Scientific Calculating, Programming, and Writing

James D Emery

7/11/2010

Contents

1 Introduction **11**

2 Programming Editors **12**

3 Some Scientific Programming Tools **13**

4 Using the Command Line **16**
4.1 Using the Command Line in Windows Systems, Adding a Directory to the Path . . . 16
4.2 Redirecting Output, and Piping Commands . . . 18
4.3 Some Windows Commands . . . 18
4.4 The Command Line in Linux . . . 19
4.5 Some Linux Commands . . . 19
4.6 The Command line in MacOS . . . 19
4.7 Some MacOSX Commands . . . 20
4.8 Common Troubles in MacOS . . . 21

5 Using grep, Creating and Using a Notebook to Document and Recall Programs and Commands **21**
5.1 Using grep . . . 22
5.2 Adding A Directory to the Path in Windows, Linux, and MacOS 23
5.3 The Windows Version of Notes . . . 23
5.4 The MacOS and Linux Version of Notes . . . 25

5.5 Installing Notes on Windows 26
5.6 Installing Notes on Linux 27
5.7 Installing Notes on MacOS 27

6 Graphing and Programmable Calculators 27

7 Plotting a Function With Utility Programs. 30
7.1 Viewing Postscript Graphics With Ghostview and Ghostscript. 31
7.2 Converting Postscript to pdf 32

8 Converting Between Text Point Files and Excel CSV Files (Comma Separated Values). 32

9 An Example of Scientific Programming: Solving a Differential Equation Numerically 32

10 Scientific Programming in Python 35
10.1 Example: Writing Function Points to a File 38
10.2 Example: Solar Heating, Using Command Line Arguments . . 39
10.3 Example: Converting degrees, minutes, seconds to a decimal number, using Command Line Arguments 40
10.4 Example: Reading points from a file 41
10.5 A List of some Python Commands and Their Use 42
10.6 Vectors 43
10.7 Formatting Output 45
10.8 Creating a Module 45
10.9 Using Eval 47
10.10 Object Oriented Programming Example 47
10.11 A Matrix Class 48
10.12 Functions With a Variable Number of Arguments 60
10.13 Class Inheritance 60
10.14 The Module Search Path 60
10.15 Google Python Tutorial 61
10.16 Computing the Matrix Inverse Using Gaussian Elimination . . 61
10.17 Examples Using Matrices 66
10.18 Python Exercises 67
10.19 Python References 75

11 Using Numerical Subroutine and Function Libraries in Scientific Programming **75**

12 Scientific Programming in Fortran **75**
12.1 Passing Function Names 75
12.2 Handling Matrices 77

13 Scientific Programming in C **88**
13.1 Passing Function Names 88
13.2 Handling Matrices and Dynamic Memory Allocation 90

14 Scientific Programming in C Using Numerical Recipes **97**
14.1 Passing Function Names 97
14.2 Handling Matrices 97

15 Scientific Programming in C Using The GNU Library **97**
15.1 Passing Function Names 97
15.2 Handling Matrices 97

16 Scientific Programming in C++ **97**
16.1 Passing Function Names 97
16.2 Handling Matrices 102

17 Scientific Programming in Basic **102**
17.1 Passing Function Names 102
17.2 Handling Matrices 102

18 Scientific Programming in Matlab and Octave **102**
18.1 Introduction 102
18.2 Downloading Octave 102
18.3 A Few Command Examples 103
18.4 Using Octave as a Calculator 103
18.5 Matlab Scripts, m-Files 104
18.6 Linear Equation Solution 105
18.7 Fourier Transform Example 105
18.8 Reading Data From a File as a Matrix 106
18.9 Solving a Differential Equation Numerically 107
18.10 Damped One-Dimensional Vibration 108
18.11 Bibliography for Matlab and Octave 112

19 Scientific Programming in Java **113**
19.1 Passing Function Names 113
19.2 Handling Matrices 113

20 Scientific Programming in Symbolic Languages Such as Maple, Mathematica, and Mupad **113**
20.1 Passing Function Names 113
20.2 Handling Matrices 113

21 Scientific Programming in Pascal **113**
21.1 Passing Function Names 113
21.2 Handling Matrices 113

22 Scientific Programming in Excel **113**
22.1 Passing Function Names 113
22.2 Handling Matrices 113

23 Example: Orbit Intersection **113**

24 Example: Boat Analysis. **115**

25 Polynomial Evaluation **115**

26 Roots of a Cubic Polynomial **118**

27 Synthetic Division and Newton's Method **119**

28 Tschebyscheff Approximation **120**

29 The Maximum Curvature of a Cubic Polynomial **122**

30 Ferguson Cubics **123**

31 Hermite Interpolation **124**

32 On The Determination Of Maximum Curvature Of A Cubic (Dixon) **125**
32.1 The Analysis 125

33 The Trapezoid Rule For Numerical Integration **128**

34 The Truncation Error of the Trapezoid Rule **129**

35 The Adams-Moulton Formula **130**

36 The Euler Method **131**

37 The Taylor Series Method **131**

38 Runge-Kutta Methods **132**

39 Derivation of a Second Order RungeKutta Method **133**

40 The Fourth Order RungeKutta Method **133**

41 Richardson Extrapolation **136**
41.1 A Single Extrapolation 136
41.2 Repeated Richardson Extrapolation 136

42 Common Applications **137**

43 Solving Differential Equations Using Finite Differences **138**

44 Central Force Example **139**

45 Stiff Equations **140**

46 Eigenvalues **142**

47 The Jacobi Method For Locating Eigenvalues **143**

48 Leverrier's Method for the Characteristic Polynomial **145**

49 Scientific Typesetting: TeX and Latex **146**

50 Downloading, Installing, and Running MikTex on Windows **147**

51 Creating a Sample LaTeX File **148**

52 A Listing of the File sample.tex **149**

53 The Command Line For Running Latex **152**

54 A LaTeX Tutorial **153**

55 Placing Figures in LaTeX **153**

56 Placing Figures and Diagrams in A Latex File **154**

57 Installation of PCTeX **154**

58 The PostScript Mode in PCTeX **154**

59 Saving A Typeset Document as PostScript **155**

60 LaTeX Online Equation Editor **156**

61 Including a PostScript Figure **156**

62 Including A Plot of a function Using psfig Under PCTeX **157**

63 Using Acrobat Distiller to Make a PDF **161**

64 Including a Windows Bitmap Figure With PCTeX **161**

65 Another Method of Placement: An Optics Figure **162**

66 Making an Acrobat PDF **164**

67 A Figure Showing Planck's Black Body Radiation Curves **164**

68 Using psfig Under Linux and Unix **165**

69 Generating A PostScript File in Linux and Unix **167**

70 Viewers **168**

71 A Figure Created in a Drawing Program **169**

72 A Figure Created Using EG Graphics **169**

73 EG Graphics in Programming. **174**

74 A CorelDraw Figure **174**

75 A Scanned Figure **177**

76 A Projective Geometry Figure **179**

77 Pasting a Figure Created in DesignCAD into MathCAD **179**

78 Creating a Postscript Figure Using AutoCad or DesignCad **182**

79 Adding Annotation to A Postscript File **183**

80 Creating Figures With AutoCad and DesignCad **186**

81 Importing A Figure That Was Generated With A Programming Language Into DesigCAD or AutoCAD **186**

82 Least Squares Approximation **187**
82.1 Elementary Formulation 187
82.2 A Geometric View of the Least Squares Problem 188
82.3 Linear Circle Fitting 194
82.4 Nonlinear Circle Fitting 197
82.5 Weighted Least Squares 201
82.6 Bilinear Interpolation Example 202
82.7 Fitting Functions With Horizontal Asymptotes, Using the general program **lsqgen**. 203
82.8 A General LSQ Program **lsqgen.ftn** 205
82.9 A Listing of **lsqgen.ftn** 205

83 References for LaTeX and Postscript **214**

84 The Matrix Singular Value Decomposition **215**
84.1 Eigenvalues 219
84.2 Range and Kernel 220
84.3 The Psuedoinverse 221
84.4 Application to Fitting a Straight Line in Space 225
84.5 Linear Least Squares Using SVD 230

85 Solving Partial Differential Equations **230**

86 An Introduction to Laplace's Equation in Physics 230
86.1 Introduction 230
86.2 Gravity 230
86.3 Gauss's Law 231
86.4 Curl, Divergence, Gradient 232
86.5 Potential 233
86.6 The Heat Equation 233
86.7 Electric Potential 234
86.8 Electric Current Flow 234
86.9 Diffusion Equation 234
86.10 Analytic and Harmonic Functions 234
86.11 Viscosity 235
86.12 Fluid Mechanics 235
86.13 Velocity Potential and Stream Function 237
86.14 Solution Methods 240
86.15 Separation of Variables 240
86.16 The Finite Difference Method 240
86.17 Variational Methods and the Finite Element Method 241
86.18 References for Laplace's Equation 241

87 A Review of Vector Analysis 241
87.1 Introduction 241
87.2 The Inner Product 241
87.3 The Vector Product 244
87.4 Heron's Formula for the Area of a Triangle 246
87.5 Curl, Divergence, Gradient 247
87.6 The Parallel Axis Theorem 252
87.7 Curvilinear Coordinates 252

88 High Performance Computing and Supercomputing 258
88.1 History 258

89 The Kalman Filter 259

90 Elementary Physics Formulas 260
90.1 Simple Motion 260
90.2 Vectors 261
90.3 Parabolic Trajectory of a Projectile 261

90.4 Centripetal Acceleration 261
90.5 Newton's Laws of Motion 261
90.6 Gravity 262
90.7 Friction 262
90.8 Momentum 262
90.9 Energy 262
90.10 Elasticity, Stress-Strain Relations 263
90.11 Torque 264
90.12 Equilibrium 264
90.13 Rotation about a fixed axis 264
90.14 Simple Harmonic Motion 265
90.15 Special Relativity 266
90.16 Fluids, Forces and Motion 267
90.17 Heat and Thermodynamics 267

91 Electrical Circuits 269
91.1 Maxwell's Equations 269
91.2 The Clamp Meter 271
91.3 Inductance and Mutual Inductance 272
91.4 Magnetic Circuits 273
91.5 Steady State Alternating Currents 274
91.6 RLC Transient Solution 275
91.7 A Piezioelectric Transducer Equivalent Circuit 278
91.8 The Ideal Transformer 279
91.9 The Perfect Transformer 279
91.10 Transformer Equivalent Circuits 281
91.11 Thevenin's Theorem 281
91.12 Resonance of an RLC Circuit and The Quality Factor 281
91.13 Quality Factor at Nonresonance 283
91.14 General Networks 287
91.15 A Spice Direct Current Example 287
91.16 A Spice Alternating Current Example 287
91.17 A Spice Transient Example 287
91.18 Bibliography For Electrical Circuits 287

92 What is a Vector Space and What is Linear Algebra? 288
92.1 Vectors and Linear Algebra 288

93 Finite Dimensional Vector Spaces **290**
93.1 Determinants 294
93.2 Permutations 296
93.3 Kernal and Range 296
93.4 Inner Product Spaces 297
93.5 Quadratic Forms 297
93.6 Canonical Forms 297
93.7 Upper Triangular Form 297
93.8 Isometries, Rotations, Orthogonal Matrices 298
93.9 Rotation Matrices 298
93.10 Exponentials of Matrices and Operators in a Banach Algebra . 299
93.11 Eigenvalues 299
93.12 Unitary Transformations 299
93.13 Transpose, Trace, Self-Adjoint Operators 299
93.14 The Spectral Theorem 299
93.15 Tensors 299
93.16 Projection Operators 300
93.17 Linear Algebra Bibliography 300

94 Projective Space **300**
94.1 Computations With Two Dimensional Straight Lines, Homogeneous Coordinates 300
94.2 A Geometric Motivation For Projective Geometry 302
94.3 Higher Dimensional Projective Space 304
94.4 Barycentric Coordinates For Triangular Elements In 2-Space . 305
94.5 Barycentric Coordinates In Euclidean N-Space 306
94.6 Barycentric Coordinates Relative To A Triangle In Space . . . 308
94.7 Barycentric Coordinates Relative To A Line Segment In Space 309
94.8 Projective Space, Conic Curves, Quadric Surfaces, and Quadric Solids 310
94.9 Overview 313
94.10 Projective Space and the Conic Sections 321
94.11 Application to the Projection of an Image to a Screen 326
94.12 A Projective Transformation That Takes Four Points To Four Points 328
94.13 A Screen Projection Program Example 336
94.14 Projective Space Bibliography 346

1 Introduction

Scientific Programming is that part of science concerned with numerical and symbolic calculation. Electronic computers were invented to do scientific programming, although soon after their invention the majority of computers were used to do simple data processing. Scientific Programming often uses mathematical functions, vectors, matrices, and complex numbers. There are certain characteristics of computer languages that are specifically needed in this type of computing. Subroutines must be able to accept the names of arbitrary functions for its calculations as arguments. For example a general subroutine that computes a integral of a function needs to know were to find that function. This may be done by passing the name of the external function. Some computer languages do not have this capability, in others this may be difficult to do. Subroutines must also handle matrices and vectors of non-fixed size. They must be able to determine the size and shape of matrices. They must be able to set the range of indices for matrices. Below we shall show how this can be done using various languages and tools. The first compiler ever written was for the Fortran language. In the beginning computers were programmed laboriously using what is called machine language, using the operators built into the machine and using actual memory addresses to store and retrieve data. A little later came symbolic assembly language where one could introduce names for data locations and some other aids to programming.

Science relies on mathematical expressions and formulas. Fortran was specifically devised for computing with such formulas. The name Fortran comes from the phrase "Formula Translation." Fortran specifically included complex numbers, elementary functions such as, sin, cosine, exponential, and complex versions of such elementary functions. Multidimensional arrays for handling matrices and vectors was also part of the language. Much scientific programming is now done in C and C++. C and C++ are very powerful languages, but scientific programming in these languages is not so transparent as in Fortran. Fortran is an old language, and old versions of the language had several problems. It was developed during the time of the 80 column IBM punched card, had a rigid format where the first six columns were reserved for line numbers, a line could be no longer than 72 columns.

Programming tended to use goto's to jump to line numbers. This was overcome with the Fortran 77 standard. Then programming could be done with no line numbers at all, although programmers often continued to use them especially for the notoriously ugly Format statements, which specified the print formats. However, Fortran still suffered from lack of dynamic memory allocation. So for example, if one wrote a general program to solve a partial differential equation, one had to dimension the arrays for the largest possible problem, because the data sizes could not be modified after the program was compiled. The original Fortran also did not support recursion. Some Fortran 77 compilers did support recursion, others did not. The Fortran 90 and 95 versions modernized the language to support dynamic memory allocation, free format statement specification, and object oriented programming. The new version also had support for pipeline computing, direct matrix computations, and vector computations. And Fortran 90/95 had general support for supercomputers and for parallel computing. To my knowledge, to this day, this is the only language that has this support for high performance scientific computing.

Other languages and tools for scientific computing, such as Matlab and Maple, as well as tools for creating scientific documents, are treated below.

2 Programming Editors

A very important tool for writing scientific programs is a good programming editor. A programming editor must be capable of displaying line numbers.

Windows

Notepad++ This is a very good programming editor and is the one I reccomend. It is free and easy to download and install.

`http://sourceforge.net/projects/notepad-plus/`

TextPad A good Windows editor. Shareware, Textpad 4.4 from Helios Software Solutions.
Winedit A good Windows editor circa 1995. But rarely it has introduced an invisible change that causes a compiler error, due to some property of the file.

