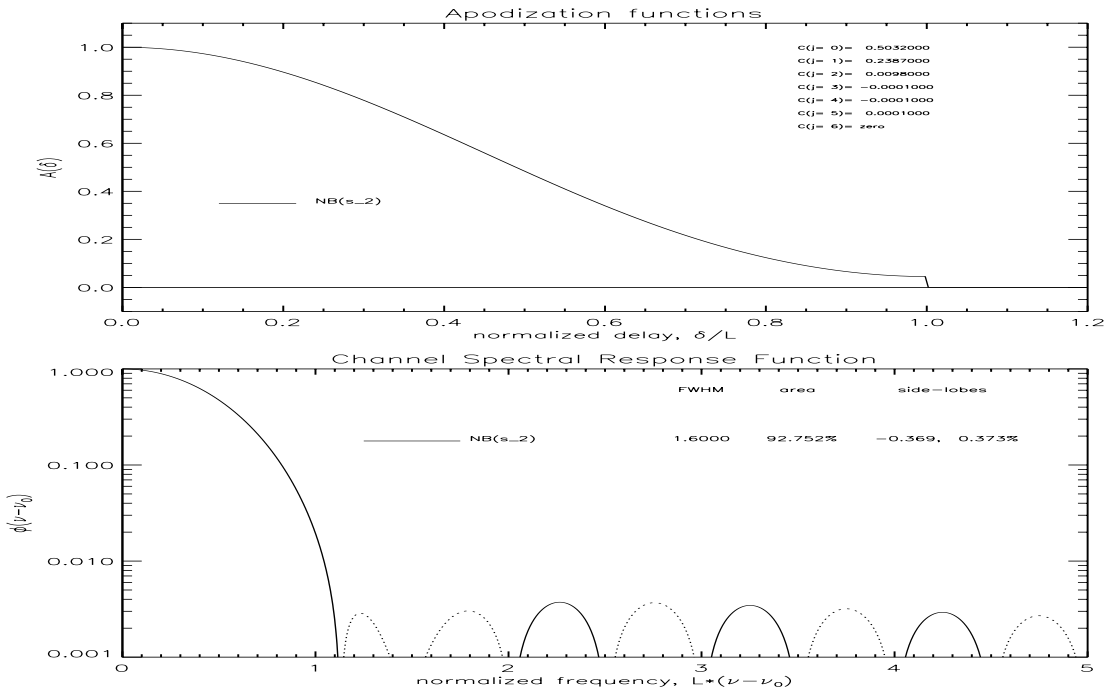


Figure 12.19: Channel response function for the Norton-Beer "strong" apodization function

12.11.2 Q_1 Function

Using the integral (CRC #394)

$$\frac{1}{L}\int_0^L \left(\frac{x^2}{L^2}\right)\cos(ax)dx = \frac{2\cos(y)}{y^2} + \frac{\sin(y)}{y} - \frac{2\sin(y)}{y^3} = \frac{2\cdot(\cos(y)-\mathrm{sinc}(y))}{y^2} - \mathrm{sinc}(y) \tag{12.85}$$

$$Q_1 = \frac{1}{L}\int_0^L \left(1-\frac{x^2}{L^2}\right)\cos(ax)dx = \frac{2\cdot(\mathrm{sinc}(y)-\cos(y))}{y^2} \tag{12.86}$$

To determine the correct normalization the *sinc* and *cosine* functions can be expanded and combined

$$\begin{aligned}\frac{\mathrm{sinc}(y)-\cos(y)}{y^2} &= \sum_{j=1}^{\infty}(-1)^j\left[\frac{y^{2j-2}}{(2j+1)!}-\frac{y^{2j-2}}{(2j)!}\right] \\ &= \sum_{j=1}^{\infty}\frac{(-1)^j}{(2j+1)!\cdot(2j)!}\left[((2j)!-(2j+1)!)\cdot y^{2j-2}\right]\end{aligned} \tag{12.87}$$

and as $y\to 0$ then only the $j=1$ term will apply and

$$\frac{\mathrm{sinc}(y)-\cos(y)}{y^2} \to \frac{6-2}{12} = \frac{1}{3} \tag{12.88}$$

Therefore, $Q_1(y)$ should be normalized by 3.

$$Q_1(y) = 3\cdot\frac{\mathrm{sinc}(y)-\cos(y)}{y^2}$$

$$\begin{aligned} &= \sum_{j=1}^{\infty} \frac{(6j)\cdot(-1)^{j+1}}{(2j+1)!}\cdot y^{2j-2} \\ &\simeq 1 \quad -\frac{y^2}{10}+\frac{y^4}{280}-\frac{y^6}{15,120}+\frac{y^8}{1,330,560} \end{aligned} \tag{12.89}$$

12.11.3 Q_2 Function

$$Q_2 = \frac{1}{L}\int_0^L \left(1-\frac{x^2}{L^2}\right)^2 \cos(ax)dx = \frac{1}{L}\int_0^L \left(1-2\cdot\frac{x^2}{L^2}+\frac{x^4}{L^4}\right)\cos(ax)dx \tag{12.90}$$

$$\frac{1}{L}\int_0^L \left(\frac{x^m}{L^m}\right)\cos(ax)dx = \mathrm{sinc}(aL) - \frac{4}{aL^{m+1}}\int_0^L x^{m-1}\sin(ax)dx \quad \text{CRC\#396} \tag{12.91}$$

$$\int_0^L x^3\sin(ax)dx = \frac{3a^2L^2-6}{a^4}\sin(aL) - \frac{a^2L^3-6L}{a^3}\cos(aL) \quad \text{CRC\#391} \tag{12.92}$$

$$\int_0^L x^m\sin(ax)dx = \frac{-L^m}{a}\cos(aL) + \frac{m}{a}\int_0^L x^{m-1}\cos(ax)dx \quad \text{CRC\#392} \tag{12.93}$$

$$\int_0^L x^3\cos(ax)dx = \frac{3a^2L^2-6}{a^4}\cos(aL) + \frac{a^2L^3-6L}{a^3}\sin(aL) \quad \text{CRC\#395} \tag{12.94}$$

recursive substitution of CRC# 392 into CRC# 396 will yield a general solution for the integral which has the form of CRC# 396b.

$$\frac{1}{L}\int_0^L \left(\frac{x^m}{L^m}\right)\cos(ax)dx = b_0(m)\cdot\sin(y) + b_1(m)\cdot\cos(y) \tag{12.95}$$

$$b_0(m) = \sum_{r=0}^{[m/2]} \frac{(-1)^r}{L^{m+1}}\frac{m!}{(m-2r)!}\cdot\frac{L^{m-2r}}{a^{2r+1}} = \sum_{r=0}^{[m/2]} \frac{m!}{(m-2r)!}\cdot\frac{(-1)^r}{y^{2r+1}} \tag{12.96}$$

$$[s] = \text{greatest integer} \qquad e.g.,\ [2.5] = 2 \tag{12.97}$$

$$b_1(m) = \sum_{r=0}^{[(m-1)/2]} \frac{(-1)^r}{L^{m+1}}\frac{m!}{(m-2r-1)!}\cdot\frac{L^{m-2r-1}}{a^{2r+2}} = \sum_{r=0}^{[(m-1)/2]} \frac{m!}{(m-2r-1)!}\cdot\frac{(-1)^r}{y^{2r+2}} \tag{12.98}$$

m	$y\cdot b_0(m)$	$b_1(m)$
0	1	0
2	$1-2/y^2$	$2/y^2$
4	$1-12/y^2+24/y^4$	$4/y^2-24/y^4$

$$Q_2 = \frac{1}{L}\int_0^L \left(1-2\cdot\frac{x^2}{L^2}+\frac{x^4}{L^4}\right)\cos(ax)dx \tag{12.99}$$

$$\begin{aligned} Q_2 &= \mathrm{sinc}(y)\cdot y\cdot[b_0(0)-2b_0(2)+b_0(4)]+\cos(y)\cdot[b_1(0)-2b_1(2)+b_1(4)] && (12.100)\\ &= \mathrm{sinc}(y)\cdot\left[1-2(1-2/y^2)+1-12/y^2+24/y^4\right] \\ &\quad +\cos(y)\cdot\left[0-4/y^2+4/y^2-24/y^4\right] && (12.101)\\ &= -(8/y^2)\cdot\left[(1-3/y^2)\cdot\mathrm{sinc}(y)+3/y^2\cdot\cos(y)\right] && (12.102)\\ &= -(8/y^2)\cdot(Q_0-Q_1) && (12.103)\end{aligned}$$

using the expansions for Q_0 and Q_1 we find the first term cancels and the second term is equal to

$$Q_2(0)=\frac{8}{y^2}\cdot\left(\frac{y^2}{6}-\frac{y^2}{10}\right)=\frac{8}{60}(10-6)=\frac{8}{15} \tag{12.104}$$

therefore the normalized function is given by

$$Q_2(y)=-(15/y^2)\cdot\left[(1-3/y^2)\cdot\mathrm{sinc}(y)+3/y^2\cdot\cos(y)\right]=-(15/y^2)\cdot(Q_0-Q_1) \tag{12.105}$$

$$\begin{aligned} Q_2(y) &= \frac{-15}{y^2}\cdot\left[\sum_{j=0}^{\infty}\frac{(-1)^j}{(2j+1)!}\cdot y^{2j}-\sum_{j=1}^{\infty}\frac{(6j)\cdot(-1)^{j+1}}{(2j+1)!}\cdot y^{2j-2}\right] \\ &= \sum_{j=2}^{\infty}\frac{60j\,(j-1)\cdot(-1)^j}{(2j+1)!}\cdot y^{2j-4} \\ &\simeq 1-\frac{y^2}{14}+\frac{y^4}{504}-\frac{y^6}{33,264}+\frac{y^8}{3,459,456} && (12.106)\end{aligned}$$

The functions $Q_1(y)$ and $Q_2(y)$ are developing a general form and we can suppose that a general solution. Note that $6=3!/1!$ and $60=5!/2!$.

$$Q_n(y)=\frac{(2n+1)!}{n!}\cdot\sum_{j=n}^{\infty}\frac{j!\cdot(-1)^{j+n}}{(j-n)!\cdot(2j+1)!}\cdot y^{2j-2n} \tag{12.107}$$

12.11.4 Q_3 Function

$$Q_3=\frac{1}{L}\int_0^L\left(1-\frac{x^2}{L^2}\right)^3\cos(ax)dx=\frac{1}{L}\int_0^L\left(1-3\cdot\frac{x^2}{L^2}+3\cdot\frac{x^4}{L^4}-\frac{x^6}{L^6}\right)\cos(ax)dx \tag{12.108}$$

This function is not used by any of the Norton-Beer functions so we will not derive its solution, however, it is given in the Norton-Beer reference as:

$$Q_3(y)=\frac{105}{y^4}\left[(1-15/y^2)\cdot\cos(y)+3(2-5/y^2)\cdot\mathrm{sinc}(y)\right] \tag{12.109}$$

$$Q_3(y)=\sum_{j=0}^{\infty}\frac{7!\cdot j(j-1)(j-2)\cdot(-1)^{j+1}}{3!\cdot(2j+1)!}\cdot y^{2j-6} \tag{12.110}$$

$$Q_3(y)\simeq 1-\frac{y^2}{18}+\frac{y^4}{792}-\frac{y^6}{61,776}+\frac{y^8}{7,413,120} \tag{12.111}$$

12.11.5 Q_4 Function

$$\begin{aligned} Q_4 &= \frac{1}{L}\int_0^L \left(1-\frac{x^2}{L^2}\right)^4 \cos(ax)dx \\ &= \frac{1}{L}\int_0^L \left(1-4\cdot\frac{x^2}{L^2}+4\cdot\frac{x^4}{L^4}-4\cdot\frac{x^6}{L^6}+\frac{x^8}{L^8}\right)\cos(ax)dx \end{aligned} \tag{12.112}$$

$$Q_4(y) = \frac{945}{y^4}\left[(1-45/y^2+105/y^4)\cdot\mathrm{sinc}(y)+(5/y^2)\cdot(2-21/y^2)\cdot\cos(y)\right] \tag{12.113}$$

$$Q_4(y) = \sum_{j=0}^{\infty}\frac{9!\cdot j(j-1)(j-2)(j-3)\cdot(-1)^j}{4!\cdot(2j+1)!}\cdot y^{2j-8} \tag{12.114}$$

$$Q_4(y) \simeq 1-\frac{y^2}{22}+\frac{y^4}{1144}-\frac{y^6}{102,960}+\frac{y^8}{14,002,560} \tag{12.115}$$

12.12 General Cosine Apodization Functions

$$A_G(x) = a_0 + 2\cdot\sum_{j=1}^{J-1} a_j\cdot\cos\left(j\cdot\pi\frac{x}{L}\right) \quad \text{for} \quad |x|\le L \tag{12.116}$$

$$a_0 \equiv \left(1-2\sum_{j=1}^{J-1}a_j\right) \tag{12.117}$$

$$\phi_G(y) = a_0\cdot\frac{\sin(y)}{y}+\sum_{j=1}^{J-1}a_j\cdot\left[\frac{\sin(y+j\pi)}{y+j\pi}+\frac{\sin(y-j\pi)}{y-j\pi}\right] \tag{12.118}$$

where,

$$y \equiv 2\pi\cdot L\cdot(\nu-\nu_k) = 2\pi\cdot L\cdot\Delta\nu \tag{12.119}$$

which can be reorganized as

$$\phi_G(y) = \frac{\sin(y)}{y}\cdot\left[1+\sum_{j=1}^{J-1}\frac{2a_j}{a_0}\cdot\left[(-1)^j\frac{y^2}{y^2-(j\pi)^2}\right]\right]. \tag{12.120}$$

and related to the Nyquist sampled unapodized radiances

$$R_G(i) = a_0\cdot R_U(i)+\sum_{j=1}^{J-1}a_j\cdot(R_U(i-j)+R_U(i+j)) \equiv \mathbf{M}_{\mathbf{G}i,i'}\cdot R_U(i'). \tag{12.121}$$

Table 12.4: Cosine expansion coefficients for common apodization functions

function	a_0	a_1	a_2	a_3	a_4
Hamming	.540	.230	0	0	
von Hann	.500	.250	0	0	
Triangle	.508	.203	0	.023	0
Blackman	.420	.250	.04	0	
NB w_1	.778	.115	-.004	-.0002	.0005
NB w_2	.701	.156	-.006	.0002	.0004
NB m_1	.634	.189	-.006	-.0008	.001
NB m_2	.586	.214	-.008	-.00007	.0008
NB s_1	.534	.227	.006	.0002	-.0003
NB s_2	.503	.239	.010	-.0002	-.0002
KB α =1	.928	.043	-.010	.0045	-.0025
KB α =2	.795	.119	-.024	.0101	-.0056
KB α =3	.684	.176	-.026	.0108	-.0060
KB α =4	.604	.211	-.018	.0081	-.0045
KB α =5	.545	.231	-.006	.0046	-.0027
KB α =6	.500	.241	.008	.0017	-.0013
KB α =7	.465	.247	.021	$3{\cdot}10^{-5}$	-.0005
KB α =8	.435	.249	.033	-.0005	$-6{\cdot}10^{-5}$
KB α =9	.412	.249	.045	.0001	$6{\cdot}10^{-5}$
KB α =10	.391	.248	.055	.0015	$5{\cdot}10^{-5}$
DC s=30 dB	.663	.184	-.022	.0092	-.0052
DC s=50 dB	.525	.234	.002	.0029	-.0021
DC s=70 dB	.444	.248	.031	-.0002	-.0004
ASE p=1,λ=.20	.437	.196	.055	.0216	.0052
ASE p=2,λ=.02	.460	.261	.044	-.0167	-.0159

12.12.1 Derivation of General Channel Response Functions

Each component of the general apodization function can be convered to a component of the channel response function

$$\Phi(\Delta\nu) = \sum_{j=0}^{J} \Phi_j(\Delta\nu) \tag{12.122}$$

The first term results in a $\Phi_0(\Delta\nu) = a_0 \cdot \sin(y)/y$, while the remaining terms are of the form

$$\Phi_j(\nu-\nu_0) = \frac{4\cdot a_j}{L\cdot a_0}\int_0^L \cos(2\pi\nu_0 x)\cos(2\pi\nu x)\cos(j\cdot\pi\frac{x}{L})dx \tag{12.123}$$

$$\cos(\alpha x)\cos(\beta x) = \frac{1}{2}[\cos((\alpha-\beta)x) + \cos((\alpha+\beta)x)] \tag{12.124}$$

$$\begin{aligned}\Phi_j(\nu-\nu_0) &= \frac{2\cdot a_j}{a_0\cdot L}\int_0^L \cos(2\pi\nu_0 x)\cos(2\pi\nu x - j\pi\frac{x}{L})dx \\ &+ \frac{2\cdot a_j}{a_0\cdot L}\int_0^L \cos(2\pi\nu_0 x)\cos(2\pi\nu x + j\pi\frac{x}{L})dx\end{aligned} \tag{12.125}$$

$$\begin{aligned}\Phi_j(\nu-\nu_0) &= \frac{a_j}{a_0\cdot L}\int_0^L \cos(2\pi\nu_0 x - 2\pi\nu x + j\pi\frac{x}{L})dx \\ &+ \frac{a_j}{a_0\cdot L}\int_0^L \cos(2\pi\nu_0 x - 2\pi\nu x - j\pi\frac{x}{L})dx \\ &+ \frac{a_j}{a_0\cdot L}\int_0^L \cos(2\pi\nu_0 x + 2\pi\nu x + j\pi\frac{x}{L})dx \\ &+ \frac{a_j}{a_0\cdot L}\int_0^L \cos(2\pi\nu_0 x + 2\pi\nu x - j\pi\frac{x}{L})dx \end{aligned} \tag{12.126}$$

$$y \equiv 2\pi\cdot L\cdot(\nu-\nu_0) \tag{12.127}$$

$$z \equiv 2\pi\cdot L\cdot(\nu+\nu_0) = 2\pi\cdot L\cdot 2\cdot\nu_0 + y \tag{12.128}$$

$$\Phi_j(\nu-\nu_0) = \frac{a_j}{a_0}\left[\frac{\sin(y+j\pi)}{y+j\pi} + \frac{\sin(y-j\pi)}{y-j\pi} + \frac{\sin(z+j\pi)}{z+j\pi} + \frac{\sin(z-j\pi)}{z-j\pi}\right] \tag{12.129}$$

$$\sin(\alpha \pm j\pi) = (-1)^j\cdot\sin(\alpha) \tag{12.130}$$

$$\Phi_j(\nu-\nu_0) = (-1)^j\cdot\frac{2\cdot a_j}{a_0}\left[\frac{\sin(y)}{y}\cdot\frac{y^2}{y^2-(j\pi)^2} + \frac{\sin(z)}{z}\cdot\frac{z^2}{z^2-(j\pi)^2}\right] \tag{12.131}$$

12.12.2 Noise Correlation of Cosine Apodized Spectra

In Section 12.12 we showed that any apodization function can be written in terms of a cosine apodization function is composed of J terms in Eqn. 12.116, which is rewritten below.

$$A(x) = a_0 + 2\sum_{j=1}^{J-1} a_j\cdot\cos\left(j\cdot\frac{\pi x}{L}\right) \tag{12.132}$$

and this was shown to be equivalent to a linear combination of Nyquist sampled de-apodized spectral channels

$$R_A(n) = a_0\cdot R(n) + \sum_{j=1}^{J-1} a_j\cdot(R(n-j)+R(n+j)) \tag{12.133}$$

We could have written the apodized radiance as a $2J-1$ point running mean with weights, w_k, given by

$$R_A(n) = \sum_{k=-J+1}^{J-1} w_k\cdot R(n+k) \tag{12.134}$$

and we can define weights as

$$w_k = a_{|k|}, \quad \text{for} \quad k = -(J-1), J-1. \tag{12.135}$$

For example, Hamming apodization has $J=2$ and $a_0=0.54$ and $a_1=0.23$. So that $w_{-1}=0.23, w_0=0.54, w_1=0.23$ and $\sum w_k = 1$.

If we assume that the instrumental noise is random, $\sigma(R(n)) \cdot r^i(n)$, on the unapodized spectrum and has Gaussian statistics (*i.e.*, $\sigma(r^i) = 1$ & $< r^i >= 0$ for a large number of samples, i). The noise on a given channel is given by its statistical standard deviation, $\sigma(R(n)$, that is the standard deviation over a large number of samples of a constant source, $R(n)$, is $\sigma(R(n))$. Further, for unapodized channels we assume the noise from one channel is uncorrelated with its neighbors, that is for a statistically large sample (large value of I) the correlation would be

$$\begin{aligned} C(m,n) &= 1 \qquad \text{for} \quad m = n \\ &= \frac{\sum_{i=1}^{I} r^i(m) \cdot r^i(n)}{\sqrt{\sum_{i=1}^{I} r^i(m)^2} \cdot \sqrt{\sum_{i=1}^{I} r^i(n)^2}} = 0 \quad \text{for} \quad m \neq n. \end{aligned} \tag{12.136}$$

To compute the noise on the apodized spectrum we can imagine adding noise spectrum to the noise free unapodized radianced, $R(n)$,

$$\tilde{R}_A^i(n) = \sum_{k=-J+1}^{J-1} w_k \cdot \big(R(n+k) + \sigma(n) \cdot r^i(n)\big) \tag{12.137}$$

$$= R_A(n) + \sigma(R(n)) \cdot \sum_{k=-J+1}^{J-1} w_k \cdot r^i(n) \tag{12.138}$$

But we know that $\sum r^i = 0$ statistically. If we assume the standard deviation, $\sigma(R)$, is both a smooth and weak function of channel number, n, then we can compute the standard deviation of $\tilde{R}_A^i(n)$ relative to the standard deviation of $R(n)$ we find that

$$\begin{aligned} \sigma^2(\tilde{R}_A(n)) &= \frac{1}{I} \sum_{i=1}^{I} \sum_{k=-J+1}^{J-1} \left(\tilde{R}_A^i(n) - R_A(n)\right)^2 \\ &= \sigma^2(R(n)) \cdot \sum_{k=-J+1}^{J-1} \left(w_k \cdot \frac{1}{I} \sum_{i=1}^{I} r^i(n)\right)^2 \\ &= \sigma^2(R(n)) \cdot \sum_{k=-J+1}^{J-1} w_k^2 \end{aligned} \tag{12.139}$$

Thus apodization has a noise reduction by a factor of

$$f \equiv \sqrt{\frac{\sigma^2(R(n))}{\sigma^2(\tilde{R}_A(n))}} = \frac{\sigma(R(n))}{\sigma(R(n) \cdot \sqrt{\sum_{k=-J+1}^{J-1} w_k^2}} = \frac{1}{\sqrt{\sum_{k=-J+1}^{J-1} w_k^2}} \tag{12.140}$$

which for Hamming apodization equals 1.58. But nothing comes for free. If we look at the statistical (*i.e.*, large I) correlation of the noise in adjacent channels m and n when $m \neq n$ we find

$$C_A(m,n) = \frac{\sum_{i=1}^{I} \tilde{R}_A^i(m) \cdot \tilde{R}_A^i(n)}{\sqrt{\sum_{i=1}^{I} \tilde{R}_A^i(m)^2} \cdot \sqrt{\sum_{i=1}^{I} \tilde{R}_A^i(n)^2}} \neq 0 \quad \text{when} \quad m \leq |n \pm J| \tag{12.141}$$

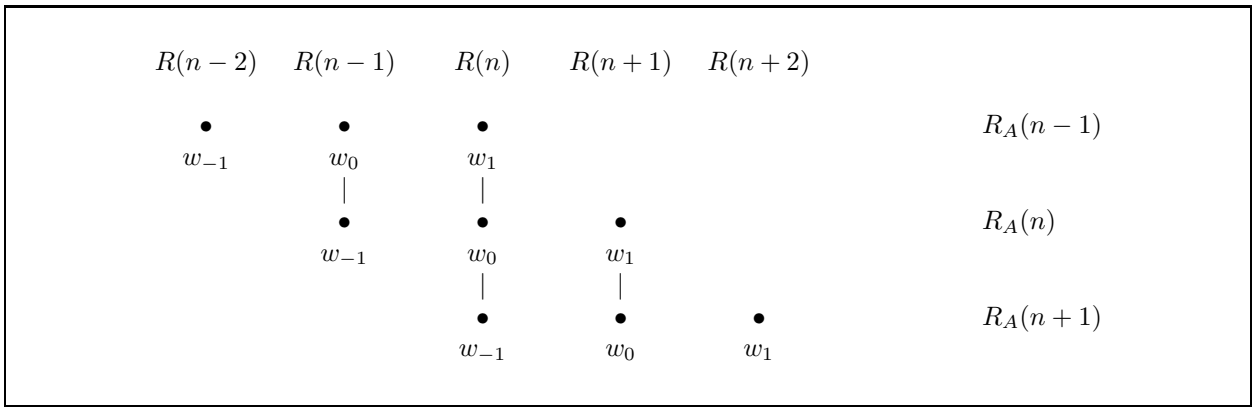


Figure 12.20: Illustration of correlation of Nyquist sampled apodized spectra for a 1 channel separation

If we assume that $\sigma(R(m)) \simeq \sigma(R(n))$ then we can expand the terms to find the correlation when $m = n \pm 1$

$$C_A(m = n \pm 1, n) = \frac{w_0 w_{-1}\sigma^2(\tilde{R}(n)) + w_1 w_0 \sigma^2(\tilde{R}(n))}{\sqrt{\frac{\sigma^2(R(n))}{f^2}} \cdot \sqrt{\frac{\sigma^2(R(n))}{f^2}}} \tag{12.142}$$

$$= f^2 \left(w_0 \cdot w_{-1} + w_1 \cdot w_0\right) \tag{12.143}$$

$$= 1.58^2 \left(0.23 \cdot 0.54 + 0.54 \cdot 0.23\right) = 62.5\% \quad \text{for Hamming}$$

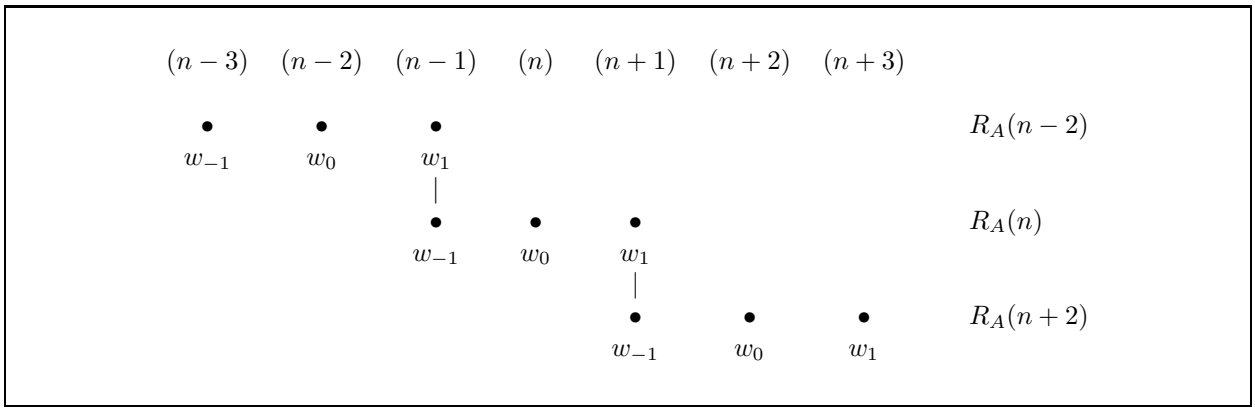


Figure 12.21: Illustration of correlation of Nyquist sampled apodized spectra for a 2 channel separation

$$C_A(m = n \pm 2, n) = f^2 \left(w_1 \cdot w_{-1}\right) = 1.58^2 \left(0.23 \cdot 0.23\right) = 13.3\% \quad \text{for Hamming} \tag{12.144}$$

In general we found that all apodization functions can be written in terms of an infinite cosine expansion, therefore, the expansion of an apodization function into a cosine series allows calculation of the noise reduction factor and spectral correlation in adjacent channels as shown above. In general,

$$f = \left[\sum_{k=1-J}^{J-1} w_k^2\right]^{-\frac{1}{2}} \tag{12.145}$$

and it can be shown that the channel correlation for a channel separated by $\pm n$ Nyquist channels is given by

$$C_n = f^2 \sum_{k=1-J}^{J-1-n} w_k \cdot w_{k+n}, \quad \text{for} \quad n = 1, 2J-2. \tag{12.146}$$

Table 12.5: Correlation characteristics of common apodization functions

apodization	FWHM	reduction	c_1	c_2	c_3	c_4	$\ldots c_{10}$
3pt running		1.733	66.6667	33.3333			
5pt running		2.236	80.0000	60.0000	40.0000	20.0000	
Hamming	1.5043	1.5863	62.5063	13.3115	0.0000	0.0000	
Triangle	1.4683	1.7147	60.3836	14.8953	6.7095	3.7238	0.5957
Blackman	1.9050	1.8119	75.5089	31.5496	6.5660	0.5253	
Kaiser-Bessel							
K=1	1.0359	1.0749	9.1370	-1.9162	0.8287	-0.4618	-0.0732
K=2	1.1286	1.2285	27.6766	-3.1053	1.3428	-0.7465	-0.1181
K=3	1.2492	1.3712	43.5409	0.0624	0.6235	-0.4029	-0.0710
K=4	1.3766	1.4838	54.3046	5.7960	0.0050	-0.0847	-0.0258
K=5	1.5011	1.5746	61.6064	12.1233	0.2256	0.0026	-0.0067
K=6	1.6196	1.6513	66.8479	18.1986	1.2951	-0.0028	-0.0013
K=7	1.7317	1.7183	70.8069	23.7595	3.0023	0.0591	-0.0002
K=8	1.8378	1.7782	73.9116	28.7617	5.1313	0.2834	$-5.1\cdot10^{-6}$
K=9	1.9386	1.8324	76.4152	33.2362	7.5132	0.7066	$-7.8\cdot10^{-7}$
K=10	2.0347	1.8822	78.4784	37.2371	10.0268	1.3296	0.0000
Connes	1.2268	1.3502	41.2239	-0.5562	0.7509	-0.4688	-0.0804
Beer	1.5780	1.5687	62.7836	11.2737	-1.1889	0.2240	-0.0602
Norton-Beer							
weak	1.1217	1.2581	28.1799	1.0934	-0.1873	0.1093	0.0265
weak	1.2000	1.3611	40.0355	2.8320	-0.2993	0.1038	0.0220
medium	1.2961	1.4531	50.1567	5.8980	-0.5895	0.1679	0.0297
medium	1.4000	1.5141	56.9287	8.3481	-0.7535	0.1452	0.0154
strong	1.5011	1.6039	63.0906	14.8601	0.7006	-0.0220	-0.0003
strong	1.6000	1.6487	66.6350	18.1314	1.1797	-0.0366	-0.00007
Dolph Chebyshev							
s=30 dB		1.4025	46.39	1.77	.406	1.27	
s=50 dB		1.6108	64.03	15.08	.758	1.54	
s=70 dB		1.7616	73.09	27.48	4.60	1.80	
ASE (Amato et al.)							
p=1,λ=.20		1.9074	71.07	34.89	15.62	1.54	
p=2,λ=.02		1.6845	74.41	28.09	-.22	1.83	

Chapter 13

Least Squares Solution of Linear Equations

Real observations have measurement noise, and as such, the solution of a set of linear equations becomes an issue of the "best" solution for a given set of observations. We will consider linear least squares fitting of a set of observations, $(y(n), t(n[,m]))$, to a linear *fitting* equation, given by

$$y_n^c = f\left(t_{n[,i]}, a_j\right) = X_{n,j} \cdot a_j \tag{13.1}$$

$$\begin{bmatrix} y^c(1) \\ y^c(2) \\ \dots \\ y^c(N) \end{bmatrix} = \begin{bmatrix} X(1,1) & X(1,2) & \dots & X(1,J) \\ X(2,1) & X(2,2) & \dots & X(2,J) \\ \dots & \dots & \dots & \dots \\ X(N,1) & X(N,2) & \dots & X(N,J) \end{bmatrix} \cdot \begin{bmatrix} a(1) \\ a(2) \\ \dots \\ a(J) \end{bmatrix} \tag{13.2}$$

where, y_n^c are the computed (modeled) values corresponding to the N dependent measurements of y_n, a_j are J parameters. The independent variable, t_n, can, in general, have multiple variables, m. The matrix $X_{n,j}$ is the linear expression of $f(t_{n[,m]}, a_j)$ which is the relationship between the observations, y_n, and the parameters, a_j. We can define some terms:

- The case is *over-determined* when there are more observations than parameters, $N > J$ and the solution is stable.
- The case can be *under-determined* if there are more parameters than observations, $J > N$, however it can be
- *ill-posed* if the linear model does not adequately describe the relationship between the observations and the parameters due to
  - an incomplete model
  - an incomplete sample of observations, such that a parameter can not be determined.
  - low signal-to-noise

The under-determined or ill-posed case will be discussed in the section on non-linear least squares (see Section 14.3) in the next chapter. Here we will only consider the over-determined case in which the model, X, is reasonable and accurate.

The *best fit* to the data via minimization of the square of the residuals is given by

$$\text{MIN}\left[\sum_{n=1}^{N}\left(y_n - f\left(t_{n[,m]}, a_j\right)\right)^2\right] \tag{13.3}$$

The problem is *linear* if $f\left(t_{n[,m]}, a_j\right)$ is linear in the components of a_j. For example, in the case where $y_n = f(t_n, a_j)$ then

$$y_n^c = f\left(t_n, a_j\right) = a_0 \cdot \phi_0(t_n) + a_1 \cdot \phi_1(t_n) + a_2 \cdot \phi_2(t_n) + \ldots \tag{13.4}$$

$$\begin{bmatrix} y_1^c \\ y_2^c \\ \ldots \\ a_N^c \end{bmatrix} = \begin{bmatrix} \phi_0(t_1) & \phi_1(t_1) & \phi_2(t_1) \\ \phi_0(t_2) & \phi_1(t_2) & \phi_2(t_2) \\ \ldots & \ldots & \ldots \\ \phi_0(t_N) & \phi_1(t_N) & \phi_2(t_N) \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \tag{13.5}$$

Another example is statistical regression where many observations are used to determine one or more parameters (see Section 13.3), $y_n = f(t_{n,i}, a_j)$

$$y_n^c = f\left(t_{n,i}, a_j\right) = a_0 \cdot \phi_0(t_{n,1}) + a_1 \cdot \phi_1(t_{n,2}) + \ldots \tag{13.6}$$

or any combination of Eqn. 13.4 or Eqn. 13.6

13.1 Example #1: fitting to a polynomial

For a polynomial, $y_n = a_0 + a_1 \cdot t_n + a_2 \cdot t_n^2 + \ldots$ the matrix $X_{n,j}$ in Eqn. 13.1 is written as

$$X_{n,j} = \begin{pmatrix} 1 & t_1 & t_1^2 & \ldots \\ 1 & t_2 & t_2^2 & \ldots \\ \ldots & \ldots & \ldots & \ldots \\ 1 & t_n & t_n^2 & \ldots \end{pmatrix} \tag{13.7}$$

which is known as a *Vandermonde* matrix.

13.2 Example #2: fitting to a sine function

Suppose we wanted to fit to a function such as,

$$y_n = a_0 + a_1 \cdot t_n + a_2 \cdot t_n^2 + b_0 \cdot \sin\left(b_1 \cdot t_n + b_2\right) \tag{13.8}$$

where, b_0 is the amplitude of the oscillation, b_1 is a known constant (for example, if we are fitting to an annual cycle and t is the date in years, then $b_0 = 2\pi$) and b_2 is an unknown, but constant phase shift. We can rewrite the sine function using the relation $\sin(\alpha + \beta) = \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta)$. The we can write

$$y_n = a_0 + a_1 \cdot t_n + a_2 \cdot t_n^2 + a_3 \cdot \sin\left(b_1 \cdot t_n\right) + a_4 \cdot \cos\left(b_1 \cdot t_n\right) \tag{13.9}$$

where $a_3 = b_0 \cdot \cos(b_2)$ and $a_4 = b_0 \cdot \sin(b_2)$. Note that is b_1 or b_2 were not constant, then this problem would be *non-linear* and we could not write the equations in the form of Eqn. 13.1. This will be discussed in the section on non-linear least squares fitting. The X in Eqn. 13.1 for this example becomes

$$X_{n,j} = \begin{pmatrix} 1 & t_1 & t_1^2 & \sin(b_1 \cdot t_1) & \cos(b_1 \cdot t_1) \\ 1 & t_2 & t_2^2 & \sin(b_1 \cdot t_2) & \cos(b_1 \cdot t_2) \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & t_n & t_n^2 & \sin(b_1 \cdot t_n) & \cos(b_1 \cdot t_n) \end{pmatrix} \tag{13.10}$$

upon solving this equation we can get the magnitude of the oscillation as $b_0 = \sqrt{a_3^2 + a_4^2}$ and the phase angle as $b_2 = \tan^{-1}(\frac{a_4}{a_3})$.

13.3 Example #3: Example of Statistical Regression

If we want to predict specific items from a large number of independent measurements we can use the form in Eq. 13.6 to fit our parameters. This method requires a *training* ensemble which is used to solve for the statistical relationship between the measurements and a physical quantity. For example, in atmospheric remote sensing we want to determine a geophysical parameter, say surface temperature, T, from a set of radiance measurements, $R(j)$. If we have a large ensemble of N cases with J observations (*e.g.*, channels of an instrument) and we have associated knowledge of the surface temperature (a *training* ensemble) we can solve for the statistical relationship

$$T_n = a_0 + R_{n,j} \cdot a_j \tag{13.11}$$

In practice, the average values of the independent and dependent variables over the training ensemble are used to *center* the equation. This helps maintain numerical stability.

$$\hat{T} \equiv \frac{1}{N} \sum_{n=1}^{N} T_n \tag{13.12}$$

$$\hat{R}_i \equiv \frac{1}{N} \sum_{n=1}^{N} R_{n,j} \tag{13.13}$$

It can be shown that $a_0 = \hat{T}$ for our N cases, so that Eqn. 13.11 becomes

$$T_n = \hat{T} + \left(R_{n,j} - \hat{R}_j\right) \cdot a_j \tag{13.14}$$

$$T_n - \hat{T} = \left(R_{n,j} - \hat{R}_j\right) \cdot a_j \tag{13.15}$$

$$y_n = X_{n,j} \cdot a_j \tag{13.16}$$

The coefficients, a_j, can be solved for via least squares methods (see Section 13.6, Eqn. 13.39). Once a_j is known then Eqn. 13.14 can be used to determine temperature directly from the radiances

$$T = \hat{T} + \left(R_j - \hat{R}_j\right) \cdot a_j \tag{13.17}$$

where, $\hat{R}_j$ and $\hat{T}$ are the average values from the training ensemble.

13.4 Least Squares Solution by the method of Maximum Likelihood

This section is adapted from Bevington (1969).

For any given single value of $t_{n=1} = t_n$ at $n = 1$, we can calculate the probability $P_{n=1}$ for making the observed measurement $y_{n=1}$, assuming a Gaussian distribution with a standard deviation of $\sigma_{n=1}$ for the observations about the actual value $y^c(t_{n=1})$. Note that Bevington's value of $\sigma^2_{n=1}$ is equivalent to the inverse a the diagonal weighting matrix, given in Section 13.5.

The probability of measuring $y_{n=1}$ is given by

$$P_{n=1} = \frac{1}{\sigma_{n=1}\sqrt{2\pi}} \cdot \exp\left(-\frac{1}{2}\left[\frac{y_{n=1} - y^c(t_{n=1})}{\sigma_{n=1}}\right]^2\right) \tag{13.18}$$

The probability for making the observed set of N measurements of the of y_n is the product of the single probabilities for the $n = 1, N$ observations. For the *true* parameters $\hat{a}_0$ and $\hat{a}_1$ this probability is

$$P(\hat{a}_0, \hat{a}_1) = \prod_{n=1}^{N} P_n = \prod_{n=1}^{N} \left(\frac{1}{\sigma_n \sqrt{2\pi}} \right) \cdot \exp\left(-\frac{1}{2} \sum_{n=1}^{N} \left[\frac{y_n - \hat{y}^c(t_n)}{\sigma_n} \right]^2 \right) \tag{13.19}$$

Similarly, for any estimated values of the parameters a_0 and a_1 we can calculate the probability that could make the observed set of measurements

$$P(a_0, a_1) = \prod_{n=1}^{N} \left(\frac{1}{\sigma_n \sqrt{2\pi}} \right) \cdot \exp\left(-\frac{1}{2} \sum_{n=1}^{N} \left[\frac{y_n - y^c(t_n)}{\sigma_n} \right]^2 \right) \tag{13.20}$$

The method of *maximum likelihood* relies on the assumption that the observed set of measurements is more likely to have come from the actual parent distribution $(\hat{a}_0, \hat{a}_1)$ than from any other similar distribution with different coefficients and, therefore, the probability given by Eqn. 13.19 is the maximum probability attainable with Eqn. 13.20.