Winedt This is not winedit. This is an editor specifically designed for TeX and LaTeX, but is also a good general purpose editor. It has a terrific spell checker, and can execute Latex from inside the program window. A shareware program that after a period generates an irritating popup that suggest paying the nominal charge.
Notepad A useable, but below average editor, that is supplied with Windows.

Linux

vi A powerful, but hard to learn editor.
gedit A good graphical editor.
Emacs A powerful editor, but also hard to learn.

MacOSX

Textedit The default Mac editor. Be careful of a wordprocessing mode not suitable for programming.
Nano
vi
Emacs

3 Some Scientific Programming Tools

Versions of the book **Numerical Recipes** have associated subroutine libraries:

```
Numerical Recipes in Fortran 77
Numerical Recipes in Fortran 90 and 95
Numerical Recipes in C
Numerical Recipes in C++
Numerical Recipes in Pascal
Numerical Recipes in Basic
Numerical Recipes in Lisp
```

Other scientific subroutine and function Libraries include:

```
IMSL (International Mathematics and Statistics Library) Fortran
GNU Scientific Library (C)
IBM Scientific Library (Fortran)
Lau's Numerical Library in C
C Mathematical Function Handbook Library, Louis Baker (From NBS Handbook)
SLATEC common math library F77
Linpack (Numerical Linear Algebra)
EISPACK (numerical computation of eigenvalues and eigenvectors of matrices, writte
ELLPACK (High level, portable system for solving elliptic boundary value problems)
Emery Library (Fortran, C, C++, Pascal)
```

Languages and Computational Programs

```
Microsoft Visual C++
Microsoft Power Station, Fortran, Digital Equipment Corporation took over this
 compiler, then whoever took over DEC, Compaq? maintained it.
Microsoft Visual Basic
Borland BCC32, c++ download freecommandlinetools.exe
Lahey Fortran 77 An old compiler. I don't know if Lahey still exists.
There is a Watcom C++ and a Watcom Fortran, from Waterloo University.
GNU compilers (free):
 gcc, g++  ,C, C++
 f77  Fortran 77, Is on most Linux systems.
 g95  Fortran 95, The download file for the GNU Fortran 95 compiler
       for windows, is g95-MinGW.exe. MinGW is a kind of Unix shell
       that runs on Windows.
Python (free http://www.python.org/download/)
Java   (free download)
Perl   (free download)
Scheme (free download)
Lisp   (free download)
Mathcad
Matlab
Octave (free download, near clone of Matlab)
Maple
Mupad  (free download Mupad_light, near clone of Maple))
maxima (free download http://sourceforge.net/projects/maxima/files/)
Mathematica
Excel
```

and many many more.

Programs for Graphics, Plotting, Modelling, Designing

Ghostscript and Ghostview Postscript viewers.
Emery eg plotting language Emery eg2ps.c (eg to postscript).
Emery pplt, pltc, pltax, pltmerge, ... (function plots)
Flux Player. (VRML Viewer)
GNU Plot
Matlab, Octave
AutoCad
TurboCad
DesignCad.
AVS (Scientific Display of Data, Triangulated Models, ...)
Blender (Geometric Modelling)
Sketchup (Geometric Modelling)
Excel
OpenCV. (Image Processing)
Cgal (Geometric Modelling)
InkSkape (Vector Graphics)
Gimp (Raster Graphics)
PhotoShop

Programs for Signal Processing, Audio, and Music

Goldwave
Audacity
Cakewalk
WinAmp
The MatLab Signal Processing Toolbox

Some Programs for Writing and Documenting

TeX, LaTeX,
PCTeX (commercial)
MikTeX (free http://miktex.org/2.8/)
LaTeX Online Equation Editor:
http://www.codecogs.com/components/equationeditor/equationeditor.php
Microsoft Equation Editor
MathCad

Various Utilities

pplt, lsq, pltc, pltmerge, pltax, addtext, f2cpp, ...

4 Using the Command Line

4.1 Using the Command Line in Windows Systems, Adding a Directory to the Path

If you know the name of a command, windows will give information on using it when the word help is typed followed by the name of the command. For example

```
help sort
help copy
help mkdir
help dir
help xcopy
help rename
help type
help cd
help set
help prompt
help path
help erase
help tree
```

To find the names of commands, the names and a short description can be placed in the notes.txt file.

To start a command line window select **start**, **run**, type **cmd** in the box and click **OK**. Go to the root directory of drive C

```
C:
cd \
```

Change to the directory called **bin**

```
cd bin
```

If there is no such directory, create a directory called **bin** where we will place some programs to run

```
mkdir bin
```

We will also need a directory called **txt** and a directory called **tmp**. So do

```
mkdir txt
mkdir tmp
```

View the current path, which is a list of directories, that are searched by the operating system for commands and programs to run. Type

```
path
```

So if you type a command, say **gojumpinthelake** chances are good that the operating system will say something like command not found, because no such program or batch file of that name was found in any of the list of directories defined in **path**.

If **bin** is not in the path, it must be added to the path. **Path** is an environment variable. To set the **path** environment variable select **start**, **control panel**, **system**, **advanced**, **environment variables**, scroll down to highlight **path**, **edit**. Scroll to the right, and type at the end of the **path** string

```
;c:\bin
```

or to make it the first directory in a search for a command or executable, add this to the front of the path string

```
c:\bin;
```

Now when a new command line window is opened, **bin** should be in the path. Check this by opening a new command line window and type

```
path
```

An example of searching for C programs in the current directory and all subdirectories:

```
dir *.c /s
```

4.2 Redirecting Output, and Piping Commands

If a command or program writes to the screen then a greater than sign can be used to redirect the output to a file. For example suppose you want to save the information displayed when you type

```
help sort
```

So

```
help sort > myfile
```

writes this information to the file myfile. Again

```
dir > myfile
```

writes a copy of the list of files in the current directory to myfile.

The output of one command can be piped to a second command as in

```
dir | more
```

So this makes the output of command dir go to the input of command more. Command more displays output one screen at a time, pausing until a space is typed, which causes the next page to be displayed, or Q is typed, which causes more to quit.

4.3 Some Windows Commands

```
sort
copy
mkdir
dir
xcopy
rename
type
cd
set
prompt
path
erase
tree
```

```
more
pwd  (print working directory, added exe command)
cp   (copy added bat command)
mv   (rename added bat command)
rm   (erase added bat command)
s    (start added bat command)
cat  (type added bat command)
notepad
start file (starts an appropriate application for a file, like clicking in windows
d: (change to disk drive d)
dir *. (list all subdirectories)
```

4.4 The Command Line in Linux

To start the command line in Linux Unix or MacOS we open a terminal or console session.

To show the current directory type **pwd** (for Print Working Directory)

To list the files in the directory

ls -la

An example of searching for C programs in the current directory and all subdirectories:

```
find . -name "*.c"  -print
```

4.5 Some Linux Commands

```
cp joe.txt moe.txt (copy files)

cd ~  (change directory to user home directory)
cd /   (change to the computer root directory)
cd ~/c++ (change to the users c++ directory assuming one exists)
mkdir c++ (create a c++ directory)
```

4.6 The Command line in MacOS

To start a terminal go to the utilities menu by selecting the **go** item.

Editors, nano, TextEdit, vi

To start a program from the command line

open -a name-of-application

An example of searching for C programs in the current directory and all subdirectories:

```
find . -name "*.c"  -print (finds names of  files in current directory and subdirectories)
```

The dot means start in the current directory. The -name indicates the filename or names to search for. The double quotes around a file specification containing a wild card such as in c.* is apparently necessary, but would not be necessary for a single file specification such as myfile.txt. The -print specifies that the location and filename found should be printed. This seems redundant, but one can direct find to do other things such as execute the found program.

4.7 Some MacOSX Commands

When a Postcript file is opened from the command line with the command

```
open myfile.ps
```

the file is converted to a pdf file and opened in program **preview** At that point one can save the file as a pdf file. So this is a technique for converting a ps file to a pdf file.

```
open -a myapplication (runs program myapplication)
find . -name "*.c"  -print (finds names of  files in current directory and subdirectories)
open -e filename  (open a file in the editor TextEdit)
open filename      (runs a file with an application}
open p.ps          (converts the postscriptfile to a pdf file and displays it.)
gcc whatever.c             (compiles a c program called whatever.c and produces a.out)
./a.out            (run the program a.out, ./ means in this directory, no permission to run as just a.out)
cp a.out ~/bin/name (copy program a.out to whatever in your bin directory)
cd ~                (change directory to your home directory)
pwd                 (displays the name of you current directory, stands for: print working directory)
ls -la               (displays the files in the directory in a ling format)
mkdir bin            (makes a bin directory as a subdirectory of the current directory)
cd /                ( change directory to the machine root directory)
wc myfile    (size of myfile)
cat myfile | more (list file myfile stopping at each page, type q to quit, space to continue)
cd ..             ( go to next higher directory)
```

4.8 Common Troubles in MacOS

The prototype for the C function **atof** resides in the header file stdlib.h for the compiler gcc. This differs from some other compilers. If

```
#include <stdlib.h>
```

is not present in a program that calls the function atof, which converts a string to a floating point number, a bogus value will be returned, thus probably screwing up the program. And even worse, the compiler remains mute on this problem.

When a zip file is downloaded, the operating system will deal with it automatically and perhaps inappropriately, by uncompressing the zip and making unwarranted assumptions.

It is not clear how to start a second terminal session when one terminal session is already running. Clicking on terminal again just returns to the first terminal session?

MacOS is a Unix system and file names are case sensitive, so this can cause problems when downloading and uploading Window-Dos files.

On a Unix system like MacOS, the file end of line character is an ascii 10, which is the lf linefeed character. On old Mac systems the end of line character was an ascii 13, a carriage return character cr. And on Windows systems the end of line is indicated by two characters 13 and 10 , a crlf combination. To check a file for its end of line characters use my program **recend.c**, and also one of my programs to convert end of line types.

On some files produced by DOS, there is a control-z to indicate end of file, this is an ascii 26 character. Some programs on Unix systems don't like this character.

When the Editor TextEdit is opened it assumes the document format. So select format and choose text to switch to a text format for a program or for a data file.

5 Using grep, Creating and Using a Notebook to Document and Recall Programs and Commands

The serious users of computers must know and remember a number of commands, procedures, programs, and algorithms. If one uses a given set of

commands daily, they can be remembered. But everyone needs to refer to notes from time to time. This is especially true when learning new material. Therefore one should keep a detailed physical notebook where commands are written down, and also one should have a computer notebook. A computer notebook can be nothing more than a text file. Items can be found using the search capability of a text editor. However it is rather time consuming and tedious searching in this manner.

5.1 Using grep

The unix program **grep** is a program that searches for text strings in a file. Technically it is a bit more than that. It searches for **regular expressions**. Regular expressions are discussed in various computer science books and in some books on unix. We can consider it to be a program that searches a text file for strings. So suppose we wanted to search a file called **foo.txt** for the word "unix." We can do that with the command:

```
grep unix foo.txt
```

The command:

```
grep unix *.txt
```

would search all txt files for "unix."

Further we could refine the search to lines that start with "unix," and set various other search conditions.

The grep command is native to Unix operating systems (thus MacOS and Linux). Versions exist for all computers.

```
grep
 start of line=^
 end  of line=$
 all lines that don't match pattern -v
 ignore case -n
 grep -v ^c p.eg > q.eg  (remove pen color change in p.eg )
```

I shall present a Windows batch file and a Unix script for searching a notebook.

5.2 Adding A Directory to the Path in Windows, Linux, and MacOS

We shall place executable programs in a directory called **bin**. In order for an operating system to execute a command or program it must know where to look. This is the role of the **path** environmental variable. So when we create a new directory containing programs, we must add the directory name to the path. This can be done in windows in a couple of ways. One can open the control panel, select system, advanced, environmental variables and edit the path variable. Say add

```
c:\bin;
```

to the front of the path string, in the case of adding a directory called bin. A path can also be set in windows as in:

```
set path=%path%;C:\python26
```

One used to set the path on a PC in the autoexec.bat file which was run at startup. I am not sure of the way autoexec.bat operates in modern windows systems.

In the case of a Unix system like MacOS or Linux, we can add a directory to the path like this

```
PATH=$PATH:~/bin
```

This adds the bin directory to the end of the path string. We can do this in an initialization file such as in **.profile** located in /etc, or alternately we can do this in a local **.profile** located in your home directory. Note that a file starting with a dot is invisible normally. A local **.profile** may not exist, but can be created. Here is an example of a local .profile for MacOS:

```
export PS1="$ "
PATH=$PATH:~/bin
```

5.3 The Windows Version of Notes

```
@echo off
rem notes.bat, Version 12/11/09, by Jim Emery, modified the help information.
rem looks in file  /txt/notes.txt
rem lists those lines in file \txt\notes.txt containing specified words
if "%1"==""" goto help
```

```
rem grep2 -i %1 c:\je\\txt\notes.txt > \tmp\tmp1.txt
grep2 -i %1 c:\txt\notes.txt > \tmp\tmp1.txt
copy \tmp\tmp1.txt \tmp\tmp2.txt
:search
shift
if "%1"==""" goto list
grep2 -i %1 \tmp\tmp2.txt > \tmp\tmp1.txt
copy \tmp\tmp1.txt \tmp\tmp2.txt
goto search
:list
semi2nl \tmp\tmp1.txt \tmp\tmp2.txt
type \tmp\tmp2.txt | more
goto end
:help
echo notes.bat, by jim emery, Version 12/12/2009,
echo Lists each line in the file \txt\notes.txt that
echo contains all of a set of specified text strings.
echo ...
echo Example: Suppose notes.txt contains the lines:
echo ...
echo The capital of colorado is denver.
echo The capital of New Jersey is Trenton.
echo The capital of California is Sacramento.
echo Jim Emery; 6021 Wornall Road; kansas city MO, 64113; (816)444-7895; jdemery1@yahoo.com
echo ...
echo The command:
echo notes capital
echo will list all lines containing the word capital.
echo The command:
echo notes capital jersey
echo will list all lines containing both capital and jersey.
echo ...
echo The parts of a line that are separated by semicolons are
echo output on separate lines.
echo So the command:
echo notes emery yahoo
echo will list
echo jim emery
echo  6021 Wornal Road
echo  Kansas City MO, 64113
echo  (816)444-7895
echo  jdemery1@yahoo.com
echo ...
echo Usage: notes string1 string2 string3 ...
:end
```

This batch file uses **grep**, and a program called **semi2nl** that converts a semicolon to a newline character. Also it searches a text file called

`c:\txt\notes.txt`

And it writes some temporary files to directory

`c:\tmp`

So these two directories need to be created if they don't yet exist.

```
cd \
mkdir tmp
mkdir txt
```

This batch file can be modified to search a different txt file in any location by changing the line

```
grep2 -i %1 c:\txt\notes.txt > \tmp\tmp1.txt
```

The program **grep2** is just a renamed version of some **grep** program, because there are several versions of grep on my computers, and I have selected one of them by renaming it **grep2**. Some versions of **grep** on windows work a little differently than others, that is for example if one issued the command

```
cd \
grep joe *.txt
```

one version of **grep** might only return lines containing the string "joe" while others might give information on the name of the file that it came from. The latter is useful if one is searching for all files containing "joe."

There is a quit similar batch file that uses **perl** in place of **semi2nl** to separate the parts of a line. This requires that the **perl** compiler be present. So a **perl.exe** version can be copied to the **bin** directory.

5.4 The MacOS and Linux Version of Notes

On Unix type systems **perl** is always present. So here is a version of **notes** for Linux or MacOS.

Here is the Unix shell script **Notes.sh** :

```
#notes.sh modified 12/18/2009
if [ $# -eq 0 ]
 then
  echo nts.sh ,Version 6/19/08, notes are in file ~/txt/note.txt
  echo each item is on one line of the file notes.txt, each field
  echo separated by a semicolon. each field will be printed on a separate line
  echo returns each address item that contains all of the words on the command line
  echo Example: nts book mechanics
  echo returns lines containing each of the two words book and mechanics
  echo Usage: nts  word1 word2 word3 ........
  exit
```

```
 fi
 # echo number of arguments $#
 cp ~/txt/notes.txt tmp1
 while [ $# -gt 0 ]
   do
    grep -i $1 tmp1 > tmp2
    cp tmp2 tmp1
    shift
   done
  perl ~/bin/splitnotes.pl < tmp1 > tmp2
  cat tmp2 | more
```

Here is the Perl script **splitnotes.pl** used in **notes.sh**:

```
#!/usr/bin/perl
# this program is used in getaddres.bat without an argument
# so if there is an argument, that is, if $#ARGV is 0 or greater,
# then print info.
 if($#ARGV > -1  ){
  print("split1.pl, Version 8/13/06, Jim Emery.\n");
  print("Split each input line into fields delimited by a semicolon.\n");
  print("Usage: perl split1.pl  < infile > outfile \n");
  exit
 }
# print $#ARGV,"\n";
while(<STDIN>){
# print "\n";
 chomp($_);
 @fields = split(/;/,$_);
 $n=@fields;
# print " number of fields =", $n,"\n";
# print @fields,"\n";
 for($i=0;$i<$n;$i++){
  print @fields[$i],"\n";
 }
}
```

5.5 Installing Notes on Windows

We need to create directories called

```
c:\bin
c:\txt
c:\tmp
```

We need to add the bin directory to our path. We need to copy or create a file called notes.txt and place it in the txt directory. We need to place the batch file notes.bat in the bin directory. we need to have the required grep programs in the bin directory, and we need the program semi2nl in bin.

5.6 Installing Notes on Linux

We need to create a directories called

```
~\bin
~\txt
```

off of our home directory, which is indicated by the tilda.

We need to add the bin directory to our path. We need to copy or create a file called notes.txt and place it in the txt directory. We need to place the script file notes.sh in the bin directory, without the sh extension. We need to give it execution permissions. we need to have the required grep programs in the bin directory. Perl should be already present.

5.7 Installing Notes on MacOS

The installation for MacOSX is about the same as for Linux.

We need to create a directories called

```
~\bin
~\txt
```

off of our home directory, which is indicated by the tilda.

We need to add the bin directory to our path. We need to copy or create a file called notes.txt and place it in the txt directory. We need to place the script file notes.sh in the bin directory, without the sh extension. We need to give it execution permissions. we need to have the required grep programs in the bin directory. Perl should be already present.

6 Graphing and Programmable Calculators

We can use calculators such as the: TI82, TI83 TI84, the HP whatever, or the Casio whatever, to do a little root finding.

Newton's method can be done on a calculator. Suppose we want to find the zero of a function $f(x)$. We want to find an x that makes the function zero. So write a Taylor series for the function including only the first order term

$$f(x) = f(x_0) + \frac{df}{dx}(x_0)(x - x_0).$$

So

$$f(x_0) + \frac{df}{dx}(x_0)(x - x_0)$$

is the best straight line approximation to $f(x)$. So we can estimate for a zero of $f(x)$ by finding where this straight line is zero. So solve for x in the formula

$$0 = f(x_0) + f'(x_0)(x - x_0).$$

Then

$$\frac{-f(x_0)}{f'(x_0)} = x - x_0.$$

Then our linear estimate of the zero is

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

If x is not a zero, then set $x_0 = x$ and repeat the technique. It can be shown that the convergence is quadratic. This means that if h is the distance from the estimate of the zero to the actual zero, then this distance will decrease to the order h^2 on the next Newton iteration.

So as an example of this technique suppose

$$f(x) = a - x^2.$$

If x is a zero of this function, then $x = \sqrt{a}$. So finding a zero is the same as finding the square root of a. Now

$$\frac{df}{dx} = -2x$$

So Newton's method applied to this example gives

$$\begin{aligned} x &= x_0 - \frac{a - x_0^2}{-2x_0} \\ &= x_0 + \frac{a}{2x_0} - \frac{x_0}{2} \\ &= \frac{x_0 + a/x_0}{2}. \end{aligned}$$

So to find the square root of a number a, start with any estimate x_0. Divide a by x_0, then add this to x_0 and divide by 2. This is the next

estimate. Continue until it converges to the square root where the estimates no longer change. One can do this on a calculator. A similar formula can be devised for the cube root. We need to store and recall the value of x_0 as we calculate.

Note. Given a function $g(x)$ of x. If $g(x) = x$, then x is called a fixed point of the function. To calculate the square root of a we are using the function

$$g(x) = \frac{x + a/x}{2},$$

and when the iteration converges we have

$$x = g(x) = \frac{x + a/x}{2}.$$

Solving this equation we find that $x = \sqrt{a}$. So actually we are locating a fixed point of $g(x)$ by iteration.

This can be done in Python in about the same way as on a calculator.

```
#squareroot.py
a=5.
print " Computing the square root of  = ",a
x=2.
count=1
while count < 8 :
  print " x=", x
  x=(x+a/x)/2
  count=count+1
```

To run the program from the command line, type

```
python squareroot.py
```

and get the output:

```
Computing the square root of  =  5.0
 x= 2.0
 x= 2.25
 x= 2.23611111111
 x= 2.23606797792
 x= 2.2360679775
 x= 2.2360679775
 x= 2.2360679775
```

7 Plotting a Function With Utility Programs.

Here we show how to create a **Postscript** and a **pdf** version of a function plot file. The plots can be viewed with Ghostview and Acrobat Viewer, or the Postscript file can be placed in a Latex document. The plot data comes from a file containing x-y pairs. These pairs are usually generated from some program. Below in the Python section we give a program **writefunc.py** that writes function pairs to a file. We introduce here some utility programs that will make a labeled plot from such file of function pairs. These utilities should be placed the **bin** directory that you created. If the name of a utility is typed without any parameters, then help usage information is provided. You might place the names of the utilities that you use regularly in the **notes.txt** file, which should be placed in the **txt** directory that you created.

We can create the function points for a function plot in various ways. One can write a program to do this in some computer language, such as Fortran, C, C++, Python, et cetera. Also points could be generated in some program such as Excel, Matlab, Maple, et cetera. These utilities are placed in a directory where the operation system can find them, for windows

```
c:\bin
```

To use the utilities, **bin** must be in known to the operating system via the **path** environmental variable. For Linux or MacOS they can be in the bin subdirectory off of the users home directory

```
~/bin
```

Here is a description of the utilities used for plotting:
Program **pnts2eg.c** converts a file containing x and y values to an eg graphics file. The second line in such a file contains a w command, that is a window command that determines the boundary of the graphics. This can be changed directly in an editor to alter the size of the plot.
Program **pltc.ftn** plots points from a file. It is an older program, and can also plot symbols for points. Also it can plot more than one function contained in the file. See the help information displayed by the program.
Program **pltax.c** adds labels and axes to an eg plot.
Program **pltmerge.c** can merge 2 or more eg files. So for example it could be used to combine two function plots into a single plot.

Program **eg2ps.c** converts an eg file into a Postscript file, which can be merged into a Latex document, or converted to a pdf file with tools such as **ps2pdf** or Acrobat Distiller.
Some Programs for Plotting:

```
writefunc.py
pnts2eg.c
pltc.ftn
pltax.c
pltmerge.c
eg2ps.c
ps2pdf
```

In windows, you could set up a batch file called doit.bat that would do the plotting with one command. Such a batch file might have content somthing like:

```
pnts2eg %1 q.eg
pltax q.eg p.eg "x label" "y label" "title information"
eg2ps p.eg p.ps
ps2pdf p.ps p.pdf
```

File **p.pdf** is the final result. So to run this on a file mypoints.txt, you would type

```
doit mypoints.txt
```

On the other hand, one could type each of the utility commands once. Then to repeat them use the up arrow on the keyboard repeatedly.

7.1 Viewing Postscript Graphics With Ghostview and Ghostscript.

The simple eg language is convenient for generating plots, graphs, and figures in a programming language. From am eg plot file generated by such programs it is very convenient to convert them to postscript because Postscript is nicely included in Latex documents. To do this conversion one may use the C program **eg2ps** . Ghostview is a postscript viewer and can send output to standard printers. Ghostview is a free program obtained from the GNU website. Recall that Postscript is a programming language for displaying graphics. A nice feature of Ghostview is that if it is left open while displaying a postscript file, such as say a file called **p.ps**, then whenever the version of **p.ps** is changed, i.e. by the program that generated **p.ps**, Ghostview

will update the display of **p.ps**. This is a great time saver. Another big advantage of converting to Postscript is the further conversion to a related pdf file. This can be done in various ways. One can use Acrobat Distiller, the ps2pdf program supplied with MiKTeX, or with the built in conversion found on MacOS systems.

7.2 Converting Postscript to pdf

Postscript can be converted to pdf in various ways. One can use Acrobat Distiller, the ps2pdf program supplied with MiKTeX, or the built in conversion program found on MacOS systems.

8 Converting Between Text Point Files and Excel CSV Files (Comma Separated Values).

Numbers contained in an Excel array can be saved as csv files, these are files where the numbers in the cells in a row are separated by commas. Many programs want to see such numbers separated by just spaces. So we have a little utility that converts csv to ssv (space separated values). **csv2ssv.c**. We also might want to go the other way to put space separated numbers back into Excel by using **ssv2csv.c**. The program **cdelim.c** also will convert a csv file to a ssv file. The program **trnsf.c** can scale data and select a subset of points of a file. The program **mvcols.c** can change the order of the columns in a file.

9 An Example of Scientific Programming: Solving a Differential Equation Numerically

Consider the differential equation

$$\frac{d^2w}{dx^2} + w = 0.$$

This is an homogeneous second order equation with constant coefficients, easy to solve. A general solution is

$$w(x) = c_1 \cos(x) + c_2 \sin(x),$$

where c_1 and c_2 are constants determined by the initial conditions. Suppose at $x = 0$

$$w(0) = 0,$$

and

$$\frac{dw}{dx}(0) = 1.$$

Then $c_1 = 0$ and $c_2 = 1$ so the solution is

$$w(x) = sin(x).$$

Let us solve this equation numerically. To do so, we must convert the second order equation to a system of two first order equations. So let

$$y_1(x) = w(x),$$

and

$$y_2(x) = \frac{dw}{dx}(x).$$

Then

$$\frac{dy_1}{dx} = \frac{dw}{dx} = y_2$$

and

$$\frac{dy_2}{dx} = \frac{d^2w}{dx^2} = -w = -y_1.$$

So the system of first order equations to be solved is

$$\frac{dy_1}{dx} = y_2$$

$$\frac{dy_2}{dx} = -y_1.$$

The initial conditions at $x = 0$ are

$$y_1(0) = 0$$

$$y_2(0) = 1.$$

```
c  rk4je.for
      implicit real*8(a-h,o-z)
      dimension y(2),dydx(2),yout(2)
      external der1
c     starting value of independent variable:
      x=0.
      pi=3.14159265358979
c     ending value of independent variable:
      xend=pi/2.
c     approximate step size:
      h=.001
c     number of steps:
      m=(xend-x)/h
c     adjusted step size:
      h=(xend-x)/m
c     approximate number of print points:
      npa=15
c     steps between prints:
      np=max(m/npa,1)
      n=2
      y(1)=0.
      y(2)=1.
      write(*,'(i6,3(g21.14,1x))')0,x,y(1),sin(x)
      do i=1,m
c      calculate derivatives of y at x
       call der1(x,y,dydx)
       call rk4je(y,dydx,n,x,h,yout,der1)
c      update values:
       x=x+h
       y(1)=yout(1)
       y(2)=yout(2)
       k=mod(i,np)
       if((k .eq. 0) .or. (i .eq. m))then
        write(*,'(i6,3(g21.14,1x))')i,x,y(1),sin(x)
       endif
      enddo
      end
c+ rk4je runge-kutta differential equation step
      subroutine rk4je(y,dydx,n,x,h,yout,derivs)
      implicit real*8(a-h,o-z)
c     modified version of rk4 (numerical recipes)
c  input:
c  y      initial values of vector function
c  dydx   initial values of vector derivative
c  n      number of first order equations
c         y, dydx are n vectors.
c  x      independent variable
c  h      independent variable step
c  output:
c  yout   value of vector function after step h
c  derivs name of external subroutine that
c         calculates vector derivative.
c  Equation that is solved:
c  dy/dx = f(x,y), where f is defined by subroutine derivs
      parameter (nmax=20)
      dimension y(*),dydx(*),yout(*),yt(nmax),dyt(nmax),dym(nmax)
      external derivs
```

```
      hh=h*0.5
      h6=h/6.
      xh=x+hh
      do i=1,n
       yt(i)=y(i)+hh*dydx(i)
      enddo
      call derivs(xh,yt,dyt)
      do i=1,n
        yt(i)=y(i)+hh*dyt(i)
      enddo
      call derivs(xh,yt,dym)
      do i=1,n
       yt(i)=y(i)+h*dym(i)
       dym(i)=dyt(i)+dym(i)
      enddo
      call derivs(x+h,yt,dyt)
      do i=1,n
       yout(i)=y(i)+h6*(dydx(i)+dyt(i)+2.*dym(i))
      enddo
      return
      end
c+ der1 a first order diff equation definition
      subroutine der1(x,y,dy)
      implicit real*8(a-h,o-z)
c     Second order differential equation
c     d^2 w/dx^2 + w = 0,
c     Initial conditions: w(0)=0, dw(0)/dx = 1
c     Exact solution: w(x) = sin(x)
c     To convert to a first order system of two equations
c     let new variables y_i equal the derivatives:
c     y_1=w, y_2= dw/dx
c     Then we get a system of two first order equations:
c     d y_1/dx = f_1(x,y) =  y_2
c     d y_2/dx = f_2(x,y) = -y_1
      dimension y(*),dy(*)
      dy(1)=y(2)
      dy(2)=-y(1)
      return
      end
```

10 Scientific Programming in Python

Python is a scripting programming language. One can use it in interactive mode as a calculator for computing formulas. Python is started at the command line by simply typing "python." Python is usually installed on Linux and Unix systems including MacOS. For windows it is easily obtained and installed. Python natively supports complex numbers, and integers of arbitrary length, provided they are designated with the suffix "L".