The first term of Eqn. 13.20 is a constant, independent of the values of a_0 and a_1. Thus, maximizing the probability (a_0, a_1) is equivalent to minimizing the sum in the exponential. We can define the quantity χ^2 to be this sum

$$\chi^2 = \sum_{n=1}^{N} \left[\frac{1}{\sigma_n^2} \left(y_n - a_0 - a_1 \cdot t_n \right)^2 \right] \tag{13.21}$$

which is simply a statement that minimization of the squares is the most probable distribution.

An example of minimum variance is given by the following. Assume we want to know how many fish are in a pond. We pull out 100 fish and mark them and toss them back in. Then we wait awhile (to allow the Gaussian distribution to occur) and select 100 fish and notice that 10 are marked. We can then infer that the pond holds 1000 fish.

13.5 Least Squares Solution by method of Gauss (c. 1800)

This section is adapted from Heath 1997 (pg. 84) and Hamilton 1964 (pg. 124-127).

In general, we may want to weight the observations. For example, a common practice is to use the inverse of the error co-variance of the observations, $W_{n,n} = \left[\delta Y_n^T \delta Y_n \right]^{-1}$. In cases where errors in the observations are uncorrelated the noise covariance matrix can be written as $\sigma^2(n) \cdot I_{n,n}$ where $\sigma(n)$ is the error (standard deviation) in y_n and $I_{n,n}$ is the identity matrix (diagonal is one). The weighting matrix is then $W_{n,n} = [\sigma(n) \cdot I_{n,n}]^{-1}$ which is a diagonal matrix with elements along the diagonal equal to $W_{n,n} = 1/\sigma(n)$. For apodized interferometer radiances the noise covariance will have significant off diagonal terms that are discussed in Sectionnoisecorr.

The residual of our linear equation is written as $r_n \equiv y_n - X_{n,j} \cdot a_j$ for a given set of parameters a_j. We can take the inner product to compute a scalar quantity that can be minimized, $r^T \cdot r$. This is in contrast to the covariance of r, given by $r \cdot r^T$

$$\left(r \cdot r^T \right)_{n,n} = \begin{bmatrix} r(1) \\ r(2) \\ \dots \\ r(N) \end{bmatrix} \cdot \begin{bmatrix} r(1) & r(2) & \dots & r(N) \end{bmatrix} = \begin{bmatrix} r(1) \cdot r(1) & r(1) \cdot r(2) & \dots & r(1) \cdot r(N) \\ r(2) \cdot r(1) & r(2) \cdot r(2) & \dots & r(2) \cdot r(N) \\ \dots & \dots & \dots & \dots \\ r(N) \cdot r(1) & r(N) \cdot r(2) & \dots & r(N) \cdot r(N) \end{bmatrix} \tag{13.22}$$

The scalar quantity we want to compute is given by

$$\left(r^T \cdot r \right)_{1,1} = \begin{bmatrix} r(1) & r(2) & \dots & r(N) \end{bmatrix} \cdot \begin{bmatrix} r(1) \\ r(2) \\ \dots \\ r(N) \end{bmatrix} \tag{13.23}$$

The weighted inner product of the residuals is given by

$$\left(r^T \cdot W \cdot r\right)_{1,1} = \left(y_n - X_{n,j} \cdot a_j\right)^T \cdot W_{n,n} \cdot \left(y_n - X_{n,j} \cdot a_j\right) \tag{13.24}$$

and expanding the terms (NOTE: $(A \cdot B)^T = B^T \cdot A^T$ and $(A + B)^T = A^T + B^T$).

$$\left(r^T \cdot W \cdot r\right)_{1,1} = y_n^T \cdot W_{n,n} \cdot y_n - a_j^T \cdot X_{j,n}^T \cdot W_{n,n} \cdot y_n - y_n^T \cdot W_{n,n} \cdot X_{j,n} \cdot a_j + a_j^T \cdot X_{j,n}^T \cdot W_{n,n} \cdot X_{n,j} \cdot a_j \tag{13.25}$$

But we can note that each term is a scalar quantity so that

$$a_j^T \cdot X_{j,n}^T \cdot W_{n,n} \cdot y_n = y_n^T \cdot W_{n,n} \cdot X_{j,n} \cdot a_j \tag{13.26}$$

so that

$$\left(r^T \cdot W \cdot r\right)_{1,1} = y_n^T \cdot W_{n,n} \cdot y_n - 2 \cdot a_j^T \cdot X_{j,n}^T \cdot W_{n,n} \cdot y_n + a_j^T \cdot X_{j,n}^T \cdot W_{n,n} \cdot X_{n,j} \cdot a_j \tag{13.27}$$

and note that

$$\begin{aligned} \frac{\partial \left(a_j^T \cdot X_{j,n}^T \cdot W_{n,n} \cdot X_{n,j} \cdot a_j\right)}{\partial a_j^T} &= X_{j,n}^T \cdot W_{n,n} \cdot X_{n,j} \cdot a_j + a_j^T \cdot X_{j,n}^T \cdot W_{n,n} \cdot X_{n,j} \\ &= X_{j,n}^T \cdot W_{n,n} \cdot X_{n,j} \cdot a_j + X_{j,n}^T \cdot W_{n,n} \cdot X_{n,j} \cdot a_j \\ &= 2 \cdot X_{j,n}^T \cdot W_{n,n} \cdot X_{n,j} \cdot a_j \end{aligned} \tag{13.28}$$

so that our equation simplifies to

$$\frac{\partial \left(r^T \cdot W \cdot r\right)_{1,1}}{\partial a_j^T} = 2 \cdot X_{j,n}^T \cdot W_{n,n} \cdot X_{n,j} \cdot a_j - 2 \cdot X_{j,n}^T \cdot W_{n,n} \cdot y_n = 0 \tag{13.29}$$

or,

$$\boxed{X_{j,n}^T \cdot X_{n,j} \cdot a_j = X_{j,n}^T \cdot y_n} \tag{13.30}$$

which is commonly known as the *normal equations*. If equations are *linearly independent*, then $X_{j,n}^T \cdot X_{n,j}$ is a non-singular matrix.

13.6 Least Squares Solution by linear algebra

The sum of residuals squared are minimized when we pick coefficients a_j such that the residuals of the observed, y_n minus computed, $y_n^c \equiv f(X_{n,j}, a_j)$, values are minimized.

$$\sum_{n=1}^{N} \left(y_n - y_n^c\right)^2 \qquad \text{is a minima} \tag{13.31}$$

This occurs when the derivative is zero

$$2 \sum_{n=1}^{N} \left(y_n - y_n^c\right) = 0 \tag{13.32}$$

and substitution of Eqn. 13.1 into the equation above yields

$$\sum_{n=1}^{N} y_n = \sum_{n=1}^{N} X_{n,j} \cdot a_j \tag{13.33}$$

One might be inclined to assume that $y_n = X_{n,j} \cdot a_j$ and solve for $a_j = X^{-1}_{j,n} \cdot y_n$; however, it can be shown (see section D.6) that for a non-square matrix $X^{-1} \equiv [X^T \cdot X]^{-1} \cdot X^T$. In reality, in a given set of measurements there is an error vector, $\epsilon(n)$, such that

$$y_n = X_{n,j} \cdot a_j + \epsilon(n) \tag{13.34}$$

and the method of least squares minimizes the sum of $\epsilon^2(n)$ which implies that $\sum_{n=1}^{N} \epsilon(n) = 0$ for a Gaussian probability density function. Note that, in general, the model, $X_{n,j}$, is not necessarily the correct model for these observations and ϵ represents the difference between the linear model $X_{n,j}$ and the real function $f(t_{n[,m]})$.

We can obtain Eqn. 13.30 in Section 13.5 if we simply multiply both sides of the $y_n = X_{n,j} \cdot a_j$ by the product of the transpose of the independent variable matrix, $X^T_{j,n}$, and the weighting function, $W_{n,n}$ as follows

$$X^T_{j,n} \cdot W_{n,n} \cdot y_n = X^T_{j,n} \cdot W_{n,n} \cdot X_{n,j} \cdot a_j \tag{13.35}$$

finally, multiply both sides by the inverse of $X^T_{j,n} \cdot W_{n,n} \cdot X_{n,j}$ to solve for the parameters of the linear least squares fit

$$a_j = \left[X^T_{j,n} \cdot W_{n,n} \cdot X_{n,j}\right]^{-1}_{j,j} X^T_{j,n} \cdot W_{n,n} \cdot y_n \tag{13.36}$$

and *voila* you have a set of coefficients that are the *best fit* to the observed data using the linear model $X_{n,j} \simeq f(t_{n[,m]})$.

NOTE: The authors of linear algebra textbooks (*e.g.*, Heath, 1997) point out that one never needs to compute the inverse in Eqn. 13.36; however, the inverse has information about the accuracy of the parameters (to be discussed in the next lecture). While it is correct that one never **needs** to compute the inverse it is sometimes easier to write the problem in this way. If you are concerned about execution time and/or propagation of roundoff errors, then you should consult Heath, 1997 on these issues.

To solve the statistical regression form of linear least squares we need to use the form of Eq. 13.14. To simplify the notation substitute $\tilde{T}_n \equiv T_n - \hat{T}$ and $\tilde{R}_{n,j} \equiv R_{n,j} - \hat{R}_j$ so that Eq. 13.14 becomes

$$\tilde{T}_n = \tilde{R}_{n,j} \cdot a_j \tag{13.37}$$

Then we can multiply both sides by the transpose of $\tilde{R}_{n,j}$

$$\tilde{R}^T_{j,n} \cdot \tilde{T}_n = \tilde{R}_{n,j} \cdot \tilde{R}_{n,j} \cdot a_j \tag{13.38}$$

and the least squares solution is given by

$$a_j = \left[\tilde{R}^T_{n,j} \cdot \tilde{R}_{n,j}\right]^{-1}_{j,j} \cdot \tilde{R}^T_{j,n} \cdot \tilde{T}_n \tag{13.39}$$

For instruments with a large number of channels n (*e.g.*, high resolution Michelson interferometers) the inverse can become difficult. Eigenvector methods can be employed to change the dimension of this inverse matrix to an reasonable size as well as stabalizing the matrix in low signal-to-noise domains (see Section 14.3.3). We can weight the equation by the inverse of the noise covariance matrix (see PHYS741 notes); however, in this case we will find that it will have no useful effect in this formulation because the statistical nature of the measurements is already represented in the radiance covariance, $R_{n,j} \cdot \tilde{R}_{n,j}$.

13.7 Estimation of the errors in the coefficients

We can only estimate the standard deviation of the errors in the root sum squared (RSS) sense. That is, we cannot estimate biases caused by the random noise added to the observations. Therefore, to estimate the

errors in the parameters, δa_j, due to error in the observations, δy_n, we can propagate the error in the RSS sense through Eqn. 13.36 by taking the error equation times its transpose

$$\left(\delta a \cdot \delta a^T\right)_{j,j} = \left(\left[X^T \cdot W \cdot X\right]^{-1}_{j,j} X^T_{j,n} \cdot W_{n,n} \cdot \delta y_n\right) \cdot \left(\left[X^T \cdot W \cdot X\right]^{-1}_{j,j} X^T_{j,n} \cdot W_{n,n} \cdot \delta y_n\right)^T \tag{13.40}$$

where we have used the notation $\left[X^T \cdot W \cdot X\right]_{j,j}$ to represent $\left[X^T_{j,n} \cdot W_{n,n} \cdot X_{j,n}\right]$. The transpose of a product of matrices is equal to the product of the transpose of the matrices in reverse order $(A \cdot B \cdot C)^T = C^T \cdot B^T \cdot A^T$. In practice, the weighting function is a square matrix such that $W^T_{n,n} = W_{n,n}$ and $\left[X^T \cdot W \cdot X\right]_{j,j}$ and its inverse are also square matrices so that

$$\left(\delta a \cdot \delta a^T\right)_{j,j} = \left[X^T \cdot W \cdot X\right]^{-1}_{j,j} X^T_{j,n} \cdot W_{n,n} \cdot \delta y_n \cdot \delta y^T_n \cdot W_{n,n} \cdot X_{j,n} \cdot \left[X^T \cdot W \cdot X\right]^{-1}_{j,j} \tag{13.41}$$

If the weighting matrix, $W_{n,n}$ is built from the inverse of the error covariance of the observations, $W_{n,n} = (\delta y_n \cdot \delta y_n)^{-1}$, then the equation for the error covariance in the parameters is simplified:

$$\begin{aligned}
\left(\delta a \cdot \delta a^T\right)_{j,j} &= \left[X^T \cdot W \cdot X\right]^{-1}_{j,j} X^T_{j,n} \cdot W_{n,n} \cdot \delta y_n \cdot \delta y^T_n \cdot W_{n,n} \cdot X_{n,j} \cdot \left[X^T \cdot W \cdot X\right]^{-1}_{j,j} \\
&= \left[X^T \cdot W \cdot X\right]^{-1}_{j,j} X^T_{j,n} \cdot W_{n,n} \cdot W^{-1}_{n,n} \cdot W_{n,n} \cdot X_{n,j} \cdot \left[X^T \cdot W \cdot X\right]^{-1}_{j,j} \\
&= \left[X^T \cdot W \cdot X\right]^{-1}_{j,j} X^T_{j,n} \cdot W_{n,n} X_{n,j} \cdot \left[X^T \cdot W \cdot X\right]^{-1}_{j,j} \\
&= \left[X^T \cdot W \cdot X\right]^{-1}_{j,j}
\end{aligned} \tag{13.42}$$

If the equation had been unweighted, or weighted by an arbitrary criteria, then we could compute the weighted value of $\delta y^T_n \cdot \delta y_n$ from the observed minus computed residuals $\delta y_n = y_n - X_{n,j} \cdot a_j$. In many applications the weighting function is a diagonal matrix with values $w_n = W_{n',n}$ when $n' = n$ and $W_{n',n} = 0$ when $n' \neq n$. The quality of the fit itself can be used to estimate $\left(\delta a \cdot \delta a^T\right)_{j,j}$. A quantity, χ^2 can be computed from the residuals of the solution and the values along the diagonal of $W_{n,n}$ as follows

$$\chi^2 = \frac{1}{N} \cdot \sum_{n=1}^{N} w_n \cdot (y_n - X_{n,j} \cdot a_j)^2 \tag{13.43}$$

$$\left(\delta a \cdot \delta a^T\right)_{j,j} = \left[X^T \cdot W \cdot X\right]^{-1}_{j,j} X^T_{j,n} \cdot W_{n,n} \cdot \left(\delta y_n \cdot \delta y^T_n \cdot W_{n,n}\right) \cdot X_{n,j} \cdot \left[X^T \cdot W \cdot X\right]^{-1}_{j,j} \tag{13.44}$$

$$= \left[X^T \cdot W \cdot X\right]^{-1}_{j,j} X^T_{j,n} \cdot W_{n,n} \cdot \left(\chi^2 \cdot I_{n,n}\right) \cdot X_{n,j} \cdot \left[X^T \cdot W \cdot X\right]^{-1}_{j,j} \tag{13.45}$$

$$= \chi^2 \left[X^T \cdot W \cdot X\right]^{-1}_{j,j} \tag{13.46}$$

In general, the error estimate is a little larger due to the number of degrees of freedom in the measurements. The correct value of χ^2 to use should reflect these degrees of freedom in the error estimate (e.g., if $N = J$ we would fit the measurements exactly; however, then we would be believing the noise 100% so that as we approach $N = J$ the uncertainty in a_j should grow.

$$\chi^2 \equiv \frac{1}{N - J} \cdot \sum_{n=1}^{N} w_n \cdot (y_n - X_{n,j} \cdot a_j)^2 \tag{13.47}$$

$$\left(\delta a \cdot \delta a^T\right)_{j,j} = \chi^2 \left[X^T \cdot W \cdot X\right]^{-1}_{j,j} \tag{13.48}$$

In the previous example where we know $\delta y_n \cdot \delta y^T_n$ either from measurement or empirical estimates (χ^2) we assumed that there was no error in the independent variable $X_{n,j}$ (*e.g.*, sampling errors). If this is not the case, then additional sources of error need to be accounted for. See Twomey 1977 for a discussion of the full error determination.

13.7.1 Error Estimation for Statistical Regression

One example where significant errors are present in both the independent and dependent variables is statistical regression. For example, in Eq. 13.37 ($\tilde{T}_n = \tilde{R}_{n,j} \cdot a_j$) both $\tilde{T}_n$ and $\tilde{R}_{n,j}$, where $\tilde{T}_n$ represents measurements of temperature for a statistically significant number of cases, N and $R_{n,j}$ represents the instrumental measurements corresponding to the N cases and j measurements (*e.g.*, channels) per case. $\tilde{T}_n$ has error because we never have complete knowledge of the geophysical data due to measurement error as well as the fact that the measurements are not necessarily measred at the same time or exact location (representation error). $\tilde{R}_{n,j}$ has measurement error. Recall that the solution to this equation was given in Eqn. 13.39

$$a_j = \left[\tilde{R}^T_{j,n} \cdot \tilde{R}_{n,j}\right]^{-1} \cdot \tilde{R}^T_{j,n} \cdot \tilde{T}_n \tag{13.49}$$

The error in the parameters, a_j, due to the training ensemble error, δT_n, is computed as analogously to the discussion in the previous section. This error may be large in practice, but is usually not known well and the χ^2 test is typically the best estimate

$$\left(\delta a \cdot \delta a^T\right)_{j,j} = \chi^2 \cdot \left[\tilde{R}^T_{j,n} \cdot \tilde{R}_{n,j}\right]^{-1}_{j,j} \tag{13.50}$$

where in this case χ^2 determined from the training ensemble of N cases and is given by

$$\chi^2 \equiv \frac{1}{N-J} \cdot \sum_{n=1}^{N} w_n \cdot \left(T_n - R_{n,j} \cdot a_j\right)^2 \tag{13.51}$$

The error in the observations, δR_j, is usually due to a random error in the instrument. For most instruments this measurement error covariance is a diagonal matrix; however, processing techiques such as apodization can make this matrix significantly off-diagonal. The error in a_j due to the error covariance of the observations used in training $\left(\delta\tilde{R}^T_{j,n}\delta\tilde{R}_{n,j}\right)$ is difficult to interpret because $\tilde{R}^T_{j,n}$ occurs in the inverse in Eqn. 13.49. One can compute multiple a_j's (with random sequences with proper instrumental statistics) are applied to the measurements and the variance in the resulting a_j's are used as an estimate of the error. This is computationally burdensome and only accurate if the instrumental statistics are accurate and a large number of random sequences are applied.

Since, in general, the error in the coefficients can be used to estimate the error propagated into the regression result ($T = R_j \cdot a_j$), that is, we really want to know the error estimate in T for a given set of radiances, R_j, with a error estimate of δR_j a *reasonable* estimate of the error in T can be given by

$$\left(\delta T \delta T^T\right)_{1,1} = \text{TRACE}\left(\delta R_{n,j} \cdot \left(\delta a \cdot \delta a^T\right)_{j,j} \delta R^T_{j,n}\right) + \text{TRACE}\left(a_j \cdot \left(\delta R^T_{j,n}\delta R_{n,j}\right) \cdot a^T_j\right) \tag{13.52}$$

This can become quite *ad hoc*, but it is computationally viable. When one considers that error analysis is only an estimate of the *statistical* error, the *ad-hoc* nature of this formulation becomes tolerable. If the measurement noise dominates then this reduces to the weighted root-sum-square (RSS) of the instrument noise for diagonal error covariance matrices

$$\delta T \simeq \sqrt{\sum_{n=1}^{N} \left(a_j \cdot \delta R_j\right)^2} \tag{13.53}$$

13.7.2 Example of order=1 polynomial least squares error analysis

We will use the IDL programs in the Appendix (Sections B.11 and B.10) to compute the Vandermonde matrix for $y(t)$) (ftp/lsq_vand) and solve (ftp/lsq_linear) for a polynomial of a specified order.

To test least squares error analysis we can simulate measurements by computing a polynomial and adding noise. We need a random number generator with a Gaussian distribution ($< r >= 0, \sigma(r) = 1$). We begin be specifying x_n and computing

$$y(n) = a_0 + a_1 \cdot x(n) + r(n) \quad \text{for} \quad n = 1, N \tag{13.54}$$

N =100 points were simulated with a_0 =8.0 and a_1=-5.0 over the interval $-10 \leq x \leq 10$. The random number sequence for these 100 points was $< r >= 0.05$ and $\sigma(r)$=0.97, which is reasonable for a sample of only 100 points. The results of a unweigted (*i.e.*, $W_{n,n} \equiv I_{n,n}$) least squares fit to these observations are shown in Fig. 13.1. The least squares fit for the 1st random sequence was a_0 =8.202 $\pm$ 0.39 and a_1=-5.004 $\pm$ 0.067, and $\chi^2 = 15.27$. The prediced error, using Eqn. 13.48, is given by

$$\left(\delta a \cdot \delta a^T\right)_{j,j} = \begin{bmatrix} 0.1527 & 2.48 \cdot 10^{-9} \\ 2.48 \cdot 10^{-9} & 0.004491 \end{bmatrix} \tag{13.55}$$

The predicted error, $\left(\delta a \cdot \delta a^T\right)_{j,j}$, depends on the Vandermone matrix which, in turn, depends on the range (*i.e,* $-10 \leq x \leq 10$) we selected for the least squares fit and the value of χ^2.

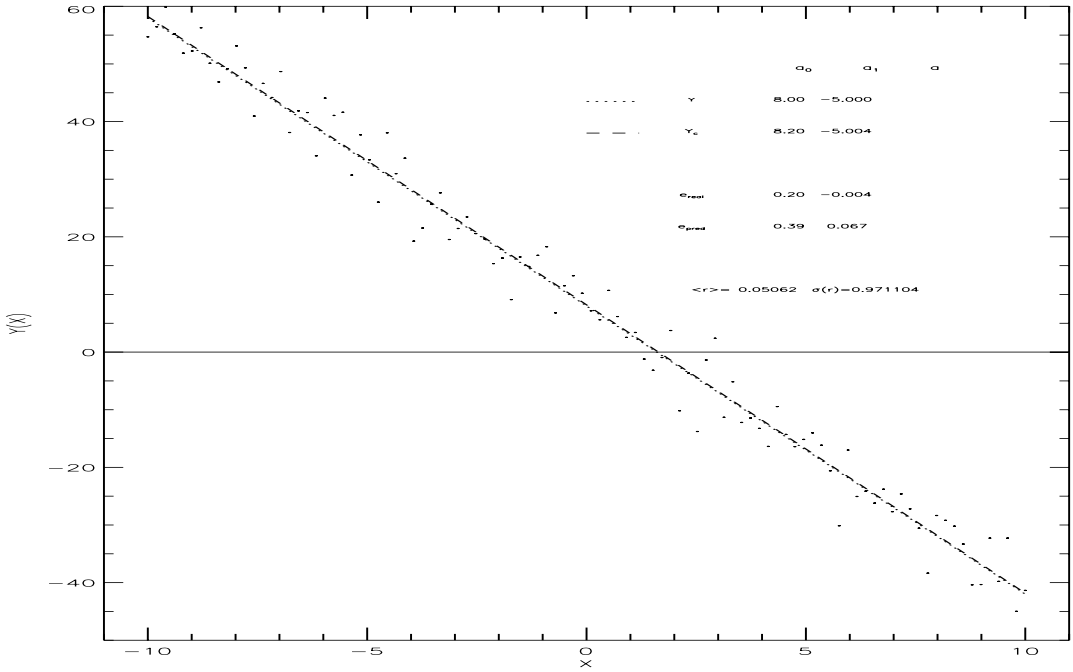


Figure 13.1: Example of a LSQ fit of a line to 1st order polynomial

We then ran this algorithm for 10 different random sequences. The determined coefficient and the computed error in the coefficients (determined value minus the known values of a_0 and a_1) are shown in the following table. Also shown is the square root of the diagonal of $\left(\delta a \cdot \delta a^T\right)_{j,j}$ for each run. Since this is a two parameter model, the diagonal of the error covariance has two terms, δa_0 and δa_1.

Least Squares Results from 10 random sequences						
	determined coefficients		computed errors		predicted errors	
sequence	a_0	a_1	δa_0	δa_1	δa_0	δa_1
1	8.202	-5.004	0.202	-0.004	0.391	0.067
2	8.481	-4.982	0.481	0.018	0.379	0.065
3	8.175	-4.920	0.175	0.080	0.344	0.059
4	7.755	-4.987	-0.245	0.013	0.416	0.071
5	7.829	-5.076	-0.171	-0.076	0.415	0.071
6	8.074	-5.089	0.074	-0.089	0.374	0.064
7	8.319	-4.941	0.319	0.059	0.457	0.078
8	8.296	-4.841	0.296	0.159	0.419	0.072
9	8.397	-5.080	0.397	-0.080	0.401	0.069
10	7.180	-5.035	-0.820	-0.035	0.410	0.070

The average, AVG(E(real)), root-mean-square, RMS(E(real)), and standard deviation, SDV(E(real)), of the 10 set's of computed errors are shown below. Also, shown is the root sum square of the predicted errors, RSS(E(pred)). Notice, that we predicted the statistical nature of the errors quite well.

statistic	for a_0	for a_1
AVG(E(real))	0.071	0.005
RMS(E(real))	0.376	0.076
SDV(E(real))	0.370	0.075
RSS(E(pred))	0.402	0.069

Notice that we can't predict the error of an individual case (if we could we would have improved the answer), but we can predict the statistical nature of the errors. The error covariance, computed from the real residuals of the $K = 10$ cases can be determined in our simulation experiment because we know the true answer, a_j^t and we compute answers for a_j^k for $k = 1, K$. The computed error covariance is given by

$$\delta a_i \delta a_j^T = \frac{1}{10} \sum_{k=1}^{10} \left(a_i^k - a_i^t\right) \left(a_j^k - a_j^t\right)^T = \begin{bmatrix} 0.14175 & 0.0088 \\ 0.0088 & 0.0057 \end{bmatrix} \tag{13.56}$$

If we use 100 sets of random numbers the computed error covariance is determined to be

$$\delta a_i \delta a_j^T = \frac{1}{100} \sum_{k=1}^{100} \left(a_i^k - a_i^t\right) \left(a_j^k - a_j^t\right)^T = \begin{bmatrix} 0.15994 & 0.00214 \\ 0.00214 & 0.00541 \end{bmatrix} \tag{13.57}$$

If we use 10000 sets of random numbers the computed error covariance is determined to be

$$\delta a_i \delta a_j^T = \frac{1}{10000} \sum_{k=1}^{10000} \left(a_i^k - a_i^t\right) \left(a_j^k - a_j^t\right)^T = \begin{bmatrix} 0.15976 & -0.000137 \\ -0.000137 & 0.00474 \end{bmatrix} \tag{13.58}$$

13.7.3 Example of order=2 polynomial least squares error analysis

100 points were simulated with a_0 =8.0, a_1=-5.0, and a_2=1.0 over the interval $-10 \le x \le 10$. The random number sequence for these 100 points was $< r >= 0.05$ and $\sigma(r)$=0.97 (same sequence as before). The results of a least squares fit to these observations are shown in Fig. 13.2. The unweighted least squares fit was a_0 =7.76 ± 0.59, a_1=-5.004 ± 0.067, a_2=1.007 ± 0.0129 with a χ^2 of 15.4. for this random sequence. The prediced error, using Eqn. 13.48, for the 1st case was

$$\left(\delta a \cdot \delta a^T\right)_{j,j} = \begin{bmatrix} 0.346164 & 4.10780 \cdot 10^{-9} & -0.00565592 \\ 4.10780 \cdot 10^{-9} & 0.00452337 & -4.72195 \cdot 10^{-11} \\ -0.00565592 & -4.72195 \cdot 10^{-11} & 0.000166318 \end{bmatrix} \tag{13.59}$$

We are beginning to see a correlation ($\delta a_0 \delta a_2$ = -0.00566) between the errors in a_0 and a_2. This can be interpreted as follows: if we increase a_0 the quadratic term must be smaller to fit the data. Therefore, the parameters a_0 and a_2 are related to a small degree.

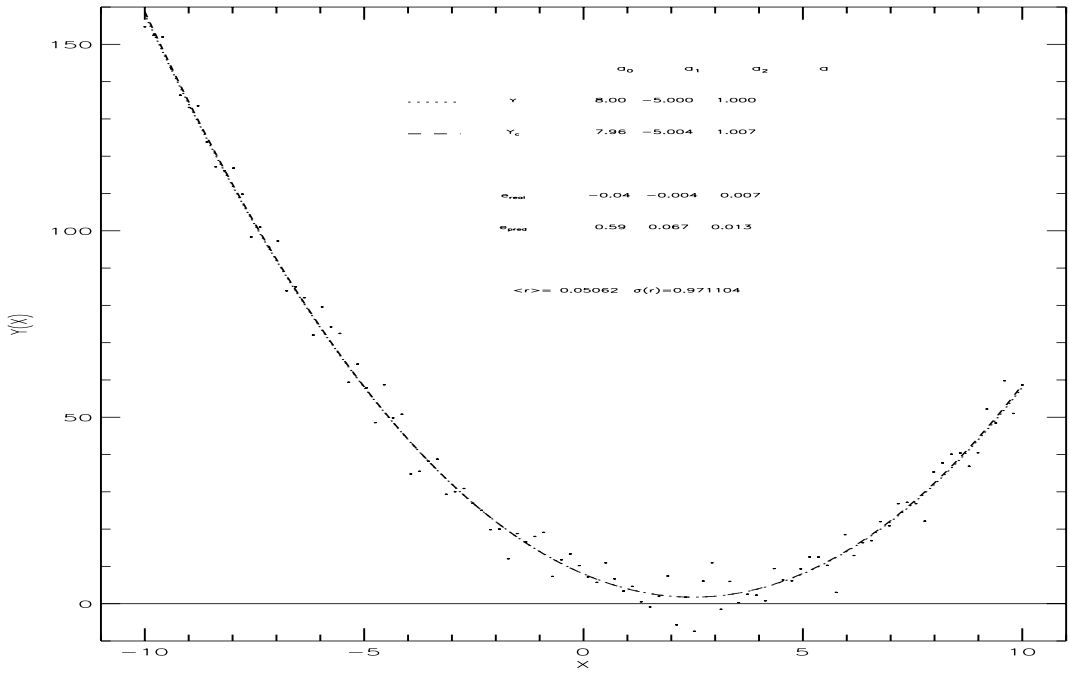


Figure 13.2: Example of a LSQ fit of a line to 2nd order polynomial

Least Squares Results from 5 random sequences						
	determined coefficients			predicted errors		
sequence	a_0	a_1	a_2	δa_0	δa_1	δa_2
1	7.96086	-5.00367	1.00710	0.58836	0.06726	0.01290
2	7.51419	-4.98244	1.02844	0.55677	0.06365	0.01220
3	8.66810	-4.92007	0.98551	0.51383	0.05874	0.01126
4	7.55177	-4.98724	1.00598	0.62596	0.07155	0.01372
5	8.13592	-5.07556	0.99097	0.62493	0.07144	0.01370

Running 10 set's of random number sequences yields the following statistics of the errors

statistic	for a_0	for a_1	for a_2
AVG(E(real))	-0.1165	0.00450	0.00551
RMS(E(real))	0.5622	0.07550	0.01298
SDV(E(real))	0.5500	0.07537	0.01176
RSS(E(pred))	0.6026	0.06888	0.01321

and the computed error covariance for $K = 10$ cases is

$$\delta a_i \delta a_j^T = \frac{1}{10}\sum_{k=1}^{10}\left(a_i^k - a_i^t\right)\left(a_j^k - a_j^t\right)^T = \begin{bmatrix} 0.3161 & 0.00565 & -0.00566 \\ 0.00565 & 0.0057 & 9.249\cdot 10^{-5} \\ -0.00566 & 9.249\cdot 10^{-5} & 0.0001686 \end{bmatrix} \quad (13.60)$$

and the computed error covariance for 1000 cases is

$$\delta a_i \delta a_j^T = \frac{1}{1000}\sum_{k=1}^{1000}\left(a_i^k - a_i^t\right)\left(a_j^k - a_j^t\right)^T = \begin{bmatrix} 0.3543 & 0.00126 & -0.00577 \\ 0.00126 & 0.00469 & 9.249\cdot 10^{-5} \\ -0.00577 & 9.249\cdot 10^{-5} & 0.000174 \end{bmatrix} \quad (13.61)$$

13.7.4 Example of order=4 polynomial least squares error analysis

$N = 100$ points were simulated with a_0 =10.0, a_1=-2.0, a_2=0.8, a_3 =-0.03, and a_4 =-0.002 over the interval $-10 \le x \le 10$. The random number sequence for these 100 points was $< r >= 0.05$ and $\sigma(r)$=0.97 (same sequence as before). The results of the least squares fit to these observations are shown in Fig. 13.3. The least squares fit was a_0 =9.741 ± 0.741, a_1=-2.081 ± 0.169, a_2=0.8287 ± 0.045, a_3=-0.029 ± 0.0025, and a_4=-0.0025 ± 0.0005 and χ^2=15.625 with this random sequence. The prediced error covariance, using Eqn. 13.48, for the 1st case was

$$\left(\delta a \cdot \delta a^T\right)_{j,j} = \begin{bmatrix} 0.549582 & 1.615\cdot 10^{-07} & -0.02516 & -2.354\cdot 10^{-09} & 0.0002221 \\ 1.615\cdot 10^{-07} & 0.0287 & -1.26\cdot 10^{-08} & -0.000394 & 1.343\cdot 10^{-10} \\ -0.02516 & -1.26\cdot 10^{-08} & 0.00207 & 1.84\cdot 10^{-10} & -2.178\cdot 10^{-05} \\ -2.354\cdot 10^{-09} & -0.000394 & 1.84\cdot 10^{-10} & 6.45\cdot 10^{-06} & -1.95\cdot 10^{-12} \\ 0.000222069 & 1.343\cdot 10^{-10} & -2.178\cdot 10^{-05} & -1.95\cdot 10^{-12} & 2.490\cdot 10^{-07} \end{bmatrix} \quad (13.62)$$

Again, 10 cases is too few to get the really low numbers, however, we can begin to see some similarities. Again, this would need to be run for 100 or 10000 cases to really see the true lack of correlation (small numbers) and to see that a_0 and a_2 are really correlated for this choice of dependent variables, which is all the predicted error is based on.

Running 10 set's of random number sequences yields the following statistics of the errors

statistic	for a_0	for a_1	for a_2	for a_3	for a_4
AVG(E(real))	-0.22216	0.00921	0.01588	-0.00008	-0.00012
RMS(E(real))	0.68362	0.17799	0.04409	0.00251	0.00043
SDV(E(real))	0.64652	0.17775	0.04114	0.00251	0.00041
RSS(E(pred))	0.75464	0.17256	0.04634	0.00258	0.00051

The computed error covariance for these $K = 10$ cases is

$$\delta a_i \delta a_j^T = \frac{1}{10}\sum_{k=1}^{10}\left(a_i^k - a_i^t\right)\left(a_j^k - a_j^t\right)^T \quad (13.63)$$

$$= \begin{bmatrix} 0.4673 & -0.0618 & -0.0219 & 0.000798 & 0.000167 \\ -0.0618 & 0.0317 & 0.00219 & -0.000406 & -1.69\cdot 10^{-5} \\ -0.0219 & 0.00219 & 0.00194 & -4.485\cdot 10^{-6} & -1.82\cdot 10^{-5} \\ 0.000798 & -0.000406 & -4.485\cdot 10^{-6} & 6.32\cdot 10^{-6} & -6.48\cdot 10^{-8} \\ 0.000167 & -1.69\cdot 10^{-5} & -1.82\cdot 10^{-5} & -6.48\cdot 10^{-8} & 1.85\cdot 10^{-7} \end{bmatrix} \quad (13.64)$$

and for 1000 cases is

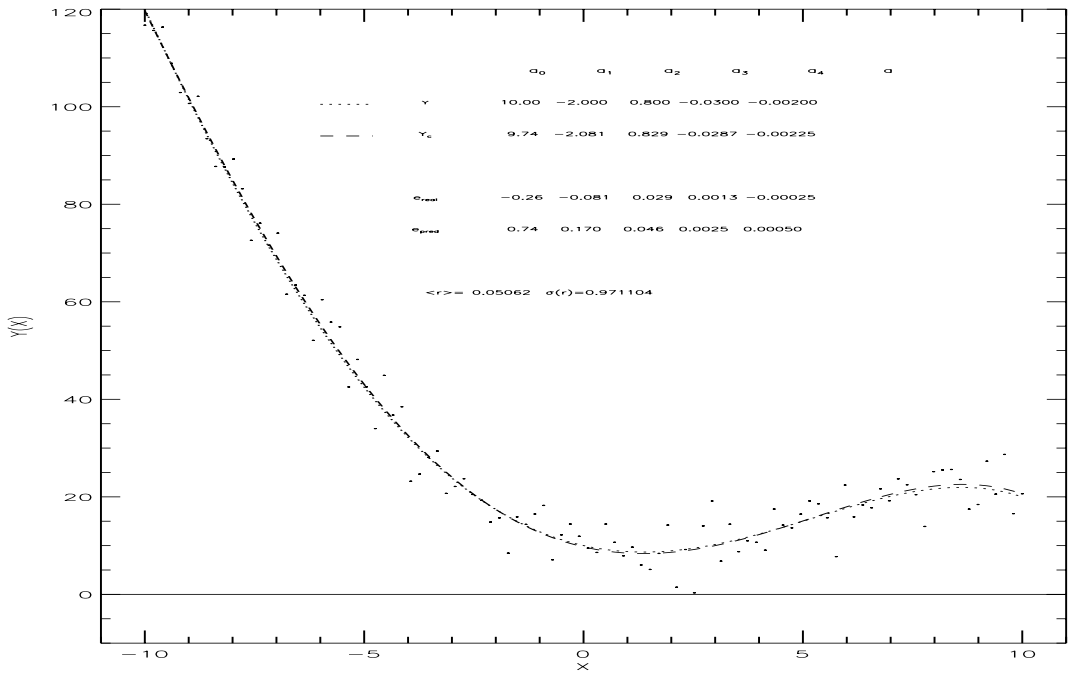


Figure 13.3: Example of a LSQ fit of a line to 4^{th} order polynomial

$$\delta a_i \delta a_j^T = \frac{1}{1000}\sum_{k=1}^{1000}\left(a_i^k - a_i^t\right)\left(a_j^k - a_j^t\right)^T \tag{13.65}$$

$$= \begin{bmatrix} 0.5670 & -0.00026 & -0.0266 & 0.000022 & 0.000237 \\ 0.00026 & 0.0285 & -8.2\cdot10^{-5} & -0.000387 & -6.88\cdot10^{-7} \\ -0.0266 & -8.2\cdot10^{-5} & 0.00221 & -4.07\cdot10^{-7} & -2.32\cdot10^{-5} \\ 0.000022 & -0.000387 & -4.07\cdot10^{-7} & 6.29\cdot10^{-6} & -4.68\cdot10^{-9} \\ 0.000237 & -6.88\cdot10^{-7} & -2.32\cdot10^{-5} & -4.68\cdot10^{-9} & 2.645\cdot10^{-7} \end{bmatrix} \tag{13.66}$$

13.8 Expansion of the equations for a linear fit (polynomial of order=1)

Fitting to a line ($y_n = a_0 + a_1 \cdot t_n$) is so common it is worth building a subroutine to compute that form directly. If we use the form of Eqn. 13.36. We will assume that the weighting matrix, $W(n,n)$ is a diagonal matrix with values along the diagonal of w_n. As an example, we will expand the matrices in Eqn. 13.36 assuming 3 equations with 3 unknowns (a_0, a_1).

$$X_{j,n}^T \cdot W_{n,n} \cdot y_n = \begin{bmatrix} 1 & 1 & 1 & \dots \\ t_1 & t_2 & t_3 & \dots \end{bmatrix} \begin{bmatrix} w_1 & 0 & 0 & \dots \\ 0 & w_2 & 0 & \dots \\ 0 & 0 & w_3 & \dots \\ 0 & 0 & 0 & \dots \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \end{bmatrix} \tag{13.67}$$

$$= \begin{bmatrix} 1 & 1 & 1 & \dots \\ t_1 & t_2 & t_3 & \dots \end{bmatrix} \begin{bmatrix} w_1 \cdot y_1 \\ w_2 \cdot y_2 \\ w_3 \cdot y_3 \\ \dots \end{bmatrix} \tag{13.68}$$

$$= \begin{bmatrix} \sum_{n=1}^{N} w_n \cdot y_n \\ \sum_{n=1}^{N} w_n \cdot t_n \cdot y_n \end{bmatrix} \tag{13.69}$$

$$\left[X^T_{j,n} \cdot W_{n,n} \cdot X_{n,j}\right] = \begin{bmatrix} 1 & 1 & 1 & \dots \\ t_1 & t_2 & t_3 & \dots \end{bmatrix} \begin{bmatrix} w_1 & 0 & 0 & \dots \\ 0 & w_2 & 0 & \dots \\ 0 & 0 & w_3 & \dots \\ 0 & 0 & 0 & \dots \end{bmatrix} \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ 1 & t_3 \\ \dots & \dots \end{bmatrix} \tag{13.70}$$

$$= \begin{bmatrix} 1 & 1 & 1 & \dots \\ t_1 & t_2 & t_3 & \dots \end{bmatrix} \begin{bmatrix} w_1 & w_1 \cdot t_1 \\ w_2 & w_2 \cdot t_2 \\ w_3 & w_3 \cdot t_3 \\ \dots & \dots \end{bmatrix} \tag{13.71}$$

$$= \begin{bmatrix} \sum_{n=1}^{N} w_n & \sum_{n=1}^{N} w_n \cdot t_n \\ \sum_{n=1}^{N} w_n \cdot t_n & \sum_{n=1}^{N} w_n \cdot t_n^2 \end{bmatrix} \tag{13.72}$$

The solution of the inverse of a 2^{nd} order matrix is given by (you can check this by multiplication).

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}^{-1} = \begin{bmatrix} \frac{a_{22}}{\Delta} & \frac{-a_{12}}{\Delta} \\ \frac{-a_{21}}{\Delta} & \frac{a_{11}}{\Delta} \end{bmatrix} \tag{13.73}$$

where,

$$\Delta \equiv a_{11}a_{22} - a_{12}a_{21} \tag{13.74}$$

Therefore, for our least squares fit the value of Δ becomes

$$\Delta = \sum_{n=1}^{N} w_n \cdot \sum_{n=1}^{N} w_n \cdot t_n^2 \quad - \quad \left(\sum_{n=1}^{N} w_n \cdot t_n\right)^2 \tag{13.75}$$

and the matrix inversion can be written as

$$\left[X^T_{j,n} \cdot W_{n,n} \cdot X_{n,j}\right]^{-1} = \begin{bmatrix} \frac{\sum_{n=1}^{N} w_n \cdot t_n^2}{\Delta} & \frac{-\sum_{n=1}^{N} w_n \cdot t_n}{\Delta} \\ \frac{-\sum_{n=1}^{N} w_n \cdot t_n}{\Delta} & \frac{\sum_{n=1}^{N} w_n}{\Delta} \end{bmatrix} \tag{13.76}$$

and the parameters are given by multiplying Eqn. 13.69 by Eqn. 13.76

and the error covariance of the parameters is given by Eqn. 13.76

$$\left(\delta a \cdot \delta a^T\right)_{j,j} = \chi^2 \cdot \left[X^T_{j,n} \cdot W_{j,j} \cdot X_{n,j}\right]^{-1} = \begin{bmatrix} \frac{\sum_{n=1}^{N} w_n \cdot t_n^2}{\Delta} & \frac{-\sum_{n=1}^{N} w_n \cdot t_n}{\Delta} \\ \frac{-\sum_{n=1}^{N} w_n \cdot t_n}{\Delta} & \frac{\sum_{n=1}^{N} w_n}{\Delta} \end{bmatrix} \tag{13.77}$$

For a linear relationship a useful test is the *linear corrrelation coefficient*. Since the relationship is linear we can easily solve for $t_n = a'_0 + a'_1 \cdot y)n$ (see Bevington (1969), p.120-121). We can then equate the two solutions as follows

$$y = -\frac{a'_0}{a'_1} + \frac{1}{a'_1} \cdot t_n = a_0 + a_1 \cdot t_n \tag{13.78}$$

The linear correlation is given by

$$r = \sqrt{a_1' \cdot a_1} \tag{13.79}$$

and the values range from $-1 \leq r \leq 1$. The sign of r is the same as the sign of a_1 and a_1' and is irrelevant. Values of the absolute value of r should approach 1 for a quality fit.

$$\begin{aligned}
\text{sumW} &= \sum_{n=1}^{N} w_n & (13.80)\\
\text{sumX} &= \sum_{n=1}^{N} w_n \cdot t_n & (13.81)\\
\text{sumXX} &= \sum_{n=1}^{N} w_n \cdot t_n^2 & (13.82)\\
\text{sumY} &= \sum_{n=1}^{N} w_n \cdot y_n & (13.83)\\
\text{sumYY} &= \sum_{n=1}^{N} w_n \cdot y_n^2 & (13.84)\\
\text{sumXY} &= \sum_{n=1}^{N} w_n \cdot t_n \cdot y_n & (13.85)\\
\Delta = \text{DEL} &= \text{sumW} * \text{sumXX} - \text{sumX} * \text{sumX} & (13.86)
\end{aligned}$$

$$\begin{aligned}
a_0 &= \frac{\sum_{n=1}^{N} w_n \cdot y_n \cdot \sum_{n=1}^{N} w_n \cdot t_n^2 - \sum_{n=1}^{N} w_n \cdot t_n \cdot y_n \cdot \sum_{n=1}^{N} w_n \cdot t_n}{\Delta} & (13.87)\\
&= \frac{\text{sumY} * \text{sumXX} - \text{sumXY} * \text{sumX}}{\text{DEL}} & (13.88)
\end{aligned}$$

$$\begin{aligned}
\delta a_0 &= \sqrt{\frac{\chi^2 \cdot \sum_{n=1}^{N} w_n \cdot t_n^2}{\Delta}} & (13.89)\\
&= \sqrt{\frac{\chi^2 \cdot \text{sumXX}}{\text{DEL}}} & (13.90)
\end{aligned}$$

$$\begin{aligned}
a_1 &= \frac{\sum_{n=1}^{N} w_n \cdot t_n \cdot y_n \cdot \sum_{n=1}^{N} w_n - \sum_{n=1}^{N} w_n \cdot y_n \cdot \sum_{n=1}^{N} w_n \cdot t_n}{\Delta} & (13.91)\\
&= \frac{\text{sumW} * \text{sumXY} - \text{sumY} * \text{sumX}}{\text{DEL}} & (13.92)
\end{aligned}$$

$$\delta a_1 = \sqrt{\frac{\chi^2 \cdot \sum_{n=1}^{N} w_n}{\Delta}} \tag{13.93}$$

$$= \sqrt{\frac{\chi^2 \cdot \mathrm{sumW}}{\mathrm{DEL}}} \tag{13.94}$$

$$\left(\delta a \cdot \delta a^T\right)_{j,j} = \chi^2 \cdot \left[X^T_{j,n} \cdot W_{j,j} \cdot X_{n,j}\right]^{-1} \tag{13.95}$$

$$= \chi^2 \cdot \begin{bmatrix} \frac{\mathrm{sumXX}}{\mathrm{DEL}} & \frac{-\mathrm{sumXX}}{\mathrm{DEL}} \\ \frac{-\mathrm{sumXX}}{\mathrm{DEL}} & \frac{\mathrm{sumW}}{\mathrm{DEL}} \end{bmatrix} \tag{13.96}$$

$$r = \frac{\sum_{n=1}^{N} w_n \cdot t_n \cdot y_n \cdot \sum_{n=1}^{N} w_n - \sum_{n=1}^{N} w_n \cdot y_n \cdot \sum_{n=1}^{N} w_n \cdot t_n}{\left[\sum_{n=1}^{N} w_n \cdot t_n^2 - \left(\sum_{n=1}^{N} w_n \cdot t_n\right)^2\right]^{\frac{1}{2}} \left[\sum_{n=1}^{N} w_n \cdot y_n^2 - \left(\sum_{n=1}^{N} w_n \cdot y_n\right)^2\right]^{\frac{1}{2}}} \tag{13.97}$$

The FORTRAN program (*ftp*/linear.f) shown in Section B.9 computes the required 5 weighted sums and then computes the parameters for the least squares fit and the estimated error in the parameters (to be discussed later).

In section B.14program *ftp*/lsq_linear.pro computes the least squares fit for an arbitrary Vandermonde matrix (A polynomial can be set up with lsq_vand.pro by constructing the Vandermonde matrix from the input of T(i), the independent array , N, the order of the polynmial). The output is a set of parameters, PAR(j), j=1,NP and the error in those parameters, PERR(j) which is computed from square root of the diagonal of the error covariance.

13.9 Example of a bad idea: A linear fit to a SINE function

It is illustrative to test ideas of numerical quality assurance on a known bad case. In the first example, shown in Fig. 13.4, the data to be fit is given by

$$y(n) = \sin(2\pi x(n)) \tag{13.98}$$

The fit is found to be $a_0 = -0.001 \pm 0.045$ and $a_1 = -9 \cdot 10^{-8} \pm 0.08$. Notice that the correlation coefficient, r, is zero and the χ^2 value is somewhat high ($\chi^2 = 0.502$). Both of these indicators are indicative of a poor fit. The error covariance computed by Eqn. 13.77 is

$$\left(\delta a \cdot \delta a^T\right)_{j,j} = \begin{bmatrix} 0.002 & -0.003 \\ -0.003 & 0.006 \end{bmatrix} \tag{13.99}$$

In the second example, shown in Fig. 13.5, we will fit to

$$y(n) = \sin(2\pi x(n) + \frac{\pi}{2}) \tag{13.100}$$

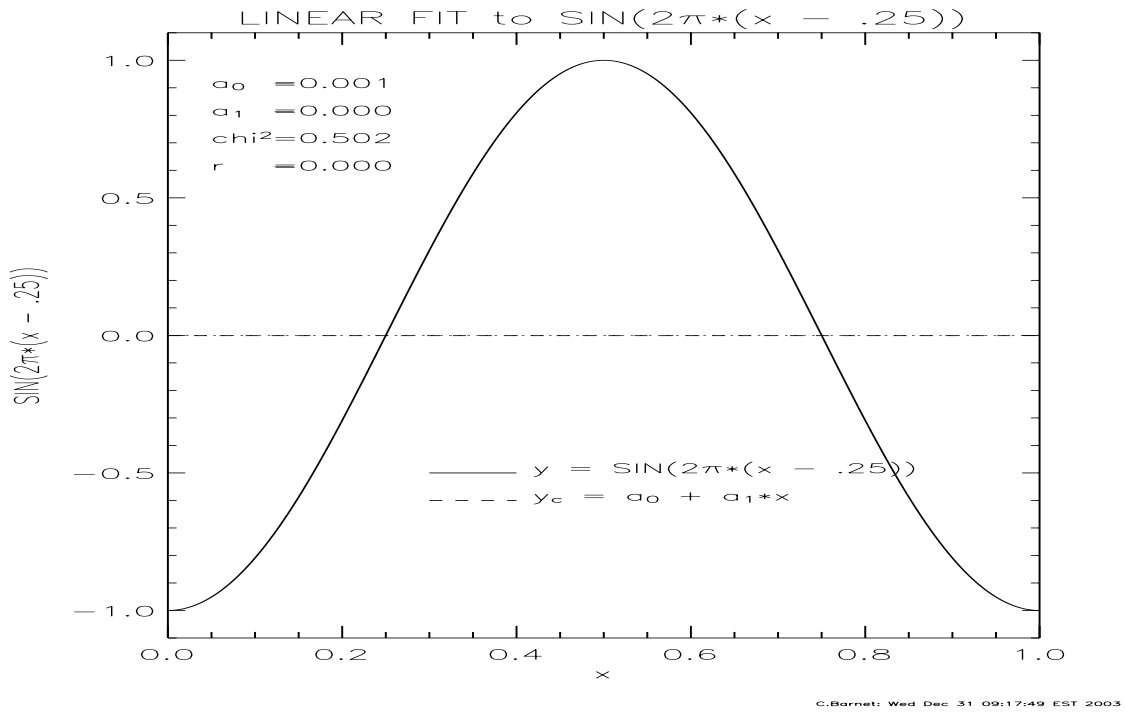


Figure 13.4: Example of a LSQ fit of a line to $\sin(2\pi x(n))$

The fit is found to be $a_0 = 0.952 \pm 0.028$ and $a_1 = -1.9 \pm 0.05$. Notice that the correlation coefficient, r, is actually quite good (0.78) and $\chi^2 = 0.197$ is better. If we hadn't plotted this we might feel that the fit was acceptable, especially if noise had been added to the data. The error covariance computed by Eqn. 13.77 is

$$\left(\delta a \cdot \delta a^T\right)_{j,j} = \begin{bmatrix} 0.00079 & -0.00118 \\ -0.00118 & 0.00236 \end{bmatrix} \tag{13.101}$$

In the third example, shown in Fig. 13.6, we will fit to

$$y(n) = \sin(3\pi x(n)) \tag{13.102}$$

The fit is found to be $a_0 = 0.212 \pm 0.043$ and $a_1 = -3.8 \cdot 10^{-8} \pm 0.074$. Here the correlation coefficient, $r = 0$, and $\chi^2 = 0.455$ indicates a poor fit.

$$\left(\delta a \cdot \delta a^T\right)_{j,j} = \begin{bmatrix} 0.001819 & -0.002727 \\ -0.002727 & 0.005455 \end{bmatrix} \tag{13.103}$$

These examples should convince you that plotting the function and the polynomial fit is always a good idea. If the assumptions of the Vandermonde matrix are met then a plot of each component of the fit should provide a reasonable solution. The error covariance computed by Eqn. 13.77 is

13.10 Problem: FIT to CO_2 data

1. Use the data provided in the file maunaloa.co2 to compute a least squares fit of the data (use all valid data) to a 2^{nd} order polynomial. Plot the data and the fit and discuss the physical interpretation of the fit as if you were publishing this data (*i.e.,* what is the meaning of a_1 and a_2 in terms of climate and how significant are your results.

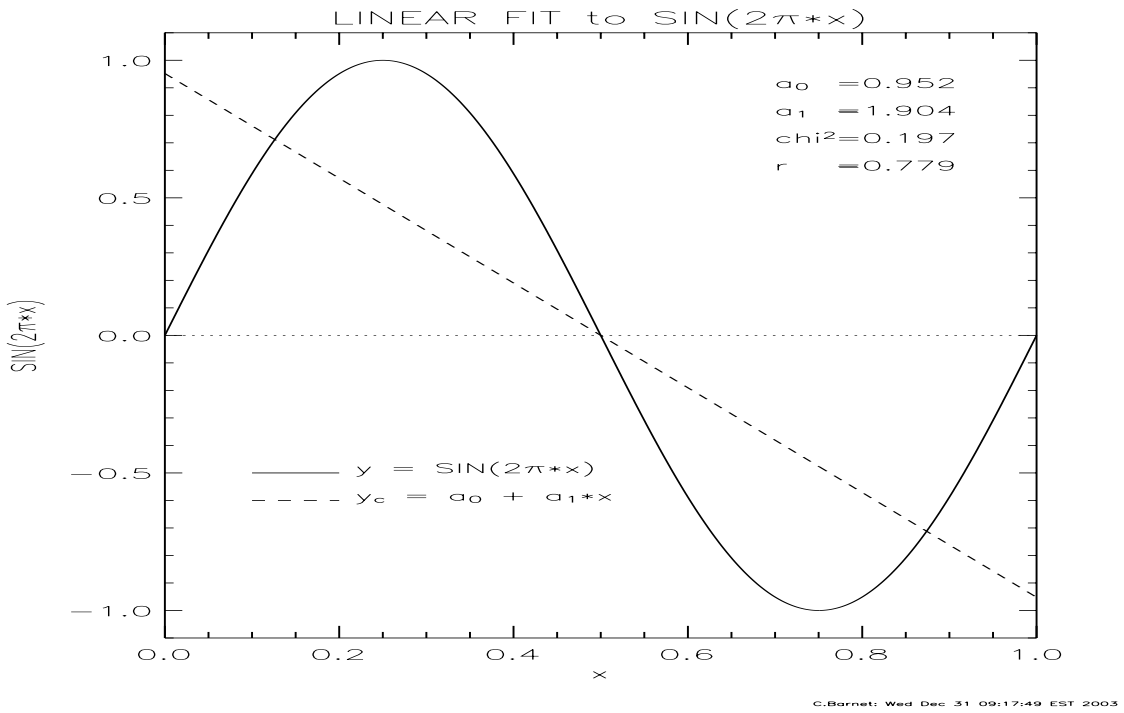


Figure 13.5: Example of a LSQ fit of a line to $\sin(2\pi x(n) + \pi/2)$

The Mauna Loa site is believed to be representative of the global atmosphere. This site was chosen as representative of global CO_2 because it is not near any sources or sinks. Charles Keeling, shown in the photograph below (taken in 2001), initiated the *in-situ* measurements in 1958.

The CO_2 in the atmosphere of the '50's was increasing by 0.78 ppmv per year. In the late 90's it was increasing at a rate of $0.78 + 0.013 \cdot 40 = 1.3$ ppmv per year. This is presumably due to the increase in global fossil fuel emissions from 2.5 Gt per year to 6 Gt per year over the same interval of time.

2. Perform a linear least squares fit of the data to a quadratic function plus a SIN() function to see if you can estimate the amplitude and phase shift of the seasonal variability of CO_2.

   Using Eqn. 13.9

$$y_n = a_0 + a_1 \cdot t_n + a_2 \cdot t_n^2 + a_3 \cdot \sin(b_1 \cdot t_n) + a_4 \cdot \cos(b_1 \cdot t_n) \tag{13.104}$$

   The error on coefficients #4 and #5 is about 0.06 and they are positively correlated with a error of $\sqrt{(\delta a' \delta a)_{3,4}} = 0.0032$ or about 5% correlated. The full error covariance is

$$(\delta a' \delta a)_{i,j} = \begin{matrix} 0.0163 & -0.00158 & 0.000032 & 0.000018 & 0.000085 \\ -0.00158 & 0.00020 & -0.00000046 & -0.0000031 & -0.0000074 \\ 0.000032 & -0.0000046 & 0.00000011 & 0.00000012 & 0.00000014 \\ 0.000018 & -0.0000031 & 0.00000010 & 0.00336 & 0.000010 \\ 0.000085 & -0.0000074 & 0.00000014 & 0.000010 & 0.00336 \end{matrix} \tag{13.105}$$

   The amplitude error (see Eqn. 13.8 and discussion relating to amplitude and phase) can be computed as follows

$$\delta A = \frac{\partial A}{\partial a_3} \cdot \delta a_3 + \frac{\partial A}{\partial a_4} \cdot \delta a_4 \tag{13.106}$$

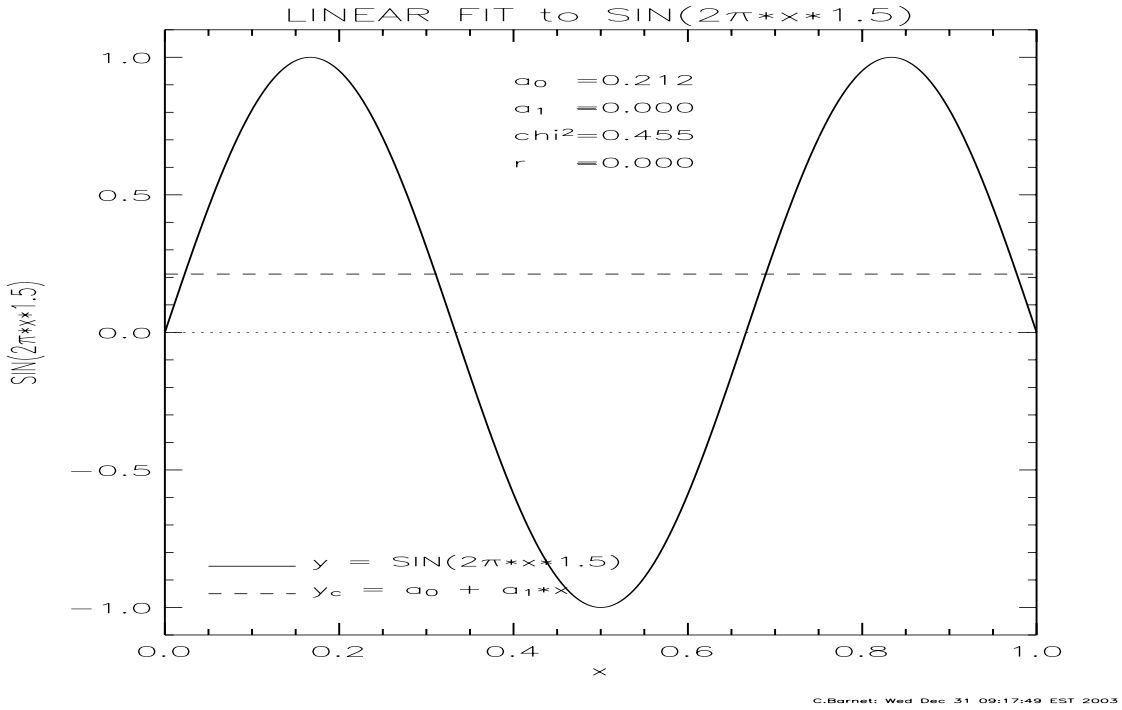


Figure 13.6: Example of a LSQ fit of a line to $\sin(3\pi x(n))$

$$= \frac{1}{2}\left(a_3^2 + a_4^2\right)^{-\frac{1}{2}} \cdot (2a_3 \cdot \delta a_3 + 2a_4 \cdot \delta a_4) \tag{13.107}$$

$$= \frac{1}{A} \cdot (a_3 \cdot \delta a_3 + a_4 \cdot \delta a_4) \tag{13.108}$$

I computed the phase error by direct substitution

$$\delta\theta = \frac{1}{2} \cdot \frac{12}{2\pi} \cdot \left[\tan^{-1}\left(\frac{a_4 - \delta a_4}{a_3 + \delta a_3}\right) + \tan^{-1}\left(\frac{a_4 + \delta a_4}{a_3 - \delta a_3}\right)\right] \tag{13.109}$$

3. What can you say about the phase of the variation of CO_2.

   The phase shift of -0.77 ± 0.02 months indicates that the mean CO_2 value occurs in late January and late July; the maximum CO_2 occurs in late April and the minimum in late October. Given that respiration of the soil is a strong function of temperature we would expect the maximum CO_2 emitted by respiration to occur in spring, after the thaw. These data support that. As leaves and tree growth occur the CO_2 is drawn out of the atmosphere; however, this is compensated, in part, by increased respiration throughout the summer. As temperatures cool the photosynthesis can dominate in late fall.

4. Why is it difficult to fit the phase shift with linear methods?

   (a) The phase is not constant with time. We really need to fit to a function that allows for phase and amplitude variation as a function of time.

   (b) The phase is not a sine wave. We could use a few terms of a Fourier series to fit the shape; however, it would still require a phase shift as a function of time for each Fourier component.

5. There is evidence that the seasonal thaw occurs ≈ a week earlier than it did 50 years ago. This should be evident in the phase of the CO_2 seasonal cycle. Design and implement a method to use linear least squares fitting to extract an estimate of the change in phase over the last 40 years.

Figure 13.7: Photo's of Charles Keeling. On the left is a shot that was released when he won the national metal of science (2002), on the right at a building dedication at the Mauna Loa Observatory in Nov. 1997 (http://www.mlo.noaa.gov/HISTORY)

Many of you fitted each year in the 40 year dataset to the 5-term function. This was unstable for the years with the missing data. Others fit the 5-term function to multiple year (5-8 year) groups, computed the phase and amplitude, and then fit those to a line. This is shown in Fig. 13.10.

The annual average from the CO_2 values for each year can be subtracted and then 10 years can be averaged together (being careful not to add bad months). The results are shown in Fig. 13.11. A least squares fit to $a_0 + a_1 \cdot \sin(2\pi \cdot m/12) + a_2 \cdot \cos(2\pi \cdot m/12)$ is shown next to each curve. I computed the errors in the same manner as the phase and amplitude error above.

In 40 years there has been a slight trend (-0.14 months), most of it occurring recently. The error bars on the measurement lead me to believe that it is barely supported by these data.

6. There is also evidence that there is a larger seasonal cycle with increasing amounts of CO_2 and/or temperature. Design and implement a method to use linear least squares fitting to extract an estimate of the trend in the amplitude over the last 40 years.

   I took the maximum peak-peak value for each year and fit that data to a line. The plot of the peak-to-peak values and the fit is shown below. Here we see a trend of $0.024 \cdot 40 = 0.97 \pm 0.24$ ppmv over 40 years which is an increase of $\approx 15\%$ over the value of 5.28 at the beginning of the period. This would tend to support an increase of photosynthesis/respiration cycles over the last 40 years.

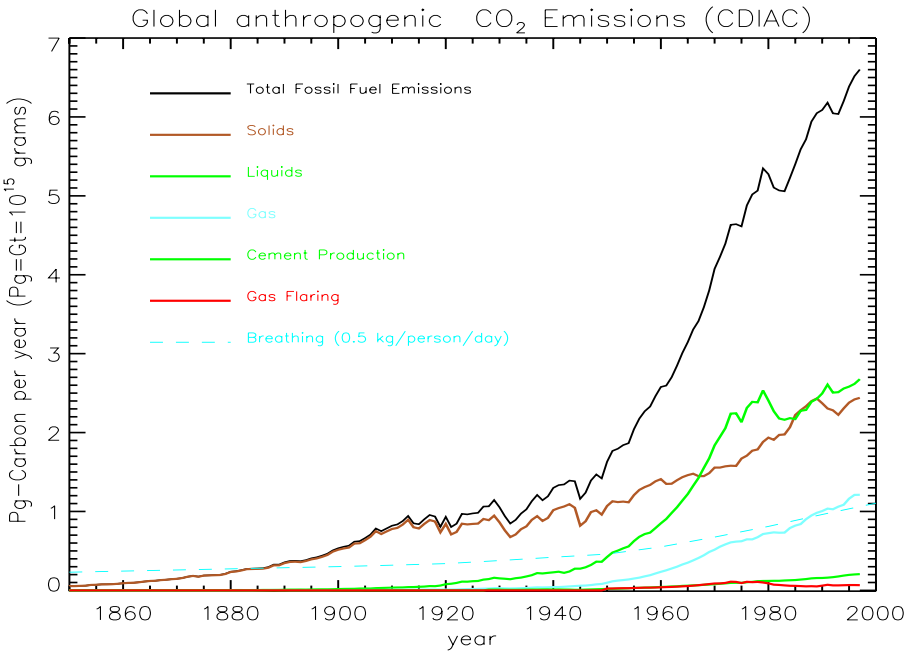


Figure 13.8: The global fossil fuel emissions from 1950 to 2000

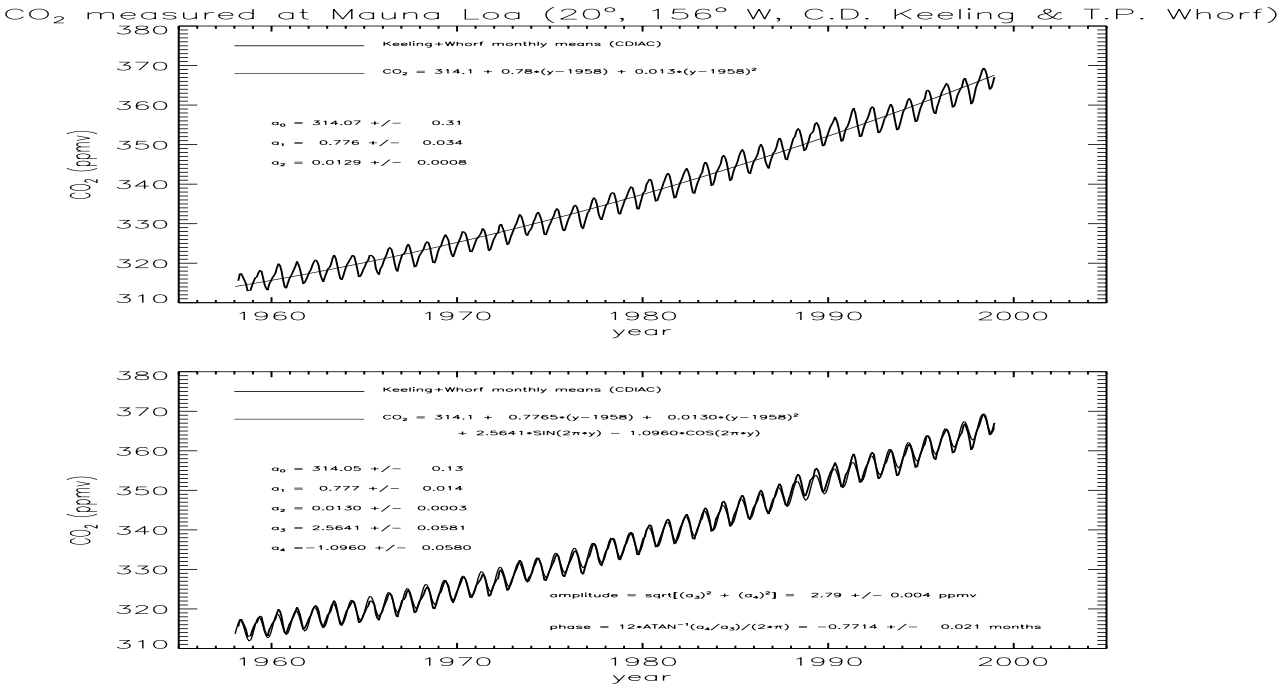


Figure 13.9: Example of a linear fit to CO_2. Top panel shows a polynomial fit and the bottom panel shows a fit to the annual cycle

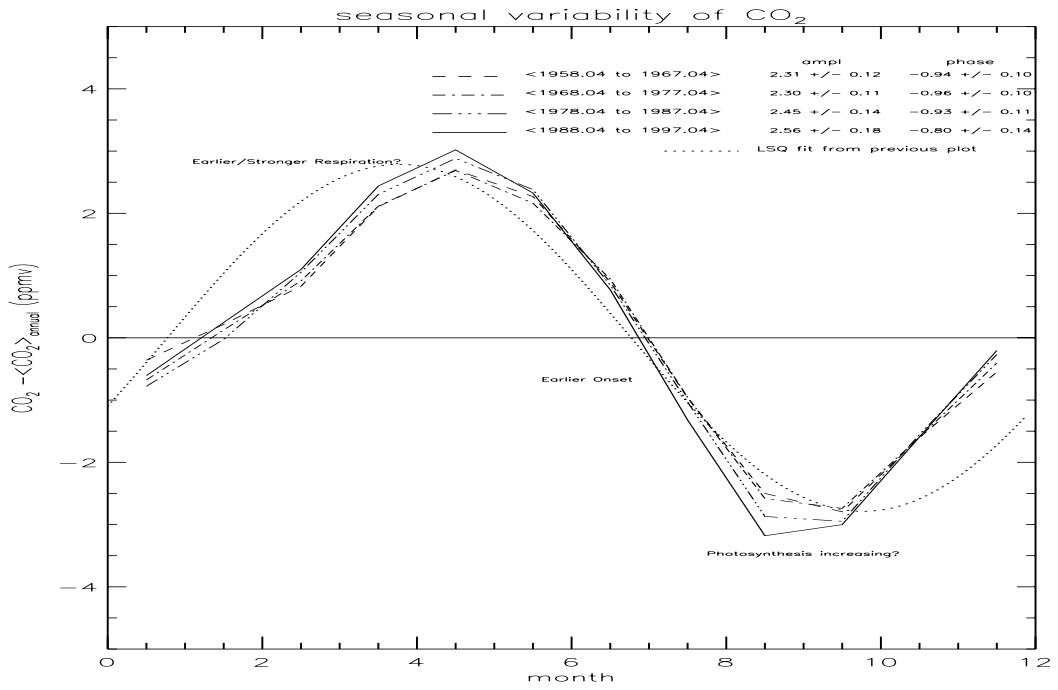


Figure 13.10: Example of using a linear fit to extract information about phase shift

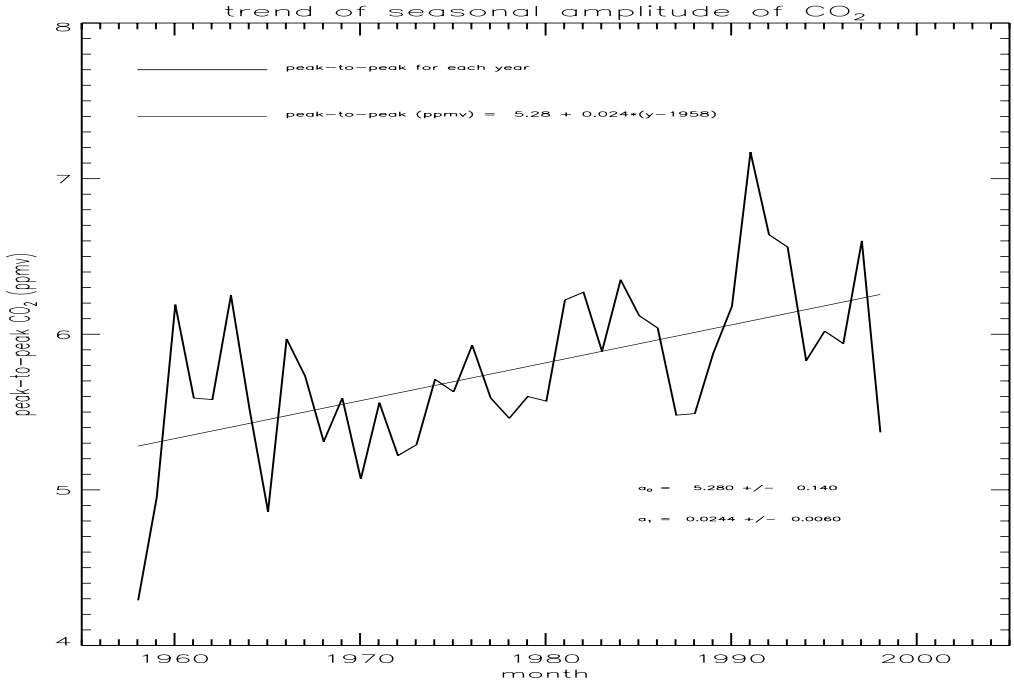


Figure 13.11: Example of using a linear fit to extract information about change in amplitude

Chapter 14

Solution of Non-Linear Equations

14.1 Finding Roots of one dimension equations

The notes for this section are taken from mostly Heath [1997] and the references listed for this chapter. Finding roots of non-linear functions or systems of equations is an ancient problem and illustrates the concept of *iteration* and *convergence* towards a solution. Iteration and convergence are concepts that are important for the solution or optimization of non-linear functions.

14.1.1 Kepler's Equation: An example of iteration

In the computation of orbits the solution of Kepler's equation (circa 1620) is typically done by iteration. The eccentric anomaly, E, is related to the mean anomaly, M, and the eccentricity of the orbit, e, by the following equation:

$$E = M + e \cdot \sin(E) \tag{14.1}$$

where, the mean anomaly is related to the period of the orbit, P, and the perihelion date, t_0, by

$$M = 2\pi \cdot (t - t_0) \tag{14.2}$$

If one makes an initial guess, $E_0 = M$ and then solves the equation iteratively, then it will converge towards a solution

$$\begin{aligned} E_1 &= M + e \cdot \sin(E_0) && (14.3)\\ E_2 &= M + e \cdot \sin(E_1) && (14.4)\\ E_3 &= M + e \cdot \sin(E_2) && (14.5)\\ \ldots &= M + e \cdot \sin(\ldots) && (14.6)\\ E_n &= M + e \cdot \sin(E_{n-1}) && (14.7) \end{aligned}$$

and the answer is *converged* when the absolute value of the difference between two *iterations*, $|E_n - E_{n-1}|$, is less that a prescribed tolerance, usually the floating point precision. For example, these are the orbital parameters of Earth and Halley's comet.

For the Earth (small eccentricity) the iteration of Kepler's equation converges very rapidly, typically with 7 iterations for a convergence threshold of 10^{-13}. For Halley's comet the convergence to 10^{13} takes more iterations, typically within 50 iterations. For very small angles $M \approx 0$; however, it can take hundreds

Table 14.1: Orbital elements for Earth and Halley's comet

	Earth	Halley's Comet
e	0.016717	0.9672759
P	365.259641	27762.429
t_0	2444242.31778	2446470.94394
	(Jan. 3, 1980)	(Feb. 9, 1986)

of iterations. Therefore, one may want to have a test for convergence and also a test for a maximum number of iterations.

Number of Iterations

M	tol=10^{-7}	tol=10^{-13}
0.00001	139	554
0.0001	208	623
0.001	274	684
0.01	200	422
0.1	44	81

Many equations can be written in terms of a converging system which meets the criteria that $f'(x) < 1$. When $f'(x)$ is near unity, however, the convergence can be quite slow.

14.1.2 Bi-section method

The bisection method searches for the zeroes of a non-linear function by searching for a change in sign. The step size, h, is chosen and the function is searched over the interval $[a, b]$. When a zero crossing is detected, then the algorithm decreases the size of h. For example, to solve

$$f(x) = x^2 - 4 \cdot \sin(x) = 0 \tag{14.8}$$

over the interval of $[a, b]$ by the bisection method we can write

```
c     by bisection method
c     -------------------
      a = 1.0
      b = 3.0
      tol = 0.1E-6

      fx1 = fuser(a)
      iter = 0
      print 305
 300    iter = iter + 1
        m = a + 0.5d00*(b-a)  ! midpoint
        fx2 = fuser(m)
        print 310, iter, a, fx1, b, m, fx2
        if(fx1*fx2.ge.0.0d00) then
          a = m
          fx1 = fx2
        else
          b = m
        endif
      if(DABS(b-a).gt.tol) goto 300
```

The bisection method only makes use of the function's sign. It is very slow to converge. On each successive iteration the length of the interval containing the solution is reduced by half, therefore, the error is reduced by half. We gain 1 bit of accuracy in the solution for each iteration.

$f(x) = x^2 - 4 \cdot \sin(x)$

i	a	$f(a)$	b	m	$f(m)$
1	1.000000	-2.365884	3.000000	2.000000	.362810
2	1.000000	-2.365884	2.000000	1.500000	-1.739980
3	1.500000	-1.739980	2.000000	1.750000	-.873444
4	1.750000	-.873444	2.000000	1.875000	-.300718
5	1.875000	-.300718	2.000000	1.937500	.019849
6	1.875000	-.300718	1.937500	1.906250	-.143255
7	1.906250	-.143255	1.937500	1.921875	-.062406
	...	...	...	...	...
17	1.933746	-.000039	1.933777	1.933762	.000041
18	1.933746	-.000039	1.933762	1.933754	.000001
19	1.933746	-.000039	1.933754	1.933750	-.000019
20	1.933750	-.000019	1.933754	1.933752	-.000009
21	1.933752	-.000009	1.933754	1.933753	-.000004
22	1.933753	-.000004	1.933754	1.933753	-.000001
23	1.933753	-.000001	1.933754	1.933754	.000000

Another interesting example is the problem originally solved by Leonardo of Pisa in the year 1225. Nobody knows what method he used to solve this to produce the result 1.368808107; however, we can use the bisection method (with a tolerance of 0.1D-10) to find the result

$f(x) = x^3 + 2 \cdot x^2 + 10 \cdot x - 20$

i	a	$f(a)$	b	m	$f(m)$
1	1.000000000	-7.000000000	3.000000000	2.000000000	16.000000000
2	1.000000000	-7.000000000	2.000000000	1.500000000	2.875000000
3	1.000000000	-7.000000000	1.500000000	1.250000000	-2.421875000
4	1.250000000	-2.421875000	1.500000000	1.375000000	.130859375
5	1.250000000	-2.421875000	1.375000000	1.312500000	-1.168701172
6	1.312500000	-1.168701172	1.375000000	1.343750000	-.524810791
7	1.343750000	-.524810791	1.375000000	1.359375000	-.198459625
8	1.359375000	-.198459625	1.375000000	1.367187500	-.034172535
	...	...	...	...	...
24	1.368808031	-.000001619	1.368808270	1.368808150	.000000896
25	1.368808031	-.000001619	1.368808150	1.368808091	-.000000361
26	1.368808091	-.000000361	1.368808150	1.368808120	.000000267
27	1.368808091	-.000000361	1.368808120	1.368808106	-.000000047
28	1.368808106	-.000000047	1.368808120	1.368808113	.000000110
29	1.368808106	-.000000047	1.368808113	1.368808109	.000000031
30	1.368808106	-.000000047	1.368808109	1.368808107	-.000000008
31	1.368808107	-.000000008	1.368808109	1.368808108	.000000012
32	1.368808107	-.000000008	1.368808108	1.368808108	.000000002
33	1.368808107	-.000000008	1.368808108	1.368808108	-.000000003
34	1.368808108	-.000000003	1.368808108	1.368808108	.000000000
35	1.368808108	.000000000	1.368808108	1.368808108	.000000001
36	1.368808108	.000000000	1.368808108	1.368808108	.000000000

14.1.3 Newton-Raphson iteration

We can use the truncated Taylor series of a non-linear function, $f(x)$, to approximate the function about a point x_k.

$$f(x+h) \simeq f(x) + f'(x) \cdot h \tag{14.9}$$

and find where the solution will cross zero.

$$0 = f(x) + f'(x) \cdot h \tag{14.10}$$

The distance to the next zero, $f(x+h) = 0$, can be approximated by the slope of the line, as follows

$$h = \frac{-f(x)}{f'(x)} \tag{14.11}$$

We can solve for the zero of the function by iteratively seeking the zero.

$$x_{k+1} = x_k - \frac{f(x)}{f'(x)\Big|_{x_k}} \tag{14.12}$$

In a sense we have transformed a non-linear equation into a linear equation and then iteratively solved the equation. In our example, the equation

$$f(x) = x^2 - 4 \cdot \sin(x) = 0 \tag{14.13}$$

and it's derivative

$$f'(x) = 2 \cdot x - 4 \cdot \cos(x) \tag{14.14}$$

are known. Therefore, the iteration equation looks like

$$x_i = x_{i-1} \quad + \quad \frac{x_{i-1}^2 - 4 \cdot \sin(x_{i-1})}{2 \cdot x_{i-1} - 4 \cdot \cos(x_{i-1})} \tag{14.15}$$

The error at iteration i can be estimated via Taylor series and is found to be

$$e_i \simeq \frac{f''(x)}{2 \cdot f'(x)} \cdot e_{i-1}^2 \tag{14.16}$$

for equation with only one root. This is known as *quadratic convergence*, since the error is roughly proportional to the sqauare of the previous iterations error. The number of correct digits almost doubles with each iteration. An example of a simple FORTRAN code for Newton's method is

```
c     by Newton's method with analytic derivatives
c     -------------------------------------------
      x = 3.0          ! initial guess for starting point

      iter = 0
      print 405
 400  iter = iter + 1
      fx = fuser(x)
      fpx = dfuser(x)  ! analytic derivative
      h = fx/fpx
      print 410, iter, x, fx, fpx, -h
      x = x - h
      if(ABS(h).gt.tol) goto 400
```

and for our two examples, we can see that the rate of convergence is very rapid.