After Python version 26 is installed on Windows, to add this directory to the path

```
set path=%path%;C:\python26
```

So suppose one wants to compute a point on an ellipse of major axis 10 and minor axis 6 at angle $\theta = 1$.

$$x = a * \cos(\theta),$$

$$y = b * \sin(\theta.$$

You could try this in python:

```
a=5
b=3
theta=1
x =a*cos(theta)
y= b*sin(theta)
```

But this would produce an error because the mathematical functions require that the math module be imported. But it will work after importing the math module:

```
import math
a=5
b=3
theta=1
x =a*math.cos(theta)
y= b*math.sin(theta)
print "{", x,y,")"
print x*x/(a*a)+ y*y/(b*b)
```

So a good place to start learning this language is to read the tutorial supplied with Python. Python is a free function type interpreter that is available for most computers. Python is an extension to operating system scripting languages like those found on Unix systems. Python can be operated in interactive mode much like a calculator. Some features that make it useful for scientific calculation are that there is no requirement to declare variables. Complex numbers are supported. Integers of arbitrary length are also supported. However, arrays are not part of the language, so computing with matrices requires an external package be imported. This is available but may or may not be awkward. Commands can also be stored in script files and then imported and executed. This prevents the interactive commands from being lost. Mathematical functions such as sin, cosine and so on are available in the math package. So if they are to be used one must import the module:

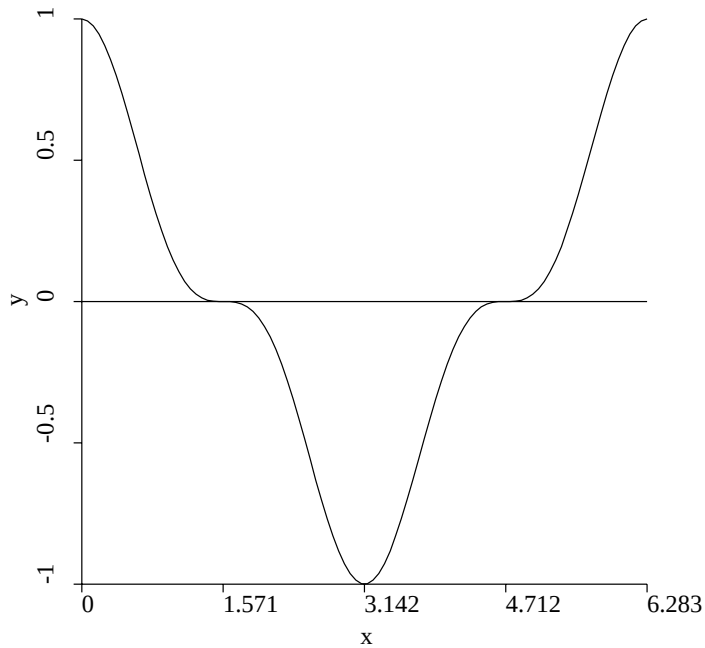


Figure 1: **Function Plotting:** The function plotted here is $f(x) = \cos^3(x)$. It was created with the sequence of programs writefunc.py, a Python program to generate function points, pts2eg.c, a C program to create an eg graphics file, pltax.c, a C program to add axes to the plot, and eg2ps.c, a C program to convert from eg to postscript. These can all be put in a single unix script file, or in a single windows batch file. The final file is beautifully displayed with Ghostview, and converted to pdf with Acrobat distiller or with the ps2pdf program, which comes with MikTex, or is installed on Linux systems, if TeX is installed. A couple of other useful programs here are pltc.ftn, a Fortran Program to create an eg graphics file, and pltmerge.c, a C program to merge together eg files.

import math

then to use one of the functions say exp access them like this:

math.exp(1)

blocks are indicated with indentation rather than with an end statement, or with a curly bracket as in most languages. So for example

```
while n < 10 :
 print n*n
 n= n+1
print " out of while loop"
```

So the while ends when the indented statements after the while end.

Functions are defined with a def statement, like this:

```
def newtonsmethod(xstart)
  x1=xstart
  "f(x2) = f(x1) + (x2-x1)*df(x1)=0"
  -f(x1)/df(x1) = x2 - x1
  x2 = x1 - f(x1)/df(x1)
  print x2
  x1=x2
```

When a script is imported, the script is run. Depending on the script, this may change the local environment, so that on reimporting reloading, may not run the same way, (local variables changed and so on). So it may be required to shut down python (cntrl z on windows, cntrl d on unix.) and start again.

10.1 Example: Writing Function Points to a File

```
#writefunc.py write the x and y pairs of a function to file outfile.txt
import math
#define function
def f(x):
 y=pow(math.cos(x),3)
 return y
file1=open('outfile.txt','w')
n=100
xmn=0.
xmx=2.*math.pi
for i in range(1,n+1):
 x=(i-1)*(xmx-xmn)/(n-1) + xmn
 y=f( x)
 s=repr(x)+" "+repr(y)+"\n"
 file1.write(s)
file1.close
```

10.2 Example: Solar Heating, Using Command Line Arguments

Here is a program to compute the time it takes to heat a gallon of water from temperature t_1 to temperature t_2. The temperature is in centigrade. It is assumed that all solar radiation is absorbed, and no heat is lost. Note that a pound sign # specifies a comment. Indentation specifies a block of code. When a block such as an if block, a for block, and so on is started, indented lines are part of the block. Brackets are not used to define the block as in C, or "begin end" pairs as in Pascal, or "do enddo" as in Fortran.

```
#solorheating.py, The time to raise a gallon of water from temperature t1 to t2
#with 1 square meter of solar radiation.
import sys
print sys.argv
print sys.argv[0]
n=len(sys.argv)
print 'number of arguments = ', n
if n < 3 :
 #An indented line starts the content of the if statement
 print "solarheating.py, The time to raise one gallon of water from t1 centigrade degrees"
 print "to t2 degrees, using one square meter of solar radiation, and assuming all"
 print "radiation is absorbed and no energy lost."
 print "Usage: python solarenergy.py t1 t2 "
 sys.exit(0)
t1=float(sys.argv[1])
t2=float(sys.argv[2])
#
#Define function.
def f(t1,t2):
 # Time in seconds to raise one gallon of water from t1 C to t2 C
 # Cubic inches in a gallon 231
 # Cubic centimeters in a cubic inch 2.54^3
 # Heat required to raise a cubic centimeter 1 degree = 1 calorie
 # Joules in a calorie 4.184
 ccg=231*pow(2.54,3)   # Cubic centimeters in a gallon
 j=ccg*4.184  #Joules needed to raise one gallon 1 degree
 sc=1366  # Solar constant watts /m^2
 t=j/sc #Time to raise 1 gallon of water 1 degree, with 1 meter^2 of solar radiation
 t12 = t*(t2-t1) #Time to raise 1 gallon of water from t1 degrees to t2 degrees
 #print t12, " seconds"
 return t12
v=f(t1,t2)
print "The time to raise 1 gallon of water from ", t1, " degrees centigrade to "
print t2 , " degrees centigrade,", " with 1 square meter of solar radiation, is "
print v/60, " minutes"
```

The program obtains the temperatures from command line arguments. When we type on the command line

python solarheating.py 25 55

The strings on the command line, namely, solarheating, 25, and 55 are the command line arguments. They are stored in the argument vector. The program can access these three arguments as sys.argv[0], sys.argv[1], and sys.argv[2]. These are strings. The last two strings are converted to floating point numbers using the built in float function. The function atof can also be used.

10.3 Example: Converting degrees, minutes, seconds to a decimal number, using Command Line Arguments

```
#dms2d.py, Convert degrees minutes seconds to a single decimal number
import sys
n=len(sys.argv)
print 'number of arguments = ', n
if( n < 2):
 print "dms2d.py, Convert degree minutes seconds to a single number."
 print "The first number is degrees, the second if present is minutes,"
 print "the third number if present is seconds."
 print "Usage: python dms2d.py d m s "
 sys.exit(0)

d=float(sys.argv[1])
m=0.
s=0.
if (n > 2):
 m=float(sys.argv[2])
if (n > 3):
 s=float(sys.argv[3])
def f(d,m,s):
 # degrees minutes seconds to a single decimal
 y=d + m/60.0 + s/3600.0
 return y

v=f(d,m,s)
print " Answer= ", v
```

The program obtains the degrees minutes and seconds from command line arguments.
Example:

```
python dms2d.py 30. 30. 30.
```

Gives: Answer 30.5083333333

10.4 Example: Reading points from a file

Program **readxy.py** gives an example of reading points from a file. We obtain the file to be read as an argument to the program. So suppose we had the following text file:

```
1. 2.
3. 4.
1.      2.
3. 4.
5. 6.
7. 8.
9. 10.

1. 2.
3. 4.
1. 2.
3. 4.
8.
1 14
7 8 9
5. 6.
7. 8.
9. 10.
```

This file has pairs of numbers separated by spaces. There are a couple of exceptions, one line has only one number, and another has no numbers.

We want to read only the pairs. If there is no filename given on the command line, then information is given about how the program is to be run. Each line of the file is read in a for loop. So the variable a is a string containing each line as it is read. The pairs are split out using the string function split into a list s. If there are 2 or more items in the list, the first two are converted from strings into floating point numbers. There will be more lines read than pairs found, because two lines do not contain pairs. Here is the program:

```
#readxy.py read point pairs, x y separated by spaces, from a file
import sys
#print sys.argv
#print sys.argv[0]
na=len(sys.argv)
#print 'number of arguments = ', na
if( na < 2):
 print "readxy.py, read point pairs, x y separated by spaces, from a file"
 print "Usage: python readxy.py file "
 sys.exit(0)
f1=open(sys.argv[1],'r')
n=0;
lns=0
for a in f1:
```

```
 lns=lns+1
 s=a.split()
 k=len(s)
 if k>1:
  x=float(s[0])
  y=float(s[1])
  n=n+1
  print "x=",x,"y=",y
print n," pairs read"
print lns," lines read"
f1.close
```

10.5 A List of some Python Commands and Their Use

```
dir() (lists current functions)
dir(math) (lists functions in module math)
import math
math.sin(x) (the sin function in the module math)
import sys
sys.path
#!/usr/bin/env python  (in unix pound bang tells system to use python to execute t
a=1.5+0.5j, a.real, a.imag , abs(a) (complex numbers)
hello = "This is a rather long string containing\n\
several lines of text just as you would do in C.\n\
    Note that whitespace at the beginning of the line is\
 significant."
s = 'supercalifragilisticexpialidocious'
 len(s)=34
list: a = ['spam', 'eggs', 100, 1234]
 b=a.append(3.14)
while b < 10:
print 'The value of i is', i
a = ['cat', 'window', 'defenestrate'], for x in a:
for i in range[2,23]:
def mean(a): .... return m (function definition)
list.append(x)
 list.extend(L)
 list.insert(i, x)
 list.remove(x)
 list.reverse()
a=[1, 66.25, 333, 333, 1234.5]
```

```
 del a[2:4]
t = 12345, 54321, 'hello!' ( tuple)
file1=open('outfile.txt','w')
 file1.write(s)
 file1.close (python)
import sys
 print sys.argv[0]
 n=len(sys.argv)
 sys.exit(0)
 t1=float(sys.argv[1])
if( n < 3):
dir() list currently defined names (python)
repr() convert values to strings (python)
f.read() reads entire file
 f.readline() reads a line
for line in f:
  print line
s="1.23 4.56 7.13"
 a=s.split()
 x=float(a[0])
 y=float(a[1])
 z=float(a[2])
for a in file1:
import sys
 print sys.argv
 print sys.argv[0]
 na=len(sys.argv)
 print 'number of arguments = ',na
x=float(s[0])
```

10.6 Vectors

We may store vectors as lists. Here is a program defining some vector operations:

```
#vectors.py, sum, difference, norm, products of vectors
#
import math
#+ vsum sum of vectors
def vsum(a,b):
```

```
 c=[]
 n=len(a)
 for i in range(0,n):
  c.append(0.)
 for i in range(0,n):
  c[i]=a[i]+b[i]
 return c
#+ vdiff difference of vectors
def vdiff(a,b):
 c=[]
 n=len(a)
 for i in range(0,n):
  c.append(0.)
 for i in range(0,n):
  c[i]=a[i]-b[i]
 return c
#+ vsm scalar multiple of vector
def vsm(s,a):
 c=[]
 n=len(a)
 for i in range(0,n):
  c.append(0.)
 for i in range(0,n):
  c[i]=s*a[i]
 return c
#+ vnorm norm of a vector
def vnorm(a):
 n=len(a)
 vn=0.
 for i in range(0,n):
  vn=vn+a[i]*a[i]
 vn=math.sqrt(vn)
 return vn
#+ crsspr cross product of vectors
def crsspr(a,b):
 c=[0.,0.,0.]
 c[0]=a[1]*b[2]-a[2]*b[1]
 c[1]=a[2]*b[0]-a[0]*b[2]
 c[2]=a[0]*b[1]-a[1]*b[0]
 return c
#+ dotpr dot product of vectors
def dotpr(a,b):
 n=len(a)
 s=0.
 for i in range(0,n):
  s=s+a[i]*b[i]
 return s


# main
a=[1.,2.,3.]
print "a=",a
b=[3.,5.,1.]
print "b=",b
s=9.
print "s=",s
aplusb=vsum(a,b)
```

```
print "aplusb=",aplusb
aminusb=vdiff(a,b)
print "aminusb=",aminusb
sa=vsm(s,a)
print "sa=",sa
norma=vnorm(a)
print "norma=",norma
axb=crsspr(a,b)
print "axb=",axb
adotb=dotpr(a,b)
print "adotb=",adotb
norma=math.sqrt(dotpr(a,a))
print "norma=",norma
```

10.7 Formatting Output

In this example we have an example of formating numbers. This is a method applied to strings.

```
#inductance.py
import sys
import math
f=float(sys.argv[1])
print "f=",f," Hertz"
omega=2.*math.pi*f
C=2.0e-6
print " C=",C*1.e6,"microfarads"
L=1./(omega*omega*C)
print " L=",L*1000.," millihenries"
fmin=500.
fmax=2000.
R=1000.
n=30
for i in range(0,n):
 f=i*(fmax-fmin)/(n-1)+fmin
 omega=2.*math.pi*f
 XL=omega*L
 XC=-1/(omega*C)
 print 'i=',i,'f={0:.1f}'.format(f),'XL={0:.4f} XC={1:.4f}'.format(XL,XC)
 #print 'XL={0:.5f}'.format(XL)
#
```

10.8 Creating a Module

Pretty much any Python program containing function definitions is a module. So for example the following program called vec.py can searve as a module for vector oerations.

```
#vec.py, sum, difference, norm, products of vectors
import math
#+ vsum sum of vectors
```

```
def vsum(a,b):
 c=[]
 n=len(a)
 for i in range(0,n):
  c.append(0.)
 for i in range(0,n):
  c[i]=a[i]+b[i]
 return c
#+ vdiff difference of vectors
def vdiff(a,b):
 c=[]
 n=len(a)
 for i in range(0,n):
  c.append(0.)
 for i in range(0,n):
  c[i]=a[i]-b[i]
 return c
#+ vsm scalar multiple of vector
def vsm(s,a):
 c=[]
 n=len(a)
 for i in range(0,n):
  c.append(0.)
 for i in range(0,n):
  c[i]=s*a[i]
 return c
#+ vnorm norm of a vector
def vnorm(a):
 n=len(a)
 vn=0.
 for i in range(0,n):
  vn=vn+a[i]*a[i]
 vn=math.sqrt(vn)
 return vn
#+ crsspr cross product of vectors
def crsspr(a,b):
 c=[0.,0.,0.]
 c[0]=a[1]*b[2]-a[2]*b[1]
 c[1]=a[2]*b[0]-a[0]*b[2]
 c[2]=a[0]*b[1]-a[1]*b[0]
 return c
#+ dotpr dot product of vectors
def dotpr(a,b):
 n=len(a)
 s=0.
 for i in range(0,n):
  s=s+a[i]*b[i]
 return s
# end
```

Here is a program that imports vec as a module:

```
#vectest.py
import vec
# main
```

```
a=[1.,2.,3.]
print "a=",a
b=[3.,5.,1.]
print "b=",b
s=9.
print "s=",s
aplusb=vec.vsum(a,b)
print "aplusb=",aplusb
aminusb=vec.vdiff(a,b)
print "aminusb=",aminusb
sa=vec.vsm(s,a)
print "sa=",sa
norma=vec.vnorm(a)
print "norma=",norma
axb=vec.crsspr(a,b)
print "axb=",axb
adotb=vec.dotpr(a,b)
print "adotb=",adotb
norma=math.sqrt(vec.dotpr(a,a))
print "norma=",norma
```

10.9 Using Eval

The eval function can be used to interpret an expression. Here is an example:

```
#eval.py
import math
def f(x,s):
 y=eval(s)
 return y
x=2
s="x**3+math.sin(math.pi/2.)"
w=f(x,s)
print w
```

10.10 Object Oriented Programming Example

```
class point:
 def __init__(self,x=0,y=0):
  self.x=x
  self.y=y
 def norm(self):
  return math.sqrt(self.x**2+self.y**2)
 def __eq__(self,other):
  return self.x==other.x and self.y==other.y
 def __repr__(self):
  return "Point({0.x!r}, {0.y!r})".format(self)
 def __str__(self):
  return "({0.x!r}, {0.y!r})".format(self)
 def __add__(self,other):
  x=self.x+other.x
  y=self.y+other.y
```

```
  return point(x,y)
a=point()
s=repr(a)
print s
b=point(3,4)
s=repr(b)
print "b=", b
print " b= ",str(b)
print " a is equal to b, is ",a==b
print " a is not equal to b, is",a!=b
c=point("hello","world")
print str(c)
d=point(2,3)
print "d=",str(d)
e=b+d
s=str(e)
print "b+d=",s
```

10.11 A Matrix Class

Here is a Python Class for handling matrices. Matrices are stored as objects. The numbers in the matrix are stored as a list of numbers. Also stored is the number of rows and columns. We can initialize a matrix in three ways. So to use the matrix class, which is stored in the file mat.py, we must first import it.

```
import mat
```

We may initialize a matrix in three ways. A matrix b can be defined by providing a list, the number of rows, and columns

```
b=mat.matrix([1.,2.,3.,4,4,1],3,2)
```

This defines the matrix

$$b = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 4 & 1 \end{bmatrix}$$

with 3 rows and 2 columns. The list in the matrix object is called "a", the rowsize is called "m" and the column size is called "n". Classes in Python are public, so the numbers in the matrix could be changed directly by manipulating the list. But better way is to use the get and put functions.