$f(x) = x^2 - 4 \cdot \sin(x)$

i	x	$f(x)$	$f'(x)$	$-f(x)/f'(x)$
1	3.000000000	8.435519968	9.959969986	-0.846942308
2	2.153057692	1.294772505	6.505771710	-0.199019050
3	1.954038642	0.108438553	5.403795436	-0.020067109
4	1.933971533	0.001151632	5.288919536	-0.000217744
5	1.933753789	0.000000136	5.287669848	-0.000000026
6	1.933753763	0.000000000	5.287669700	0.000000000

$f(x) = x^3 + 2 \cdot x^2 + 10 \cdot x - 20$

i	x	$f(x)$	$f'(x)$	$-f(x)/f'(x)$
1	3.000000000	55.000000000	49.000000000	-1.122448980
2	1.877551020	12.444644663	28.085797584	-.443093867
3	1.434457153	1.411541795	21.910830587	-.064422103
4	1.370035050	.025892934	21.111128313	-.001226506
5	1.368808543	.000009190	21.096144659	-.000000436
6	1.368808108	.000000000	21.096139339	.000000000

Press *et. al* (1986, pg. 258) discuss a method of Newton-Raphson iteration for finding local minima combined with the bisection method to ensure global minimization. This combined method is very powerful for finding all roots of an equation.

14.1.4 Determining Square Roots w/ Newton-Raphson Iteration

In implementing a floating point package it is efficient to compute the square root operator ($y = f(x) = \sqrt{x}$) without using logarithms. The square root can be found most simply by iteration of the Newton-Raphson approximation. We illustrated this example in Chapter 1. In Section 1.4.1, Eqn. 1.6 we determined that the Newton-Raphson approximation for $f(y) = y^2 - x$ can be used an initial value, y_0, to compute an approximation for the derivative about the initial value and a perturbaton.

In Eqn. 1.4 we showed that our equation has has the analytic derivative equal to

$$f'(y) = 2y \tag{14.17}$$

and we can write the derivative so that $f(y)$ goes to zero. In general, for any iteration n this can be written

$$f'(y) \simeq \frac{f(y_n) - f(y_{n+1})}{y_n - y_{n+1}} = \frac{f(y_n) - 0}{y_n - y_{n+1}} = \frac{f(y_n)}{y_n - y_{n+1}} \tag{14.18}$$

Equating Eqn. 14.17 and Eqn. 14.18 and solving for y_{n+1} will yield

$$y_{n+1} \simeq y_n - \frac{f(y_n)}{2y_n} = \frac{1}{2} \cdot \left(y_n + \frac{x}{y_n} \right) \tag{14.19}$$

The ability to perform the square-root efficiently depends on having a reasonable first guess, y_0. In the example in Section 1.4.1, we began with $x = 16$ and $y_0 = 1$, and showed that the answer converged quickly after a more reasonable estimate was found. Here is the table from that section.

y_n	$f(y_n)$	y_{n+1}	$f(y_{n+1})$
1.0	-15.0	$\frac{1}{2}(1 + 16/1) = 8.5$	56.25
8.5	56.25	$\frac{1}{2}(8.5 + 16/8.5) = 5.19$	10.95
5.19	10.95	$\frac{1}{2}(5.19 + 16/8.5) = 4.14$	1.11
4.14	1.11	$\frac{1}{2}(4.14 + 16/4.14) = 4.002$	0.02
4.002	0.02	$\frac{1}{2}(4.002 + 16/4.002) = 4.0000005$	$0.5 \cdot 10^{-6}$

Notice that the method improves the estimate by a couple of decimal places in each iteration if the estimate is close to the correct answer. This is an example of quadratic convergence, that is, the error is approximately equal to the square of the previous iterations error.

In a robust algorithm, the argument, x, must be evaluated for zero and negative values. A zero should be trapped for execution time reasons. An excellent first guess can be obtained for the value of x by separating x into an binary exponent, k, and mantissa, m. For example, with IEEE floating point the mantissa is guaranteed to be $0 \leq m \leq 1$ and the exponent is given as a power of 2. If the exponent is an odd number then the matissa can be shifted (multiplied or divided by 2) to make the exponent even, so that $x = 2^i \cdot m = 4^{\frac{k}{2}} \cdot m$, $0 \leq m \leq 2$. A initial guess for the square root is $y^0 = 2^{\frac{k}{2}} \cdot (0.223607 + 0.894427 \cdot m)$. The coefficients are derived from a simple linear fit to the square root function and only needs to be approximate.

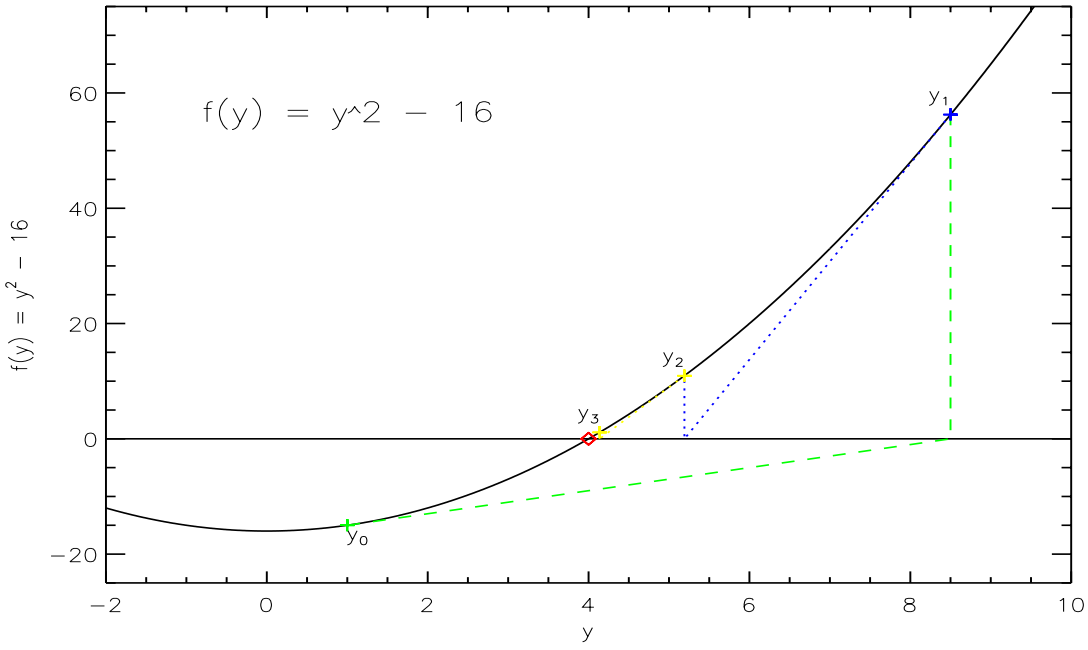


Figure 14.1: Illustration of the Newtonian iteration scheme applied to the square root function

Four iterations of this argument will yield the square root within 10 decimal digits. Since this is a small number of iterations, it is faster to always do four iterations than to build a convergence test. Note that the absolute value should be taken in case the interations cross zero (can happen for very small values of x), since the square-root should always be a positive value. See *ftp*/primfunc.f for an example of a FORTRAN implementation. In realistic applications, this function would be implemented in the native assemble code to take advantage of ALU operations such as shift and boolean operators.

14.1.5 Secant Method

This method is the same as Newton's method (Eq. 14.12), but does not require an analytic derivative. Instead, the derivative is evaluated by a finite difference approximation on the successive iterations.

$$x_i = x_{i-1} - f(x_{i-1}) \cdot \frac{\Delta x}{\Delta f(x)} \tag{14.20}$$

$$x_i = x_{i-1} - f(x_{i-1}) \cdot \frac{x_{i-1} - x_{i-2}}{f(x_{i-1}) - f(x_{i-2})} \tag{14.21}$$

For example, to solve our examples we begin by specifying a starting point, x1 = x_{i-2}, and a point to compute the first finite difference, x2 = x_{i-1}. A simple FORTRAN program is used

```
c     by secant method
c     ----------------
      x1 = 1.0d00  ! lower bound for initial derivative
      x2 = 3.0d00  ! upper value for initial derivative

      fx1 = fuser(x1)
      iter = 0
      print 505
      print 510, iter, x1, fx1
 500  iter = iter + 1
      fx2 = fuser(x2)
      h = fx2*(x2-x1)/(fx2-fx1)
      print 510, iter, x2, fx2, -h
      x1 = x2
      fx1 = fx2
      x2 = x2 - h
      if(ABS(h).gt.tol) goto 500
```

and for our two examples, we can see that the rate of convergence is still quite rapid.

$f(x) = x^2 - 4 \cdot \sin(x)$

iter	x	$f(x)$	$-f(x) \cdot \Delta x / \Delta f(x)$
0	1.000000000	-2.365883939	
1	3.000000000	8.435519968	-1.561930290
2	1.438069710	-1.896774492	0.286734911
3	1.724804621	-0.977705597	0.305028632
4	2.029833253	0.534304488	-0.107789074
5	1.922044179	-0.061522557	0.011129840
6	1.933174019	-0.003064531	0.000583457
7	1.933757476	0.000019632	-0.000003714
8	1.933753762	-0.000000006	0.000000001
9	1.933753763	0.000000000	0.000000000

$f(x) = x^3 + 2 \cdot x^2 + 10 \cdot x - 20$

iter	x	$f(x)$	$-f(x) \cdot \Delta x / \Delta f(x)$
0	1.000000000	-7.000000000	
1	3.000000000	55.000000000	-1.774193548
2	1.225806452	-2.894834010	.088712506
3	1.314518958	-1.127453942	.056591826
4	1.371110784	.048609961	-.002339096
5	1.368771688	-.000768318	.000036396
6	1.368808084	-.000000512	.000000024
7	1.368808108	.000000000	.000000000

14.2 Optimization

Optimization is finding minima or maxima in multiple dimensions. Local extrema are dimples on the multi-dimensional surface while global extrema are the extreme minima or maxima over the entire extent of the

surface. The function can be highly non-linear and is not necessarily expressible analytically. Optimization is related to finding roots because the extrema of a smooth function is found by solving for roots of it's derivative.

14.2.1 Optimization using steepest descent method

For a function of multiple dimensions we are finding the extrema of a multi-dimensional surface and the minima is found by simultaneously solving for roots of the partial derivatives. The method of steepest decent uses the local gradient to search for a new value of an extrema

$$\vec{x}^i = \vec{x}^{i-1} - \alpha^i \cdot \nabla f(\vec{x}^{i-1}) \tag{14.22}$$

where α^i is a *line search* parameter that determines how far to go in the given direction. One should continue along the gradient until a minimum along that path is reached. Therefore, a reasonable value of α^i is the value at which the value of the function reaches a minimum, which is a one-dimensional minimization problem:

$$min\left[f(\vec{x}_{i-1} - \alpha^i \cdot \nabla f(\vec{x}_{i-1})\right] \tag{14.23}$$

In Heath (1997) page 188 there is an example of this method for a 2-dimensional optimization of

$$f(\vec{x}) = 0.5 \cdot x_1^2 + 2.5 \cdot x_2^2 = [\,0.5 \quad 2.5\,] \cdot \begin{bmatrix} x_1^2 \\ x_2^2 \end{bmatrix} = A \cdot \vec{x}^2 \tag{14.24}$$

$$\nabla f(\vec{x}) = \hat{i}x_1 + \hat{j}5x_2 = 2 \cdot \sum A(i) \cdot x(i) = \begin{bmatrix} x_1 \\ 5 \cdot x_2 \end{bmatrix} \tag{14.25}$$

If we take $\vec{x}^{i=0} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$ as a starting point, then the gradient at iteration=0 is $\begin{bmatrix} 5 \\ 5 \end{bmatrix}$. The minimization of

$$f\left(\left[\vec{x}^{i-1} - \alpha^i \nabla f(\vec{x}^{i-1}))\right]\right) \tag{14.26}$$

at iteration zero

$$min\left[f\left(A \cdot \left(\vec{x}^0 - \alpha \cdot \nabla f(\vec{x}^0)\right)^2\right)\right] \tag{14.27}$$

$$\begin{aligned} &= min\left[f\left(\begin{bmatrix} 5 - 5\alpha \\ 1 - 5\alpha \end{bmatrix}\right)\right] && (14.28)\\ &= min\left[0.5 \cdot (5 - 5\alpha)^2 + 2.5 \cdot (1 - 5\alpha)^2\right] \\ &= min\left[0.5 \cdot (25 - 50\alpha + 25\alpha^2) + 2.5 \cdot (1 - 10\alpha + 25\alpha^2)\right] \\ &= min\left[15 - 50\alpha + 75\alpha^2\right] \\ &= min\left[1 - \frac{10}{3}\alpha + 5\alpha^2\right] && (14.29) \end{aligned}$$

as a function of α. Setting the derivative of Eqn. 14.29 and setting it to zero yields $\alpha_0^0 = \frac{1}{3}$ so that $\vec{x}^1$ is equal to $\begin{bmatrix} 3.333 \\ -0.667 \end{bmatrix}$. A simple iterative IDL program to search for the minima of this function by the method of steepest descent is

```
A = TRANSPOSE([0.5,2.5])  ; function coefficients
x1 = [5.0,1.0]            ; initial guess
Niter = 10
for i = 1, Niter do begin
  fx = A#x1^2      ; function
```

```
    g = 2.0*A*x1     ; gradient(fx)
;   now compute the line search parameter by finding where MIN(x1 - alpha*g)
    alpha  = 2.0*TOTAL(A*X1*g)/(2.0*TOTAL(A*g^2))
    x1 = x1 - alpha*g
  endfor
```

The contour of $f(\vec{x}^i)$ and the gradient path for each $\vec{x}^i$ is shown in Fig. 14.2.

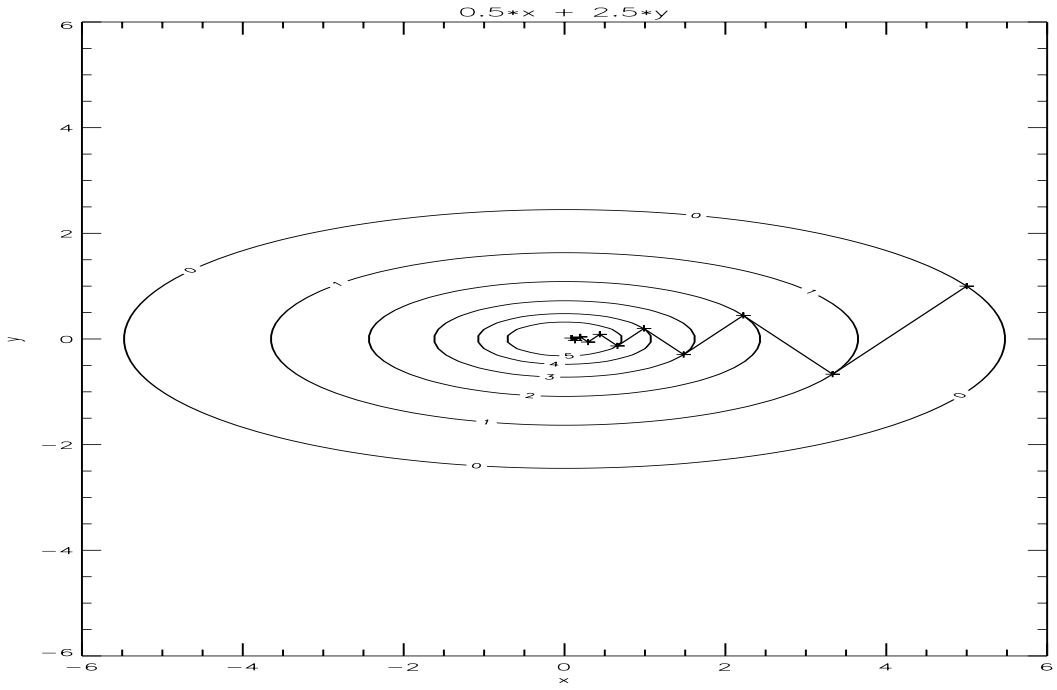


Figure 14.2: Example of the steepest descent method

x_{i-1}		$f(\vec{x}_{i-1})$	$\nabla f(x_{i-1})$	
5.000	1.000	15.00	5.000	5.000
3.333	-.0667	6.667	3.333	-3.333
2.222	0.444	2.963	2.222	2.222
1.481	-0.296	1.317	1.481	-1.481
0.988	0.198	0.585	0.988	0.988
0.658	-0.132	0.260	0.658	-0.658
0.439	0.088	0.116	0.439	0.439
0.293	-0.059	0.051	0.293	-0.293
0.195	0.039	0.023	0.195	0.195
0.130	-0.026	0.010	0.130	-0.130

In the figure we see that this method is stepping slowly towards the solution of $\vec{x} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. The gradient direction is orthogonal to the contours (lines of constant $f(\vec{x})$). The gradient direction approaches the new minima along the contour, since it is a minima along the direction of the gradient. Therefore, the search zig-zag's towards the minima).

In general, the steepest descent method is very reliable in that it can always make progress provided the gradient is nonzero. In general, the convergence rate of steepest descent is only linear, with a constant factor very close to unity.

It is important to use an initial guess that maximizes the gradient. In the example problem we could have started at a more optimal starting point. Here are two more examples of iterations.

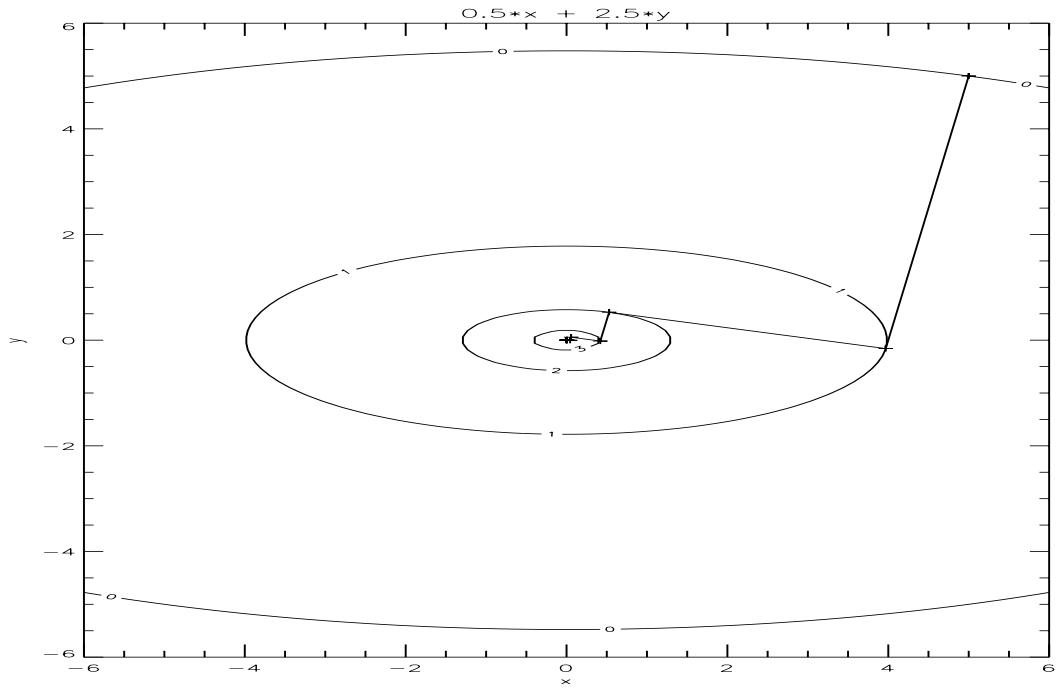


Figure 14.3: Example of the steepest descent method with a better first guess

x_{i-1}		$f(\vec{x}_{i-1})$	$\nabla f(x_{i-1})$	
5.000	5.000	75.000	5.000	25.000
3.968	-0.159	7.937	3.968	-0.794
0.529	0.529	0.840	0.529	2.646
0.420	-0.017	0.089	0.420	-0.084
0.056	0.056	0.009	0.056	0.280
0.044	-0.002	0.001	0.044	-0.009
0.006	0.006	0.000	0.006	0.030
0.005	0.000	0.000	0.005	-0.001
0.001	0.001	0.000	0.001	0.003
0.000	0.000	0.000	0.000	0.000

x_{i-1}		$f(\vec{x}_{i-1})$	$\nabla f(x_{i-1})$	
1.000	5.000	63.000	1.000	25.000
0.800	-0.006	0.320	0.800	-0.032
0.005	0.025	0.002	0.005	0.127
0.004	0.000	0.000	0.004	0.000
0.000	0.000	0.000	0.000	0.001

14.2.2 Optimization using Newton's method

Here we look for extrema in the Taylor expansion of the function and it's first derivative

$$f(x_i) = f(x_{i-1}) + f'(x_{i-1}) \cdot (x_i - x_{i-1}) \quad (14.30)$$

$$f'(x_i) = f'(x_{i-1}) + f''(x_{i-1}) \cdot (x_i - x_{i-1}) \equiv 0 \quad (14.31)$$

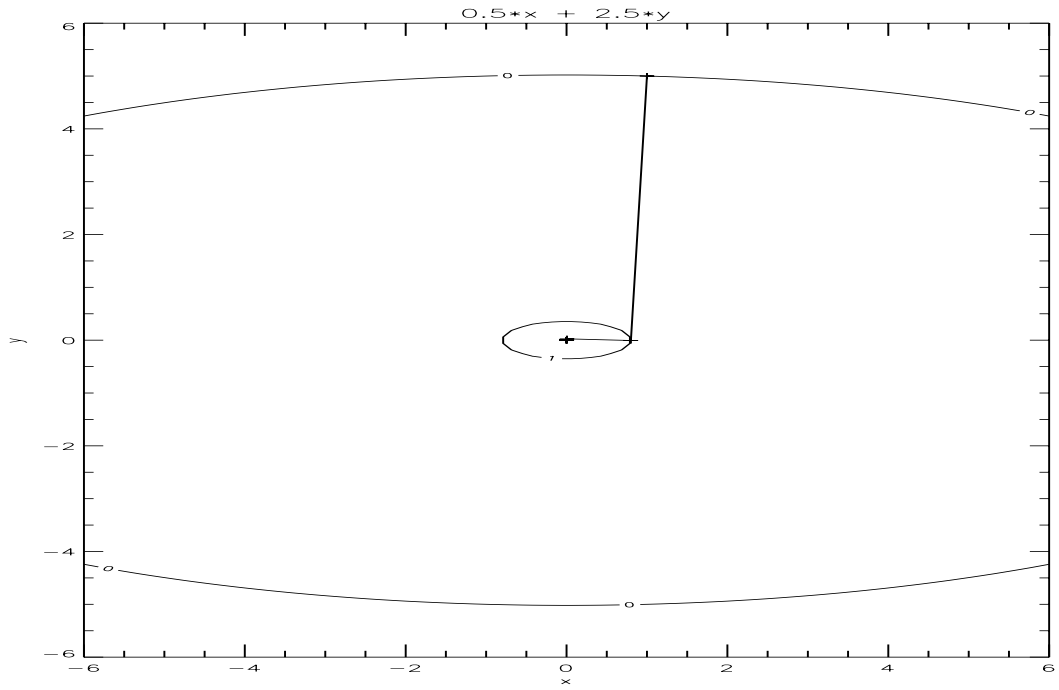


Figure 14.4: Example of the steepest descent method for an ideal first guess

Newton's method (see Eqn. 14.12) can be applied to a multi-dimensional problem to find the minima of $f(\vec{x})$ by solving for the value of $\vec{x}$ where $f'(\vec{x}) = 0$ in Eqn. 14.31.

$$\vec{x}_i = \vec{x}_{i-1} - \frac{f'(\vec{x}_{i-1})}{f''(\vec{x}_{i-1})} \tag{14.32}$$

We can use the Hessian matrix, $H_f(\vec{x})$, of second partial derivatives to find the extrema in a multi-dimensional function

$$(H_f(\vec{x}))_{i,j} = \frac{\partial^2 f(\vec{x})}{\partial x_i \cdot \partial x_j} \tag{14.33}$$

- If $H_f(\vec{x})$ is positive definite, then $\vec{x}$ is a minimum of f
- If $H_f(\vec{x})$ is negative definite, then $\vec{x}$ is a maximum of f
- If $H_f(\vec{x})$ in indefinite, then $\vec{x}$ is a saddle point of f.

$$\vec{x}_i = \vec{x}_{i-1} - (H_f(\vec{x}_{i-1}))^{-1} \cdot \nabla f(\vec{x}_{i-1}) \tag{14.34}$$

In Heath (1997) page 190 there is an example of this method for a 2-dimensional optimization of

$$f(\vec{x}) = 0.5 \cdot x_1^2 + 2.5 \cdot x_2^2 \tag{14.35}$$

whose gradient and Hessian are given by

$$\nabla f(\vec{x}) = \hat{i}x_1 + \hat{j}5x_2 = \begin{bmatrix} x_1 \\ 5x_2 \end{bmatrix} \qquad \textbf{and} \qquad H_f(\vec{x}) = \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix} \tag{14.36}$$

If we take $\vec{x}_0 = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$ as a starting point, the gradient is $\begin{bmatrix} 5 \\ 5 \end{bmatrix}$. While Heath makes the point that we never have to solve for the inverse, the inverse of the Hessian in this case is trivial

$$(H_f(\vec{x}_{i-1}))^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 0.2 \end{bmatrix} \tag{14.37}$$

and the solution becomes

$$\vec{x}_1 = \begin{bmatrix} 5 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 0.2 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 5 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \end{bmatrix} - \begin{bmatrix} 5 \\ 1 \end{bmatrix} \tag{14.38}$$

Thus, Newton's solution converges in 1 iteration. This is a result of the fact that $f(\vec{x})$ is quadratic; however, in general Newton's method will converge much more quickly than the steepest descent method.

Intuitively, unconstrained minimization is like finding the bottom of a bowl by rolling a marble down the side. If the bowl is oblong then the marble will rock back and forth along the valley. In Newton's method the metric of the space is redefined so that a quadratic bowl becomes circular, and hence the marble rolls directly to the bottom. (Taken from Numerical Recipes, Press *et al.*.)

14.2.3 Levenberg-Marquardt

Marquardt, 1962 published a technique for combining Newton's approach with that of the steepest descent method. His work referenced earlier work by Levenberg, 1943 and one can find this method attributed to one or both of these authors. We can rewrite the steepest descent method, Eqn. 14.22, as

$$-\frac{1}{\alpha_i} \cdot (\vec{x}_i - \vec{x}_{i-1}) = \nabla f(\vec{x}_{i-1}) \tag{14.39}$$

and Newton's method, Eqn. 14.34, as

$$-\left(H_f(\vec{x}_{i-1})\right) \cdot (\vec{x}_i - \vec{x}_{i-1}) = \nabla f(\vec{x}_{i-1}) \tag{14.40}$$

By combining these two equations one can achieve the speed of Newton's method with the stability of the steepest descent method.

$$\left(H_f(\vec{x}_{i-1}) + \lambda \cdot \frac{1}{\alpha_i}\right) \cdot (\vec{x}_i - \vec{x}_{i-1}) = -\nabla f(\vec{x}_{i-1}) \tag{14.41}$$

When $\lambda \to 0$ the method is the quadratic Hessian approach and then λ is large the method is diagonal and is using steepest descent.

In Press *et al.*, 1986, pg. 523 the Levenbert-Marquardt is discussed in terms of a solution to a function $f(t, \vec{a})$, where $\vec{a}$ is a vector of parameters with components a_j. The merit function is as a root-sum-square of the weighted residuals, given by

$$\chi^2(a) = \sum_{n=1}^{N} \left[\frac{y_n - f(t_{n[,m]}, \vec{a})}{\sigma_n}\right]^2 \tag{14.42}$$

where σ_n is the standard deviation or error in the measurements, y_n. $t_{n[,m]}$ are the independent measurements. The solution of J parameters a_j from the residuals. NOTE: a_j are analygous to the x used in the previous discussions. For a given iteration, i, the previous value of $\vec{x}_i = a_j^i$.

$$\nabla f \to \frac{\partial \chi^2}{\partial a_j} = -2 \sum_{n=1}^{N} \frac{\left[y_n - f(t_{n[,m]}, \vec{a})\right]}{\sigma_n} \cdot \frac{\partial f(t_{n[,m]}, \vec{a})}{\partial a_j} \qquad j = 1, 2, \ldots, J \tag{14.43}$$

$$H \to \frac{\partial^2 \chi^2}{\partial a_k \partial a_j} = 2 \sum_{n=1}^{N} \frac{1}{\sigma_n^2} \cdot \left[\frac{\partial f(t_{n[,m]}, \vec{a})}{\partial a_k} \frac{\partial f(t_{n[,m]}, \vec{a})}{\partial a_j} - \left[y_n - f(t_{n[,m]}, \vec{a})\right] \frac{\partial^2 f(t_{n[,m]}, \vec{a})}{\partial a_k \partial a_j}\right] \tag{14.44}$$

The variable λ is initially set to a small value ($\lambda \approx 0.001$) and a solution is computed. If the $\chi^2(\vec{a}+\Delta\vec{a})$ of the residuals decrease with the new solution, then λ is decreased by an order of magnitude and that solution

is kept. If the $\chi^2(\vec{a} + \Delta\vec{a})$ of the residuals increase, then λ is increased by an order of magnitude and the solution is not kept (*i.e.*, go back to the steepest descent method).

In the least squares terminology, the Jacobian, given by J in the next Section, Eqn. 14.50 is related to the gradient and partial derivatives in the Hessian. Therefore, we can think of the the Marquart-Levenberg approach as using the departure from the model (fitting function) as a regularization component. We can write the solution for the desired change in x in terms of the generalized sensitivity matrix, J

$$\Delta x = \left[S^T W S - \Delta y \frac{\partial^2 f(t_{n[,i]}, \vec{a})}{\partial a_k \partial a_j} \right]^{-1} \cdot S^T \cdot \Delta y \tag{14.45}$$

14.3 Solution via Non-linear Least Squares methods

Non-linear least squares solution of a set of equations is another kind of optimization problem. This is a generalization of Chapter 13 and follows from section 13.6 in that chapter. Given a set of N observations, $(y(n), t(n[,m]))$, we can fit them to a non-linear model *fitting equation* with J parameters, a_j, given by

$$y_n^c = f\left(t_{n[,m]}, a_j, b_k\right) \tag{14.46}$$

where, y_n^c are the computed (modeled) values corresponding to the N dependent measurements of y_n, and $t_{n[,m]}$ is the independent variable (y_n can, in general, have multiple independent variables, $t_{n,m}$). In addition, we presume that the computation may include other parameters, b_k, that affect the computation of the measurements but are not solved for.

In contrast to linear method, the expression of $f(t_{n[,m]}, a_j, b_k)$ can be any linear or non-linear function including logical expression (if statements), tabular (interpolated) forms of data or models, or analytic forms which describe the relationship between the observations, y_n, and the parameters, a_j. As before, we separate the cases into

- The case is *over-determined* when there are more observations than parameters, $N > J$
- The case is *under-determined* when there are more parameters than observations, $J > N$

The over-determined case for linear equations was discussed in the linear least squares fitting chapter (Chapter 13). Here we will consider the under-determined case and unstable forms (*i.e.,* extremely noisy, insufficient sampling, or incomplete model) of the over-determined case and will discuss the *best fit* to the data via minimization of the square of the residuals

$$\text{MIN}\left[\sum_{n=1}^{N} \left(y_n - f\left(t_{n[,m]}, a_j, b_k\right)\right)^2 \right] \tag{14.47}$$

In the 70's a program developed at Argonne Lab's was used to solve for Mössbauer spectral fitting parameters using Powell's (1966) approach. This program minimized the squares by solving for the coefficients using finite difference derivatives of Jacobians. Jacobian's are known by many names, including kernel (the K in Rodger's papers), tangent-linear model, adjoint, etc. The coefficients would represent physical parameters, such as absorber amount, temperature, Lorentzian half-width, isomer shift, magnetic and Boltzmann factors field strength, etc. Thus, a very complicated spectrum, consisting of 10's of lines could be fit with a half-dozen parameters. I have copied the program into *ftp*/bar_mosf.for. The program was built into a library and required the user to write a main program which

- Provided a subroutine (called CALFUN) which described the functional relationship between parameters
- Initialized (or read) the data arrays, t = XD(1:NY) and y= YD(1:NY) for the NY data points via a common block called /usrdat/

- Provided an initial guess for all the parameters. A plot of the data and a ruler was used to manually determine initial guesses for the Mössbauer program.
- Provide the desired accuracy for all the parameters. This could be set to a fraction of the initial guess of the parameter (i.e., a per-cent error).
- call MOSFIT() subroutine

The general methodology of all non-linear least squares routines is that we can iteratively search for a vector of parameters, a_n, which are a solution to a set of equations $f(t_{n[,m]}, a_j, b_k)$ by using the sensitivity of the non-linear function to each parameter. At iteration i we can write a Taylor expansion for the calculated estimate, $f(t_{n[,m]}, a_j^i, b_k)$, of the observations, y_n, in terms of the current estimate of the parameters we are solving for, a_j^i, and parameters that affect the computation but are not solved for, b_k.

$$y_n \simeq f(t_{n[,m]}, a_j^i, b_k) \quad + \quad \sum_{j=1}^{J} \left.\frac{\partial f(t_{n[,m]}, \vec{a}^i, \vec{b})}{\partial a_j}\right|_{\vec{a}} \cdot \Delta a_j^i \quad + \quad \sum_{k=1}^{K} \left.\frac{\partial f(t_{n[,m]}, \vec{a}^i, \vec{b})}{\partial a_j}\right|_{\vec{b}} \cdot \Delta b_k \tag{14.48}$$

The partial derivatives with respect to b_k, the parameters not solved for, become part of the error assessment of the model ($f(t_n[,m], \vec{a}^i, \vec{b})$) and the partial derivatives with respect to the parameters we wish to solve for, a_j^i are assembled into a 2-d matrix called the Jacobian, $J_{n,j}^i$,

$$\boxed{\left(y_n - f(t_{n[,m]}, \vec{a}^i, \vec{b})\right) = J_{n,j}^i \cdot \Delta a_j^i} \tag{14.49}$$

where, the Jacobian at iteration i, J^i, is defined by

$$J_{n,j}^i \left(f(t_{n[,m]}, a)\right) = \left.\frac{\partial f_n(t, \vec{a}^i, \vec{b})}{\partial a_j^i}\right|_{\vec{a}^i} \tag{14.50}$$

The Jacobian is computed using the current value of the parameters (initially specified as a first guess, a_j^0). Eqn. 14.49 assumes that the residuals are completely due to errors in the parameters. We can solve for the change in the parameters, $\Delta\vec{a}$, which will minimize the residuals at each iteration with the linear least squares method.

We assume that observations can have uncertainty that the uncertainty is both a function of channel n and that it can be correlated with neighboring channels (*e.g.*, apodized interferogram radiances). This is given by a error co-variance matrix, $N_{n,n'}$ and we will weight by it's inverse, $W_{n,n'} \equiv N_{n,n'}^{-1}$. The error co-variance matrix should also include any uncertainties in the computation of $y_n^i = f(t_{n[,m]}, \vec{a}^i, \vec{b})$ (*e.g.*, due to parameters held constant in a given step).

$$W_{n,n'} = \left[N_{n,n'} + \sum_{k=1}^{K} \left(\left.\frac{\partial f(t_{n[,m]}, \vec{a}^i, \vec{b})}{\partial a_j}\right|_{\vec{b}} \right) \left(\delta b_k \delta b_k^T\right) \left(\left.\frac{\partial f(t_{n[,m]}, \vec{a}^i, \vec{b})}{\partial a_j}\right|_{\vec{b}} \right)^T \right]^{-1} \tag{14.51}$$

where $\delta b_k \delta b_k^T$ is an estimate of the error covariance in the parameters not solved for. The weighting matrix can change as $\vec{a}^i$ is iterated due to the non-linear nature of $f(t_{n[,m]}, \vec{a}^i, \vec{b})$. Also, in some highly non-linear situations (*e.g.*, atmospheric remote sensing) the retrievals might be separated into steps, where in step #1 $\vec{b}$ is held constant and $\vec{a}$ is solved for and then in step #2 $\vec{a}$ is held constant and $\vec{b}$ is solved for.

To solve Eqn. 14.49 we first multiply both sides of Eqn. 14.49 by the weighting matrix yields

$$W_{n,n} \cdot \left(y_n - f(t_{n[,m]}, a)\right) = W_{n,n} \cdot J_{n,j} \cdot \Delta a_j \tag{14.52}$$

then we multiply both sides by the transpose of the Jacobian

$$J_{j,n}^T \cdot W_{n,n} \cdot \left(y_n - f(t_{n[,m]}, a)\right) = J_{j,n}^T \cdot W_{n,n} \cdot J_{n,j} \cdot \Delta a_j \tag{14.53}$$

then multiply both sides by the inverse of $J_{j,n}^T \cdot W_{n,n} \cdot J_{n,j}$

$$\Delta a_j = \left[J_{j,n}^T \cdot W_{n,n} \cdot J_{n,j}\right]^{-1} \cdot J_{j,n}^T \cdot W_{n,n} \cdot \left(y_n - f(t_{n[,m]}, a)\right) \tag{14.54}$$

Also note that we can compute an error estimate at each iteration

$$\delta a_j^i \simeq \chi^2 \cdot \left[J_{j,n}^T \cdot W_{n,n} \cdot J_{n,j}\right]^{-1} \tag{14.55}$$

where the diagnonal component of $W_{n,n}$ is used to compute χ^2 as follows

$$\chi^2 = \sum_{n=1}^{N} \left[W_{n,n} \cdot \left(y_n - f(t_{n[,m]}, a^i)\right)^2\right] \tag{14.56}$$

If the problem were linear then we have obtained *the best fit* to the observed data. For a non-linear system, we have approximated the minima, based on a linear extrapolation (analogous to Newton's iteration method when finding roots). For a non-linear system of equations are iterated as follows

1. Compute the observed minus calc's,$\left(y_n - f(t_{n[,m]}, a^i)\right)$, using the current value of the parameters, a_j^i
2. If the noise is a function of the parameters, then recompute $W_{n,n}$.
3. Compute the Jacobian for this iteration, $J_{n,j}^i$, using the current parameters, a_j^i from Eqn. 14.50
4. Compute the new parameters $a_j^{i+1} = a_j^i + \Delta a_j^i$, using Eqn. 14.54 for the value of Δa_j^i.

And repeat the iterations until convergence is reached. Convergence can be specified by a χ^2 test, a maximum number of iterations, or a test on the size of Δa^i. This methodology will work if the inverse of $J_{j,n}^T \cdot W_{n,n} \cdot J_{n,j}$ is well-behaved. That is to say, if we are

- over-determined (*i.e.*, $N > J$)
- and we have significant signal-to-noise (Jacobian is a signal of parameter a_j and noise is the noise in the measurements or computation.

14.3.1 damping

For under-determined systems we need to stabilize our solution for Δa^i. There are a variety of methods to accomplish this but they are all based on the idea of damping the solution (i.e., taking smaller steps in Δa). This is the "art" of non-linear least squares fitting and is the topic of many recent publications by Eyre, Hanel, Rodgers, Twomey, and others, given in the references section.

For example, in constrained linear least squares fitting we apply a constraint by a parameter γ (a Lagrangian multiplier) and we replace our equation with

$$\Delta \tilde{a}_j = \left[J_{j,n}^T \cdot W_{n,n} \cdot J_{n,j} \quad + \quad \gamma \cdot I_{j,j}\right]^{-1} J_{j,n}^T \cdot W_{n,n} \cdot \left(y_n - f(t_{n[,m]}, a)\right) \tag{14.57}$$

where $I_{j,j}$ is the identity matrix. This method can be dervied in a variety of ways and is called the *linear constrained* and *minimum information* solution. It has been utilized by Twomey, Conrath, and others. In *minimum information* the value of γ can be derived from the variance of the first guess field, a_j^0, the variance of the observations,

$$\gamma \simeq \frac{\sigma^2(y)}{\sigma^2(a)} \tag{14.58}$$

The value of gamma, γ, can also be determined empirically. When the value of γ is small the retrieval converges quickly with small residuals but the resulting parameters can be sensitive to noise and have significant undesirable variability. With large γ the solution will converge more slowly and typically will have larger residuals, but the resulting solution will be more behaved. For this reason some call *gamma* a "smoothness" constrain, since in remote sensing it typically results in smoother profiles.

Where the signal-to-noise (S/N) ratio is high, the solution will be unaffected; however, where S/N is small the corresponding component of Δa will be small and thus the solution will "stick" to the first guess values.

14.3.2 background term

Iteration of a damped solution raises a serious issue. When we constrain the solution we have chosen to NOT make a change that was determined from the residuals (obs-calc's). Therefore, we have chosen to not believe a portion of the residuals. If the damping simply stabilizes the non-linear nature of the problem (i.e., some of the residuals will vanish as we approach the correct solution) then then no harm is done. If, on the other hand, the portion of the residuals we are damping is due to instrumental noise then the residuals on the next iteration will attempt to make the same change to the parameters again. To the extent that we are noise dominated, the solution will ultimately respond to the noise. Rodgers discusses a *background* term which keeps the solution within a certain distance of the first guess, thus he does not allow successive iterations to keep responding to the noise.

Another approach is to determine the S/N and to compute the portion of the observations which are presumed to be damped due to noise and remove that component from subsequent iterations. This technique will be illustrated in the remote sensing example. At this juncture we can imagine computing the component of the residual estimated to be noise as

$$\Phi_n^i = J_{n,j}^i \cdot (\Delta a - \Delta \tilde{a}) \tag{14.59}$$

which is the component of the observations we chose to NOT believe. Then we would subtract with value on the next iteration.

$$\Delta \tilde{a}_j^{i+1} = \left[J_{j,n}^T \cdot W_{n,n} \cdot J_{n,j} \quad + \quad \gamma \cdot I_{j,j}\right]^{-1} J_{j,n}^T \cdot W_{n,n} \cdot \left(y_n - f(t_{n[,m]}, a^i) - \Phi_n^i\right) \tag{14.60}$$

On each iteration we could compute both Δa and $\Delta \tilde{a}$ and recompute Φ. In this way, we would allow for changes due to non-linearity, but continuously remove the effect due to our S/N estimate. For this method to work the noise co-variance matrix must be accurate.

14.3.3 Eigenvectors

Eigenvectors of covariance matrices can be used to

- Determine the *information content* of a set of observations (see Hanel *et al.* 1992, Eyre, 1990, Rogers, 1996, Viera and Box, 1985).
- Reducing dimensionality of regression least squared fitting (Smith and Woolf, 1976).
- stabilize a covariance matrix, called *ridge filtering* when solving an over-determined case and damping when it is an iterative under-determined case.
- transform a set of measurements to an orthogonal set of measurements
- stabilize the inverse of a low S/N LINEAR least squares fitting equation.

Smith and Woolf [1976] discuss the use of eigenvectors of radiance and temperature covariance matrices for atmospheric remote sounding. The eigenfunctions, also known as empirical orthogonal functions are derived from a set of observations, $R_{n,k}$, where n is the channel index, and k is the index for one of the K observations. The covariance of the observations is given by

$$C_{n,n'} = \frac{1}{K} \sum_{k=1,K} R_{n,k} \cdot R_{k,n'}^T \tag{14.61}$$

which is a square symmetric matrix. We can transform a square symmetric matrix, $C_{n,n'}$, into a diagonal set

$$\lambda_j = U_{j,n} \cdot C_{n,n'} \cdot U^T_{n,j} \tag{14.62}$$

where U is a matrix of eigenvectors and λ is the associated eigen-value. The matrix U is unitary and orthogonal, therefore,

$$U_{j,n} \cdot U^T_{n,j} = I_{j,j} \tag{14.63}$$

The eigenvalues typically fall off quite quickly and the number of significant eigenvalues is a measure of the degrees of freedom in the original data. We can rationalize this though as follows: if a number of observations are correlated, they will be combined into one eigenfunction.

The smaller eigenfunctions represent the noise in the measurements (since noise is uncorrelated) and removal of these can, in effect, remove the noise in the measurements. Thus for an unknown radiance case, $R_{n,u}$, the radiances can be transformed to a smaller number of *principal components*

$$P_{j,u} = \sum_{n=1}^{N} U_{j,n} \cdot R_{n,u} \tag{14.64}$$

The top $J < N$ principal components (determined by those eigenvectors associated with the highest λ_j's) can then be used in a regression least squares fit instead of the raw radiances.

Another interesting example is to reduce noise by transforming the $J < N$ most significant principle components back to radiance space

$$\hat{R}_{n,u} = \sum_{j=1}^{J} U^T_{n,j} \cdot P_{j,u} \tag{14.65}$$

If $J = N$ then $\hat{R}_{n,u} = R_{n,u}$, but if $J < N$ $\hat{R}$ will effectively have noise removed. For example, with AIRS regression retrieval we use the best 1960 of the 2378 available channels and $\approx$ 20,000 cases to determine the eigenvector matrix. We select the top 100 λ's. Therefore, we have reduced the dimensionality of the problem by a factor of 10. In linear eigenvector regression we solve for a matrix A that relates the independent variable(s), X, to the observations, R for the K known cases.

$$\Delta X_{L,k} = X_{L,k} - < X_{L,k} >= A(j) \cdot (U_{j,n} \cdot R_{n,k}) \tag{14.66}$$

and the least squares linear regression fit is given by

$$A_{L,j} = \Delta X_{L,k} \cdot \cdot (U_{j,n} \cdot R_{n,k})^T \cdot \left[U_{j,n} \cdot R_{n,k} \cdot R^T_{k,n} \cdot U^T_{n,j}\right]^{-1} \tag{14.67}$$

Therefore, the dimensionality was reduced from N to J and we only need to invert a $J \times J$ matrix instead of a $N \times N$ matrix. The inversion if more stable since we have removed the low noise component. If the S/N is low, an additional *ridge filter* can be applied to stabilize the solution.

We use the routines in numerical recipes to solve for $U_{j,n}$ and λ_j (see Press *et. al* 1986, pgs. 350-363). These routines are shown in Sections B.20, B.22, and B.3. To call these routines the calling program looks like

```
c Initially, U(n,j) is a square symmetric covariance matrix.  It is
c   replaced by the eigenvectors by these these routines.
c U(n,j) has FORTRAN memory indexing order.  The transformed space
c   is denoted by j.  The original space is denoted by n.
c maxdim is the FORTRAN dimension of Unj() and lambda() arrays
c ndim is the actual size of Unj() that is used.

      real*4 Unj(maxdim,maxdim), lambda(maxdim), e2(maxdim)
      CALL TRED2(Unj, ndim, maxdim, lambda, e2)  ! intermediate U,lambda,e2
      CALL TQLI(lambda,e2,ndim,maxdim, Unj, iret)! compute lambda(j), U(n,j)
      if(iret.ne.0) goto 4200
```

```
CALL EIGSRT (lambda, Unj, ndim, maxdim )   ! SORT lambda, U(n,j)
do j = 1,ndim
 lambda(j) = DMAX1(lambda(j),0.1D-8)  ! make sure all are positive
enddo
```

14.3.4 Recipe for Non-Linear Least Squares with Damping

1. Build a function, $f_n(t,a)$, which is a model representation of your system as a function of your independent variable $t_{n[,m]}$ and a set of unknown parameters, a_j, j =1,J, which are chosen to adequately represent this problem. This function will be used to compute the dependent measurements, y_n, as a function of these parameters.

2. Build a function to perturb your model in step # 1, $f'_{n,j}(t,a,\Delta\hat{a}_j)$. The perturbation parameters may be additive or multiplicative perturbation, $\Delta\hat{a}_j$, which is a small (*linear*) perturbation that is large enough to be numerically stable.

   This function will be used in steps # 6 and #8 and can be written as

   - an analytic function, $f'(t,a)\cdot\Delta\hat{a}_j$,
   - a centered finite difference, $f'(t,a,\Delta\hat{a}_j)$,

$$f'_{n,j}(t,a,\Delta\hat{a}_j) \simeq f_n\left(t,a+\frac{\Delta\hat{a}_j}{2}\right)-f_n\left(t,a-\frac{\Delta\hat{a}_j}{2}\right) \quad \textbf{additive} \tag{14.68}$$

$$\simeq f_n\left(t,a\left(1+\frac{\Delta\hat{a}_j}{2}\right)\right)-f_n\left(t,a\left(1-\frac{\Delta\hat{a}_j}{2}\right)\right) \quad \% \tag{14.69}$$

   - or a one-sided finite difference. A one-sided finite difference can save significant computational time since you can pass the values of the function computed in step # 5, $f_n(t,a)$, instead of re-computing them.

$$f'_{n,j}(t,a,\Delta\hat{a}_j) \simeq f_n(t,a+\Delta\hat{a}_j)-f_n(t,a) \quad \textbf{additive} \tag{14.70}$$

$$\simeq f_n(t,a\,(1+\Delta\hat{a}_j))-f_n(t,a) \quad \% \tag{14.71}$$

3. determine a reasonable first guess for the J parameters, a_j^0, an error estimate for the J parameters, δa_j^0, and the perturbation step size, $\Delta\hat{a}_j$. The superscipt zero represents the value at iteration $i=0$.

4. set the background term for iteration $i=1$ equal to zero

$$\Phi_n^1 = 0 \tag{14.72}$$

5. compute an estimate of the observations for the current state, $f_n(t,a_j^{i-1})$, using the function defined in step #1. It is useful to print the observations, y_n, the computed estimate of the observations, $f_n(t,a_j^{i-1})$, and the observed minus computed values, $y_n - f_n(t,a_j^{i-1})$, for all points $n=1,N$

6. compute the sensitivity matrix, $S^i_{n,j}$, for each observation n and each parameter, a_j, to be solved for in this step. The sensitivity matrix is a scaled dimensionless representation of the Jacobian and is computed using the function defined in step #2

$$S^i_{n,j} = J^i_{n,j}\otimes\Delta\hat{a}_j = \left.\frac{\partial f_n(t,X))}{\partial a_j}\right|_{a_j^{i-1}}\otimes\Delta\hat{a}_j = f'(t,a^{i-1},\Delta\hat{a}_j) \tag{14.73}$$

The $\otimes$ symbol indicates that it is a scalar multiplication, which does not change the dimensionality of the matrix. The parameter $\Delta\hat{a}_j$ scales each column of the Jacobian.

7. Print the sensitivity matrix and look to see which observations are determining each parameter and to check to see if the parameters are being determined by independent observations.

   - If any observation n is insensitive to all the parameters begin solved for at this step, then that observation can be removed from the computation. This can save considerable computation time.
   - The rows (j's) of the sensitivity matrix should be of similar magnitude. After reviewing this printout the perturbation size, $\Delta\hat{a}_j$, can be adjusted to *balance* the matrix and the process can be restarted.

8. compute the observed-calculated error covariance matrix. The error covariance matrix has both instrumental noise estimates, $N^i_{n,n'}(y_n)$, and *computational* error estimates, $N^i_{n,n'}(f(t_{n[,m]}, a_j^{i-1}, \vec{b}))$, due to any parameters held constant during this retrieval. If all parameters are being solved for then the computational error component is zero, otherwise we can estimate the impact of that parameter on the observations

$$E^i_{n,k}\left(f_n(t,X,\vec{b})\right) = \left.\frac{\partial f_n(t,X))}{\partial b}\right|_{a_j^{i-1}} \cdot \delta b_k \quad \simeq \quad f'_{n,k}(t,a,\delta\vec{b}) \tag{14.74}$$

and compute the error covariance in the sum squared sense. This assumes each parameter represents an uncorrelated error.

$$N^i_{n,n'} = N_{n,n'}(y_n) + N^i_{n,n'}(f_n(t,a_j^{i-1},\vec{b})) = N_{n,n'}(y_n) + \sum_{k=1}^{K} E^i_{n,k}\cdot\left(E^i\right)^T_{k,n'} \tag{14.75}$$

9. Invert the error covariance matrix

$$W^i_{n,n'} = \left(N^i_{n,n'}\right)^{-1} \tag{14.76}$$

10. Analyse the information content of the parameter co-variance matrix by computing eigenvectors and associated eigenvalue (see Press *et al.* [1996] routines TQLI, TRED2, EIGSRT)

$$\lambda^i_k \equiv \left(U^i\right)^T_{k,j}\cdot\left[\left(S^i\right)^T_{j,n}\cdot W^i_{n,n'}\cdot S^i_{n,j}\right]\cdot U^i_{j,k} \tag{14.77}$$

Eigenvalues that are small should be damped. A simple *ridge* damping can be defined as

$$\Delta\lambda^i_k = 0 \quad \textbf{for} \quad \lambda^i_k \geq \lambda_c \tag{14.78}$$

$$\Delta\lambda^i_k = \infty \quad \textbf{for} \quad \lambda^i_k < \lambda_c \tag{14.79}$$

Alternatively, the damping, $\Delta\lambda^i_j$, can be a gradual function. The fraction of the transformed functions we are going to believe is given by

$$\phi^i_k = \sqrt{\frac{\lambda^i_k}{\lambda^i_k + \Delta\lambda^i_k}} \tag{14.80}$$

Print a table of λ^i_k, $U^i_{j,k}$, $\Delta\lambda^i_k$, and ϕ^i_k. It is sometimes useful to print $S^i_{n,j}\cdot U^i_{j,k}$ which is the transformed sensitivity matrix.

Note that the value of the damping parameter, $\Delta\lambda_k^i$, is analygous to the H matrix used in many forms of regularization, such as Twomey's smoothing operators. See PHYS 741 notes for more details on these methods. Here we note that

$$\begin{aligned}
\Delta\lambda_k^i &= \left(U^i\right)_{k,j}^T \cdot H_{j,j} \cdot U_{j,k}^i && (14.81)\\
H_{j,j} &= U_{j,k}^i \cdot \Delta\lambda_k^i \cdot \left(U^i\right)_{k,j}^T && (14.82)\\
\left(S^i\right)_{j,n}^T W_{n,n}^i S_{n,j}^i + H_{j,j} &= U_{j,k}^i \left(\lambda_k^i + \Delta\lambda_k^i\right) \left(U^i\right)_{k,j}^T && (14.83)\\
\left(U^i\right)_{k,j}^T \left(\left(S^i\right)_{j,n}^T W_{n,n}^i S_{n,j}^i + H_{j,j}\right) U_{j,k}^i &= \left(U^i\right)_{k,j}^T U_{j,k}^i \left(\lambda_k^i + \Delta\lambda_k^i\right) \left(U^i\right)_{k,j}^T U_{j,k}^i && (14.84)\\
& && (14.85)
\end{aligned}$$

Note that a property of a Unitary (orthogonal) transform is that

$$\left(U^i\right)_{k,j}^T U_{j,k}^i \equiv I_{k,k} = I_{j,j} \equiv U_{j,k}^i \left(U^i\right)_{k,j}^T \quad \text{since} \quad J \equiv K \tag{14.86}$$

$$\left(U^i\right)_{k,j}^T \left(S^i\right)_{j,n}^T W_{n,n}^i S_{n,j}^i U_{j,k}^i + \left(U^i\right)_{k,j}^T \Delta\lambda_k^i U_{j,k}^i = \lambda_k^i + \Delta\lambda_k^i \tag{14.87}$$

Since $\Delta\lambda_k^i$ is a function of λ_k^i then $H_{j,j}$ is large when the measurement or model errors are large and small when the measurement or model errors are small.

11. Compute the "best" least squares fit to the parameters, with damping and background term (initially zero) subtracted

$$\begin{aligned}
\Delta\tilde{a}_j^i &= U_{j,k}^i \cdot \Delta\tilde{b}_k^i \qquad \textbf{and} && (14.88)\\
\Delta\tilde{b}_k^i &= \frac{1}{\lambda_k^i + \Delta\lambda_k^i} \cdot \left(U^i\right)_{k,j}^T \cdot \left(S^i\right)_{j,n}^T \cdot W_{n,n}^i \cdot \left(y_n - f_n(t, a_j^{i-1}) - \Phi_n^i\right) && (14.89)
\end{aligned}$$

12. Compute the new background term for the <u>next</u> iteration

$$\begin{aligned}
\Phi_n^{i+1} &= S_{n,j}^i \cdot \left(\Delta a_j^i(0) - \Delta a_j^i\right) \qquad \textbf{or} && (14.90)\\
\Phi_n^{i+1} &= S_{n,j}^i U_{j,k}^i \cdot \left(\Delta b_k^i(0) - \Delta b_k^i\right) && (14.91)
\end{aligned}$$

where, $\Delta_j^i a(0)$ is the desired change without damping or background term (NOTE: if λ_k^i is approaching zero then that eigenfunction is also tending towards zero and these components can be ignored).

$$\begin{aligned}
\Delta a_j^i(0) &= U_{j,k}^i \cdot \Delta b_k^i(0) \qquad \textbf{and} && (14.92)\\
\Delta b_k^i(0) &= \frac{1}{\lambda_k^i} \cdot \left(U^i\right)_{k,j}^T \cdot \left(S^i\right)_{j,n}^T \cdot W_{n,n}^i \cdot \left(y_n - f_n(t, a_j^{i-1})\right) && (14.93)
\end{aligned}$$

and Δa_j^i is the desired change with damping, but without the background term

$$\begin{aligned}
\Delta a_j^i &= U_{j,k}^i \cdot \Delta b_k^i \qquad \textbf{and} && (14.94)\\
\Delta b_k^i &= \frac{1}{\lambda_k^i + \Delta\lambda_k^i} \cdot \left(U^i\right)_{k,j}^T \cdot \left(S^i\right)_{j,n}^T \cdot W_{n,n}^i \cdot \left(y_n - f_n(t, a_j^{i-1})\right) && (14.95)
\end{aligned}$$

13. check for convergence. A number of convergence criteria should be tested

    (a) The residuals are less than the expected error. One way to test this is to compute the expected error in the transformed parameter space (a function of the residuals)

    $$\delta b_k^i = \sqrt{\frac{1}{\lambda_k^i + \Delta\lambda_k^i}} \tag{14.96}$$

    If the sum-squared of the changes made (Eqn. 14.88) $\sum \left(\Delta\tilde{b}_k^i\right)^2 \leq 0.1 \cdot \sum \left(\delta b_k^i\right)^2$ then we should not adjust any of the parameters any further.

    (b) If $\sum \left(\Delta\tilde{b}_k^i\right)^2 > \sum \left(\Delta\tilde{b}_k^i\right)^2$ then the solution if diverging and we should stop.

    (c) Stop if we exceed a maximum (preset) number of iterations.

    (d) $(\chi^i)^2$ can be computed in step #5 and then compared to $(\chi^{i-1})^2$ to determine if we are converging. However, this can be computationally expensive, since we will not be using the computed values.

14. Adjust the state, if solution is NOT diverging. The change is in proportion to the original method and perturbation value, $\Delta\hat{a}_j$, used to compute the Jacobian.

    $$a_j^i = a_j^{i-1} + \Delta\tilde{a}_j^i \otimes \Delta\hat{a}_j \qquad \textbf{for additive parameters} \tag{14.97}$$

    $$a_j^i = a_j^{i-1} \cdot \left[1 + \Delta\tilde{a}_j^i \otimes \Delta\hat{a}_j\right] \quad \textbf{for multiplicative parameters} \tag{14.98}$$

15. To compute an error estimate we can use the fraction of the transformed parameters we believed, ϕ_k^i, and compute the root-sum-squared (RSS) of the error in our parameter space

    $$\delta a_j^i = \sqrt{\sum_{k=1}^{J} \left[\left(U_{j,k}^i \phi_k^i \frac{1}{\sqrt{\lambda_k^i}} \right)^2 + \left(1 - \phi_k^i\right)^2 \cdot \sum_{j=1}^{J} \left(\left(U^i\right)_{k,j}^T \cdot \delta a_j^{i-1} \right)^2 \right]} \tag{14.99}$$

    If the convergence is diverging or we exceeded a preset number of iterations then the last computed change, Δa_j^i, could be added (in the RSS sense) to the error estimate.

    Since the eigenvector matrix makes a linear combination of the original functions we can compute and estimate of the fraction of the original parameters solved with

    $$\Phi_j^i = \sqrt{\sum_{k=1}^{J} \left(U_{j,k}^i\right)^2 \cdot \phi_k^i} \tag{14.100}$$

    A useful printout at the end of each iteration includes, a_j^{i-1}, $\Delta a_j^i \otimes \Delta\hat{a}_j$, , a_j^{i+1}, Φ_j^i, and the error estimate, δa_j^i.

16. if NOT converged, then goto step #5

Appendix A

References

A.1 General References for Numerical Methods

Abramowitz, M. and I. A. Stegun, **Handbook of Mathematical Functions** Dover NYC, 1965.

Bevington, P.R. 1969, "Data reduction and error analysis for the physical sciences." McGraw-Hill Book Company 336 pgs. (Note: in 1992 the 2nd edition was published, 328 pgs.

Burden, R.L., J.D. Faires 1985. "Numerical Analysis, 3rd Edition." Prindle, Weber, and Schmidt, Boston. 676 pgs.

Alejandro Garcia 2000, "Numerical Methods for Physics, 2nd Edition.", Prentice Hall. 423 pgs.

Gould, H. and J. Tobochnik 1996. "An Introduction to Computer Simulation Methods, 2nd Ed.." *Addison-Wesley Series in Physics* 695 pps.

Hamilton, W.C. 1964. "Statistics in physical science: Estimation, hypothesis testing, and least squares." The Ronald Press Company, New York. 225 pages.

Hamming, R.W., E.A. Feigenbaum 1971. Introduction to applied numerical analysis. McGraw Hill, Inc. NY.

Hayes, Monson H. 1999. **Schaum's outline series on Digital Signal Processing** (McGraw-Hill, New York).

Heath, M. T. "Scientific Computing: An Introductory Survey " McGraw-Hill NYC, 1st edition, 1997, 408 pages.

Luehrmann, A.W. 1974. Orbits in the solar wind - a mini-research problem. Amer. J. Phys. v.42 p.361-371.

Merrill, J.R. and R.A. Morrow 1979. An introductory scattering experiment by simulation. Amer. J. Phys. v.38 p.1104-1107.

Milne, W.E. 1970. Numerical Solution of Differential Equations. Dover Publications, Inc. New York, NY, 259 pgs.

Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vettering 1986. **Numerical Recipes in FORTRAN : the art of scientific computing** (Cambridge Univ. Press), 818 pgs.

Scheid, F. 1989. **Schaum's outlines of Theory and Problems of Numerical Analysis, 2nd Ed.** (McGraw-Hill, New York).

Turner, Peter R. 2001. **Guide to Scientific Computing, 2nd Ed.** (MacMillan & CRC Press).

Twomey, Sean 1977. **Introduction to the mathematics of inversion in remote sensing and indirect measurements**, Elsevier Scientific Publ., 243 pgs. **NOTE: This text was re-printed in 1996 by Dover Publications, Inc.**

A.2 References for Monte-Carlo methods

Frenkel, D., B. Smit 2001. Understanding Molecular Simulations: from algorithms to applications, 2nd Ed.

Metropolis, N., Rosenbluth, A. Rosenbluth, M., Teller, A., and Teller, E. 1953. *J. Chem. Phys.* **21**, p. 1087.

Pozdnyakov, L.A., Sobol, I.M., and Syunyaev, R.A. 1983. "Comptonization and shaping of X-ray source spectra: Monte Carlo calculations" in *Soviet Scientific Reviews, Sect. E: Astrophysics and Space Phys. Reviews*, Harwood-Academic, New York, 2, 189-331.

Sobol, I.M., 1994. A primer for the Monte Carlo method, CRC Press,

A.3 References for Fourier Series and Transforms

Bell, R. J. (1972). *Introductory Fourier Transform Spectroscopy*, (Academic Press, New York, 382 pp).

Brigham, E. O. 1988. "The Fast Fourier Transform and Its Applications", Prentice Hall, Englewood Cliffs, NJ.

Briggs, W. L., V. E. Hensen 1995. "The DFT: An Owner's Manual for the Discrete Fourier Transform." *SIAM* 423 pg.

Churchill, R.V. 1969. "Fouruer Series and Boundary Value Problems. 2^{nd} Ed.", McGraw Hill, 248 pgs.

Cochran, W. T., J. W. Cooley, D. L. Favin, H. D. Helms, R. A. Kaenel, W. M. Lang, G. C. Maling, D. E. Nelson, C. M. Rader, P. D. Welch 1967. "What is the Fast Fourier Transform." *IEEE Trans. Audio Electroacoust.* **15**, 45-55.

Cooley, J. W. and J. W. Tukey, 1965. "An algorithm for the machine calculation of complex Fourier series." *Math. and Comput.* **19**, 297-301.

Fourier Transforms: Apodization

Amato, U, I. De Feis, V. Cuomo, and C. Serio, 1994. "Regularization Methods to solve inverse problems: an investigation in the context of Fourier Spectroscopy from satellite," in *Proc. 5th Workshop Atmospheric Science from Space using Fourier Transform Spectroscopy*, Tokyo, Japan, Nov. 30 - DeC. 2, 1994

Amato, U. V. Cuomo, and C. Serio 1993a. "An advanced optimal spectral estimation algorithm in Fourier spectroscopy with application to remote sensing of the atmosphere," *J. Appl. Met.*, vol. 32, pp. 1508-1520, Sep. 1993.

Amato, U, V. Cuomo, and C. Serio 1993b. "Inverting interferogram signals to retrieve atmospheric temperature profiles," in *Tech. Proc. 7th Int. TOVS study Conf.*, Ed. J. Eyre, Igls, Austria, Feb. 10-16, 1993, ECMWF, Reading UK, pp. 1-10.

Bartlett, M. S. (1950). Periodogram analysis and continuous spectra. *Biometrika* **37**, p.1-16.

Beer, R. 1992. Remote Sensing by Fourier Transform Spectrometry. John Wiley and Sons, Inc, New York. 153 pgs.

Blackman, R. B. and J. W. Tukey (1959). The Measurement of Power Spectra. Dover, New York,190 pp. (TK5101.B62)

Kuo, F. F and J. F. Kaiser (1966). "System Analysis by Digital Computer," John Wiley and Sons, New York, 438 pp. (TA 345.K96)

Buijs, H.L. 1969. "Fast Fourier transformation of large arrays of data." *Applied Optics* **8** p. 211-213.

Hamming, R. W. (1977). Digital Filters, 2^{nd} edition. Prentice Hall, New Jersey, 257 pp. (QA 297.H26)

Lucy, L.B. 1974. An iterative technique for the rectification of observed distributions. Astron. J. v.79 p.745-754.

Norton, R.H. and R. Beer (1976). "New apodizing functions for Fourier spectroscopy." *J. Opt. Soc. of America* **66** p.259-264.

Norton, R. H. and R. Beer (1977). errata. *J. Opt. Soc. of America* **67** p.419.

Singleton, R. 1967. "A method for computing the fast Fourier transform with auxiliary memory and limited high-speed storage." *IEEE Trans. Audio Electroacoust.* **15** p. 263-270.

Winograd, J.M. and S.H. Nawab. 1995. "Mixed radix approach to incremental DFT refinement. *SPIE* **2563** p.418-429.

Ward, H. R. (1973) "Properties of Dolph-Chebyshev weighting functions." *IEEE Trans. Aerospace and Elec. Sys.*, **9**, No. 5, p. 785

A.4 References for Non-linear Least Squares

Abbas, M.M., G.L. Shapiro, B.J. Conrath, V.G. Kunde and W.C. McGuire 1984. Method for correction of errors in observation angles for limb thermal emission measurements. Applied Optics v.23 p.1862-1866.

Backus, G. and F. Gilbert 1970. Uniqueness in the inversion of inaccurate gross earth data. Phil. Trans. Roy. Soc. Lond. v.266 p.123-192.

Bjerhammar, Arne 1951. Rectangular reciprocal matrices, with special reference to geodetic calculations. Bull. Geodesique v.20 p.188-229.

Greville, T.N.E. 1959. The pseudoinverse of a rectangular or singular matrix and its application to the solutions of systems of linear equations. SIAM Rev. v.1 p.38-43.

Conrath, B.J. 1977. Backus-Gilbert theory and its application to retrieval of ozone and temperature profiles. in "Inversion Methods in Atmospheric Remote Sounding" (ed. Deepak) p.155-193.

Conrath, B.J. 1972. Vertical resolution of temperature profiles obtained from remote sensing radiation measurements. J. Atmos. Sci. v.29 p.1262- 1271.

Conrath, B.J. and I. Revah 1972. A review of nonstatistical techniques for the estimation of vertical atmospheric structure from remote infrared measurements. NASA TMX-62,150 (Math. of Profile Inversion) v.1 p.36-49.

Eyre, J.R. 1990. The information content of data from satellite sounding systems: a simulation study. Quart. J. Roy. Meteor. Soc. v.116 p.401- 434.

Eyre, J.R. 1989a. Inversion of cloudy satellite sounding radiances by nonlinear optimal estimation. I: Theory and simulation for TOVS. Quart. J. Roy. Meteor. Soc. v.115 p.1001-1026.

Eyre, J.R. 1989b. Inversion of cloudy satellite sounding radiances by nonlinear optimal estimation. II: Application to TOVS data. Quart. J. Roy. Meteor. Soc. v.115 p.1027-1037.

Johnson, L.E. and F. Gilbert 1972. Inversion and inference for teleseismic data. Method in Computational Physics (ed. Bolt, Academic Press) v.12 p.231-266.

Kursinski, E.R., S.S. Leroy, J.R. Eyre and et al. 1996. Initial results of radio occultation observations of Earth's atmosphere using the Global Positioning System. Science v.271 p.1107-1110.

Hanel, R.A., B.J. Conrath, D.E. Jennings and R.E. Samuelson 1992. Exploration of the solar system by infrared remote sensing. Cambridge Univ. Press 458 pgs.

Marquardt, D.W. 1963. An algorithm for least-squares estimation of nonlinear parameters. J. Soc. Indust. Appl. Math v.11 p.431-441.

Powell, M.J.D. 1965. A method for minimizing a sum of squares of non- linear functions without calculating derivatives. The Computer Journal v.7 p.303-307.

Rodgers, C.D. 2000. Inverse methods for atmospheric sounding: Theory and practice. *World Scientific Publishing*, 238 pgs.

Rodgers, C.D. 1996. Information content and optimization of high spectral resolution measurements. SPIE v.2830 p.136-147.

Rodgers, C.D. 1990. Characterization and error analysis of profiles retrieved from remote sounding measurements. J. Geophys. Res. v.95 p.5587-5595.

Rodgers, C.D. 1977. Statistical principles of inversion theory. in "Inversion Methods in Atmospheric Remote Sounding (ed. Deepak) p.117- 138.

Rodgers, C.D. 1976. Retrieval of atmospheric temperature and composition from remote measurements of thermal radiation. Rev. Geop. & Space Phys. v.14 p.609-624.

Rodgers, C.D. 1972. Statistical retrieval techniques for sounding the meteorological structure of the atmosphere. NASA TMX-62,150 (Math. of Profile Inversion) v.1 p.26-35.

Viera, G. and M.A. Box 1985. Information content analysis of remote sensing experiments using eigenfunctions. in Adv. in Remote Sensing Retrieval Methods (Deepak, Fleming, Chahine) p.287-302.

A.5 References for Differential Equations

A.5.1 Example of relaxation methods

Barnet, C.D., R.F. Beebe and B.J. Conrath 1992. A seasonal radiative- dynamic model of Saturn's troposphere. Icarus v.98 p.94-107.

Barnet, C.D. 1990. Saturn's seasonal winds and temperature: the effect of the ring system on the troposphere and stratosphere. PhD Dissertation 377 pgs.

Boas, M. 1966. in "Mathematical Methods in the Physical Sciences", Wiley

A.5.2 References for Shallow Water Model

Andrews, D.G., J.R. Holton and C.B. Leovy 1987. Middle Atmospheric Dynamics. Academic Press 489 pgs.

Dowling, Timothy and A.P. Ingersoll 1989. Jupiter's great red spot as a shallow water system. *J. Atmos. Sci.* **46** p.3256-3278.

Gill, A.E. 1982. *Atmosphere-Ocean Dynamics* Academic Press, New York.

Holton, J.R. 1979. The dynamic meteorology of the stratosphere and mesosphere. Met. Monographs, 15, No. 37 218 pgs.

Houghton, D., A. Kasahara and W. Washington 1966. Long-term inte- gration of the barotropic equations by the Lax-Wendroff method. Monthly Weather Review v.94 p.141-150.

Kasahara, A. 1966. The dynamical influence of orography on the large- scale motion of the atmosphere. J. Atmos. Sci. v.23 p.259-271.

Pedlosky, J. 1987. Geophysical fluid dynamics. Springer-Verlag Publ. 710 pgs.

Semazzi, F.H.M. 1980. Stationary barotropic flow induced by a mountain over a tropical belt. *Monthly Weather Review* **108** p.922-930.

Zehnder, J.A. 1991. The interaction of planetary-scale tropical easterly waves with topography: a mechanism for the initiation of tropical cyclones. *J. Atmos. Sci.* **48** p.1217-1230.

Appendix B

Programs: (source code also given in ftp/phys_640)

NOTE: IDL arrays are indexed from (0) to (N-1).

B.1 BINNER: program to group data into bins (IDL)

```
pro binner2,xax,yax,bin,xout,yout,wstart,wend
;+
; NAME:
;      binner
; PURPOSE:
;      Created to bin data in the form of wavelength and intensity
;      to a uniform binned data set.
; CALLING SEQUENCE:
;      binner,xax,yax,bin
; INPUTS:
;      xax = input x-axis data (ie. wavelength)
;      yax = input y-axis data (ie. intensities)
;      bin = size of bin in x-axis space
; OPTIONAL INPUT PARAMETERS:
;      None.
; KEYWORD PARAMETERS:
;      None.
; OUTPUTS:
;      xout = output x-axis binned scale
;      yout = output y-axis binned data
; RESTRICTIONS:
;      You will be asked to set a beginning and ending xax value. The
;      first xax value entered will also be the first xout value, and
;      the xout increment will be half of the bin size. The yout value
;      will be the weighted average of all yax values from xout-bin/2
;      to xout+bin/2.  Some prefer the xout increment to be equal to
;      the bin size. If this is the case, then you must double the value
;      of _bin_ when running the program.
```

```
;      In the cases where no yax data exists to be binned for an xout
;      value, a linear interpolation (or exterpolation if needed) is
;      performed. More accurate methods could probably be defined.
; PROCEDURE:
;      Note that the outputs are one dimensional arrays.
;      The program will ask for a beginning value from which to start, and
;      a maximum value which the program will not exceed.
; MODIFICATION HISTORY:
;       Created 1991 Oct. by H. Peter White  ISTS/SAL.
;+
xax=float(xax)
wave=make_array(fix(n_elements(xax)+2))
wave(1:n_elements(wave)-2)=xax
yax=float(yax)
ints=make_array(fix(n_elements(yax)+2))
ints(1:n_elements(ints)-2)=yax
if(n_elements(wave) eq n_elements(ints))then begin
  numb=fix(n_elements(wave))-1
  bin=float(bin)
;  print,' '
;  print,'At what wavelength do you wish to start?'
;  read,wstart
  wstart=float(wstart)
;  print,'At what wavelength do you wish to end?'
;  read,wend
  wend=float(wend)
;
;  Create the 'new' first data point via an extrapolation method.
;
  if(wstart lt wave(1)) then begin
    wave(0)=wstart-bin
    m=(ints(2)-ints(1))/(wave(2)-wave(1))
    ints(0)=ints(1)+(m*(wave(0)-wave(1)))
  endif else begin
    wave(0)=wave(1)-bin
    m=(ints(2)-ints(1))/(wave(2)-wave(1))
    ints(0)=ints(1)+(m*(wave(0)-wave(1)))
  endelse
;
;  Create the 'new' last wavelength point via an extrapolation method.
;
  if(wend gt wave(numb-1))then begin
    wave(numb)=wend+bin
    m=(ints(numb-1)-ints(numb-2))/(wave(numb-1)-wave(numb-2))
    ints(numb)=ints(numb-1)+(m*(wave(numb)-wave(numb-1)))
  endif else begin
    wave(numb)=wave(numb-1)+bin
    m=(ints(numb-1)-ints(numb-2))/(wave(numb-1)-wave(numb-2))
    ints(numb)=ints(numb-1)+(m*(wave(numb)-wave(numb-1)))
  endelse
;
;  By creating a new beginning and end data point, the program is now able
;  to find the points within the data set from which to start and end the
```

```
;  binning process if these points existed outside the data range
;
;
;  Set the first wavelength of the output data set equal to the first point
;  inputed as the beginning value to start binning from.
;
  xout=make_array(fix((wend-wstart)/(bin/2.))+1)
  yout=xout*0.0
  for j=0,n_elements(xout)-1 do xout(j)=wstart+((bin/2.)*float(j))
  b2=float(bin/2.)
;
;  Start binning procedure for xout value, using a weighted average for
;  the output intensities. The combined intensities from the input data
;  within the bin are wieghted so that the intensity nearest the centre
;  of the bin is given the highest wieght, and the intensity furthest
;  from the centre of the bin is given the lowest weight. This weighting
;  parameter is  deltax, and is a relative parameter.
;
  for j=0,n_elements(xout)-1 do begin
;
;  The intensity points are averaged for wavelength values greater than
;  the lower edge of the bin to wavelength values less than or equal to the
;  higher edge of the bin. Thus find these points if they exist
;
    pts=fix(where((wave le (xout(j)+(b2))) and (wave gt (xout(j)-(b2)))))
;
;  If a point is unique in this bin, then do the following.
;
    h3=fix(size(pts))
    if((h3(0) eq 1) and (h3(1) eq 1)) then begin
      yout(j)=yout(j)+(ints(pts(0))*(bin))
    endif
;
;  If a point is not unique within this bin, then find its wieghted
;  intensity contrabution, based on its position in the bin.
;  ie, is it the furthest to one side of the points in the bin, or
;  does it have valid points to both sides.
;
    if(h3(0) eq 1 and h3(1) ge 2)then begin
;  For the smallest wavelength in the bin
      deltaw=((wave(pts(1))+wave(pts(0)))/2.)-(xout(j)-b2)
      yout(j)=yout(j)+(deltaw*ints(pts(0)))
;  For the largest wavelength in the bin
      deltaw=(xout(j)+b2) -  $
       ((wave(pts(n_elements(pts)-1))+wave(pts(n_elements(pts)-2)))/2.)
      yout(j)=yout(j)+(deltaw*ints(pts(n_elements(pts)-1)))
;  For the other wavelengths in the bin, if any
      if(h3(1) gt 2)then begin
        for k=1,n_elements(pts)-2 do begin
          deltaw=((wave(pts(k+1))+wave(pts(k)))/2.) - $
                 ((wave(pts(k))+wave(pts(k-1)))/2.)
          yout(j)=yout(j)+(deltaw*ints(pts(k)))
        endfor
```

```
      endif
    endif
;
;  If no origional intesity values were found, then interpolate a value
;  from the two surrounding origional intesity values as the binned value.
;
    if((h3(0) eq 0)) then begin
      ira=fix(where(wave lt xout(j)))
      irb=fix(where(wave ge xout(j)))
      m=(ints(min(irb))-ints(max(ira)))/(wave(min(irb))-wave(max(ira)))
      yout(j)=ints(max(ira))+(m*(xout(j)-wave(max(ira))))
      yout(j)=yout(j)*bin
    endif
;
;  This step is required due to the weighted summation
;   done in the above steps.
;
    yout(j)=yout(j)/(bin)
  endfor
endif else begin
  print,'The input arrays must be 1 dimensional arrays only!'
endelse
return
end
```