A second way of initializing a matrix is to define it as a string using semicolons in a manner similar to matlab, where a semicolon indicates a new row. So

```
b=mat.matrix("1. 2.;3. 4;4 1")
```

gives

$$b = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 4 & 1 \end{bmatrix}$$

Notice in the explicit list definition method there are commas, in the string definition there are no commas. Another difference is that in the explicit list definition method the matrix elements are evaluated, but in the string definition they are not. So for example if a python variable x has value 2, then

```
b=mat.matrix([1.,x,3.,4,4,1],3,2)
```

produces

$$b = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 4 & 1 \end{bmatrix}$$

whereas

```
b=mat.matrix("1. x ;3. 4;4 1")
```

gives

$$b = \begin{bmatrix} 1 & x \\ 3 & 4 \\ 4 & 1 \end{bmatrix}$$

The third method creates a matrix using a tuple defining the number of rows and columns.

```
b=mat.matrix((2,2))
```

produces a 2 by 2 matrix of zeroes,

$$b = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Then the elements can be changed using put

```
b.put(1,1,1)
b.put(2,2,1)
```

gives

$$b = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

These statements mean set the element at row 1 column 1 to 1, and set the element at row 2 and column 2 to 1. Similarly the statement

```
y=b.get(2,1)
```

retrieves the element in the second row and first column of matrix b and places it in y.

Matrix addition, multiplication, scalar multiplication, transpose, and inverse are implemented. Operator overloading is done so that we may write matrix multiplication as

```
c=a+b
c=a*b
```

For transpose and inverse we write

```
b=a.transpose()
b=a.inverse()
```

To print a matrix b we may type

```
print b
```

The current version of mat.py may be found in

```
stem2.org/je/py.zip
```

Here is the definition of the matrix class in file mat.py:

```
#mat.py matrix class, 3/27/2010
import math
class matrix:
 def __init__(self,md,*args):
  """ Initializes a matrix
  md - defines the numbers in the matrix.
       It can be a number string, a number list, or a tuple defining the
       matrix size.
  m  - is the number of rows
  n  - is the number of columns.
       m,n may be implied or explicitly given.
       Example of a string definition:
        a=matrix("1. 2. 3.;4. 5. 6.")
```

```
        a semicolon separates rows, so this is a 2 row by 3 column matrix
        with the first row being
        1. 2. 3.
        m and n are determined by the placement of semicolons.
This initialization converts the string elements separated by
spaces to floating point numbers. So if you want the elements of
the matrix to be say complex numbers, or strings, use the list
definition.
       Example of a list definition:
        a=matrix([1., 2., 3., 4., 5., 6.,7.,8.],4,2)
        The second argument 4, is taken as m, the number of rows,
        The third argument 2, is taken as the number of columns.
        So this is a 4 row by 2 column matrix,
        the first row being
        1. 2.
        If only one argument is present the matrix is taken to be a
        matrix with a single row.
   Example of a tuple definition:
        a=matrix((50,17))
The items in the tuple are m,n the size of the matrix.
So in this example the matrix is a 50 row by a 17 column
matrix. When a matrix is initialized with a (m,n) tuple, it
is filled with floating point zeroes. Use the put function
to change an element in the matrix.
   """
  self.a=[]
  if (type(md)==type(""")):
   nl=len(md)
   s=md.split(";")
   m=len(s)
   self.m=m
   for i in range(0,self.m):
    r=s[i].split()
    n=len(r)
    self.n=n
    for j in range(0,n):
     self.a.append(float(r[j]))
  if (type(md)==type([])):
   nl=len(md)
   largs=len(args)
   if largs==2:
    self.m=args[0]
    self.n=args[1]
   else:
    self.m=1
    self.n=nl
   for i in range(0,nl):
    self.a.append(md[i])
  if (type(md)==type(())):
   self.m=md[0]
   self.n=md[1]
   zero=0.
   nl=self.m*self.n
   for i in range(0,nl):
    self.a.append(zero)
 #end init
#++ p create a print string for a matrix
```

```
 def p(self):
  m=self.m
  n=self.n
  s=""
  for i in range(0,m):
   for j in range(0,n):
    s=s + " " + repr(self.a[i*n + j])
   s=s+"\n"
  return s
#++ add  add two matrices
 def __add__(self,other):
  m=self.m
  n=self.n
  s=[]
  for i in range(0,m*n):
   s.append(self.a[i]+other.a[i])
  sum=matrix(s)
  sum.m=m
  sum.n=n
  return sum
#++ mult  multiply two matrices
 def __mul__(self,other):
  m=self.m
  p=self.n
  n=other.n
  s=[]
  for i in range(0,m):
   for j in range(0,n):
    b=0.
    for k in range(0,p):
     b=b+self.a[i*p +k]*other.a[k*n + j]
    s.append(b)
  prod=matrix(s)
  prod.m=m
  prod.n=n
  return prod
#++ get  return an element of a matrix at i,j
 def get(self,i,j):
  n=self.n
  e=self.a[(i-1)*n + (j-1)]
  return e
#++ put put a value in a matrix at i,j
 def put(self,i,j,v):
  n=self.n
  self.a[(i-1)*n + (j-1)]=v
 def transpose(self):
  m=self.m
  n=self.n
  t=matrix((n,m))
  for i in range(1,m+1):
   for j in range(1,n+1):
    v=self.get(i,j)
    t.put(j,i,v)
  return t
 #compute the inverse of a matrix
 def inverse(self):
  m=self.m
```

```
  n=self.n
  #allocate the matrix b for the inverse
  det=1.0
  b=matrix((n,n))
  #make b the n by n identity matrix
  for i in range(1,n+1):
   v=1.
   b.put(i,i,v)
  #make a copy so that the original matrix is not changed
  a=matrix((n,n))
  for i in range(1,n+1):
   for j in range(1,n+1):
    a.put(i,j,self.get(i,j))
  print " initial a="
  print a
  print "initial b="
  print b
  #normalize a by dividing each row by the element of largest magnitude in the row.
  for i in range(1,n+1):
   bigest=a.get(i,1)
   for j in range(2,n+1):
    ab=a.get(i,j)
#    print "ab= ",ab
    if (math.fabs(ab) > math.fabs(bigest)):
     bigest=ab
#   print "i=",i," bigest= ",bigest
   det=det*bigest
   if(det == 0.):
    return b
   for j in range(1,n+1):
    a.put(i,j,a.get(i,j)/bigest)
#  Divide the corresponding row of b by the largest element.
   for j in range(1,n+1):
    b.put(i,j,b.get(i,j)/bigest)
  print "normalized a="
  print a
  print "normalized b="
  print b
  #Start of the elimination to put the matrix a into upper triangular form
  #At the jth step, we make the elements of the jth column 0
  #below the main diagonal. To start we interchange rows if
  #necessary to make the element a_jj the largest element in magnitude.
  #This element is called the pivot.
  #Then we subtract multiples of this row from the lower rows.
  j=1
  while(j<n):
    print "column j=",j
#   print "column j=",j,"a(j,j)=",a.get(j,j)
    kk=j+1
    l=j
    for i in range(kk,n+1):
 if (math.fabs(a.get(i,j)) > math.fabs(a.get(l,j))):
  l=i
#    print "l= ",l
    if(l != j):
#   interchange rows l and j
     det=-det
```

```
     print "interchange rows ",l," and ",j
     for k in range(1,n+1):
      c=a.get(l,k)
      a.put(l,k,a.get(j,k))
      a.put(j,k,c)
     for k in range(1,n+1):
      c=b.get(l,k)
      b.put(l,k,b.get(j,k))
      b.put(j,k,c)
    print " after interchange a="
    print a
# divide row j by pivot
    c=a.get(j,j)
    det=det*c
    if(det == 0.):
     return b
    print " pivot=",c
    for k in range(j,n+1):
     a.put(j,k,a.get(j,k)/c)
    for k in range(j,n+1):
     b.put(j,k,b.get(j,k)/c)
    print " after dividing by pivot a="
    print a
#   Add multiple of row j to lower rows
#   to eliminate coefficients below the diagonal
    jj=j+1
    for i in range(jj,n+1):
     am=a.get(i,j)
     for k in range(1,n+1):
      a.put(i,k,a.get(i,k)-am*a.get(j,k))
     for k in range(1,n+1):
      b.put(i,k,b.get(i,k)-am*b.get(j,k))
    print "after eliminating lower column coefficients a="
    print a
    j=j+1
#  end of while and elimination
  print "after completed elimination a="
  print a
  print "after completed elimination b="
  print b
  am=a.get(n,n)
  det=det*am
  if(det == 0.):
    return b
  for k in range(1,n+1):
   b.put(n,k,b.get(n,k)/am)
#  Back substitute to compute n-i component
#     i=1,2,3,...
  nn=n-1
  for i in range(1,nn+1):
   ni=n-i
   for j in range(1,n+1):
    nj=ni+1
    for ki in range(nj,n+1):
     b.put(ni,j,b.get(ni,j)-a.get(ni,ki)*b.get(ki,j))
  print "after back substitution and replacement b="
  print b
```

```
  print "det= ",det
  return b
 def __str__(self):
  """ Returns the string representation of the matrix.

  Example:
  >>> Matrix([1,2,3,4], 2, 2)
   1 2
   3 4
  """
  matrix_as_string = ""
  for i in range(0, self.m):
   for j in range(0, self.n):
    matrix_as_string = "%s %-10s" % (matrix_as_string, self.a[i * self.n + j])
   matrix_as_string =  "%s\n" % matrix_as_string
  return matrix_as_string

 def __repr__(self):
  """ Returns the representation of a matrix.

  Example:
  >>> x = matrix([1,2,3,4], 2, 2)
  >>> x.__repr___
  Matrix([1, 2, 3, 4], 2, 2)
  """
  return "matrix(%s, %s, %s)" % (self.a, self.m, self.n)
def main():
 """ Exersize some matrix functions.
  defining a main function in the class allows
  one to execute some examples using the code
  if __name__ == "__main__":
    main()
  that is if there is a function called "main" in the class,
  call it. In this way we can hide the examples when the class
  definition module is imported.
 """
 mat1 = matrix([1, 2, 3, 4, 5, 6, 7, 8, 9], 3, 3)
 mat2 = mat1

 print
 print "mat1 as a string:"
 print mat1
 print
 print "mat1 repr is %s" % mat1.__repr__()
 print
 print "Let mat2 = mat1"
 print "mat1 + mat2:"
 print mat1 + mat2
 s="1. 2. 3.;4. 5. 6.;7. 8. 9."
 a=matrix(s)
 print "rows=",a.m
 print "columns",a.n

 print " (printing matrix a directly) a= "
 print a

 print " (a print string representation of a) a= "
```

```
s=a.p()
print s

print "Check what happens if the row numbers are unequal"
s="10. 20. 30.;4. 5. ;7. "
print "String definition= ", s
b=matrix(s)
print "rows=",b.m
print "columns",b.n

print " b= "
s=b.p()
print s
c=matrix("1 2 3 4 5")
print " c="
print c.p()
d=matrix("1 ;2 ;3 ;4 ;5")
print " d="
print d.p()
e=matrix([1001.,1002.,1003.,1004.])
print "e="
print e.p()
e.m=2
e.n=2
print "e="
print e.p()
z=[5,10,15,20]
e.a=z
print "e="
print e.p()
print "Redefine e to be a 2 by 3 matrix, and"
print "fill it with complex numbers."
z=[1.+ 2j,3-1j,15,6-5j,25.8,3.14159]
e.a=z
e.m=2
e.n=3
print "e="
print e.p()
print "(again) e="
print e
print "Define a matrix with 4 rows and 3 columns, "
print "it is filled with zeroes, by default"
f=matrix((4,3))
print " f= "
print f.p()
print "Add matrices"
print " a= "
print a.p()
print "g=a"
g=a
print " g= "
print g.p()
h=g+a
print " h=a+g "
print " h= "
print h.p()
print " h(2,2) = ",h.get(2,2)
```

```
 print " h(3,2) = ",h.get(3,2)
 print " h(1,1) = ",h.get(1,1)
 print " Set the element in the 2nd row and 3rd column of h to 27.0"
 v=27.
 h.put(2,3,v)
 print " h= "
 print h.p()
 aa=matrix("2 3;1 2")
 print " aa="
 print aa.p()
 bb=matrix("3 1;4 2")
 print " bb="
 print bb.p()
 print "cc=aa*bb="
 cc=aa*bb
 print "cc= "
 print cc.p()
 print "cc repr is %s" % cc.__repr__()
 print "cc= "
 print cc
 t=matrix(["x","x^2","x^3","x^4"],2,2)
 print " t= "
 print t
 m27=matrix("1 2;3 4;5 6")
 print " m27="
 print m27
 m28=m27.transpose()
 print "m28 is the transpose of m27"
 print "m28="
 print m28
 m29=matrix([1. ,2. ,5. , 3. ,2. ,3. ,4., 0., 1.],3,3)
 print "m29="
 print m29
 b=m29.inverse()
 print " inv(m29)= "
 print b

 print "m29*m29.inverse()="
 print m29*b

print "Here we can call main() to execute some examples"
if __name__ == "__main__":
 main()
```