B.2 CFFT: FFT routine (FORTRAN)

```
      subroutine cfft(Npts, cx, idir)
      implicit none
      integer*4  Npts   ! # of points in array
      complex*8  cx(*)  ! complex array of input/output data
      integer*4  idir   ! -1 is FORWARD, +1 is INVERSE FFT
      integer*4  bitinv ! external function

c     local variables
      integer*4  k,k1,k2,nu,pass,subpas,npw
      integer*4  zero, one, two
      real*4     pi, sgn, arg, a0
      complex*8  X1,X2,Wp, iwt
      parameter (zero=0, one=1, two=2)
      parameter (pi=3.14159265)

c      cx(i),  1 <= i <= NPTS, single precision complex values of X
c       - Radix-2 Decimation in Time Algorithm
c       - In-Place-Calculation
c       - Bit-inversion performed prior to FT passes.
c         IDIR=-1 for forward and +1 for reverse FFT
c     nu = # of decimation passes
      nu = int(0.1+alog(float(Npts))/alog(2.0))
      k1 = 2**nu   ! check that Npts has 2^nu points
      if(k1.ne.Npts) then
        print *, 'CFFT: require 2^nu number of points'
```

```
        stop
      endif

c     swap the complex words (bit-reversal operation)
c     -----------------------------------------------
      do k1=one,Npts
         k2 = bitinv(nu,k1)
         if(k2.gt.k1) then
           x1 = Cx(k1)
           Cx(k1) = Cx(k2)     ! Cx(k1) <=> Cx(k2)
           Cx(k2) = x1
         endif
       enddo

c     *** Degenerate Pass *** (COS(w)=1, SIN(w)=0)
c     --------------------------------------------
      if(idir.gt.0) then          ! inverse FFT
        sgn = -1.0
        a0  = 1.0/float(Npts)
      else                        ! FORWARD FFT
        sgn = 1.0
        a0  = 1.0
      endif
      Wp = cmplx(a0,0.0)
      do K1 = one,Npts-one,two
        K2 = K1 + one
        X1 = Cx(K1)
        X2 = Cx(K2)
        Cx(K1) = Wp*(X1+X2)
        Cx(K2) = Wp*(X1-X2)
      enddo

c     *** Rest of Passes ***
c     ----------------------
      do pass = one,NU-one
        npw = 2**pass
        A0 = sgn*pi/float(npw)
        do K = one,NPW
          arg = A0*float(one-K)
          iwt = CMPLX(0.0,arg)
          Wp =  CEXP(iwt)
          do Subpas=zero,Npts-one,two*NPW
            K1 = Subpas+K
            K2 = K1+NPW
            X1 = Cx(K1)
            X2 = Cx(K2)*Wp
            Cx(K1) = X1+X2
            Cx(K2) = X1-X2
          enddo
        enddo
      enddo
      return
      end
```

B.3 EIGSRT: sorting of eigenvalues and functions (FORTRAN)

To sort the eigenvalues and eigenfunction from largest, D(1), to smallest, D(N), the FORTRAN subroutine EIGSRT() is used.

```
      SUBROUTINE EIGSRT(D,V,N,NP)
      implicit none
      integer*4 n, np
      real*8    d(np),v(np,np)

c*****************************************************************************
C.! ROUTINE: EIGSRT
C.! PURPOSE: This routine given the eigenvalues and eigenvectors
C.!          sorts the eigenvalues into ascending order, and rearranges
C.!          the columns of square matrix correspondingly.  The method
C.!          is straight insertion.
C.!
C.!          see pg. 348 w/ explanation pgs.335-376
C.!          Numerical Recipes: The Art of Scientific Programming
C.!          (FORTRAN version), 1st edition
C.!          W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling
C.!          Cambridge Univ. Press., 1986
C.!
c*****************************************************************************

      integer*4 i,j,k
      real*8    p

      do 13 i=1,n-1
        k = i
        p = d(i)
        do j=i+1,n
          if(d(j).ge.p) then
            k = j
            p = d(j)
          end if
        enddo
        if( k .ne. i ) then
          d(k) = d(i)
          d(i) = p
          do j = 1,n
            p = v(j,i)
            v(j,i) = v(j,k)
            v(j,k) = p
          enddo
        end if
  13  continue

      return
      end
```

B.4 FOUR_FIT: program to compute Fourier Series coefficients (IDL)

```
      pro four_fit, yx, kmax, Cn

;     note: The traditional Fourier series returns A0/2, in
;           this formulation A0 is the true average of the data.
;
;     yx(n) = a0 + sum{Ak*COS(2*pi*k*n/N) + Bk*sin(2*pi*k*n/N}
;       for n=0 to N-1 and k=1,kmax
;
;     Ak,Bk returned in complex variable, Ck = Ak - i*Bk

      zero = 0.0d00
      two = 2.0d00

      i1 = size(yx)
      nd = i1(1)              ; SIZE of input data array, yx(0:nd-1)

      Ck = DCOMPLEXARR(kmax)  ; make array for output

      xn = !dpi*2.0d00/DOUBLE(nd)

;     trapeziodal integration on even sample of f(i)
;     (yx(0) + yx(nd))/2 + sum(yd(1:nd-1)) = sum(yx)

      Ak = TOTAL(yx)/DOUBLE(nd)
      Ck(0) = DCOMPLEX(Ak,zero)

      for k = 1,kmax-1 do begin
        Ak = yx(0)
        Bk = zero
        xk = xn*DOUBLE(k)
        for n = 0L,nd-1 do begin
          theta = xk*DOUBLE(n)
          Bk = Bk + SIN(theta)*yx(n)
          Ak = Ak + COS(theta)*yx(n)
        endfor
        Bk = two*Bk/DOUBLE(nd)
        Ak = two*Ak/DOUBLE(nd)
        Cn(k) = DCOMPLEX(Ak,-Bk)
      endfor

      end
```

B.5 GASDEV: Gaussian distributed random number generator (IDL)

```
;   seed is double precision
    pro gasdev, seed, ran2
```

```
    ix = LONG(seed)
    one = 1.0d00
    two = 2.0d00

     n = 0L
    lpcn1:
     rx = ran3(ix)   ; SEED is set when ix <> 0
     v1 = two*rx - one
     n = n + 1L

     ix = 0L
     seed = DOUBLE(ix)  ; SEED has been set, now zero it
     rx = ran3(ix)
;    -----------------------
     v2 = two*rx - one
     r = v1^2 + v2^2
     if(n ge 50L) then begin
       print, "gasdev: error on n=50"
       stop
     endif
     if(r ge one) then goto, lpcn1
     fac = sqrt(-two*alog(r)/r)
     ran2 = v2*fac
    end
```

B.6 GAULEG: zeroes of Legendre polynomials (FORTRAN)

```
      subroutine gauleg(Npts,x0,x1,Xpts,Wgt)
      implicit none
      integer*2 Npts
      real*4    x0,x1,Xpts(*),Wgt(*)

      real*8    eps,pi,qtr,half,one,two
      parameter (pi=3.141592654d00,qtr=0.25d00,half=0.5d00)
      parameter (one=1.0d00,two=2.0d00,eps = 3.0d-14)

c     input:
c       npts = desired # of points in integral
c         x0 = lower limit in X
c         x1 = upper limit in X
c
c     output:
c       Xpts(1)..Xpts(npts) = X positions to calculate integrand
c        Wgt(1).. Wgt(npts) = weights associated with X values, these
c                             are simply passed to call quasquad().
c
c     see Press et al., 1986.  Numerical Recipes: The art of Scientific
c           Computing  p.125 for a description of this algorithm.

      integer*2 i,j,m,ik
      real*8    P1,P2,Xm,Xl,P3,Z,dx0,dx1,PP,Z1
```

```
      dx0 = dble(x0)
      dx1 = dble(x1)

      m = (npts+1)/2
      Xm = half*(dx1+dx0)
      Xl = half*(dx1-dx0)

      do 100 i = 1,m

c       Z is a  guess for m^th zero of Pm(x)
        Z = dcos(pi*(dfloat(i)-qtr)/(dfloat(npts)+half))

  110   P1 = one
        P2 = 0.0d00
        do j = 1,npts
           P3 = P2
           P2 = P1
           P1 = (dfloat(2*j-1)*Z*P2-dfloat(j-1)*P3)/dfloat(j)
        enddo

c       P1 is the Legendre polynomial, PP is its derivative

        PP = dfloat(npts)*(Z*P1-P2)/(Z*Z-one)
        Z1 = Z
        Z = Z1-P1/PP
        if(dabs(Z-Z1).gt.eps) goto 110

        ik = npts + 1 - i
         Xpts(i) = sngl(XM-XL*Z)
        Xpts(ik) = sngl(XM+XL*Z)
         Wgt(i) = two*XL/((one-Z*Z)*PP*PP)
        Wgt(ik) = Wgt(i)
  100 continue
      return
      end
```

B.7 LUDCMP: compute the LU decomposition of a matrix (FORTRAN)

```
c*****************************************************************
C.! ROUTINE:  LUDCMP
C.!
C.! PURPOSE: replaces matrix A(N,N) with the LU decomposition of a
C.!          rowwise permutation of itself.  NP is the dimension
C.!          of array A.  INDX(N) is an output vector (dimensioned
C.!          to NP) which records the row permutation.  D is the
C.!          value +/- 1.0 for even/odd row interchanges, respectively.
C.!          This routine is used with LUBKSB to find the inverse
C.!          of a matrix or to solve linear equations.
C.!
```

```
C.!          see pg. 35 w/ explanation pgs.19-39
C.!          Numerical Recipes: The Art of Scientific Programming
C.!          (FORTRAN version), 1st edition
C.!          W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling
C.!          Cambridge Univ. Press., 1986
C.!
c**********************************************************************

      SUBROUTINE LUDCMP(A,N,NP,INDX,D)
      implicit  none
      integer*4 N, NP, INDX(NP)
      real*4    A(NP,NP),D

      integer*4 I,J,K, IMAX, NMAX
      parameter (NMAX=600)
      real*4    AAMAX,VV(NMAX),SUM,DUM, TINY
      parameter (TINY=1.0E-20)

      D=1.
      DO 12 I=1,N
        AAMAX=0.0
        DO J=1,N
          IF(ABS(A(I,J)).GT.AAMAX) AAMAX=ABS(A(I,J))
        enddo
        IF(AAMAX.EQ.0.0) then
           print *, 'LUDCMP: ... Singular matrix ...'
           J = 9
           call softexit('LUDCMP  ',J)
        end if
        VV(I) = 1.0/AAMAX
12    CONTINUE

      DO 19 J=1,N

        IF (J.GT.1) THEN
          DO I=1,J-1
            SUM = A(I,J)
            IF (I.GT.1)THEN
              DO K=1,I-1
                SUM = SUM-A(I,K)*A(K,J)
              ENDDO
              A(I,J) = SUM
            ENDIF
          enddo
        ENDIF

        AAMAX=0.0
        DO I=J,N
          SUM = A(I,J)
          IF(J.GT.1)THEN
            DO K=1,J-1
              SUM = SUM-A(I,K)*A(K,J)
            ENDDO
```

```
            A(I,J) = SUM
          ENDIF
          DUM = VV(I)*ABS(SUM)
          IF(DUM.GE.AAMAX) THEN
            IMAX  = I
            AAMAX = DUM
          ENDIF
        enddo

        IF(J.NE.IMAX)THEN
          DO K = 1,N
            DUM = A(IMAX,K)
            A(IMAX,K) = A(J,K)
            A(J,K) = DUM
          ENDDO
          D = -D
          VV(IMAX) = VV(J)
        ENDIF
        INDX(J) = IMAX

        IF(J.NE.N) THEN
          IF(A(J,J).EQ.0.) A(J,J)=TINY
          DUM = 1.0/A(J,J)
          DO I = J+1,N
            A(I,J) = A(I,J)*DUM
          ENDDO
        ENDIF
19    CONTINUE

      IF(A(N,N).EQ.0.0) A(N,N) = TINY

      RETURN
      END
```

B.8 LUBSKB: matrix inverse via back-substitution (FORTRAN)

```
c*****************************************************************
C.! ROUTINE:  LUBKSB
C.!
C.! PURPOSE: solves  A*X = B.  A(N,N) and B(N) are input.  A(N,N)
C.!          is the LU decomposition found with LUDCMP.  INDX is an
C.!          output array from LUDCMP.  NP is the dimension of
C.!          A(NP,NP), B(NP), INDX(NP).
C.!
C.!          see pg. 36 w/ explanation pgs.19-39
C.!          see pg. 38 for example of finding inverse.
C.!          Numerical Recipes: The Art of Scientific Programming
C.!          (FORTRAN version), 1st edition
C.!          W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling
C.!          Cambridge Univ. Press., 1986
C.!
c*****************************************************************
```

```
      SUBROUTINE LUBKSB(A,N,NP,INDX,B)
      implicit none
      integer*4 N,NP,INDX(*)
      real*4    A(NP,NP),B(*)

      integer*4 I,J,LL,II
      real*4    SUM

      II = 0
      DO I = 1,N
        LL  = INDX(I)
        SUM = B(LL)
        B(LL) = B(I)
        IF(II.NE.0)THEN
          DO J = II,I-1
            SUM = SUM-A(I,J)*B(J)
          enddo
        ELSEIF(SUM.NE.0.0) THEN
          II = I
        ENDIF
        B(I) = SUM
      enddo

      DO I = N,1,-1
        SUM = B(I)
        IF(I.LT.N) THEN
          DO J = I+1,N
            SUM = SUM-A(I,J)*B(J)
          enddo
        ENDIF
        B(I) = SUM/A(I,I)
      enddo

      RETURN
      END
```

B.9 LINEAR: A linear least squares solution for a line (FORTRAN)

```
c     subroutine linear(npts,XD,YD,wgt,par,err)
c     calls:  -none-
c     fncts:  -none-
c     common: -none-
c     input:  npts[i2], XD([r4(*)], YD[r4(*)], wgt[r4(*)]
c     output: par[r4(*)], err[r4(*)]
c     use:    The user enters values of XD(i) and YD(XD(i)) for
c             1 <= i <= npts.  Also wgt(i) must be specified for
c             each point.  A value of 1.0 should be used if no
c             weighting in desired.
c             On return the following values will be in output arrays:
```

```
      subroutine linear(npts,XD,YD,wgt,par,err)
      implicit none
      integer*2  npts,i
      real*4  XD(*),YD(*),wgt(*),par(*),err(*)
      real*8  sumX,sumY,sumXX,sumYY,sumXY,sumWT
      real*8  XV,YV,WV,R,CHI2,delta,A0,A1,zero
      parameter (zero=0.0d00)

c     input :
c         npts = number of data points
c         XD(i) = absicca values       1 <= i <= npts
c         YD(i) = function(X(i))
c         wgt(i) = weight to apply in least squares fit
c
c     output :
c         YC = slope*XD(i) + intercept
c         par(1) = intercept
c         par(2) = slope
c         err(1) = error in intercept
c         err(2) = error in slope

      sumX  = zero
      sumY  = zero
      sumXX = zero
      sumXY = zero
      sumYY = zero
      sumWT = zero
      do i = 1,npts
        XV = dble(XD(i))
        YV = dble(YD(i))
        WV = dble(WGT(i))
        if(WV.gt.zero) then
          sumX  =  sumX + XV*WV
          sumY  =  sumY + YV*WV
          sumXX = sumXX + XV*XV*WV
          sumXY = sumXY + XV*YV*WV
          sumWT = sumWT + WV
        endif
      enddo

      delta = sumXX*sumWT - sumX*sumX
       A0 = ( sumXX* sumY - sumX*sumXY)/delta
       A1 = ( sumWT*sumXY - sumX*sumY )/delta

      PAR(1) = sngl(A0)
      PAR(2) = sngl(A1)

       sumY = zero
      sumYY = zero
      do i=1,Npts
        R = a0 + a1*dble(XD(i)) - dble(YD(i))  ! residual
        WV = dble(WGT(i))
        if(WV.gt.zero) then
```

```
          sumY  =  sumY + WV*R       ! weighted sum of residuals
          sumYY = sumYY + WV*R*R     ! weighted sum of residuals**2
        endif
      enddo
      R = DBLE(float(NPTS))
      CHI2 = (sumYY/sumWT)*R/(R-2.0d00)  ! chi squared
      A0 = dsqrt(CHI2*sumXX/delta)
      A1 = dsqrt(CHI2*sumWT/delta)
      ERR(1) = sngl(A0)
      ERR(2) = sngl(A1)
      return
      end
```

B.10 lsq_linear: solve for polynomial coefficients using the Vandermonde matrix (IDL)

The IDL program below (*ftp*/lsq_linear.pro) computes the linear least squares fit to a data array (Y(n), T(n), n=0,N-1) for a general fitting matrix X via intrinsic matrix manipulation routines in the language. For polynomial fitting it requires the Vandermonde matrix computed in lsq_vand.pro.

```
pro lsq_linear, T, Y, X, PVAL, PERR

; input:
;  T() is the independent variable
;  Y() is the dependent variable
;  X(Npts,Ncoef) = matrix for use in Normal Eqn's
;
; output;
;  PVAL = value of parameters
;  PERR = estimated error in parameters

;  compute the normal equations and solve for coefficients

   i1 = size(X)
   Npts = i1(1)        ; # of data points in y(t) arrays
   Ncoef = i1(2)       ; # of parameters to solve for

   XT = TRANSPOSE(X)        ; X transpose
   B = INVERT(XT#X)         ; [X'X]
   PVAL = B#XT#Y               ; solution of parameters

;  compute the residuals and chi^2 from the solved parameters
   Yc = X#PVAL                 ; Ycomputed
   resid = Y-Yc             ; residuals
   chi2= TOTAL(resid*resid)/float(Npts-Ncoef) ; chi^2
;  compute the parameter error covariance matrix & error in parameters

   E = chi2*B               ; predicted parameter error covariance
   idx = make_array(Ncoef,/index)
   PERR = SQRT(E(idx,idx))  ; error in parameters (predicted)

end
```

B.11 lsq_vand: build the Vandermonde matrix (IDL)

The IDL program below (*ftp*/lsq_poly.pro) computes the Vandermonde matrix for use in solving polynomial coeffients (*ftp*/lsq_linear.pro).

```
; Builds the Vandermonde Matrix for Least Squares Fitting
; of polynomials specified order (# coefficients = order + 1)
;
; After this call use lsq_linear to compute the solution of the
; parameters

pro lsq_vand, T, Norder, X

; set's up Vandermonde Matrix for y(t)
; for a polynomial of order = Norder
;
; input:
;  T() is the independent variable for n=0,Npts-1
;  Norder = the derired polynomial order
;
; output:
;  X(Npts,Ncoef) = Vandermonde matrix for use in Normal Eqn's

  i1 = size(T)
  Npts = i1(1)

  Ncoef = LONG(Norder)+1L

  X = fltarr(Npts,Ncoef)        ; declares space for my equations
  X(*,0) = replicate(1.0,Npts)  ; 1st columns is 1's
  X(*,1) = T                    ; 2nd column is independent variable
  for i = 2L, Ncoef-1L do begin
    X(*,i) = T^i
  endfor

end
```

B.12 MX_LSQR: a symmetric matrix inverter (FORTRAN)

In general, for least squares fitting we need a program to invert a square symmetric matrix. The following program (*ftp*/mx_lsqr.f) will solve the normal equations and perform and error analysis with a relatively small amount of memory of memory

```
!ALGORITHM REFERENCES:  None
C   This program was originally written by someone at LICK observatory
C   for use in a program called GLSQ.  It was obtained from Bernard
c   McNamara at NMSU in 1979.
C
      subroutine mx_lsqr(NP,AL,AR,Param,Qerr)
      integer*2  NP
```

```
      integer*2  npm
      parameter  (npm=10)
      real*8    AR(npm,npm),AL(npm),Param(npm),Qerr(npm,npm)

      integer*2 i,j,k,l,ni,nii,nj,njj,npa
      real*8    R(npm,npm),RL(npm),Q(npm,npm)
      real*8    zero,one,sum
      parameter (zero=0.0d00, one=1.0d00)

c     ------------------------------------------------
c     General Least Squares Solution of NDP equations
c          with NP unknowns.
c     ------------------------------------------------
c
c    uses AL(i),AR(i,i),RL(i),R(i,i),Q(i,i), for i=1 to NP
c
c    Inverse of Normal Matrix = Q(i,j), i=1 to NP, j=1 to NP
c    where,
c    YD(i) = Param(1)*X(1,i) + Param(2)*X(2,i) + ... Param(NP)*X(NP,i)
c            for i=1 to NDP
c
c     set up right hand side of normal equations
c     ( AR(I,J)=sum(X(I,i)*X(J,i)), i=1 to NDP )
c
c     note : normal equation matrix is symmetric and square
c     ------------------------------------------------

      do 100 i = 1,np
        Param(i) = zero
        RL(I)    = zero
        do j = 1,np
          Q(i,j)  = zero
          R(I,J)  = zero
        enddo
 100  continue

c     ------------------------------------------------
c     AL(i) = AR(i,j)*Param(j), for j=0 to NP
c    then Param(j) = inv{AR(i,j)}*AL(i))=Q(j,i)*AL(i)
c     calc. intermediate R(i,j) and AR(i,j) matrices
c     ------------------------------------------------

      NPA = NP-1
      do 400 I = 1,NPA
        K = I+1
        do J = K,NP
          R(I,J) = AR(I,J)/AR(I,I)
          SUM = AR(K,J)
          do L = 1,I
            SUM = SUM - R(L,K)*AR(L,J)
          enddo
          AR(K,J) = SUM
        enddo
```

```
  400 continue

c     ------------------------------------------------
c     calc. intermediate Rl(i), AL(i) matices
c     ------------------------------------------------
      RL(1)=AL(1)/AR(1,1)
      do 500 I=2,NP
        SUM = AL(I)
        do J = 1,I
          SUM = SUM - R(J,I)*AL(J)
        enddo
        AL(I) = SUM
        RL(I) = AL(I)/AR(I,I)
  500 continue

c     ------------------------------------------------
c     calc. solution to normal eqns. Param(i)
c     ------------------------------------------------

      Param(NP) = RL(NP)
      do 600 I = 1,NPA
        NI  = NP-I
        NII = NI+1
        SUM = zero
        do J = NII,NP
          SUM = SUM - R(NI,J)*Param(J)
        enddo
        Param(NI) = RL(NI) + SUM
  600 continue

c     ------------------------------------------------
c     calc. inverse matix of normal eqns. Q(i,j)
c     note : Q(i,i) terms are used in error determination.
c     ------------------------------------------------

      do I = 1,NP
        do J = 1,NP
          Q(I,J) = zero
        enddo
      enddo

      Q(NP,NP) = one/AR(NP,NP)
      do 700 I = 1,NPA
        NI = NP-I+1
        do J = I,NPA
          NJ = NP-J
          NJJ = NJ+1
          SUM = zero
          do K = NJJ,NP
            SUM = SUM - R(NJ,K)*Q(NI,K)
            Q(NI,NJ) = SUM
            Q(NJ,NI) = SUM
          enddo
```

```
        enddo
        NII = NI-1
        SUM = zero
        do L = NI,NP
          SUM = SUM - R(NII,L)*Q(L,NII)
        enddo
        Q(NII,NII) = SUM + one/AR(NII,NII)
  700 continue

      do i = 1,np
        do j = 1,np
          Qerr(i,j) = Q(i,j)
        enddo
      enddo

      return
      end
```

B.13 PLANCK: Compute the Planck function

The Planck function is used in radiation transfer problems and can be written as a function of wavelength, λ,

$$B_\lambda(T) = \frac{2 \cdot h \cdot c^2}{\lambda^5 \cdot \left(e^{(hc/\lambda kT)} - 1\right)} \tag{B.1}$$

where h is Planck's constant, c is the speed of light, k is Boltzmann's constant. It can also be written as a function of frequency, f, (NOTE: $B_f(T) \cdot \partial f = B_\lambda(T) \cdot \partial\lambda$)

$$B_f(T) = \frac{2 \cdot h \cdot f^3}{c^2 \cdot \left(e^{(hf/kT)} - 1\right)} \tag{B.2}$$

In this subroutine we will write it as

$$B_\nu(T) = \frac{2 \cdot h \cdot c^2 \nu^3}{e^{(hc\nu/kT)} - 1} \tag{B.3}$$

where, ν is the frequency expressed in wavenumbers which is related to the frequency in Hertz, f, as $\nu \equiv f/c$. as (NOTE: $B_\nu(T) \cdot \partial\nu = B_f(T) \cdot \partial f$). This is the form typically used in infrared work.

```
      function planck ( nu, bt )
      implicit none

c     Purpose: Real function to compute radiances from frequency and
c       temperature.  The unit of radiances is milliwatts per
c       ( steradian meter^2 cm^{-1}) = mW/steradian/M^2/cm^{-1}

      real*4   planck  ! output radiance in mW/M^2/steradian/cm^{-1}
      real*4   nu      ! frequency in wavenumbers(cm^{-1})
c                      ! nu = f/c where f is in units of Hz (s^{-1})
c                      !      and c is the speed of light in cm/s
      real*4   bt      ! brightness temperature in K

c     local constants: reference for these constants (see pg. 448,452):
c     Mohr, P.J. and B.N. Taylor 2000.  CODATA recommended values of the
```

```
c     fundamental physical constants: 1998.  Rev. Mod. Phys. v.72
c     p.351-495.  NOTE: MKS is converted to CGS units
      real*4   BOLTZMNS, PLANCKS, CLIGHT
      parameter( BOLTZMNS = 1.3806503E-16 )  ! k=Boltzmann's constant, erg/K
      parameter( PLANCKS  = 6.62606876E-27 ) ! h=Planck's constant, erg*s
      parameter( CLIGHT   = 2.99792458E+10 ) ! c=speed of light (cm/s)

c     reference for equation (pg. 21, Eqn. 1.7.2):
c     Hanel, R.A., B.J. Conrath, D.E. Jennings and R.E. Samuelson 1992.
c     Exploration of the solar systen by infrared remote sensing.
c     Cambridge Univ. Press 458 pgs.
c
c     limitations and assumptions:
c     this program is intended for terrestrial applications where
c     the EXP() function never approaches 1 (i.e., argument is never
c     small).  We will assume that
c         temp is always less than 500
c         nu   is always greater than 500

      real*4 PLCON1, PLCON2
      parameter ( PLCON1   = 2.0*PLANCKS*CLIGHT*CLIGHT) ! 2hc^2
      parameter ( PLCON2   = PLANCKS*CLIGHT/BOLTZMNS )  ! hc/k

c     pg. 21, Eqn. 1.7.2  B(v,T) = 2*h*c^2*nu^3/(e^{h*c*nu/k*T} - 1)
      planck = PLCON1*nu**3/(EXP(PLCON2*nu/bt)-1.0)

      return
      end

      program testplanck
      implicit none
c     this program tests the planck function.  Here is the example
c     output from this program:
c       600.00  250.00   84.081985
c       600.00  300.00  153.401703
c       600.00  350.00  238.632584
c
c     local variables:
      real*4 nu        ! wavenumber
      real*4 temp      ! temperature
      integer*4  i     ! loop index

      nu = 600.0
      do i = 250,350,50
        temp = FLOAT(i)
        print 100, nu, temp, planck(nu, temp)
      enddo
 100  format(f8.2,f8.2,f12.6)
      end
```

B.14 POLY: build the Vandermonde matrix and solve for polynomial coefficients (FORTRAN)

```
      subroutine poly(NDP,NP,XD,YD,WGT,PAR,Perr,chi2)
      integer*2  NP,NDP
      integer*2  ndpm,npm
      parameter  (ndpm=100,npm=10)
      real*4     chi2
      real*4     XD(*),YD(*),WGT(*),PAR(npm),Perr(npm,npm)

      integer*2  i,j,k
      real*8     YDCALC,Var,sum1,sum2
      real*8     Qerr(npm,npm),AR(npm,npm),AL(npm),Xi,Xj
      real*8     zero,one,Param(npm)
      parameter (zero=0.0d00, one=1.0d00)

c     ****************************************************
c      Least Squares Solution of Polynomial YD(x)
c     ****************************************************
c
c     ----------------
c     Input Variables
c     ----------------
c
c     NDP   = # OF DATA POINTS    1 <= i <+ NDP
c     XD(i) = independent variable
c     YD(i) = dependent variable = Var(XD(i))
c     NP    = Order of desired polynomial L.S. fit
c
c
c     calculate normal equations from XD(i)

      if(NP.lt.2) then
        print 10, NP
        stop
      elseif(NP.gt.npm) then
        print 11, NP,npm
        stop
      elseif(NDP.le.NP) then
        print 12, NDP,NP
        stop
      endif

      do i = 1,NP
        do j = 1,NP
          AR(i,j) = zero
        enddo
      enddo

      do 100 k = 1,NDP
        if(WGT(k).eq.0.0) WGT(k) = 1.0
        do i = 1,NP
          if(i.eq.1) then
```

```
            Xi = one
          else
            Xi = Xi*dble(XD(k))
          endif
          do j = 1,i
            if(j.eq.1) then
              Xj = one
            else
              Xj = Xj*dble(XD(k))
            endif
            AR(i,j) = AR(i,j) + Xi*Xj*dble(WGT(k))
          enddo
        enddo
  100 continue

      do i = 1,NP
        do j = 1,i
          AR(j,i) = AR(i,j)  ! symmetric
        enddo
      enddo

c     ------------------------------------------------
c     calculate left side of normal equation
c
c     AL(I)=sum(X(I,i)*YD(i)), i=1 to NDP
c     ------------------------------------------------

      do i = 1,NP
        AL(I) = zero
      enddo

      do 200 k = 1,NDP
        do i = 1,NP
          if(i.eq.1) then
            Xi = one
          else
            Xi = Xi*dble(XD(k))
          endif
          AL(i) = AL(I) + Xi*dble(YD(k))*dble(WGT(k))
        enddo
  200 continue

      call mx_lsqr(NP,AL,AR,Param,Qerr)

      sum1 = zero
      sum2 = zero

      do k = 1,NDP
        YDcalc = zero
        do i = 1,NP
          if(i.eq.1) then
            Xi = one
          else
```

```
            Xi = Xi*dble(XD(k))
          endif
          YDcalc = YDcalc + Param(i)*Xi
        enddo
        Var = YDcalc - dble(YD(k))
        sum1 = sum1 + Var
        sum2 = sum2 + Var*Var*dble(WGT(k))
      enddo

c     Calc. error associated with PAR(i) for poly.

      chi2 = sngl(sum2/dfloat(NDP-NP))
      do i = 1,NP
        PAR(i) = sngl(Param(i))
        do j = 1,NP
          var = dsqrt(sum2*dabs(Qerr(i,j))/dfloat(NDP-NP))
          Perr(i,j) = sngl(var)
        enddo
      enddo

      return
   10 format(/,5x,' *** poly(), NP=',i3,' < 2 ***')
   11 format(/,5x,' *** poly(), NP=',i3,' >',i3,' ***')
   12 format(/,5x,' *** poly(), NDP=',i4,'<=  NP=',i2,' ***')
      end
```

B.15 RAN3: a uniform deviate random number generator (IDL)

NOTE: A common block is used to for some local variables that must be saved between calls to this routine.

```
      ; input:
      ;    idum     LONG        used as SEED
      ; I/O:
      ;    MA()     fltarr(57)  generator common block

      ;ALGORITHM REFERENCES:  None
      ;  Numerical Recipes for FORTRAN,  pg. 199
      ;  Press, , Falnnery, Teukolsky, Vetterling, 1986
      ;
      ;  Knuth, D.E., 1981 Seminumerical algorithms.  2nd ed.
      ;  vol.2 of The Art of Computed Programming (Reading, Mass,
      ;  Addison-Wesly).  sections 3.2-3.3.
      ;
      ;KNOWN BUGS AND LIMITATIONS:  None
      ;
      ;ROUTINE HISTORY:
      ;   Date      Programmer      Description
      ;   12/98     C.Barnet        original code
      ;
      ;END=============================================================
      ;
      ;
      ;================================================================
```

```
function ran3, ix
common rad3blk, INEXT, INEXTP, MA

  MBIG=1000000000L
  MSEED=161803398L
  MZ=0L
  FAC=1.0/float(MBIG)

  idum = LONG(ix)

  NMA=55L
  I30=30L
  if(IDUM gt 0) then begin
    INEXT = 0L     ; prepare indices for our first generated number
    INEXTP = 31L   ; the constant is special, see Knuth.
    MA = fltarr(NMA+1)   ; MA(0) is not used

    MJ = MSEED - IDUM
    MJ = MJ MOD MBIG
    MA(NMA) = MJ
    MK = 1
    for I = 1,NMA-1 do begin ; initialize table with numbers
      II = (21L*I) MOD NMA   ; that are not especially random
      MA(II) = MK
      MK = MJ - MK
      if(MK lt MZ) then MK = MK + MBIG
      MJ = MA(II)
    endfor
    for K = 1, 4 do begin    ; randomize by "warming up the generator"
      for I = 1, NMA do begin
        II = 1L + ((I+30L) MOD NMA)
        MA(I) = MA(I) - MA(II)
        if(MA(I) lt MZ) then MA(I) = MA(I) + MBIG
      endfor
    endfor
  endif

  INEXT = INEXT + 1L
  if(INEXT gt NMA) then INEXT = 1L
  INEXTP = INEXTP + 1L
  if(INEXTP gt NMA) then INEXTP = 1L
  MJ = MA(INEXT) - MA(INEXTP)       ; new random #, subtractively
  IF(MJ lt MZ) then MJ = MJ + MBIG  ; make sure it is in range
  MA(INEXT) = MJ                ; store it

; output derived uniform deviate
  ran0 = MJ*FAC
  return, ran0
end
```

B.16 REAL_FFT: perform a "real" FFT using complex routine (FORTRAN)

```
      subroutine real_fft(Npts, fx, IDIR, gx)
      implicit none

      integer*4 Npts   ! number of points for DFT
      real*4    fx(*)  ! input function
      integer*4 IDIR   ! forward FFT, IDIR < 0
c                        reverse FFT, IDIR > 0
      complex*8 gx(*)  ! output function

c     local variables
c     ---------------
      integer*4 IDIRLOC
      integer*4 k, n, M, M2
      real*4    pi
      parameter (pi=3.14159265)
      real*4    invflg, scl
      real*4    Rk1,Rk2,Ik1,Ik2   ! components of Y(k1) and Y(k2)
      real*4    w0, W
      real*4    Cw, Sw
      integer*4 k1, k2
      real*4    R1, R2, I1, I2
      real*4    Xr, Xi

      IDIRLOC = IDIR
      if(IDIR.le.0) IDIRLOC = -1   ! forward FFT
      if(IDIR.gt.0) IDIRLOC = +1   ! inverse FFT

      M = Npts/2     ! # of points in 1/2 the data
      M2 = M/2       ! # of points in 1/4 the data (to do it in-place)

      if(IDIRLOC.gt.0) then
        invflg = -1.0  ! negate terms     ! inverse FFT
        scl = 0.25     ! inverse has 1/N which requires extra 1/2
      else
        invflg = 1.0   ! assume forward DFT
        scl = 0.50     ! forward is NOT scaled
      endif            ! 1/2 for Even/Odd determination

      if(2*M.ne.Npts .or. 4*M2.ne.Npts) then
        print *, 'real_fft: 4*M2 or 2*M <> Npts, ', 4*M2, 2*M, Npts
        stop
      endif

      do n = 1, M
c       "even" index for f(0:M-1) is 0,2,4,6,...,Npts-2
c       "odd" index for f(0:M-1) is 1,3,5,6,...,Npts-1
c       for f(1:M) we add one
        k1 = 2*n - 1     ! 1,3,5,7,9,...,Npts-1
        k2 = k1 + 1
        gx(n) = CMPLX(fx(k1),fx(k2))
```

```
      enddo

c     in-place - temporarily use the output array for
c     intermediate FFT
      call cfft(M, gx, IDIRLOC)

!     DO the 1st and Mth point separately, since Y(N)=Y(0)
!
!     cw = 1.0                 ! COS(2*pi*0/N) = 1
!     sw = 0.0                 ! SIN(2*pi*0/N) = 0
      R1 = scl*2.0*REAL(gx(1)) ! (R(N)+R(0))/2 = R(0)
!     R2 = 0.0                 ! (R(N)-R(0))/2 = 0
!     I1 = 0.0                 ! (I(N)-I(0))/2 = 0
      I2 = scl*2.0*AIMAG(gx(1))! (I(N)+I(0))/2 = I(0)
      Xr = R1 + I2             ! R1 + sw*R2 + cw*I2
      Xi = 0.0                 ! cw*R2 - I1 - sw*I2
      gx(1) = CMPLX(Xr,Xi)
      Xr = R1 - I2             ! R1 - sw*R2 - cw*I2
      Xi = 0.0                 ! sw*I2 - cw*R2 - I1
      gx(M+1) = CMPLX(Xr,Xi)
c
c     NOW separate even and odd functions and do FINAL FFT pass
      w0 = invflg*pi/float(M)  ! 2*pi/Npts
c     in-place method: we can do k and M-k operation at the
c     same time noting that:
c       cos(pi*k/M) = -cos(pi*(M-k)/M)
c       sin(pi*k/M) = +sin(pi*(M-k)/M)
      do k = 1, M2
        w = w0*float(k)
        sw = SIN(w)
        cw = COS(w)   ! for M-k this is negated
        k1 = k + 1    ! 2,3,4,...,M/2+1
        k2 = M-k + 1  ! M,M-1,...,M/2+1 (M/2 done twice, but its equal)
        RK1 = REAL(gx(k1))
        RK2 = REAL(gx(k2))
        IK1 = AIMAG(gx(k1))
        IK2 = AIMAG(gx(k2))
        R1 = scl*(RK2 + RK1)
        R2 = scl*(RK2 - RK1)  ! for M-k this is negated
        I1 = scl*(IK2 - Ik1)  ! for M-k this is negated
        I2 = scl*(Ik2 + Ik1)
        Xr = R1 + sw*R2 + cw*I2
        Xi = cw*R2 - I1 - sw*I2
        gx(k1) = CMPLX(Xr,Xi)
        Xr = R1 - sw*R2 - cw*I2
        Xi = I1 + cw*R2 - sw*I2
        gx(k2) = CMPLX(Xr,Xi)    ! in-place part k=(M-1)
        Xr = R1 - sw*R2 - cw*I2
        Xi = sw*I2 - cw*R2 - I1
        gx(M+k1) = CMPLX(Xr,Xi)
        Xr = R1 + sw*R2 + cw*I2
        Xi = sw*I2 - cw*R2 + I1
        gx(M+k2) = CMPLX(Xr,Xi)  ! in-place part k=(M-1)
```

```
      enddo

      return
      end
```

B.17 SIMPSON: definite integral via Simpson's rule (FORTRAN)

```
      function simpson(npts,x0,x1,ydata)
      integer*2 i,npts
      real*4    simpson,x0,x1,ydata(*)
      real*8    h,sum,zero,two,three,four
      parameter (zero=0.0d00,two=2.0d00,three=3.0d00,four=4.0d00)

c     calculates integral of npts evenly spaced function ydata(i)
c     where  1 <= i <= npts, and npts is an odd integer
c     using simpsons integral approximation

      i = npts/2
      if(npts.ne.2*i+1.or.npts.lt.5) then
        stop ' *** simpson : number of points is not odd ***'
      endif

      sum = zero
        h = (dble(x1)-dble(x0))/DFLOAT(npts-1)
      if(h.eq.zero) goto 200

      sum = dble(ydata(1)) + dble(ydata(npts))

      i = npts
  100 i = i - 1
      sum = sum + four*dble(ydata(i))
      i = i - 1
      if(i.eq.1) goto 200
      sum = sum + two*dble(ydata(i))
      goto 100

  200 sum = sum*h/three

      simpson = sngl(sum)
      return
      end
```

B.18 SPLINE, SPLINT, SPINE_int: cubic spline routines (FORTRAN)

In the derivation for the cubic spine in Section 4.10.1 we solved a cubic equation for each interval, j, using points sampled at x_j with value $f(x_j)$. Eqn 4.59 from that section is reproduced here.

$$S_j(x) = a_j + b_j \cdot (x - x_j) + c_j \cdot (x - x_j)^2 + d_j \cdot (x - x_j)^3 \qquad \text{for} \quad j = 1, n-1 \tag{B.4}$$

and the solution could be expresed in terms of a tridiagonal solution in terms of the c_j coefficients for

each interval, j, in out input data, (x(j), y(j), j=1,n). For the $n = 5$ case we derived the tridiagonal form as follows:

$$\begin{bmatrix} 2h_1 & h_1 & 0 & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & 0 \\ 0 & 0 & h_3 & 2(h_3+h_4) & h_4 \\ 0 & 0 & 0 & h_4 & 2h_4 \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix} = \begin{bmatrix} 3(a_2-a_1)/h_1 - 3f'(x_1) \\ 3(a_3-a_2)/h_2 - 3(a_2-a_1)/h_1 \\ 3(a_4-a_3)/h_3 - 3(a_3-a_2)/h_2 \\ 3(a_5-a_4)/h_4 - 3(a_4-a_3)/h_3 \\ 3f'(x_n) - 3(a_5-a_4)/h_4 \end{bmatrix} \tag{B.5}$$

and using the following relationships the coefficients for the cubic polynomials can be solved for

$$h_j \equiv (x_{j+1} - x_j) \quad \text{for} \quad j = 1, n-1 \tag{B.6}$$

$$a_j = f(x_j) = y(j) \quad \text{for} \quad j = 1, n \tag{B.7}$$

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2 \cdot c_j + c_{j+1}) \quad \text{for} \quad j = 1, n-1 \tag{B.8}$$

$$d_j = \frac{1}{3 \cdot h_j}(c_{j+1} - c_j) \quad \text{for} \quad j = 1, n-1 \tag{B.9}$$