Here is the output of the program:

```
rows= 3
columns 3
 (printing matrix a directly) a=
 1.0        2.0        3.0
 4.0        5.0        6.0
 7.0        8.0        9.0

 (a print string representation of a) a=
 1.0 2.0 3.0
```

```
 4.0 5.0 6.0
 7.0 8.0 9.0

Check what happens if the row numbers are unequal
String definition=  10. 20. 30.;4. 5. ;7.
rows= 3
columns 1
 b=
 10.0
 20.0
 30.0

 c=
 1.0 2.0 3.0 4.0 5.0

 d=
 1.0
 2.0
 3.0
 4.0
 5.0

e=
 1001.0 1002.0 1003.0 1004.0

e=
 1001.0 1002.0
 1003.0 1004.0

e=
 5 10
 15 20

Redefine e to be a 2 by 3 matrix, and
fill it with complex numbers.
e=
 (1+2j) (3-1j) 15
 (6-5j) 25.800000000000001 3.1415899999999999

(again) e=
 (1+2j)     (3-1j)     15
 (6-5j)     25.8       3.14159

Define a matrix with 4 rows and 3 columns,
it is filled with zeroes, by default
 f=
 0.0 0.0 0.0
 0.0 0.0 0.0
 0.0 0.0 0.0
 0.0 0.0 0.0

Add matrices
 a=
 1.0 2.0 3.0
 4.0 5.0 6.0
 7.0 8.0 9.0
```

```
g=a
 g=
 1.0 2.0 3.0
 4.0 5.0 6.0
 7.0 8.0 9.0

 h=a+g
 h=
 2.0 4.0 6.0
 8.0 10.0 12.0
 14.0 16.0 18.0

 h(2,2) =  10.0
 h(3,2) =  16.0
 h(1,1) =  2.0
 Set the element in the 2nd row and 3rd column of h to 27.0
 h=
 2.0 4.0 6.0
 8.0 10.0 27.0
 14.0 16.0 18.0

 aa=
 2.0 3.0
 1.0 2.0

 bb=
 3.0 1.0
 4.0 2.0

cc=aa*bb=
cc=
 18.0 8.0
 11.0 5.0

cc repr is matrix([18.0, 8.0, 11.0, 5.0], 2, 2)
cc=
 18.0       8.0
 11.0       5.0

 t=
 x          x^2
 x^3        x^4

Here we call main()

mat1 as a string:
 1          2          3
 4          5          6
 7          8          9


mat1 repr is matrix([1, 2, 3, 4, 5, 6, 7, 8, 9], 3, 3)

Let mat2 = mat1
mat1 + mat2:
 2          4          6
 8          10         12
```

```
14        16        18
```

10.12 Functions With a Variable Number of Arguments

Being able to call a function with an unspecified number of arguments is very useful, especially for the initilizer of a class. Here is an example of using variable arguments, and variable keyword arguments:

```
def var_args(farg, *args):
  print "the fixed arg=", farg
  n=len(args)
  for k in range(0,n):
   print "vararg", k,"=",args[k]

def var_kwargs(farg, **kwargs):
 print "the fixed arg=", farg
 for key in kwargs:
  print "kwargs[%s]= %s" % (key, kwargs[key])

print "calling the var_args function"
var_args(1, "two", 3,[1,2,3,4])
print
print "calling the var_kwargs function"
var_kwargs(farg=1, my_string="two", my_number=3, my_list=[9,99,999])
print "Notice that the call order is not necessarily the print order"
```

Here is the output generated by this program:

```
calling the var_args function
the fixed arg= 1
vararg 0 = two
vararg 1 = 3
vararg 2 = [1, 2, 3, 4]

calling the var_kwargs function
the fixed arg= 1
kwargs[my_list]= [9, 99, 999]
kwargs[my_number]= 3
kwargs[my_string]= two
```

Notice that the call order is not necessarily the print order in the case of keyed values, kwargs.

10.13 Class Inheritance

10.14 The Module Search Path

When a module named spam is imported, the interpreter searches for a file named spam.py in the current directory, and then in the list of directories

specified by the environment variable PYTHONPATH. This has the same syntax as the shell variable PATH, that is, a list of directory names. When PYTHONPATH is not set, or when the file is not found there, the search continues in an installation-dependent default path; on Unix, this is usually .:/usr/local/lib/python.

Actually, modules are searched in the list of directories given by the variable sys.path which is initialized from the directory containing the input script (or the current directory), PYTHONPATH and the installation- dependent default. This allows Python programs that know what theyre doing to modify or replace the module search path. Note that because the directory containing the script being run is on the search path, it is important that the script not have the same name as a standard module, or Python will attempt to load the script as a module when that module is imported. This will generally be an error. See section Standard Modules for more information.

10.15 Google Python Tutorial

Below is a link to a Google Tutorial on the Python language.

`http://code.google.com/edu/languages/google-python-class/`

10.16 Computing the Matrix Inverse Using Gaussian Elimination

A set of linear equations such as

$$a_{11}x + a_{12}y + a_{13}z = b_1$$

$$a_{21}x + a_{22}y + a_{33}z = b_2$$

$$a_{31}x + a_{32}y + a_{33}z = b_3$$

is solved by repeatedly adding a multiples of one equation to a second equation, to eliminate coefficients, to solve for x, y, z. This technique is called Gaussian elimination. The set of equations can be written as a matrix equation

$$A \begin{bmatrix} x \\ y \\ z \end{bmatrix} = b$$

A systematic way of solving this is to eleminate all coefficients below the main diagonal of A, that is to set them to zero. To compute an inverse we need to find a matrix B

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

So that $A * B$ is the identity

$$A * B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This is equivalent to solving three sets of equations where the right sides are equal to the three columns of the identity. But this elimination can be done all at once. So here is an example:

So suppose we have

$$A = \begin{bmatrix} 1 & 2 & 5 \\ 3 & 2 & 3 \\ 4 & 0 & 1 \end{bmatrix}$$

and

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

To prevent any possible overflow and to control error we normalize all the elements of A and B to be between 0 and 1 in magnitude, so we divide each row by the element of largest magnitude c in each row of A. We simultaneously perform all row operations also on B in all future row operations. So here we divide each row of B by the c value found in the row of A. Then A becomes after normalization

$$A = \begin{bmatrix} 0.2 & 0.4 & 1.0 \\ 1.0 & 0.666666666667 & 1.0 \\ 1.0 & 0.0 & 0.25 \end{bmatrix}$$

and B becomes

$$B = \begin{bmatrix} 0.2 & 0.0 & 0.0 \\ 0. & 0.333333333333 & 0.0 \\ 0. & 0. & .25 \end{bmatrix}$$

The next step is to select what is called the pivot in the first column. We select a row with a largest magnitude element. In this case the second row has an element of magnitude 1 in the first column, so we interchange rows 1 and 2 and get

$$A = \begin{bmatrix} 1.0 & 0.666666666667 & 1.0 \\ 0.2 & 0.4 & 1.0 \\ 1.0 & 0.0 & 0.25 \end{bmatrix}$$

The next step is to divide the row by the pivot. In this case the pivot is 1, so the row is unchanged.

$$A = \begin{bmatrix} 1.0 & 0.666666666667 & 1.0 \\ 0.2 & 0.4 & 1.0 \\ 1.0 & 0.0 & 0.25 \end{bmatrix}$$

The next step is to eliminate the coefficients below the pivot by adding multiples of the row to the lower rows. Then A becomes

$$A = \begin{bmatrix} 1.0 & 0.666666666667 & 1.0 \\ 0.0 & 0.266666666667 & 0.8 \\ 0.0 & -0.666666666667 & -0.75 \end{bmatrix}$$

Now we work on the second column. We interchange rows 2 and 3 so that the largest element becomes the pivot

$$A = \begin{bmatrix} 1.0 & 0.666666666667 & 1.0 \\ 0.0 & -0.666666666667 & -0.75 \\ 0.0 & 0.266666666667 & 0.8 \end{bmatrix}$$

Dividing by the pivot we have

$$A = \begin{bmatrix} 1.0 & 0.666666666667 & 1.0 \\ 0.0 & 1.0 & 1.125 \\ 0.0 & 0.266666666667 & 0.8 \end{bmatrix}$$

Eliminating the coefficient in the third row we have put A in upper triangular form

$$A = \begin{bmatrix} 1.0 & 0.666666666667 & 1.0 \\ 0.0 & 1.0 & 1.125 \\ 0.0 & 0.0 & 0.5 \end{bmatrix}$$

Now we have also been doing the row operations on B, and they give

$$B = \begin{bmatrix} 0.0 & 0.333333333333 & 0.0 \\ 0.0 & 0.5 & -0.375 \\ 0.2 & -0.2 & 0.1 \end{bmatrix}$$

Now we can solve for the third unknown of the third equation of the first set of linear equations using the first column of B. Then using this value, we solve for the second unknown using the value of the first, and so on to solve for all three unknowns. This process is called back substitution. Then we repeat with the second column of B, then with the third, eventually finding the complete inverse of A. Our Python inverse algorithm returns this inverse in matrix B. So we find the inverse

$$A^{-1} = \begin{bmatrix} -0.1 & 0.1 & 0.2 \\ -0.45 & 0.95 & -0.6 \\ 0.4 & -0.4 & 0.2 \end{bmatrix}$$

Here are the diagnostic values printed out by the inverse function of the matrix class mat.py:

```
 initial a=
 1.0        2.0        5.0
 3.0        2.0        3.0
 4.0        0.0        1.0

initial b=
 1.0        0.0        0.0
 0.0        1.0        0.0
 0.0        0.0        1.0

normalized a=
 0.2        0.4        1.0
 1.0        0.666666666667 1.0
 1.0        0.0        0.25

normalized b=
 0.2        0.0        0.0
 0.0        0.333333333333 0.0
```

```
 0.0        0.0        0.25

column j= 1
interchange rows  2  and  1
 after interchange a=
 1.0        0.666666666667 1.0
 0.2        0.4        1.0
 1.0        0.0        0.25

 pivot= 1.0
 after dividing by pivot a=
 1.0        0.666666666667 1.0
 0.2        0.4        1.0
 1.0        0.0        0.25

after eliminating lower column coefficients a=
 1.0        0.666666666667 1.0
 0.0        0.266666666667 0.8
 0.0        -0.666666666667 -0.75

column j= 2
interchange rows  3  and  2
 after interchange a=
 1.0        0.666666666667 1.0
 0.0        -0.666666666667 -0.75
 0.0        0.266666666667 0.8

 pivot= -0.666666666667
 after dividing by pivot a=
 1.0        0.666666666667 1.0
 0.0        1.0        1.125
 0.0        0.266666666667 0.8

after eliminating lower column coefficients a=
 1.0        0.666666666667 1.0
 0.0        1.0        1.125
 0.0        0.0        0.5
```

```
after completed elimination a=
 1.0        0.666666666667 1.0
 0.0        1.0        1.125
 0.0        0.0        0.5

after completed elimination b=
 0.0        0.333333333333 0.0
 0.0        0.5        -0.375
 0.2        -0.2       0.1

after back substitution and replacement b=
 -0.1       0.1        0.2
 -0.45      0.95       -0.6
 0.4        -0.4       0.2



a*b=
 1.0        0.0        0.0
 0.0        1.0        -1.11022302463e-16
 -1.11022302463e-16 1.11022302463e-16 1.0
```

10.17 Examples Using Matrices

Here is an example program, matinverse.py, that computes a determinant and an inverse.

```
#Example of calculating a determinant and an inverse
#matinverse.py
import mat
a=mat.matrix([1. ,2. ,5. , 3. ,2. ,3. ,4., 0., 1.],3,3)
print "a="
print a
d=a.det()
print "determinant=", d
ainv=a.inverse()
print " ainv= "
```

```
print ainv
print "a*ainv="
print a*ainv
```

Here is the output of the program.

```
a=
 1.0        2.0        5.0
 3.0        2.0        3.0
 4.0        0.0        1.0

determinant= -20.0
 ainv=
 -0.1       0.1        0.2
 -0.45      0.95       -0.6
 0.4        -0.4       0.2

a*ainv=
 1.0        0.0        0.0
 0.0        1.0        -1.11022302463e-16
 -1.11022302463e-16 1.11022302463e-16 1.0
```

The elements of order 10^{-16} are of course actually zero. So this is the identity matrix.

10.18 Python Exercises

Exercise 1. Suppose a batter hits a baseball, so that the ball initially makes an angle of 30 degrees with the ground and lands 300 feet from homeplate. Assume no air resistance. What was the initial velocity magnitude when the ball left the bat in meters per second, in miles per hour?
Solution Let $\theta = \pi/6$. Then the initial vertical velocity component is

$$v_y = v \sin(\theta),$$

where v is the unknown initial velocity. Let t be the time of flight. Then in time $t/2$ the ball has reached its maximum height and the vertical velocity

is

$$v_y - g(t/2) = 0,$$

where g is the acceleration of gravity. So

$$t = \frac{2v\sin(\theta)}{g}.$$

Now the horizontal distance travelled in meters is

$$d = 300\text{ft}\frac{12\text{in}}{\text{ft}}\frac{2.54\text{cm}}{\text{in}}\frac{\text{m}}{100\text{cm}}$$

The horizontal distance travelled is

$$d = v\cos(\theta)t = v\cos(\theta)\frac{2v\sin(\theta)}{g}.$$

or

$$v = \sqrt{\frac{dg}{\sin(2\theta)}},$$

in meters per second.

A mile per hour equals 0.44704 meters per second.

```
# baseball.py
import math
pi=math.pi
theta=pi/6.
print  " angle = ",theta*180./pi," degrees"
#theta=theta/10
d=300.*12.*2.54/100.
print " distance =",d," meters"
g=9.81
v=math.sqrt((g*d)/math.sin(2.*theta))
print " velocity = ",v," meters/second"
print " velocity = ",v*3.6," kilometers/hour"
print " velocity = ",v/.44704," miles/hour"
vx=v*math.cos(theta)
t=d/vx
print " time = ",t," seconds"
vy=v*math.sin(theta)
t2=t/2
h=vy*t2-g*pow(t2,2)/2
print " height = ",h," meters"
```

The output is

```
angle =  30.0  degrees
 distance = 91.44  meters
 velocity =  32.1837982069  meters/second
 velocity =  115.861673545  kilometers/hour
 velocity =  71.9931062251  miles/hour
 time =  3.28071337481  seconds
 height =  13.1982271537  meters
```

Exercise 2. If a baseball is dropped from an airplane, what is its terminal velocity? That is, at what velocity is the force of the air equal to the gravitational force on the ball (Yes, the airplane is in the air, not at the airport, and yes I know that the terminal velocity is zero when it hits the ground.) Can you repeat the first exercise when there is air resistance?
Solution Hint The drag force is given by

$$F_d = -\frac{1}{2}\rho v^2 AD,$$

where ρ is the air density, v is the velocity, A is the area, and D is the drag coefficient. The terminal velocity occurs where the drag force equals the gravitational force. The drag coefficient for a sphere is approximately .47, but depends on the Reynolds number. The air density is about

$$\rho = 1.2$$

kilograms per cubic meter at 20 degrees Centigrade.

Let

$$C = \frac{\rho AD}{2m},$$

where m is the mass. Then we have

$$F_d = -mCv^2.$$

Now the gravitational force is $-mg$, so equating the drag force magnitude to the gravitational magnitude we have

$$mg = mCv^2.$$

So

$$v = \sqrt{g/C}.$$

Notice that for a sphere of constant density, the terminal velocity is proportional to $\sqrt{r}$. So an object of small radius has a small terminal velocity.

The diameter of a baseball is approximately 6.9 cm, so the radius is

$$r = \frac{6.9}{(2)(100.)} = .0345$$

meters. A baseball mass is about 135 grams,

$$m = .135$$

kilograms.