The numerical recipes code solves for $S''_j(x_j) = 2 \cdot c_n$ instead of c_n so we need to rewrite our equation. In addition, we can divide through by the diagnonal component to obtain

$$\begin{bmatrix} 2 & 1 & 0 & 0 & 0 \\ \frac{h_1}{h_1+h_2} & 2 & \frac{h_2}{h_1+h_2} & 0 & 0 \\ 0 & \frac{h_2}{h_2+h_3} & 2 & \frac{h_3}{h_2+h_3} & 0 \\ 0 & 0 & \frac{h_3}{h_3+h_4} & 2 & \frac{h_4}{h_3+h_4} \\ 0 & 0 & 0 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} S''_1 \\ S''_2 \\ S''_3 \\ S''_4 \\ S''_5 \end{bmatrix} = \begin{bmatrix} \frac{6(a_2-a_1)/h_1 - 6f'(x_1)}{h_1} \\ \frac{6(a_3-a_2)/h_2 - 6(a_2-a_1)/h_1}{h_1+h_2} \\ \frac{6(a_4-a_3)/h_3 - 6(a_3-a_2)/h_2}{h_2+h_3} \\ \frac{6(a_5-a_4)/h_4 - 6(a_4-a_3)/h_3}{h_3+h_4} \\ \frac{6f'(x_n) - 6(a_5-a_4)/h_4}{h_4} \end{bmatrix} \tag{B.10}$$

and similarly for the *natural spline* we have

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{h_1}{h_1+h_2} & 2 & \frac{h_2}{h_1+h_2} & 0 & 0 \\ 0 & \frac{h_2}{h_2+h_3} & 2 & \frac{h_3}{h_2+h_3} & 0 \\ 0 & 0 & \frac{h_3}{h_3+h_4} & 2 & \frac{h_4}{h_3+h_4} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} S''_1 \\ S''_2 \\ S''_3 \\ S''_4 \\ S''_5 \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{6(a_3-a_2)/h_2 - 6(a_2-a_1)/h_1}{h_1+h_2} \\ \frac{6(a_4-a_3)/h_3 - 6(a_3-a_2)/h_2}{h_2+h_3} \\ \frac{6(a_5-a_4)/h_4 - 6(a_4-a_3)/h_3}{h_3+h_4} \\ 0 \end{bmatrix} \tag{B.11}$$

Then we could employ the tridiagonal solution (see TRIDAG.FOR, Section B.21); however, this can be done in-place. We end up solving for $S''_j = 2 \cdot c_j$. Here is the numerical recipes routine (with my comments added). The input is x(j) = x_j, y(j), yp1=$f'(x_1)$, ypn=$f'(x_n)$. The output is y2(j) = $S''(x_j) = 2 \cdot c_j$.

```
SUBROUTINE spline(x,y,n,yp1,ypn,y2)
INTEGER n,NMAX            ! n = # of input data points
REAL x(n),y(n)            ! input: data  y(x(n))
REAL yp1,ypn              ! input: boundary conditions
REAL y2(n)                ! output: second derivative at x(n)
PARAMETER (NMAX=500)      ! NMAX >= n
INTEGER i,k               ! local
REAL p,qn,sig,un,u(NMAX)  ! local
if (yp1.gt..99e30) then   ! natural spline, d^y/dx^2 = 0 at x(1)
  y2(1)=0.
  u(1)=0.
```

```
      else                        ! set yp1 = dy/dx at x(1)
        y2(1)=-0.5
        u(1)=(3./(x(2)-x(1)))*((y(2)-y(1))/(x(2)-x(1))-yp1)
      endif
      do 11 i=2,n-1                          ! TRIDAG terms
        sig=(x(i)-x(i-1))/(x(i+1)-x(i-1))  ! h(i-1)/(h(i)+h(i-1)) = a_j term
        p=sig*y2(i-1)+2.                     ! beta(i) term
        y2(i)=(sig-1.)/p                     ! y2(i)=gamma(i+1) term
        u(i)=(6.*((y(i+1)-y(i))/(x(i+1)-x(i))
     &          - (y(i)-y(i-1))/(x(i)-x(i-1)) )  ! g(i) term
     &         / (x(i+1)-x(i-1))-sig*u(i-1))/p
   11 continue
      if (ypn.gt..99e30) then   ! natural spline, d^y/dx^2 = 0 at x(n)
        qn=0.
        un=0.
      else                        ! set ypn = dy/dx at x(n)
        qn=0.5
        un=(3./(x(n)-x(n-1)))*(ypn-(y(n)-y(n-1))/(x(n)-x(n-1)))  ! g(n)
      endif
      y2(n)=(un-qn*u(n-1))/(qn*y2(n-1)+1.)        ! TRIDAG: gamma(n)
      do 12 k=n-1,1,-1
        y2(k)=y2(k)*y2(k+1)+u(k)   ! TRIDAG: u(k) = gamma(k+1)*u(k+1) + g(k)
   12 continue
      return
      END
C  (C) Copr. 1986-92 Numerical Recipes Software
```

using the cubic splines to interpolate a function

The array y2(j) $= S''_j = 2 \cdot c_j$ can be used to re-derive the coefficients for our polynomial. Substitution of Eqn. B.6 to Eqn. B.9 into Eqn. B.4 would yield

$$S_j(x) = a_j + b_j \cdot (x - x_j) + c_j \cdot (x - x_j)^2 + d_j \cdot (x - x_j)^3 \qquad \text{for} \quad j = 1, n-1 \tag{B.12}$$

$$S_j(x) = y(j) + \left[\frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2 \cdot c_j + c_{j+1})\right] \cdot (x - x_j) + c_j \cdot (x - x_j)^2 + \left[\frac{1}{3 \cdot h_j}(c_{j+1} - c_j)\right] \cdot (x - x_j)^3 \qquad \text{for} \quad j = 1, n-1 \tag{B.13}$$

If we define

$$B_j \equiv \frac{x - x_j}{x_{j+1} - x_j} = \frac{x - x_j}{h_j} \qquad \textbf{and} \tag{B.14}$$

$$A_j \equiv \frac{x_{j+1} - x}{x_{j+1} - x_j} = \frac{h_j - x - x_j}{h_j} = 1 - B_j \tag{B.15}$$

$$S_j(x) = y(j) + B_j \cdot (y(j+1) - y(j)) - [2 \cdot c_j + c_{j+1}] \cdot B_j \cdot \frac{h_j^2}{3}$$

$$+ \quad c_j \cdot B_j^2 \cdot h_j^2 \quad + \quad \left[\frac{1}{3}\left(c_{j+1} - c_j\right)\right] \cdot B_j^3 \cdot h_j^2 \tag{B.16}$$

$$S_j(x) = y(j) \cdot (1 - B_j) + y(j+1) \cdot B_j \quad + \quad \frac{2 \cdot c_j}{6} \cdot h_j^2 \cdot \left[-2B_j + 3 \cdot B_j^2 - B_j^3\right] + \quad \frac{2 \cdot c_{j+1}}{6} \cdot h_j^2 \cdot \left[-B_j + B_j^3\right] \tag{B.17}$$

$$S_j(x) = y(j) \cdot A_j + y(j+1) \cdot B_j + \frac{2 \cdot c_j}{6} \cdot \left[A_j^3 - A_j\right] + \frac{2 \cdot c_{j+1}}{6} \cdot h_j^2 \cdot \left[B_j^3 - B_j\right] \tag{B.18}$$

$$S_j(x) = y(j) \cdot A_j \quad + \quad y(j+1) \cdot B_j + \quad \frac{h_j^2}{6} \cdot \left[2 \cdot c_j \cdot \left(A_j^3 - A_j\right) \quad + \quad 2 \cdot c_{j+1} \cdot \left(B_j^3 - B_j\right)\right] \qquad \text{for} \quad j = 1, n-1 \tag{B.19}$$

Which is the form used in the following numerical recipes routine

```
      SUBROUTINE splint(xa,ya,y2a,n,x,y)
      INTEGER n            ! original number of data points
      REAL xa(n),ya(n)     ! original data (x_j, y(x_j)) used in spline() fit
      REAL y2a(n)          ! second derivative computed at xa(n), from spline()
      REAL x,y             ! pair of interpolated values, y(x)
      INTEGER k,khi,klo    ! local variables to determine index
      REAL a,b,h           ! local variables

c     determine index for cubic function to use
c     in interpolation/extrapolation of y(x).
      klo=1
      khi=n
   1  if (khi-klo.gt.1) then
        k=(khi+klo)/2
        if(xa(k).gt.x)then
          khi=k
        else
          klo=k
        endif
      goto 1
      endif
      h = xa(khi)-xa(klo)                  ! h_j in notes  j=klo  khi=j+1
      if (h.eq.0.) pause 'bad xa input in splint'
      A = (xa(khi)-x)/h                                    ! A_j in notes
      B = (x-xa(klo))/h                                    ! B_j in notes
      y = A*ya(klo)  +  B*ya(khi)  +  ((A**3-A)*y2a(klo) ! S_j(x) in notes
     &     +  (B**3-B)*y2a(khi))*(h**2)/6.
      return
      END
C  (C) Copr. 1986-92 Numerical Recipes Software
```

Computing the integral of a spline

$$\int_0^{h_j} S_j(x' + x_j)\, dx' = a_j \cdot h_j + \frac{b_j}{2} \cdot h_j^2 + \frac{c_j}{3} \cdot h_j^3 + \frac{d_j}{4} \cdot h_j^4 \qquad \text{for} \quad j = 1, n-1 \tag{B.20}$$

$$\int_{x_1}^{x_n} \sum_{j-1}^{n-1} S_j\left(x' + x_j\right) dx' = \sum_{j=1}^{n-1} a_j \cdot h_j + \frac{b_j}{2} \cdot h_j^2 + \frac{c_j}{3} \cdot h_j^3 + \frac{d_j}{4} \cdot h_j^4 \qquad \text{for} \quad j = 1, n-1 \tag{B.21}$$

```
      SUBROUTINE spline_int(xa,ya,y2a, yint)
      INTEGER*4 n             ! original number of data points
      REAL*4 xa(n),ya(n)      ! original data (x_j, y(x_j)) used in spline() fit
      REAL*4 y2a(n)           ! second derivative computed at xa(n), from spline()
      REAL*4 yint             ! integral of y(x) dx from x(1) to x(n)
      INTEGER*4 j             ! local variables to determine index
      REAL*8 a,b,c,d,h,sum    ! local variables

c     determine index for cubic function to use
c     in interpolation/extrapolation of y(x).

      sum = 0.0d00
      do j = 1, n-1
        h = DBLE(xa(j+1))-DBLE(xa(j))         ! h_j in notes
        a = DBLE(ya(j))
        b = DBLE(y(j+1)-y(j))/h - h*DBLE(y2a(j)+y2a(j+1)/2.0)/3.0d00
        c = DBLE(y2a(j))/2.0d00
        d = DBLE(y2a(j+1)-y2a(j))/(6.0d00*h)
        sum = sum + (((d*h/4.0d00 + c/3.0d00)*h + b/2.0d00)*h + a)*h
      enddo
      return
      END
```

B.19 SUN_DISTANCE: compute solar distance (FORTRAN)

```
!ALGORITHM REFERENCES:
C
C     primary reference:
C     ------------------
C     Bretagnon, P. 1986.  Planetary programs and tables from
C     -4000 to +2800: tables for the motion of the sun and the
c     five bright planets. Richmond: Willmann-Bell.
C
C     AENA = The Astronomical Almanac, 1994
C
!KNOWN BUGS AND LIMITATIONS:
C
C     Corrections for topography, Lunar nutation, and terrestrial dynamic
C     time (TDT or ephemeris time, ET) have NOT been made.
C
C     The uncorrected geo-centric distance is more than adequate for
C     the computation of the solar radiance outside of the Earth's
C     atmosphere.
C
C     The error in solar longitude and distance is such that geo-centric
C     computations of the position of the sun are within a few arcsec's
```

```
C     over the published range of years  (-4000 to +2800).
C
C     The errors in AU are on the order of 0.000003 AU = 450 km
C     Topocentric Errors are +/- 7000 km = 0.000047 AU
C                                          -----------
C     total error in AU is about           0.00005  AU
C
C     annual signal  +/- 0.017 AU   or   +/- 3.4% of the solar radiance
C     AU error = 0.3% of the variation  or  0.01% of the solar radiance
C     which is well within the certainty of the solar radiance
C


      subroutine sun_distance(TDT,slon,AU)
      implicit none
      real*8     TDT
      real*4     slon, AU

      integer*2  i
      integer*4  IL(50),IR(50)
      real*8     AL(50),BL(50)
      real*8     r0,arg
      real*8     PI,zero,pi2,y100,j2000,U
      real*8     DL,DB,DR
      parameter (PI=3.141592653589793d00)
      parameter (zero=0.0d00,pi2=2.0d00*PI)
      parameter (j2000=0.2451545d07)   ! Jan. 1, 2000 at 12h UT
      parameter (y100=0.36525d07)      ! 10000 years

c     IL = AMPLITUDES OF THE PERIODIC TERMS OF THE LONGITUDE
c     IR = AMPLITUDES OF THE PERIODIC TERMS OF THE RADIUS VECTOR
c     AL = PHASES OF THE PERIODIC TERMS
c     BL = FREQUENCIES OF THE PERIODIC TERMS

      DATA IL/ 403406,195207,119433,112392,3891,2819,1721,0,660,350,334,
     2 314,268,242,234,158,132,129,114,99,93,86,78,72,68,64,46,38,37,32,
     3 29,28,27,27,25,24,21,21,20,18,17,14,13,13,13,12,10,10,10,10/
      DATA AL/ 4.721964D0,5.937458D0,1.115589D0,5.781616D0,5.5474D0,
     2 1.512D0,4.1897D0,1.163D0,5.415D0,4.315D0,4.553D0,5.198D0,5.989D0,
     3 2.911D0,1.423D0,0.061D0,2.317D0,3.193D0,2.828D0,0.52D0,4.65D0,
     4 4.35D0,2.75D0,4.50D0,3.23D0,1.22D0,0.14D0,3.44D0,4.37D0,1.14D0,
     5 2.84D0,5.96D0,5.09D0,1.72D0,2.56D0,1.92D0,0.09D0,5.98D0,4.03D0,
     6 4.27D0,0.79D0,4.24D0,2.01D0,2.65D0,4.98D0,0.93D0,2.21D0,
     9 3.59D0,1.50D0,2.55D0/
      DATA BL/ 1.621043D0,62830.348067D0,62830.821524D0,62829.634302D0,
     2 125660.5691D0,125660.9845D0,62832.4766D0,0.813D0,
     3 125659.310D0,57533.850D0,-33.931D0,777137.715D0,78604.191D0,
     4 5.412D0,39302.098D0,-34.861D0,115067.698D0,15774.337D0,
     5 5296.670D0,58849.27D0,5296.11D0,-3980.70D0,52237.69D0,
     6 55076.47D0,261.08D0,15773.85D0,188491.03D0,-7756.55D0,
     7 264.89D0,117906.27D0,55075.75D0,-7961.39D0,188489.81D0,
     8 2132.19D0,109771.03D0,54868.56D0,25443.93D0,-55731.43D0,
     9 60697.74D0,2132.79D0,109771.63D0,-7752.82D0,188491.91D0,
     . 207.81D0,29424.63D0,-7.99D0,46941.14D0,-68.29D0,
```

```
     1 21463.25D0,157208.40D0/
      DATA IR/ 0,-97597,-59715,-56188,-1556,-1126,-861,941,-264,-163,0,
     2 309,-158,0,-54,0,-93,-20,0,-47,0,0,-33,-32,0,-10,-16,0,0,-24,-13,
     3 0,-9,0,-17,-11,0,31,-10,0,-12,0,-5,0,0,0,0,0,0,-9/

c     TDT = terrestrial dynamic time (Julian day + UTZ/24 + correct)
c          where correct = tabulated correction  AENA pg. K8-K9
c          e.g., 1/9/1992  at 4:55:01 UTZ   1992 correction = 58.31s
c          2.4486305d06 + 0.204873d00 + 58.31d00/3.6d03/2.4d01
c            = 2448630.705548 days

      U=(TDT-j2000)/y100

      DL = zero
      DB = zero   ! zero latitude of SUN
      DR = zero
      do I=50,1,-1
        r0 = DBLE(IL(i))
        arg = AL(I)+BL(I)*U
        DL = DL + r0*DSIN(arg)
        r0 = DBLE(IR(i))
        DR = DR + r0*DCOS(arg)
      enddo
      DL = DL*1.D-7 + 4.9353929D0 + 62833.1961680D0*U
      DL = DMOD(DL,PI2)
      IF(DL.LT.zero) DL = DL + PI2
      DR = DR*1.D-7 + 1.0001026D00

c      DR = solar ecliptic longitude in degrees  (mean ecliptic of date)
c      DB = solar ecliptic latitude in degrees   (mean ecliptic of date)
c      DR = solar geocentric distance in AU (1 AU = 1.49597870E+13 cm)

      slon = SNGL(DL)
      AU   = SNGL(DR)

      RETURN
      END
```

B.20 TRED2: Householder reduction of a real symmetric matrix (FORTRAN)

```
      SUBROUTINE TRED2(A,N,NP,D,E)
      implicit none

c*********************************************************************
C.! ROUTINE: TRED2
C.
C.! PURPOSE: Householder reduction of a real, symmetric, matrix
C.!          reduced to a real, symmetric tri-diagonal matrix.
C.!          all arrays dimensioned to NP
C.!          input:
```

```
C.!            A(N,N) is a real symmetric matrix
C.!          output:
C.!            D(N) is the diagaonal component of the tri-diagonal matrix
C.!            E(N) are the sub-diagonal components (E(0) is arbitrary).
C.!
C.!          see pg. 355 w/ explanation pgs.335-376
C.!          Numerical Recipes: The Art of Scientific Programming
C.!          (FORTRAN version), 1st edition
C.!          W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling
C.!          Cambridge Univ. Press., 1986
C.!
c**********************************************************************
      integer*4 N, NP
      real*8    A(NP,NP),D(NP),E(NP)

      integer*4 I,J,K,L
      real*8    F, G, H, HH, SCALE, zero, one
      parameter (zero=0.0d00, one=1.0d00)

      IF(N.GT.1)THEN
        DO 18 I=N,2,-1
          L = I-1
          H = zero
          SCALE = zero
          IF(L.GT.1)THEN
            DO K = 1,L
              SCALE = SCALE + DABS(A(I,K))
            enddo
            IF(SCALE.EQ.zero) THEN ! Skip transformation if row is zero
              E(I) = A(I,L)
            ELSE
              DO K = 1,L
                A(I,K) = A(I,K)/SCALE ! Use scales a' for transformation
                H=H+A(I,K)**2       ! Form sigma in H
              enddo
              F = A(I,L)
              G = -DSIGN(DSQRT(H),F)
              E(I) = SCALE*G
              H = H - F*G           ! Now H is equation 11.2.4
              A(I,L) = F - G        ! Store u in the Ith row of A
              F = zero
              DO 15 J = 1,L
                A(J,I) = A(I,J)/H   ! Store u/H in Ith column of A
                G = zero            ! Form an element of A*u in G.
                DO K = 1,J
                  G = G + A(J,K)*A(I,K)
                enddo
                IF(L.GT.J)THEN
                  DO K = J+1,L
                    G = G + A(K,J)*A(I,K)
                  enddo
                ENDIF
                E(J) = G/H          ! Form element of p in temporarily
```

```
              F = F + E(J)*A(I,J) ! unused element of E.
15          CONTINUE
            HH=F/(H+H)            ! Form K, equation 11.2.11
            DO J = 1,L            ! Form q and store in E overwriting p.
              F = A(I,J)          ! NOTE that E(L)-E(I-1) survives
              G = E(J)-HH*F
              E(J) = G
              DO K = 1,J          ! Reduce A, equation 12.2.13
                A(J,K) = A(J,K) - F*E(K) - G*A(I,K)
              enddo
             enddo
          ENDIF
        ELSE
          E(I) = A(I,L)
        ENDIF
        D(I) = H
18    CONTINUE
    ENDIF
    D(1) = zero
    E(1) = zero
    DO 23 I = 1,N ! Begin accumulation of transformation matrices
      L = I-1
      IF(D(I).NE.zero)THEN ! This block is skipped when I=1
        DO J = 1,L
          G = zero
          DO K = 1,L      ! Use u and u/H stored in A to form P*Q
            G = G + A(I,K)*A(K,J)
          enddo
          DO K = 1,L
            A(K,J) = A(K,J) - G*A(K,I)
          enddo
        enddo
      ENDIF
      D(I) = A(I,I)
      A(I,I) = one           ! reset fow and column of A to identity
      IF(L.GE.1)THEN         !  matrix for next iteration
        DO J=1,L
          A(I,J) = zero
          A(J,I) = zero
        enddo
      ENDIF
23  CONTINUE

    RETURN
    END
```

B.21 TRIDAG: Solve tridiagonal form of simultaneous equations (FORTRAN)

A tridiagonal matrix is a matrix of the following form

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & 0 \\ 0 & 0 & a_4 & b_4 & c_4 \\ 0 & 0 & 0 & a_5 & b_5 \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \end{bmatrix} \tag{B.22}$$

This form can be easily solved for u_j in two passes. First note that the first equation can be written

$$b_1 \cdot u_1 + c_1 \cdot u_2 = r_1 \tag{B.23}$$

which can be rewritten in terms of known items and a function of the solution to equation 2,

$$u_1 = \frac{r_1}{b_1} - \frac{c_1}{b_1} \cdot u_2 \quad \text{or} \tag{B.24}$$

$$u_1 = g_1 - \gamma_2 \cdot u_2 \tag{B.25}$$

which can be decomposed into

$$g_1 = \frac{r_1}{\beta_1} \qquad \beta_1 = b_1 \tag{B.26}$$

$$\gamma_2 = \frac{c_1}{\beta_1} \tag{B.27}$$

The second equation is given by

$$a_2 \cdot u_1 + b_2 \cdot u_2 + c_2 \cdot u_3 = r_2 \tag{B.28}$$

We can substitute the expression for u_1 and reorganized

$$a_2 \cdot (g_1 - \gamma_2 \cdot u_2) + b_2 \cdot u_2 + c_2 \cdot u_3 = r_2 \tag{B.29}$$

$$(b_2 - a_2 \cdot \gamma_2) \cdot u_2 + c_2 \cdot u_3 = r_2 - a_2 \cdot g_1 \tag{B.30}$$

which can be decomposed into two functions.

$$\beta_2 = b_2 - a_2 \cdot \gamma_2 \tag{B.31}$$

$$g_2 = \frac{r_2 - a_2 \cdot g_1}{\beta_2} \tag{B.32}$$

$$\gamma_3 = \frac{c_2}{\beta_2} \tag{B.33}$$

$$u_2 = g_2 - \gamma_3 \cdot u_3 \tag{B.34}$$

In general, we can write

$$a_1 \equiv 0 \qquad \textbf{and} \qquad g_0 \equiv 0 \tag{B.35}$$

$$\beta_j = b_j - a_j \cdot \gamma_j \quad \text{for} \quad j = 1, n-1 \tag{B.36}$$

$$\gamma_j = \frac{c_{j-1}}{\beta_{j-1}} \qquad \text{for} \quad j = 2, n \tag{B.37}$$

$$g_j = \frac{r_j - a_j \cdot g_{j-1}}{\beta_j} \quad \text{for} \quad j = 1, n-1 \tag{B.38}$$

$$\tag{B.39}$$

The last equation is written

$$a_n \cdot u_{n-1} + b_n \cdot u_n = r_n \tag{B.40}$$

$$a_n \cdot (g_{n-1} - \gamma_n \cdot u_n) + b_n \cdot u_n = r_n \tag{B.41}$$

can be solved for without any forward reference

$$u_n = g_n = \frac{r_n - a_n \cdot g_{n-1}}{b_n - a_n \cdot \gamma_n} \tag{B.42}$$

and, finally, the g_j's and γ_j's can be used to solve for the remaining u_j's. In this case, we work backwards through the arrays.

$$u_{j-1} = g_{j-1} - \gamma_j \cdot u_j \qquad \text{for} \quad j = n, 2 \tag{B.43}$$

Here is a numerical recipes subroutine, with my comments added, which performs this operation.

```
      SUBROUTINE tridag(a,b,c,r,u,n)
      INTEGER n,NMAX
      REAL a(n),b(n),c(n),r(n),u(n)
      PARAMETER (NMAX=500)
      INTEGER j
      REAL bet,gam(NMAX)
      if(b(1).eq.0.)pause 'tridag: rewrite equations'
      bet=b(1)                    ! beta(1) in notes
      u(1)=r(1)/bet               ! g(1) in notes (use output array)
      do 11 j=2,n
        gam(j)=c(j-1)/bet         ! gamma(j) = c(j=1)/beta(j-1)
        bet=b(j)-a(j)*gam(j)      ! beta(j) = b(j)-a(j)*gamma(j)
        if(bet.eq.0.)pause 'tridag failed'
        u(j)=(r(j)-a(j)*u(j-1))/bet  ! g(j) = (r(j)-a(j)*g(j-1))/beta(j)
11    continue
      do 12 j=n-1,1,-1
        u(j)=u(j)-gam(j+1)*u(j+1)    ! u(j) = g(j) - gamma(j+1)*u(j+1)
12    continue
      return
      END
C  (C) Copr. 1986-92 Numerical Recipes Software
```

B.22 TQLI: determine eigen-vectors of a tridiagonal matrix (FORTRAN)

```
      SUBROUTINE TQLI(D,E,N,NP,Z, iret) implicit none

c     input parameters
c     ----------------
      integer*4 N, NP
      real*8    D(NP),E(NP),Z(NP,NP)

c     output parameters
c     -----------------
      integer*4 iret

c*********************************************************************
C.! ROUTINE:  TQLI
```

```
C.!
C.! PURPOSE: This routine determines the the eigenvalues
C.!          and eigenvectors of a real symmetric, tridiagonal matrix.
C.!          A real symmetric matrix can be reduced to tridiagonal
C.!          form with TRED2.
C.!          all arrays dimensioned to NP
C.!          on input:
C.!            D(N) is the diagaonal component of the tri-diagonal matrix
C.!            E(N) are the sub-diagonal components (E(0) is arbitrary).
C.!          on return:
C.!            D(N)   are the eigen values
C.!            Z(N,N) are the eigen vectors  Z(N,K) correspond to D(K)
C.!
C.!          see pg. 362 w/ explanation pgs.335-376
C.!          Numerical Recipes: The Art of Scientific Programming
C.!          (FORTRAN version), 1st edition
C.!          W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling
C.!          Cambridge Univ. Press., 1986
C.!
c**********************************************************************

      integer*4 iter, I, K, L, M
      real*8    B, C, DD, F, G, P, R, S
      real*8    zero, one, two
      parameter (zero=0.0, one=1.0, two=2.0)

      IF (N.GT.1) THEN

        DO I = 2,N
          E(I-1) = E(I)   ! Renumber E() for convenience
        enddo

        E(N) = zero
        DO 15 L = 1,N
          ITER = 0
   1      DO M = L,N-1    ! Look for single small subdiagonal element
            DD = DABS(D(M)) + DABS(D(M+1)) ! to split the matrix
            IF(DABS(E(M))+DD.EQ.DD) GOTO 2
          enddo
          M = N
   2      IF(M.NE.L)THEN
            IF(ITER.EQ.30) then
              print *, 'TQLI: exceeded 30 iterations'
              iret = 10
              goto 9999
            ENDIF
            ITER = ITER+1
            G = (D(L+1)-D(L))/(two*E(L))  ! form shift
            R = DSQRT(G**2+one)
            G = D(M) - D(L) + E(L)/(G+DSIGN(R,G))  ! This is d_m - k_s
            S = one
            C = one
            P = zero
```

```
            DO 14 I=M-1,L,-1   ! A plane rotation as in the original QL,
              F = S*E(I)       ! followed by Givens rotations to restore
              B = C*E(I)       ! tridiagonal form.
              IF(DABS(F).GE.DABS(G))THEN
                C = G/F
                R = DSQRT(C**2+one)
                E(I+1) = F*R
                S = one/R
                C = C*S
              ELSE
                S = F/G
                R = DSQRT(S**2+one)
                E(I+1) = G*R
                C = one/R
                S = S*C
              ENDIF
              G = D(I+1)-P
              R = (D(I)-G)*S + two*C*B
              P = S*R
              D(I+1) = G+P
              G = C*R-B
              DO K = 1,N
                F = Z(K,I+1)             ! Form eigenvectors
                Z(K,I+1) = S*Z(K,I) + C*F
                Z(K,I)   = C*Z(K,I) - S*F
              enddo
  14          CONTINUE
              D(L) = D(L)-P
              E(L) = G
              E(M) = zero
              GO TO 1
            ENDIF
  15      CONTINUE
        ENDIF
v
        iret = 0

 9999 RETURN
        END
```

Appendix C

Derivation of Shallow Water Model: An example of solving differential equations

C.1 Vertical Momentum Equations

Assume hydrostatic equilibrium. For the shallow water model this is equivalent to assuming that the depth scale, D_s is much smaller than the length scale, L_s.

$$\frac{\partial P}{\partial z} = -\rho g_* + O(\delta^2), \quad \delta = D_s/L_s \tag{C.1}$$

The pressure at any given level z can be found by integration of the hydrostatic equation:

$$P(x, y, z, t) = -\rho g_* z + A(x, y, t) \tag{C.2}$$

C.2 Definition of Levels

lower surface:

at the base of the model there is a topography that consists of two components:

$$h_b(x, y) = h_m(x, y) + h_u(y) \tag{C.3}$$

where $h_u(y)$ is the bottom topography due to the lower level zonal wind $U_0(y)$ and $h_m(x, y)$ is the bottom topography due to a "mountain" or pressure disturbance.

middle surface:

we can treat the middle surface as a fixed equilibrium depth between the bottom and middle level, H_1, and a perturbation from that level

$$z_1(x, y, t) = H_1 + h_1(x, y, t) \tag{C.4}$$

upper surface:

$$z_2(x, y, t) = H_2 + h_2(x, y, t) \tag{C.5}$$

where, H_2 = equilibrium height of the upper level of region #2 from the middle level. At $z \geq z_2$ the pressure is constant with a value of P_3.

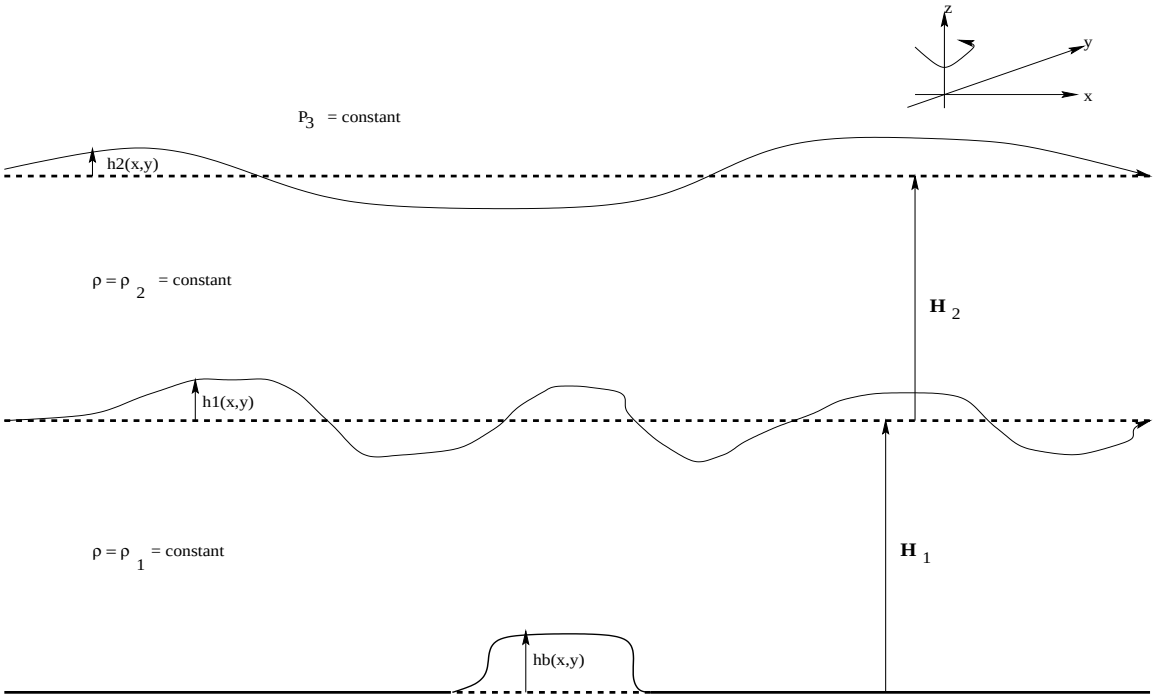


Figure C.1: Illustration of the shallow water model

Pressure in region #2 (between middle and top level)

For $z_1 \leq z \leq z_2$ the density is constant and equal to ρ_2. In this region, denoted as region #2, the pressure varies as:

$$P_2(x, y, z, t) = A_2(x, y, t) - \rho_2 g_* z \tag{C.6}$$

and at $z = z_2(x, y, t) = H_2 + h_2(x, y, t)$ the pressure is equal to P_3 and, therefore,

$$A_2(x, y, t) = P_3 + \rho_2 g_*(H_2 + h_2(x, y, t)) \tag{C.7}$$

and

$$P_2(x, y, z, t) = P_3 + \rho_2 g_*(H_2 + h_2 - z) \quad \text{for,} \quad z_1(x, y, t) \leq z \leq z_2(x, y, t) \tag{C.8}$$

region #1 (between bottom and middle level:

At the middle level z_1 the pressure must be continuous. Below z_1 the density is constant and equal to ρ_1 and the pressure can be written as:

$$P_1(x, y, t, z) = A_1(x, y, t) - \rho_1 g_* z \tag{C.9}$$

and at the interface, $z = z_1$ the pressure in the upper level must be equal in the middle level, $P_1(x, y, z_1, t) = P_2(x, y, z_1, t)$, so that:

$$A_1(x, y, t) = P_3 + \rho_2 g_*(H_2 + h_2) + (\rho_1 - \rho_2) g_* z_1 \tag{C.10}$$

but, $z_1(x, y, t) = H_1 + h_1(x, y, t)$ so that,

$$A_1(x, y, z, t) = P_3 + \rho_2 g_*(H_2 + h_2(x, y, t)) + (\rho_1 - \rho_2) g_*(H_1 + h_1(x, y, t)) \tag{C.11}$$

and over the limits of $z_0(x, y) \leq z \leq z_1(x, y, t)$ the pressure in region #1 is given as:

$$P_1(x, y, z, t) = P_0 + \rho_2 g_* h_2(x, y, t) + (\rho_1 - \rho_2) h_1(x, y, t) - \rho_1 g_* z \tag{C.12}$$

where, $P_0 \equiv P_3 + \rho_2 g_*(H_2 - H_1) + \rho_1 g_* H_1$

C.3 Continuity Equation:

In general the equation of continuity is written as:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) = 0 \tag{C.13}$$

In the shallow water model the density is assumed to be constant within each region. In region #1 $\rho \equiv \rho_1$, so that the equation of continuity is given as:

$$\frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} + \frac{\partial W}{\partial z} = 0 \tag{C.14}$$

Integration of the continuity equation yields an equation for the vertical wind at any point within region #1:

$$W(x, y, z, t,) = -z \left(\frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} \right) + \tilde{w}(x, y, t) \tag{C.15}$$

at $z = z_0(x, y, t) = h_b(x, y)$ the vertical wind is defined as:

$$W(x, y, z_2, t) = \frac{D z_0}{D t} = \frac{\partial h_b}{\partial t} + U \frac{\partial h_b}{\partial x} + V \frac{\partial h_b}{\partial y} \tag{C.16}$$

and equating the Eqn. C.16 and Eqn. C.15 evaluated at $z = z_0$ gives the constant of integration $\tilde{w}(x, y, t)$:

$$\tilde{w}(x, y, t) = \frac{\partial h_b}{\partial t} + U \frac{\partial h_b}{\partial x} + V \frac{\partial h_b}{\partial y} + h_b \left(\frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} \right) \tag{C.17}$$

The integrated continuity equation can now be written as:

$$W(x, y, z, t) = (h_b - z) \left(\frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} \right) + \frac{\partial h_b}{\partial t} + U \frac{\partial h_b}{\partial x} + V \frac{\partial h_b}{\partial y} \tag{C.18}$$

at $z = z_1(x, y, t)$ the vertical wind is defined as:

$$W(x, y, z_1, t) = \frac{D z_1}{D t} = \frac{\partial (H_1 + h_1)}{\partial t} + U \frac{\partial (H_1 + h_1)}{\partial x} + V \frac{\partial (H_1 + h_1)}{\partial y} \tag{C.19}$$

The integrated continuity equation evaluated for region #1 can be found by equating Eqn. C.19 and Eqn. C.18 evaluated at $z = z_1$. If we substitute $\varphi \equiv (H_1 + h_1) - h_b$ the equation of continuity for region #1 can be written as:

$$\frac{\partial \varphi}{\partial t} + U \frac{\partial \varphi}{\partial x} + V \frac{\partial \varphi}{\partial y} + \varphi \left(\frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} \right) = 0 \tag{C.20}$$

or equivalently,

$$\frac{d\varphi}{dt} + \varphi\left(\frac{\partial U}{\partial x} + \frac{\partial V}{\partial y}\right) = 0 \tag{C.21}$$

or equivalently,

$$\frac{\partial \varphi}{\partial t} + \frac{\partial (U\varphi)}{\partial x} + \frac{\partial (V\varphi)}{\partial y} = 0 \tag{C.22}$$

The momentum equations can also be transformed into a form similar to Dowling and Ingersoll (1989) using:

$$\frac{D\vec{v}}{Dt} \equiv \frac{\partial \vec{v}}{\partial t} + \vec{\zeta} \times \vec{v} + \nabla(\frac{\vec{v}^2}{2}), \quad \vec{\zeta} \equiv \vec{\nabla} \times \vec{v} \tag{C.23}$$

C.4 Horizontal Momentum Equations

At the middle level the horizontal momentum equations can be written as:

$$\frac{\partial U}{\partial t} + U\frac{\partial U}{\partial x} + V\frac{\partial U}{\partial y} - fV = -\frac{1}{\rho_1}\frac{\partial P_1}{\partial x} = g\frac{\partial h_1}{\partial x} + \frac{\rho_2 g_*}{\rho_1}\frac{\partial h_2}{\partial x} \tag{C.24}$$

$$\frac{\partial V}{\partial t} + U\frac{\partial V}{\partial x} + V\frac{\partial V}{\partial y} + fU = -\frac{1}{\rho_1}\frac{\partial P_1}{\partial y} = g\frac{\partial h_1}{\partial y} + \frac{\rho_2 g_*}{\rho_1}\frac{\partial h_2}{\partial y} \tag{C.25}$$

$$\frac{\partial \varphi}{\partial t} + \frac{\partial (U\varphi)}{\partial x} + \frac{\partial (V\varphi)}{\partial y} = 0 \tag{C.26}$$

where the reduced gravity, g, is defined as:

$$g \equiv g_*\left(\frac{\rho_1 - \rho_2}{\rho_1}\right) \tag{C.27}$$

If we assume that the upper surface is rigid, *i.e.* $h_2(x, y, t) \equiv 0$ then the two level shallow water equations reduce to a set of equations for a one layer model with bottom topography and are written as:

$$\frac{\partial U}{\partial t} + U\frac{\partial U}{\partial x} + V\frac{\partial U}{\partial y} - fV = -g\frac{\partial h}{\partial x} \tag{C.28}$$

$$\frac{\partial V}{\partial t} + U\frac{\partial V}{\partial x} + V\frac{\partial V}{\partial y} + fU = -g\frac{\partial h}{\partial y} \tag{C.29}$$

$$\frac{\partial \varphi}{\partial t} + \frac{\partial (U\varphi)}{\partial x} + \frac{\partial (V\varphi)}{\partial y} = 0 \tag{C.30}$$

where,

$$\varphi \equiv H_1 + h_1(x, y, t) - h_b(x, y) \tag{C.31}$$

C.5 Flux Form of Equations of motion

To build the flux form of the momentum equations take U times the continuity equation (Eqn. C.30) and add it to φ times the meridional momentum equation (Eqn. C.28)

$$U\frac{\partial\varphi}{\partial t}+\varphi\frac{\partial U}{\partial t}+U\frac{\partial(U\varphi)}{\partial x}+\varphi U\frac{\partial U}{\partial x}+U\frac{\partial(V\varphi)}{\partial y}+\varphi V\frac{\partial U}{\partial y}=\varphi fV-g\varphi\frac{\partial h}{\partial x} \tag{C.32}$$

define

$$\begin{aligned} m &\equiv \varphi U \quad \textbf{and} \\ n &\equiv \varphi V \end{aligned} \tag{C.33}$$

and the result of substituting the definition in Eqn. C.32 is

$$\frac{\partial m}{\partial t}+\frac{\partial(m^2/\varphi)}{\partial x}+\frac{\partial(nm/\varphi)}{\partial y}=nf-g\varphi\frac{\partial\varphi}{\partial x}-g\varphi\frac{\partial h_b}{\partial x} \tag{C.34}$$

and since g is a constant we can simplify Eqn. C.34

$$\frac{\partial m}{\partial t}+\frac{\partial(m^2/\varphi+g\varphi^2/2)}{\partial x}+\frac{\partial(nm/\varphi)}{\partial y}=nf-g\varphi\frac{\partial h_b}{\partial x} \tag{C.35}$$

Similarly, if we take V and multiply it times Eqn.C.30 and add it to φ times Eqn. C.29 we get

$$\frac{\partial n}{\partial t}+\frac{\partial(nm/\varphi)}{\partial x}+\frac{\partial(n^2/\varphi+g\varphi^2/2)}{\partial y}=mf-g\varphi\frac{\partial h_b}{\partial y} \tag{C.36}$$

and the equation of continuity is written:

$$\frac{\partial\varphi}{\partial t}+\frac{\partial m}{\partial x}+\frac{\partial n}{\partial y}=0 \tag{C.37}$$

equations Eqn.'s C.35 through C.37 have the form:

$$\frac{\partial\chi}{\partial t}+\frac{\partial F}{\partial x}+\frac{\partial G}{\partial y}=Q \tag{C.38}$$

where,

$$\chi=[m,n,\varphi] \tag{C.39}$$

$$F=\left[\left(\frac{m^2}{\varphi}+\frac{g\varphi^2}{2}\right),\quad\left(\frac{mn}{\varphi}\right),\quad m\right] \tag{C.40}$$

$$G=\left[\left(\frac{mn}{\varphi}\right),\quad\left(\frac{n^2}{\varphi}+\frac{g\varphi^2}{2}\right),\quad n\right] \tag{C.41}$$

$$Q=\left[\left(fn-g\varphi\frac{\partial h_b}{\partial x}\right),\quad\left(-fm-g\varphi\frac{\partial h_b}{\partial y}\right),\quad 0\right] \tag{C.42}$$

and $m\equiv\varphi U,\quad n\equiv\varphi V$. These are solved using the Lax-Wendroff two step time integration algorithm (Kasahara, 1966; Houghton et al., 1966).

$$\chi^{l+1}=\bar{\chi}^{l}-\Delta t\left(\frac{\partial F^{l}}{\partial x}+\frac{\partial G^{l}}{\partial y}\right)+\Delta t\left(\frac{\tilde{Q}^{l+1}+\bar{Q}^{l}}{2}\right) \tag{C.43}$$

$$\chi^{l+2}=\chi^{l}-2\Delta t\left(\frac{\partial F^{l+1}}{\partial x}+\frac{\partial G^{l+1}}{\partial y}\right)+2\Delta t\left(\tilde{Q}^{l+1}\right) \tag{C.44}$$

where $\tilde{Q}$ is the value of Q where φ has been averaged over x and y. $\bar{Q}$ is the value of Q where n, m, φ have been averaged over x and y.