Here is the Python calculation:

```
#terminal velocity
import math
#density of air kg/m^3
rho=1.2
#drag coefficient
d=.47
#radius meters
r=6.9/(2.*100.)
#cross sectional area
a=math.pi*r**2
# mass kgm
m=.135
# acceleration of gravity m/s^2
g=9.81
c=(rho*a*d)/(2.*m)
v=math.sqrt(g/c)
print "Terminal velocity=",v, "meters per second"
print "                 =",v* 0.44704  ,"miles per hour"
```

The velocity is about 15 miles per hour. What would be the terminal velocity of a ping bong ball?

The second part of the problem requires the numerical solution of a differential equation, where the force on the ball is the sum of the gravitational force and the drag force. This is a nonlinear differential equation and generally requires an approximate numerical solution. Set up the first order differential equation for this problem. Historically, this problem drove the invention of the computer. The army wanted to compute artillery range tables.

The diferential equation is given by equating the mass times the acceleration to the forces

$$m\frac{d^2\mathbf{r}}{dt^2} = -gm\mathbf{j} - mC\frac{\mathbf{v}\|\mathbf{v}\|^2}{\|\mathbf{v}\|}.$$

So the x and y component equations are

$$\frac{d^2x}{dt^2} = -C\frac{dx}{dt}\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}$$

and

$$\frac{d^2y}{dt^2} = -g - C\frac{dy}{dt}\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}.$$

To solve the equations numerically, we need to convert them to a first order system. So let

$$\begin{aligned} u_1 &= x \\ u_2 &= \frac{dx}{dt} \\ u_3 &= y \\ u_4 &= \frac{dy}{dt}. \end{aligned}$$

Then

$$\begin{aligned} \frac{du_1}{dt} &= \frac{dx}{dt} = u_2 \\ \frac{du_2}{dt} & \\ &= \frac{d^2x}{dt^2} \\ &= -C\frac{dx}{dt}\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2} \\ &= -Cu_2\sqrt{u_2^2 + u_4^2} \\ \frac{du_3}{dt} &= \frac{dy}{dt} = u_4 \\ \frac{du_4}{dt} & \\ &= \frac{d^2y}{dt^2} \\ &= -g - C\frac{dy}{dt}\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2} \\ &= -g - Cu_4\sqrt{u_2^2 + u_4^2} \end{aligned}$$

So we have a system of four first order differential equations.

$$\frac{du_1}{dt} = \frac{dx}{dt} = u_2$$

$$\frac{du_2}{dt} = -Cu_2\sqrt{u_2^2 + u_4^2}$$

$$\frac{du_3}{dt} = u_4$$

$$\frac{du_4}{dt} = -g - Cu_4\sqrt{u_2^2 + u_4^2}$$

The initial values are

$$u_1 = 0, u_2 = v\cos(\theta), u_3 = 0, u_4 = v\sin(\theta),$$

where v is the initial velocity magnitude of the baseball. However, we are given a boundary problen rather than an initial value problem. That is we do not know v, we know the initial position of the ball, and the final position of the ball, namely on the ground 300 feet away. So to solve the problem we assume an initial velocity and calculate the final location. Then we vary the initial velocity until we find a velocity that takes us to the 300 foot position. This method of finding the solution of a boundery value problem is called the shooting method.

Now we can use Python to calculate a numerical solution by using the simple Euler method. We take a time step Δt and compute the changes of u_i for $i = 1, 2, 3, 4$ by adding

$$\frac{du_i}{dt}\Delta t.$$

The relative error per step is quite large, so extremely small time steps must be taken, with resulting possible roundoff error. To reduce the required number of steps we must use a more sophisticated higher order algorithm. Euler's method though, does illustrate the general method of solving an ordinary differential equation numerically.

Exercise 3. Given the vertices of a triangle $A = (1.5, 3.2), B = (2.6, 9.1), C = (4.9, 15.2)$. What is the area of the triangle? Use the cross product function.

Solution Hint We can consider A,B ,and C to be three dimensional vectors with zero as the z component. Then the magnitude of the cross product

$$(B - A) \times (C - A)$$

is $\|B - A\|\|C - A\| \sin(\theta)$ where θ is the angle between the vectors. This is the magnitude of the z component, because here the x and y components are zero. So the area of the triangle is one half of this. Write a Python function to compute the cross product, where a vector is represented by a list of the x,y and z components. Then write a python function to compute the area of a plane triangle using the cross product. Note, this problem can also be solved by calculating the lengths of the sides and using Heron's Formula (see the Vector Analysis review). Here is a program:

```
#areat.py area of a triangle
#
import math

#+ vdiff difference of vectors, a-ba2,a1)
def vdiff(a,b):
 c=[]
 n=len(a)
 for i in range(0,n):
  c.append(0.)
 for i in range(0,n):
  c[i]=a[i]-b[i]
 return c

#+ crsspr cross product of vectors
def crsspr(a,b):
 c=[0.,0.,0.]
 c[0]=a[1]*b[2]-a[2]*b[1]
 c[1]=a[2]*b[0]-a[0]*b[2]
 c[2]=a[0]*b[1]-a[1]*b[0]
 return c
#+ vnorm norm of a vector
def vnorm(a):
 n=len(a)
 vn=0.
 for i in range(0,n):
  vn=vn+a[i]*a[i]
 vn=math.sqrt(vn)
 return vn



#+ areat area of a plane triangle
def areat(v1,v2,v3):
 a1=[0.,0.,0.]
 a2=[0.,0.,0.]
 a3=[0.,0.,0.]
 for i in range(0,2):
  a1[i]=v1[i]
  a2[i]=v2[i]
  a3[i]=v3[i]
 a2minusa1=vdiff(a2,a1)
 a3minusa1=vdiff(a3,a1)
 cp=crsspr(a2minusa1,a3minusa1)
```

```
 area=math.fabs(cp[2])/2.
 return area

# main
#v1=[0.,0.]
#v2=[1.,0.]
#v3=[0.,1.]
v1=[1.5, 3.2]
v2=[2.6, 9.1]
v3=[4.9, 15.2]
print "v1=",v1
print "v2=",v2
print "v3=",v3
area=areat(v1,v2,v3)
print "area=",area
#
#length of sides
a=vnorm(vdiff(v2,v1))
print "length of side a",a
b=vnorm(vdiff(v3,v2))
print "length of side b",b
c=vnorm(vdiff(v1,v3))
print "length of side c",c
#half perimeter
s=(a+b+c)/2.
#Heron's formula for the area
area=math.sqrt((s-a)*(s-b)*(s-c)*s)
print "area=",area
```

Exercise 4. Let a 1000 ohm resistor be in series with a 1 microfarad capacitor and an inductor of inductance L. Suppose an alternating voltage is applied to this circuit. If using an oscilloscope and a function generator, we find that the maximum voltage on the resistor occurs at 5000 Hertz, then what is the value L of the inductance? If the applied voltage is 20 volts, make a plot of current magnitude as the frequency varies from 0 to 10000 Hertz. Modify the program writefunc.py to generate points for the plot. Then use programs pnts2eg, pltax, eg2ps, and ps2pdf to display the plot. See the section on plotting a function with utility programs.
Solution Hint The impedance for a series circuit is

$$Z = R + (\omega L - \frac{1}{\omega C})j$$

Given a constant applied voltage, the maximum voltage drop occurs across R when the imaginary term is zero.

```
#inductance.py
```

```
import sys
import math
f=float(sys.argv[1])
print "f=",f," Hertz"
omega=2.*math.pi*f
C=2.0e-6
print " C=",C*1.e6,"microfarads"
L=1./(omega*omega*C)
print " L=",L*1000.," millihenries"
fmin=500.
fmax=2000.
R=1000.
n=30
for i in range(0,n):
 f=i*(fmax-fmin)/(n-1)+fmin
 omega=2.*math.pi*f
 XL=omega*L
 XC=-1/(omega*C)
 print 'i=',i,'f={0:.1f}'.format(f),'XL={0:.4f} XC={1:.4f}'.format(XL,XC)
 #print 'XL={0:.5f}'.format(XL)
#
```

10.19 Python References

Python Essential Reference (4th Edition) (Paperback) David M. Beazley David M. Beazley (Author) Visit Amazon's David M. Beazley Page

11 Using Numerical Subroutine and Function Libraries in Scientific Programming

12 Scientific Programming in Fortran

Fortran was devised for scientific programming. Arguments are passed by reference, so pointers are not used in traditional Fortran. Fortran has been modernized and expanded in recent years to include object oriented programming and dynamic memory allocation. Reference:
Akin Ed , **Object-Oriented Programming via Fortran 90/95** , Cambridge University Press, 2003. (Linda Hall Library).

12.1 Passing Function Names

In the function **finv** in the following example, the function name g is passed to the subroutine **bisectm** as an argument. There is an external statement

in **finv** that identifies g as an external function. Otherwise g would be considered just a number. Then in the linking step an error would be generated, because **bisectm** knows that its first parameter is a function name, because f is being called as a function in the body of **bisectm**.

```
c invfun.ftn an example of computing a function inverse using a bisection root finder
c     an example of passing a global variable using a common block
c     set the variables starting with a-h or o-z, to double precision
      implicit real*8 (a-h,o-z)
      character*100 frmt
      frmt='(a,g20.13)'
c     find an x so that, f(x)=y
      y=.5
      write(*,'(a,g20.13)')'  y= ',y
      write(*,*)' Computing the inverse of f at y'
      x=finv(y)
      write(*,'(a,g20.13)')'  x = ',x
      y=f(x)
      write(*,'(a,g20.13)')'  f(x) =',y
      end
c
c+ f  Definition of the function whose inverse is to be evaluated
      function f(x)
       implicit real*8 (a-h,p-z)
       y=1/(2.+cos(x)) + tan(x)
       f=y
       return
      end
c
c+ finv  inverse function f^{-1}
      function finv(y)
      implicit real*8 (a-h,p-z)
      common /blk1/y0
      external g
      y0=y
      a0=0.
      b0=.7
      rel=1.e-10
      ab = 1.e-10
      itmax=100
      call bisectm(g,a0,b0,rel,ab,itmax,x,ier)
      finv=x
      return
      end
c
c+ g  function g(x)=f(x)-y0
      function g(x)
       implicit real*8 (a-h,p-z)
       common /blk1/y0
       g=f(x)-y0
       return
      end
c
c+ bisectm a root of f(x) by the bisection method
      subroutine bisectm(f,a0,b0,rel,ab,itmax,x,ier)
        implicit real*8(a-h,o-z)
```

```
c     f-external function
c a0,b0-interval containing root
c   rel-x accepted if it differs from previous iteration
c       by less than rel*abs(x)
c    ab-x accepted if  abs(f(x)) is less than ab
c itmax-maximum number of iterations
c   ier-
c       =1, convergence by the relative criterion
c       =2, convergence by the absolute criterion
c       =3, f(a)*f(b) greater than zero, no convergence
c       =4, maximum iterations exceeded, no convergence
      data zero/0.0d0/
      a=a0
      b=b0
      ier=3
      c=f(a)*f(b)
      if(c.gt.zero)return
      xp=a
      do i=1,itmax
       x=(a+b)/2.
       c=f(x)*f(b)
       if(c .le. zero)then
        a=x
       else
        b=x
       endif
       ier=1
       if(abs(x-xp).lt.rel*abs(x))return
       ier=2
       if(abs(f(x)).lt.ab)return
       xp=x
      end do
      ier=4
      return
      end
```

12.2 Handling Matrices

Here is an example of using matrices in Fortran. In the first dimension statement in the main program ia, ib, and so on define the row dimension of the matrix. This defines how the elements are stored in the array and how addresses are located in the array. This is the maximum number of rows that the matrix can hold. But the mathematical size of the square matrix is determined by how many numbers are read into the program. So in the call to the matrix inversion subroutine **gaussr**, the matrix to be inverted is an n by n matrix. n and ai are different because the mathematical size of the matrix is not known at compile time. If dynamic memory allocation were used, it would be possible to allocate memory after the numbers have been read. However, older versions of standard Fortran could not dynamically