The time step is calculated from the grid spacing Δx as follows:

$$\Delta t \leq \frac{\Delta x}{\sqrt{2} \cdot (U_m + \sqrt{g \cdot h_m})} \tag{C.45}$$

where, U_m is the magnitude of the maximum possible flow velocity and h_m is the maximum height of the free surface (including $U(y)$).

C.6 Scaling of Equations

U_s is the velocity scale
L_s is the length scale
D_s is the depth scale
$U' = U/U_s$ and $V' = V/U_s$
$x' = x/L_s$ and $y' = y/L_s$
$h' = h/D_s$ and $h'_b = h_b/D_s$
$t' = t/T_s$ and $f' = f \cdot T_s$ and $T_s = L_s/U_s$
$c \equiv \sqrt{gD_s}$ and $\alpha^2 \equiv L_s^2 \beta / c$ and $\gamma^2 \equiv c/U_s^2$

Specify Basic State and Initial Free Surface

$@t = 0$ $V(x, y) = 0$ and $U(x, y) = U_1(y)$
using Eqn's (1) and (2)

$$\frac{\partial h}{\partial y} = -\frac{1}{g} f(y) U_1(y) \tag{C.46}$$

or

$$h(y, t = 0) = H_1 - \frac{1}{g} \int_0^y f(\chi) U_1(\chi) d\chi \tag{C.47}$$

or in non-dimensional form

$$h'(y, t = 0) = H'_1 - \frac{U_s^2}{gD_s} \int_0^{y'} f'(\chi) U'^0(\chi) d\chi \tag{C.48}$$

where H_1 is a free parameter
Example: for $U_1(y) = U_0 + U_1 y + U_2 y^2$ and $f(y) = \beta y$,

$$h'(y) = \frac{H_1}{D_s} - \frac{1}{gD_s} \left[y^2 \frac{\beta U_0}{2} + y^3 \frac{\beta U_1}{3} + y^4 \frac{\beta U_2}{4} \right] \tag{C.49}$$

Specify Bottom Topography

$h_b(x, y) = h_m(x, y) + h_u(x, y)$
The mountain topography is given as:

$$h_m(x, y) = H_m \cdot \exp \left[- \left(\left(\frac{(x - c_x)}{a_x} \right)^2 + \left(\frac{(y - c_y)}{a_y} \right)^2 \right)^2 \right] \tag{C.50}$$

where, a_x, a_y, c_x, c_y, and H_m specify the "mountain".
The lower level wind is assumed to be related to the upper level wind. In this study we will assume that the winds decay with height, such that:

$$\frac{\partial U}{\partial z} = c \cdot U \tag{C.51}$$

$$U_2(y) = U_1(y) \cdot e^{(c \cdot (z_2 - z_1))} \approx U_1(y) \cdot e^{(c \cdot H_1)} \tag{C.52}$$

The lower topography due to the lower level winds, $h_u(x, y)$, is derived from:

$$f \cdot U_2 = \frac{-1}{\rho_2} \frac{\partial P_2}{\partial y} \tag{C.53}$$

where the equilibrium value (i.e., $h_1(x, y, t) = 0 \quad \text{and} \quad h_2(x, y, t) = 0$) of $P_2 = \rho_1 g_* H_1 + (\rho_2 - \rho_1) g_* h_b$ and the meridional momentum balance yields:

$$f \cdot U_2 = g_* \frac{\rho_2 - \rho_1}{\rho_2} \frac{\partial h_b}{\partial y} \equiv g \frac{\partial h_b}{\partial y} \tag{C.54}$$

which can be integrated directly to obtain:

$$h_u(y) = -\frac{1}{g} \int_0^y f(\chi) U_2(\chi) d\chi \tag{C.55}$$

C.7 Dampening

$$\chi^{l+1}(i, j) = \chi^{l+1}(i, j) - \Delta t \cdot \lambda(i) \cdot \left(\chi^l(i, j) - \chi^0(i, j) \right) \tag{C.56}$$

where χ^l represents $m, n, \text{and} \, \varphi$ at time step l.
$x \equiv x_0 + i \cdot \Delta x \quad \text{and} \quad y \equiv y_0 + j \cdot \Delta y$

$$\lambda(i) = \lambda^0 \cdot g(i), \quad \text{for} \quad 1 \le i \le 40 \tag{C.57}$$

$$\lambda(i) = 0, \quad \text{for} \quad i \ge 40 \tag{C.58}$$

and $g(i)$ = 0,0,0,.1,.2,.4,.8,1.0 ... 1.0, .8, .4, .2, 0.1

λ^0 is a free parameter, typically of the order of 1 day^{-1} for the earth (Zehnder, 1990) and on the order of 1/400 day^{-1} for Jupiter (Dowling and Ingersoll, 1989).

Δt is the time step in the Lax-Wendroff algorithm.

C.8 Wall Filter

first: do a spatial filter in x direction at boundary

$$\chi(i, j) = \frac{\chi(i+1, j) + 2\chi(i, j) + \chi(i-1, j)}{4} \tag{C.59}$$

for $1 \le i \le N_x \quad \text{and} \quad j = 1, 2, N_y - 1, N_y$
x axis is periodic, $x(N_x + i, j) = x(i, j)$
and χ represents $m, n, \text{and} \, \varphi$ at current time step.
second: do a spatial filter in y at boundary

$$\chi(i, j) = \frac{\chi(i, j+1) + 2\chi(i, j) + \chi(i, j-1)}{4} \tag{C.60}$$

for $1 \le i \le N_x \quad \text{and} \quad 2 \le j \le N_y - 1$

C.9 High Frequency Filter

first: do a spatial filter in x direction

$$\chi(i,j) = \frac{\chi(i+1,j) + 2\chi(i,j) + \chi(i-1,j)}{4} \tag{C.61}$$

for $1 \le i \le N_x$ and $2 \le j \le N_y - 1$
x axis is periodic, $x(N_x + i, j) = x(i,j)$
and χ represents $m, n,$ and φ at current time step.
second: do a spatial filter in y

$$\chi(i,j) = \frac{\chi(i,j+1) + 2\chi(i,j) + \chi(i,j-1)}{4} \tag{C.62}$$

for $1 \le i \le N_x$ and $j = 2, N_y - 1$

Derived Quantities

Vorticity:

$$\zeta = \frac{\partial V}{\partial x} - \frac{\partial U}{\partial y} \tag{C.63}$$

$+$ in northern hemisphere is cyclonic
$-$ in southern hemisphere is cyclonic

Vertical Wind:

$$W(x,y,h,t) = \varphi\left(\frac{\partial U}{\partial x} + \frac{\partial V}{\partial y}\right) + U\frac{\partial h_b}{\partial x} + V\frac{\partial h_b}{\partial y} \tag{C.64}$$

Potential Vorticity:

$$q = \frac{\zeta + f}{\varphi} \tag{C.65}$$

C.10 Summary of Model Parameters for Neptune

This is an example to study the Great Dark Spot on Neptune. The simulation was of the equatorial region with a 12 km disturbance.
Planet: $\Omega = \frac{2\cdot\pi}{18.4^h}$ and $R = 24800$ Km.
$-40000 \le x \le 40000$ and $-20000 \le y \le 10000$
$N_x = 191$ and $N_y = 80$
Damping: $\lambda_0 = 1/400$ day^{-1}
Pressure disturbance: $H_m = 12$ Km
$c_x = -10000$ and $c_y = -6000$ Km
$a_x = -6000$ and $c_y = -4080$ Km
Free Height: $H_1 = 50$ Km
reduced Gravity: $g = 6$ ms^{-1}.
Scale: $D_s = 50$ Km, $L_s = 10000$ Km, $U_s = 50$ ms^{-1}.
Filters: High frequency filter every 60^h of simulation. Wall filter and dampening on every time step.

The Voyager 2 spacecraft flew within 29,240 km of Neptune on August 24, 1989, 11:56 EDT.. The 3 images shown here are all taken with the orange filter and are all map projected to the same region on Neptune,

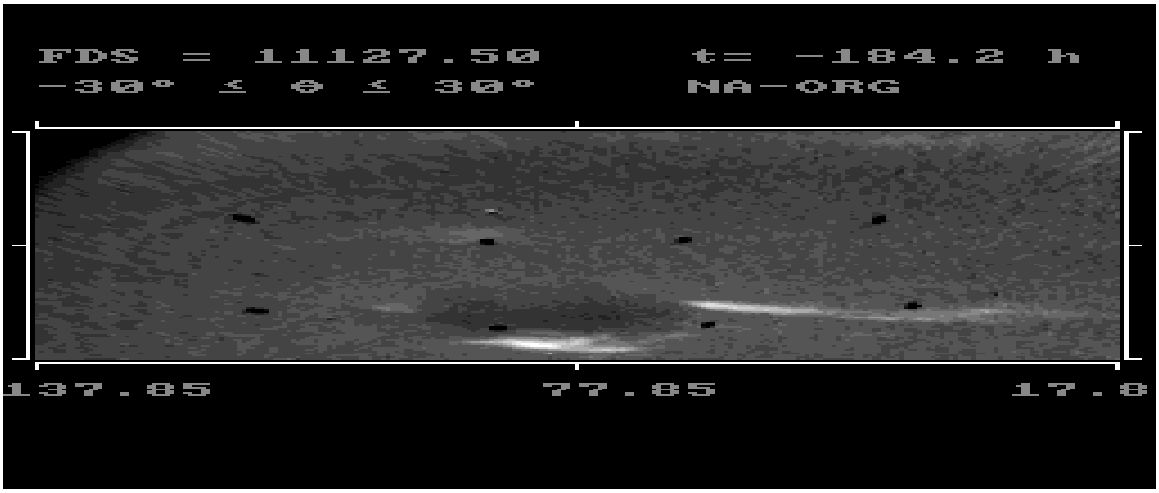


Figure C.2: Image of Neptune taken 181 hours before closest encounter

tracking the "Great Dark Spot" near -20° latitude. The image in Fig. C.2 was taken 181 hours before the encounter. The images in Fig. C.3 and C.4 are taken -19, and 0 hours before encounter, respectively.

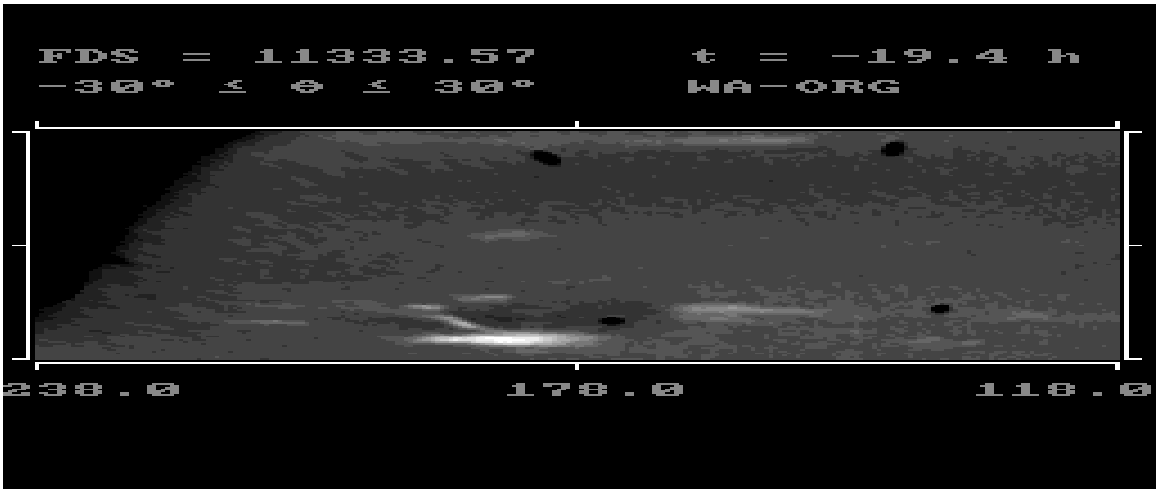


Figure C.3: Image of Neptune taken 19 hours before closest encounter

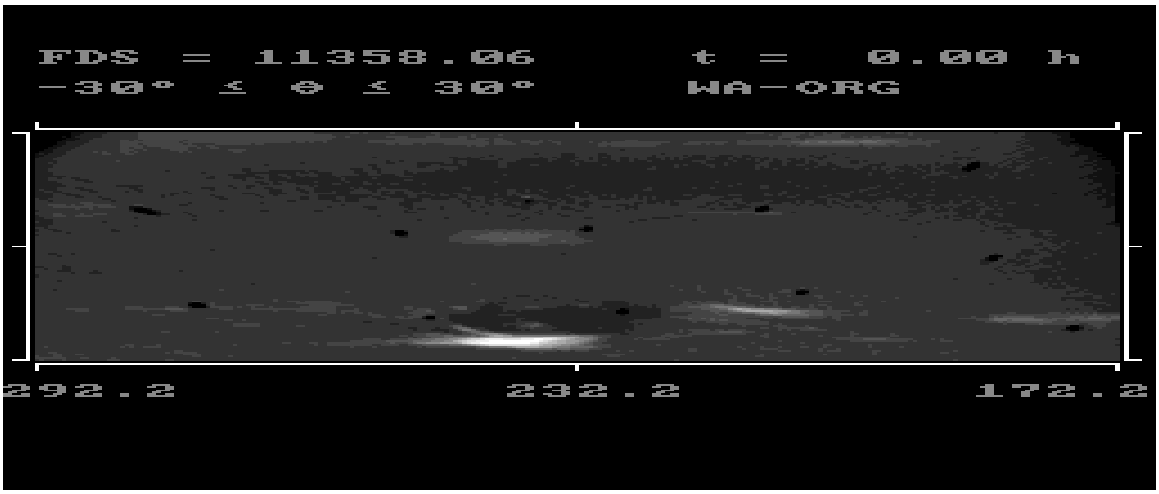


Figure C.4: Image of Neptune taken at closest encounter

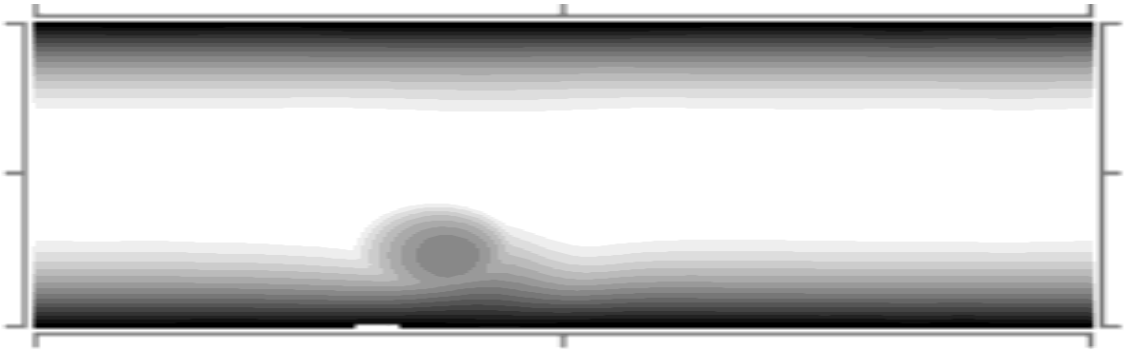


Figure C.5: The Shallow water model of the Great Dark Spot height at 2.5 days.

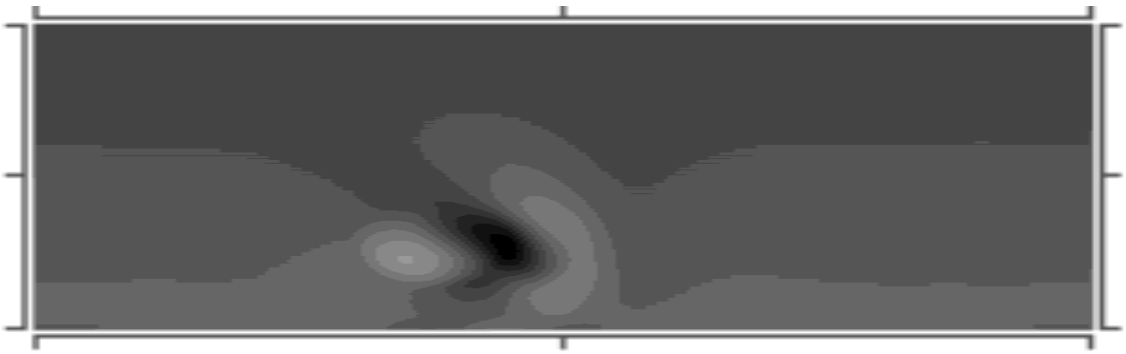


Figure C.6: Shallow Water Model Of The Great Dark Spot vorticity at 2.5 days.

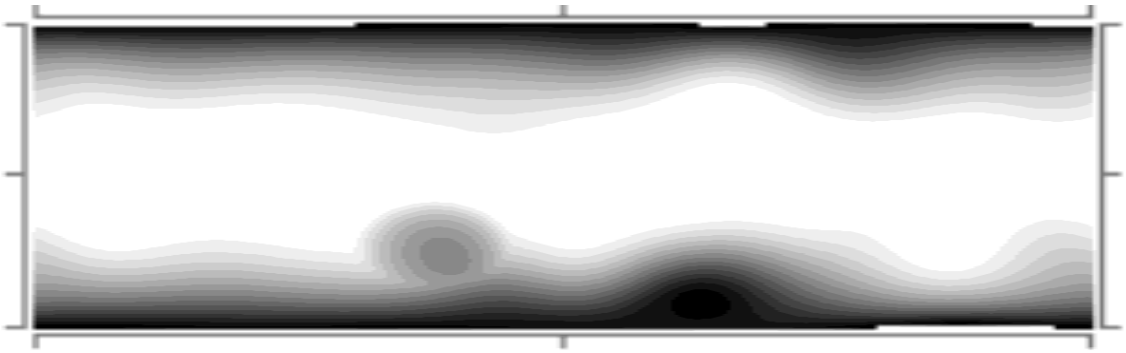


Figure C.7: The Shallow water model of the Great Dark Spot height at 40 days.

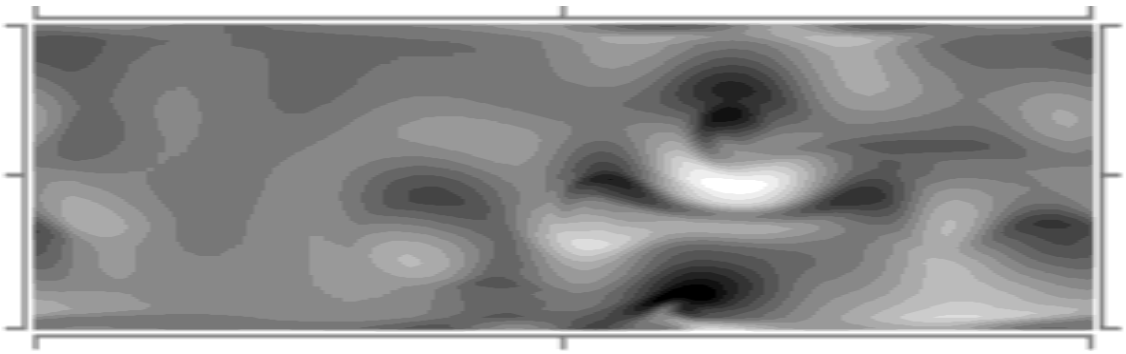


Figure C.8: Shallow Water Model Of The Great Dark Spot vorticity at 40 days.

C.11 Summary of Model Parameters for Saturn

Planet: $\Omega = \frac{2 \cdot \pi}{10.24^h}$ and $R = 56000$ Km.
$-24500 \leq x \leq 24500$ and $-58643 \leq y \leq 83078$
$N_x = 128$ and $N_y = 48$
Damping: $\lambda_0 = 1/400$ day^{-1}
Pressure disturbance: $H_m = 10$ Km
$c_x = -73792$ and $c_y = 1000$ Km
$a_x = 1000$ and $c_y = 800$ Km at an angle of 90°
Free Height: $H_1 = 120$ Km
reduced Gravity: $g = 1$ ms^{-1}.
Scale: $D_s = 100$ Km, $L_s = 10000$ Km, $U_s = 100$ ms^{-1}.
Filters: High frequency filter every 5^d of simulation. Wall filter and dampening on every time step.

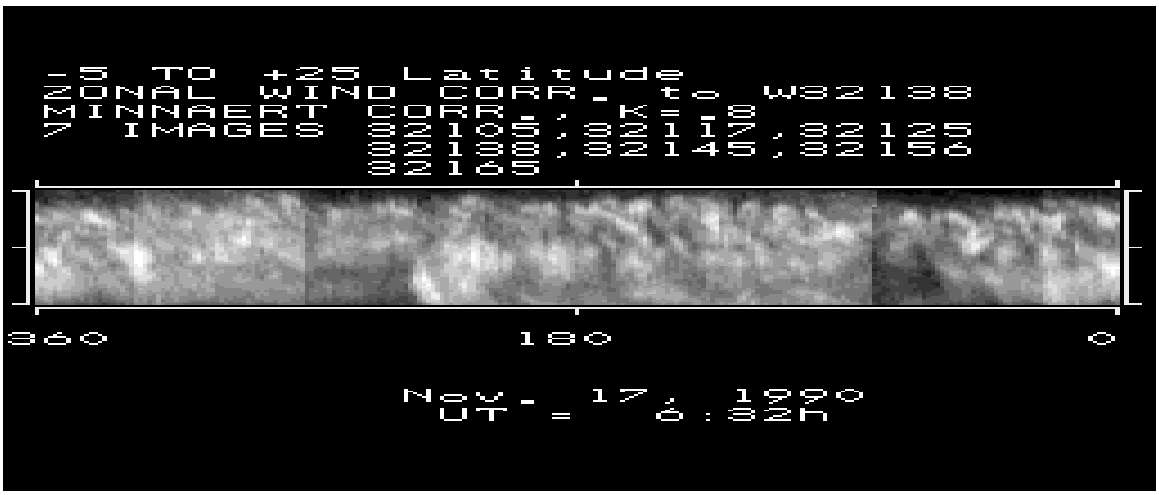


Figure C.9: Saturn storm mosaic for Nov. 17, 1990 days

On Nov. 17, and 18, 1990 the Hubble Space Telescope took a series of images for 2 full Saturnian days. In mid-September of 1990 a "storm" had erupted near the equator and was being ripped apart by Saturn's 500 km equatorial jet. The images map projected and corrected for both limb darkening and cloud motions within the equatorial jet and a mosiac was constructed for each day. The image in Fig. C.10 shows the equatorial region (-5° $\leq$ latitude $\leq$ 25°) on the 17$^{\text{th}}$ and the image shown in Fig. C.10 is the same format but for the 18$^{\text{th}}$.

Shallow water vorticity at 37.5 days for a pressure disturbance at 5° latitude is shown in Fig. C.11. The Saturnian zonal wind measurements from Voyager and Hubble Space Telescope, including the 500 km/s equatorial jet, were imposed as a boundard condition in this model (*e.g.*, see Barnet, C.D., J.A. Westphal, R.F. Beebe and L.F. Huber 1992). Hubble Space Telescope observations of the 1990 equatorial disturbance on Saturn: zonal winds and central meridian albedos. Icarus v.100 p.499-510). Notice the alternating vorticity field corresponding to the region of high wind shear.

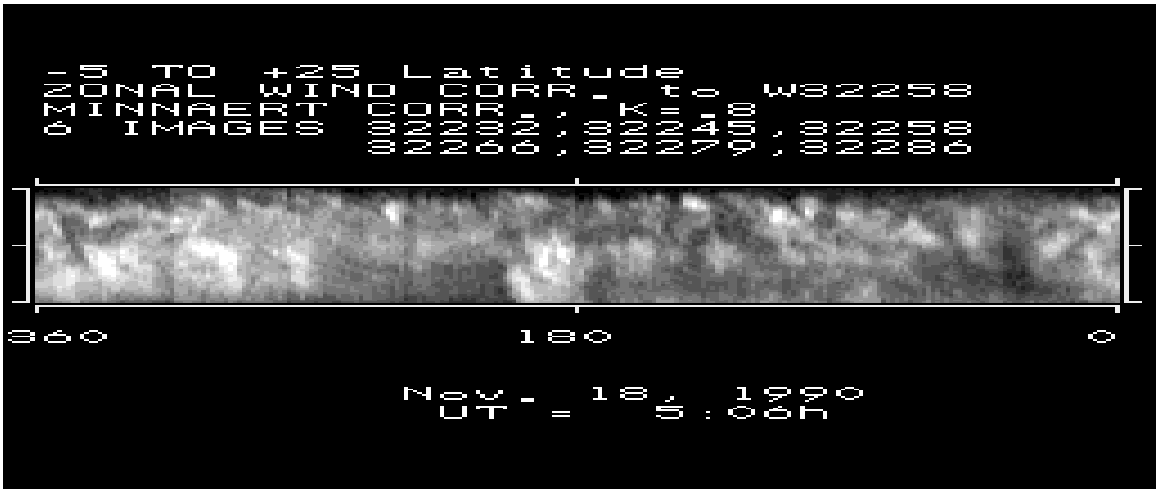


Figure C.10: Saturn storm mosaic for Nov. 18, 1990 days

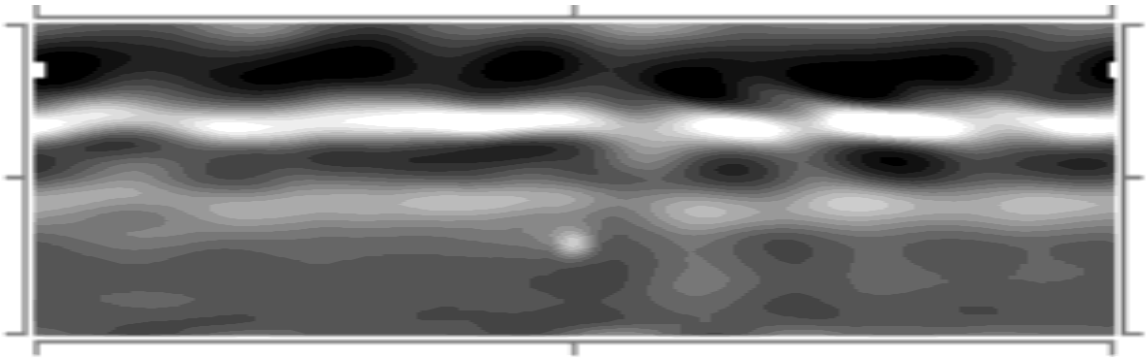


Figure C.11: Shallow water model of the Saturn storm: vorticity at 37.5 days

Appendix D

Summary of Useful Mathematical Formula & Constants

D.1 Trigonometric Identities

$$\cos(\alpha \pm \beta) = \cos(\alpha)\cos(\beta) \pm \sin(\alpha)\sin(\beta) \tag{D.1}$$

$$\sin(\alpha \pm \beta) = \sin(\alpha)\cos(\beta) \mp \cos(\alpha)\sin(\beta) \tag{D.2}$$

$$\cos(m\delta)\cos(n\delta) = \frac{1}{2}[\cos((m-n)\delta) + \cos((m+n)\delta)] \tag{D.3}$$

$$\cos(\alpha\delta)\cos(\beta\delta) = \frac{1}{2}[\cos((\alpha-\beta)\delta) + \cos((\alpha+\beta)\delta)] \tag{D.4}$$

$$\sin(\alpha \pm j\pi) = (-1)^j \cdot \sin(\alpha) \tag{D.5}$$

D.2 Vector Mathematics

The gradient:

$$\nabla\Psi = \hat{m}\frac{\partial\Psi}{\partial x} + \hat{n}\frac{\partial\Psi}{\partial y} \tag{D.6}$$

D.3 Useful Summations

$$\sum_{n=1}^{\infty}\frac{1}{n^2} = \frac{\pi^2}{6} \tag{D.7}$$

$$\sum_{n=1}^{\infty}\frac{(-1)^{n-1}}{n} = \log_e(2) \tag{D.8}$$

$$\lim_{N\to\infty}\left(\sum_{n=1}^{N}\frac{1}{n}-\log_e(N)\right)=\gamma=0.5772 \tag{D.9}$$

D.4 Derivatives & Integrals

$$\int_0^y \log_e(y)\cdot dy = e^{-y}\cdot dy \tag{D.10}$$

$$\frac{1}{L}\int_0^L \cos(a\delta)d\delta = \frac{\sin(aL)}{aL} = \mathrm{sinc}(y) \quad \text{CRC\#291} \tag{D.11}$$

$$\int_a^b \frac{dx}{1+25\cdot x^2} = 0.2\cdot\left[\tan^{-1}(5\cdot x)\right]_a^b \quad (\text{CRC26}^{\text{th}}\text{ Edition, Integral \#62, pg. 293}) \tag{D.12}$$

$$\frac{2}{L}\int_0^L \cos(2\pi\nu_0\delta)\cos(2\pi\nu\delta)d\delta = \frac{\sin(2\pi L(\nu-\nu_0))}{2\pi L(\nu-\nu_0)} + \frac{\sin(2\pi L(\nu+\nu_0))}{2\pi L(\nu+\nu_0)} \quad \text{CRC\#317} \tag{D.13}$$

$$\int_0^L \delta^3\sin(a\delta)d\delta = \frac{3a^2L^2-6}{a^4}\sin(aL) - \frac{a^2L^3-6L}{a^3}\cos(aL) \quad \text{CRC\#391} \tag{D.14}$$

$$\int_0^L \delta^m\sin(a\delta)d\delta = \frac{-L^m}{a}\cos(aL) + \frac{m}{a}\int_0^L \delta^{m-1}\cos(a\delta)d\delta \quad \text{CRC\#392} \tag{D.15}$$

$$\int_0^L \delta^3\cos(a\delta)d\delta = \frac{3a^2L^2-6}{a^4}\cos(aL) + \frac{a^2L^3-6L}{a^3}\sin(aL) \quad \text{CRC\#395} \tag{D.16}$$

$$\frac{1}{L}\int_0^L \left(\frac{\delta^m}{L^m}\right)\cos(a\delta)d\delta = \mathrm{sinc}(aL) - \frac{4}{aL^{m+1}}\int_0^L \delta^{m-1}\sin(a\delta)d\delta \quad \text{CRC\#396} \tag{D.17}$$

recursive substitution of CRC# 392 into CRC# 396 will yield a general solution for the integral which has the form of CRC# 396b.

$$\frac{1}{L}\int_0^L \left(\frac{\delta^m}{L^m}\right)\cos(a\delta)d\delta = b_0(m)\cdot\sin(y) + b_1(m)\cdot\cos(y) \tag{D.18}$$

$$\frac{1}{a}\cdot\int_{-\infty}^{\infty}\frac{\sin(ax)}{x}dx = \frac{\pi}{a} \quad \text{CRC\#621} \tag{D.19}$$

$$\int_0^{\infty}\frac{\sin^2(ax)}{x^2}dx = \frac{a\pi}{2} \quad \text{CRC\#630} \tag{D.20}$$

D.5 MacLauren and Taylor Expansions

$$\begin{aligned} f(x) &= f(a) + f'(a)\cdot(x-a) + \frac{1}{2!}f''(a)\cdot(x-a)^2 + \frac{1}{3!}f'''(a)\cdot(x-a)^3 \\ &+ \ \ldots \ + \ \frac{1}{(n-1)!}f^{n-1}(a)\cdot(x-a)^{n-1} + R_n \end{aligned} \tag{D.21}$$

where,

$$R_n = \frac{x^n \cdot f^(n)(\theta x)}{n!}, \qquad 0 < \theta < 1 \tag{D.22}$$

The binomial expansion

$$\frac{1}{1\pm x} = \sum_{n=0}^{\infty} (\mp x)^n \simeq 1 \mp x + x^2 + \ldots \qquad \text{for} \quad |x| < 1 \tag{D.23}$$

$$\frac{1}{(1\pm x)^2} = \sum_{n=0}^{\infty} (n+1)\,(\mp x)^n \simeq 1 \mp 2x + 3x^2 \ldots \qquad \text{for} \quad |x| < 1 \tag{D.24}$$

In a MacLaurin series $a = 0$

$$e^x = \sum_{j=0}^{\infty} \frac{x^j}{j!} \simeq 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots \tag{D.25}$$

$$\sin(x) = \sum_{j=0}^{\infty} (-1)^j \cdot \frac{x^{2j+1}}{(2j+1)!} \simeq x - \frac{x^3}{(3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots = x - \frac{x^3}{6} + \frac{x^5}{120} - \frac{x^7}{5040} + \ldots \tag{D.26}$$

$$\cos(x) = \sum_{j=0}^{\infty} (-1)^j \cdot \frac{x^{2j}}{(2j)!} \simeq 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + ... = 1 - \frac{x^2}{2} + \frac{x^4}{20} - \frac{x^6}{720} + ... \tag{D.27}$$

$$\log_e(1+x) = x - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \ldots \tag{D.28}$$

$$\log_e(a+x) = \log_e(a) + 2\left[\frac{x}{2a+x} + \frac{1}{3}\left(\frac{x}{2a+x}\right)^3 + \frac{1}{5}\left(\frac{x}{2a+x}\right)^5 + \ldots\right] \qquad a > 0, \quad -a < x < +\infty \tag{D.29}$$

$$\log_e\left(\frac{1+x}{1-x}\right) = 2\Big[x + \frac{x^3}{3} + \frac{x^5}{5} + \ldots + \frac{x^{2n-1}}{2n-1} \qquad -1 < x < 1 \tag{D.30}$$

$$\mathrm{sinc}(y) = \sum_{j=0}^{\infty} (-1)^j \cdot \frac{y^{2j}}{(2j+1)!} \simeq 1 - \frac{y^2}{6} + \frac{y^4}{120} - \frac{y^6}{5,040} + \frac{y^8}{362,880} - ... \tag{D.31}$$

$$\sinh(x) = \frac{e^x - e^{-x}}{2} \tag{D.32}$$

$$\sinh(x) = \sum_{j=0}^{\infty} \frac{x^{2j+1}}{(2j+1)!} \simeq x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \ldots \tag{D.33}$$

$$\sinh(ix) = i \cdot \sin(x) \tag{D.34}$$

D.6 Linear Algebra

$$(A+B)' = A' + B' \tag{D.35}$$

$$(A \cdot B)' = B' \cdot A' \quad (\text{CRC } 26^{th}\text{Ed. pg. 28, Eqn. 2.5}) \tag{D.36}$$

which can be extended *ad infinitum*

$$(A \cdot B \cdot C)' = ((A \cdot B) \cdot C)' = C' \cdot (A \cdot B)' = C' \cdot B' \cdot A' \tag{D.37}$$

and

$$[A \cdot B \cdot C]^{-1} = [C]^{-1} \cdot [B]^{-1} \cdot [A]^{-1} \quad (\text{CRC } 26^{th}\text{Ed. pg. 33, Eqn. 6.2}) \tag{D.38}$$

Given a linear set of equations where the number of equations, $A_{n,j}, b_n$, is greater than the number of parameters, x_j

$$A_{n,j} \cdot x_j = b_n \tag{D.39}$$

$$x_j = A^{-1}_{j,n} \cdot b_n \tag{D.40}$$

from the definition of an inverse

$$\begin{aligned} A_{n,j} \cdot A^{-1}_{j,n} &= I_{n,n} && \text{(D.41)} \\ A^T_{j,n} \cdot \left(A_{n,j} \cdot A^{-1}_{j,n}\right) &= A^T_{j,n} \cdot I_{n,n} && \text{(D.42)} \\ \left(A^T_{j,n} \cdot A_{n,j}\right) \cdot A^{-1}_{j,n} &= A^T_{j,n} && \text{(D.43)} \end{aligned}$$

for a non-square matrix

$$A^{-1}_{j,n} = \left[A^T_{j,n} \cdot A_{n,j}\right]^{-1} \cdot A^T_{j,n} \tag{D.44}$$

The inverse of a compound matrix given by

$$\mathbf{M} = \mathbf{M}_3 \cdot \mathbf{M}_2 \cdot \mathbf{M}_1 \tag{D.45}$$

is given by the inverses taken in the opposite order:

$$\mathbf{M}^{-1} = \mathbf{M}_1^{-1} \cdot \mathbf{M}_2^{-1} \cdot \mathbf{M}_3^{-1} \tag{D.46}$$

If the elements of a matrix $Y_{m,n}$ are functions of a scalar, x, then

$$\frac{\partial Y}{\partial x} = \frac{\partial y(m,n)}{\partial x} \tag{D.47}$$

If $Y = U \cdot V$

$$\frac{\partial U \cdot V}{\partial x} = V \cdot \frac{\partial U}{\partial x} + U \cdot \frac{\partial V}{\partial x} \tag{D.48}$$

If $A \neq f(x)$ then

$$\frac{\partial A \cdot Y}{\partial x} = A \cdot \frac{\partial Y}{\partial x} \tag{D.49}$$

$$\frac{\partial A^T \cdot Y \cdot A}{\partial x} = A^T \cdot \frac{\partial Y}{\partial x} \cdot A \tag{D.50}$$

$$\frac{\partial Y^T \cdot A \cdot Y}{\partial x} = \frac{\partial Y^T}{\partial x} \cdot A \cdot Y + Y^T \cdot A \cdot \frac{\partial Y}{\partial x} \tag{D.51}$$

D.7 Table of Constants

Table D.1: Constants

symbol	value	units	description
π	3.14159265359		
N_a	6.02214199E+23	molecules/mole	Avogadro's number
k	1.3806503E-16	erg/K	Boltzmann's constant
h	6.62606876E-27	erg· s	Planck's constant
c	2.99792458E+10	cm/s	speed of light
R_*	$N_a \cdot k$	erg/mole/K	universal gas constant
R_a	R_*/mw_d	erg/gm/K	gas constant of dry air
mw_w	18.0151	gm/mole	molecular weight of water
mw_d	28.9644	gm/mole	molecular weight of dry air
mw_{O_3}	47.9982	gm/mole	molecular weight of ozone
mw_{CH_4}	16.04303	gm/mole	molecular weight of methane
mw_{CO}	28.0104	gm/mole	molecular weight of CO
mw_{CO_2}	44.00995	gm/mole	molecular weight of CO_2

```
Nloschmidt= 2.6867775E+19   Loschmidt's number
                            molecules/cm^3 per atm @ STP
P_std = 1013.250        ! PT99, standard pressure, milli-bar
T_std =  273.150        ! standard temperature, K
G_std =  980.665       ! PT99, standard gravity, cm/s
<R> = (Req*Req*Rpl^(1/3 (volume of oblate sphere
Rearth = 6371.004  ! ESAA values of Req, Rpl
PLCON1   = 2.0*PLANCKS*CLIGHT*CLIGHT
PLCON2   = PLANCKS*CLIGHT/BOLTZMNS
```