allocate memory. Even when it is allowed in a programming language, the programmer may choose not to use it because the programming is a bit more complex. And in that case one needs also to deallocate memory. This is error prone. Fortran versions 90 and after do handle dynamic memory allocation, so this program could be rewritten to avoid the necessity of passing row dimensions.

```
c matrixv2.ftn revision 3/27/2010, example of matrix inverse calculation
      parameter(ia=20,ib=20,ic=20,iv=60,iu=43)
      implicit real*8(a-h,o-z)
      dimension a(ia,30),acopy(ia,40),b(ib,20),c(ic,40),v(iv),u(iu)
      dimension ain(20)
      m=0
1     continue
      write(*,*)' enter row ',m+1
      call readr(0,ain,nr)
      if(nr .gt.  0)then
       n=nr
       m=m+1
       do  j=1,n
        a(m,j)=ain(j)
        acopy(m,j)=ain(j)
       enddo
       go to 1
      endif
      if(m .ne. n)then
       write(*,*)' rows not equal to columns'
       stop
      endif
      inv=1
      eps=1.e-10
      mm=1
      call gaussr2(acopy,ia,b,ib,n,mm,inv,eps,idet,det,ier)
      if(ier .ne. 0)then
       write(*,*)' error parameter= ',ier
      endif
      write(*,*)' b= inverse(a) = '
      do  i=1,n
```

```
       write(*,'(20(g12.5,1x))')(b(i,j),j=1,n)
      enddo
      n1=n
      n2=n
      n3=n
c     show that a*inverse(a) = identity
      call matm(a,ia,n1,n2,b,ib,n3,c,ic)
      write(*,*)' a*inverse(a) ='
      do i=1,n
       write(*,'(20(g12.5,1x))')(c(i,j),j=1,n)
      enddo
      do i=1,n
       write(*,*)' enter element',i, ' of vector v'
       call readr(0,ain,nr)
       if(nr .gt.  0)then
        v(i)=ain(1)
       else
        stop
       endif
      enddo
      n1=n
      n2=n
      n3=1
      call matm(b,ib,n1,n2,v,iv,n3,u,iu)
      write(*,*)' inverse(a)*v = u ='
      do i=1,n
       write(*,'(20(g12.5,1x))')u(i)
      enddo
c     construct lines here to read vector v and
c     solve system a*u = v as u = inverse(a)*v
      end
c+ gaussr2 linear equations, inverse, determinant (modern version of gaussr)
      subroutine gaussr2(a,ia,b,ib,n,m,inv,eps,idet,det,ier)
c  last revision: 10/31/96
c  This is a modern version of gaussr (no statement labels)
c  solves the equation a*c=b for c, where a is an n by n matrix
c  c and b are n row by m column matrices. c is returned as b.
c  algorithm       -gaussian elimination with partial pivoting.
```

```
c  parameters     a-n by n matrix containing the coefficients of
c                    the linear system.
c                 ia-row dimension of a in defining routine
c                    e.g. in the routine where a is defined,
c                    a might be dimensioned as:
c                    dimension a(nr,nc)
c                    then ia must be set to nr.  we may have n < ia, but
c                    must not have n > ia, or n*n > nr*nc.
c                    ia is needed for proper addressing of matrix a.
c                    fortran stores by column first: a(i,j)=a(i+(j-1)*ia))
c                  b-n by m matrix containing the m right sides
c                    of the equations, on entering.  on returning, b
c                    contains the solutions.  the inverse of a is
c                    returned in b when inv=1
c                 ib-row dimension of b in defining routine
c                  n-row and column dimension of a.
c                  m-column dimension of b (usually 1)
c                    the program changes m to n when inv=1
c                inv-the inverse of a is calculated
c                    and returned in b when inv=1
c                    b must be large enough to hold the inverse
c                eps-each equation is normalized so that the
c                    coefficients are <= 1 in magnitude.
c                    when a pivot is less than eps the matrix is
c                    considered nearly singular, and ier is set to 1
c                    if a pivot is zero the matrix is singular, and
c                    ier is set to 2. one may set eps=1.e-5 for
c                    single precision, and 1.e-12 for double.
c                    eps does not effect any calculation.
c                    normalization may also prevent exponent overflow.
c               idet-compute determinant only if idet = 1
c                     determinants are products of n numbers.
c                     overflow can occur if the elements of the
c                     matrix have large exponents.
c                     set idet=0 if the determinant is not needed.
c                det-determinant of a.
c                ier-return parameter,
c                ier=0  normal return
```

```
c               ier=1  matrix is nearly singular
c               ier=2  matrix is singular
c
c  warning!! the subroutine changes a and b. if they need to be
c  saved, copies must be made before calling the subroutine.
c  the subroutine can be converted to different number type
c  by uncommenting the appropriate implicit statement.
      implicit real*8(a-h,o-z)
c     implicit complex(a-h,o-z)
c     implicit complex*16(a-h,o-z)
      logical cdet
      dimension a(ia,*),b(ib,*)
      cdet = idet .eq. 1
      zero=0.
      ier=0
      det=1.
      if(m.le.0)m=1
      if(inv.eq.1)then
c      set b equal to the identity
       do i=1,n
        do j=1,n
         b(i,j)=0.
         if(i.eq.j)b(i,j)=1.
        enddo
       enddo
       m=n
      endif
c
c     Normalize rows by dividing by largest element
      do i=1,n
       bigest=a(i,1)
       do j=2,n
        ab=a(i,j)
        if(abs(ab).gt.abs(bigest))bigest=ab
       enddo
       if(bigest.eq.zero)then
        ier=2
        det=0.
```

```
         return
        endif
        if(cdet)det=det*bigest
        do j=1,n
         a(i,j)=a(i,j)/bigest
        enddo
        do j=1,m
         b(i,j)=b(i,j)/bigest
        enddo
       enddo
       j=1
c
c      Start of elimination
       do while( j .lt. n)
        kk=j+1
        l=j
c       find row l with largest pivot
        do i=kk,n
         if(abs(a(i,j)).gt.abs(a(l,j)))then
          l=i
         endif
        enddo
        if(abs(a(l,j)).eq.zero)then
         ier=2
         det=0.
         return
        endif
        if(abs(a(l,j)) .le. abs(eps))then
         ier=1
        endif
        if(l .ne. j)then
c        interchange rows l and j
         do k=1,n
          c=a(l,k)
          a(l,k)=a(j,k)
          a(j,k)=c
         enddo
         do k=1,m
```

```
          c=b(l,k)
          b(l,k)=b(j,k)
          b(j,k)=c
         enddo
         if(cdet)then
          det=det*(-1.)
         endif
        endif
        if(cdet)then
         det=det*a(j,j)
        endif
c       divide row by pivot
        c=a(j,j)
        do k=j,n
         a(j,k)=a(j,k)/c
        enddo
        do k=1,m
         b(j,k)=b(j,k)/c
        enddo
c       Add multiple of row j to lower rows
c       to eliminate jth coefficients.
        jj=j+1
        do i=jj,n
         am=a(i,j)
         do k=1,n
          a(i,k)=a(i,k)-am*a(j,k)
         enddo
         do k=1,m
          b(i,k)=b(i,k)-am*b(j,k)
         enddo
        enddo
        j=j+1
       enddo
c      End of elimination
c
       am=a(n,n)
       if(abs(am).eq.zero)then
        ier=2
```

```
       det=0.
       return
      endif
      if(abs(am) .le. abs(eps))then
       ier=1
      endif
      if(cdet)then
       det=det*am
      endif
c
c     Matrix a is now in triangular form.
c     We compute the nth component of the solution.
      do k=1,m
       b(n,k)=b(n,k)/am
      enddo
c     Back substitute to compute n-i component
c     i=1,2,3,...
      nn=n-1
      do i=1,nn
       ni=n-i
       do j=1,m
        nj=ni+1
        do ki=nj,n
         b(ni,j)=b(ni,j)-a(ni,ki)*b(ki,j)
        enddo
       enddo
      enddo
      return
      end

c
c+ readr read a row of floating point numbers
      subroutine readr(nf, a, nr)
      implicit real*8(a-h,o-z)
c   numbers are separated by spaces
c   examples of valid numbers are:
c   12.13 34 45e4 4.78e-6 4e2,5.6D-23,10000.d015
c  nf=file number, 0 for standard input file
```

```
c   a=array of returned numbers
c  nr=number of values in returned array,
c     or 0 for empty or blank line,
c     or -1 for end of file on unit nf.
c requires functions val and length
      dimension a(*)
      character*200  b
      character*200  c
      character*1 d
        c=' '
      if(nf.eq.0)then
         read(*,'(a)',end=99)b
      else
         read(nf,'(a)',end=99)b
      endif
      nr=0
      l=lenstr(b)
      if(l.ge.200)then
        write(*,*)' error in readr subroutine '
        write(*,*)' record is too long '
      endif
      do 1 i=1,l
       d=b(i:i)
       if (d.ne.' ') then
        k=lenstr(c)
        if (k.gt.0)then
           c=c(1:k)//d
        else
           c=d
        endif
       endif
       if( (d.eq.' ').or.(i.eq.l)) then
        if (c.ne.' ') then
          nr=nr+1
           call valsub(c,a(nr),ier)
          c=' '
        endif
       endif
```

```
1     continue
      return
99    nr=-1
      return
      end
c
c+ valsub converts string to floating point number (r*8)
      subroutine valsub(s,v,ier)
      implicit real*8(a-h,o-z)
c     examples of valid strings are: 12.13 34 45e4 4.78e-6 4E2
c     the string is checked for valid characters,
c     but the string can still be invalid.
c      s-string
c      v-returned value
c      ier- 0 normal
c           1 if invalid character found, v returned 0
c
      logical p
      character s*(*),c*50,t*50,ch*15
      character z*1
      data ch/'1234567890+-.eE'/
      v=0.
      ier=1
      l=lenstr(s)
      if(l.eq.0)return
      p=.true.
      do 10 i=1,l
      z=s(i:i)
      if((z.eq.'D').or.(z.eq.'d'))then
          s(i:i)='e'
      endif
      p=p.and.(index(ch,s(i:i)).ne.0)
 10   continue
      if(.not.p)return
      n=index(s,'.')
      if(n.eq.0)then
          n=index(s,'e')
          if(n.eq.0)n=index(s,'E')
```

```
          if(n.eq.0)n=index(s,'d')
          if(n.eq.0)n=index(s,'D')
          if(n.eq.0)then
             s=s(1:l)//'.'
          else
            t=s(n:l)
            s=s(1:(n-1))//'.'//t
          endif
          l=l+1
      endif
      write(c,'(a30)')s(1:l)
      read(c,'(g30.23)')v
      ier=0
      return
      end
c+ matm  matrix multiplication
      subroutine matm(a,ia,m,n,b,ib,l,c,ic)
      implicit real*8(a-h,o-z)
c arguments
c  a-matrix
c ia-row dimension of a in calling program
c  m-number of rows of a
c  n-number of columns of a
c  b-matrix
c ib-row dimension of b in calling program
c  l-number of columns of b
c  c-product matrix: c=a*b
c ic-row dimension of c in calling program
c     c=a*b
      dimension a(ia,*),b(ib,*),c(ic,*)
      do i=1,m
       do j=1,l
        c(i,j)=0.
        do k=1,n
         c(i,j)=c(i,j)+a(i,k)*b(k,j)
        enddo
       enddo
      enddo
```

```
      return
      end
c+ lenstr   nonblank length of string
      function lenstr(s)
c  length of the substring of s obtained by deleting all
c  trailing blanks from s.  thus the length of a string
c  containing only blanks will be 0.
      character   s*(*)
      lenstr=0
      n=len(s)
      do i=n,1,-1
       if(s(i:i) .ne. ' ')then
        lenstr=i
        return
       endif
      enddo
      return
      end
```

13 Scientific Programming in C

C was constructed as a system programming language to allow accessing the deep internals of a machine. Consequently C can do most anything. But scientific programming in C can be awkward.

13.1 Passing Function Names

Here is a simple example showing how to pass a function name as an argument in C. This program computes the integral of a function using the trapezoid method. The function **trapez** integrates a function whose name is passed to the function as an argument. So the call in main

```
v=trapez(f,a,b,n);
```

returns the value of the integral of f, integrated from a to b, where n points are used in defining the trapezoids. The prototype for the call is

```
double trapez(double (*fp)(double),double,double,int);
```

The declaration

`double (*fp)(double)`

Means that fp is a pointer to function that has a double as an argument, and returns a double.

```
//trapezoid.c revision 1/5/2010, integrating an external function
#include <stdio.h>
#include <math.h>
double f(double);
double trapez(double (*fp)(double),double,double,int);
void main(){
 double a,b,v;
 int n,i;
 a=0.;
 b=4.;
 for(n=3;n < 1000000;n=2*n+1 ){
  v=trapez(f,a,b,n);
  printf(" v = %21.15g n= %d \n",v,n);
 }
 v= sin(b)-b*cos(b);
 printf(" sin(b)-b*cos(b) = %21.15g \n",v);
}
//c+ trapez integration of a function using the trapezoid method
double trapez(double (*fp)(double),double a,double b,int n){
//parameters
//    f-external function to be integrated
//  a,b-integration interval
//    n-interval divided into n-1 pieces
//    v-value returned for integral
 double x,y,v,h;
 int i;
 for(i=1,v=0.;i<=n;i++){
  x=(i-1)*(b-a)/(n-1)+a;
  y=(*fp)(x);
  if((i == 1)||(i == n)){
   y=y/2;
  }
  v=v+y;
 }
 h=(b-a)/(n-1);
 v*=h;
 return v;
 }
//c+ f  function
 double f(double x){
 double v;
 v=x*sin(x);
 return v;
}
```

13.2 Handling Matrices and Dynamic Memory Allocation

For static size matrices one can define a standard C two dimensional array. However this is not satisfactory for writing algorithms that compute with variable size matrices. For example the function in this example program called **gaussr** solves a system of linear equations using Gaussian elimination. So for such problems one technique is to use one dimensional arrays. So if one had a matrix having 200 rows and 50 columns, one can store the values in a one dimensional array by storing the first 50 numbers of column 1, followed by the second fifty numbers of column 2, and so on. When doing this we have the choice of storing by columns as just mentioned or by rows. Below we use columns. This is the Fortran convention. The second problem for general storage of matices is that array indices in C always start with zero. In mathematics, vectors usually start with index 1. So we must offset our indices.

Dynamic memory allocation in C is done with a call to malloc. An example of this occurs in function **llsq** below.

Least Squares Example.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
int llsq(double*,int,int,int,double*,double*);
int matrixp(double*,int,int,int);
int gaussr(double*,int,double*,int,int,int,int,double,int,double*);
main(){
 double a[6];
 int ia,m,n;
 double b[3];
 double c[2];
 ia=3;
 m=3;
 n=2;
 a[0+ia*0]=1.;a[0+ia*1]=1.;
 a[1+ia*0]=2.;a[1+ia*1]=3.;
 a[2+ia*0]=1.;a[2+ia*1]=2.;
 b[0]=1.;
 b[1]=5.;
 b[2]=3.;
 printf(" a= \n");
 matrixp(a,ia,m,n);
 printf(" b= \n");
 matrixp(b,1,m,1);
 llsq(a,ia,m,n,c,b);
 printf(" c= \n");
 matrixp(c,1,n,1);
```

```
 return 0;
}
//c+ llsq  solves overdetermined system a*c=b for c, using least squares
int llsq(double* a,int ia,int m,int n,double* c,double* b){
 //12/27/99
 // Input:
 //   a  m by n matrix. Declared row dimension ia. m>=n
 //       a(i,j)=a[i+j*ia], 0<= i < m, 0<= j < n
 //   ia row size used for addressing, where ia >= m
 //   b  right side vector of size m
 // Output:
 //   c  column vector of size n  a*c = b
 //
 // ier  return parameter: ier=0 normal return,ier=1 normal equations
 //       nearly singular,ier=2 normal equations singular.
 // Warning!
 //  Solves normal equations, which can be ill-conditioned.
 //  The matrix A is destroyed.
 //  For ill-conditioned problems, use the singular value decomposition.
 //Example of use:
 //    double a[6];
 //    int ia,m,n;
 //    double b[3];
 //    double c[2];
 //    ia=3;
 //    m=3;
 //    n=2;
 //    a[0+ia*0]=1.;a[0+ia*1]=1.;
 //    a[1+ia*0]=2.;a[1+ia*1]=3.;
 //    a[2+ia*0]=1.;a[2+ia*1]=2.;
 //    b[0]=1.;
 //    b[1]=5.;
 //    b[2]=3.;
 //    printf(" a= \n");
 //    matrixp(a,ia,m,n);
 //    printf(" b= \n");
 //    matrixp(b,1,m,1);
 //    llsq(a,ia,m,n,c,b);
 //    printf(" c= \n");
 //    matrixp(c,1,n,1);
 //    solution: c=(-2/3)
 //                ( 2  )
 double det;
 double e;
 double eps;
 int i;
 int ic;
 int ier;
 int idet;
 int inv;
 int j;
 int k;
 int mm;
 double s;
 double* ws;
 //
 //Compute lower elements of jth column  of transpose(a)*a
```

```
 ws=(double*)malloc(sizeof(double)*m);
 for(j=1;j<=n ;j++){
  for(i=j;i<=n ;i++){
   s=0.;
   for(k=1;k<=m ;k++){
    s = s + a[(k-1)+(i-1)*ia]*a[(k-1)+(j-1)*ia];
   }
   ws[i-1]=s;
  }
  //compute jth element of right side vector
  s=0.;
  for(k=1;k<=m;k++){
   s=s+a[(k-1)+(j-1)*ia]*b[k-1];
  }
  c[j-1]=s;
  //store lower elements of jth column in a
  for(i=j;i<=n ;i++){
   a[(i-1)+(j-1)*ia]=ws[i-1];
  }
 }
 //fill in upper values
 for(i=1;i<=n ;i++){
  for(j=i;j<=n ;j++){
   a[(i-1)+(j-1)*ia] = a[(j-1)+(i-1)*ia];
  }
 }
 ic=1;
 mm=1;
 eps=1.e-12;
 inv=0;
 idet=0;
 //solve normal equations
 printf("normal a= \n");
 matrixp(a,ia,n,n);
 printf("normal c= \n");
 matrixp(c,ic,n,mm);
 ier=gaussr(a,ia,c,ic,n,mm,inv,eps,idet,&det);
 free(ws);
 return(ier);
}
//c+ matrixp print a matrix
int matrixp(double *a,int ia,int m,int n){
 //Input:
 // a   m by n matrix
 // ia  column size used in addressing:
 //     a(i,j)=a[i+ia*j], 0 <= i < m <= ia,0 <= j < n
 //
 int i,j;
 for(i=0; i<m; i++){
  for(j=0; j<n; j++){
   printf("%11.5g ",a[i+j*ia]);
  }
  printf("\n");
 }
 return(0);
}
//c+ gaussr solution of real linear equations, inverse, determinant
```

```
int gaussr(double* a,int ia,double* b,int ib,int n,int m,int inv,
 double eps,int idet,double* det){
 //Version: 12/27/99
 //Input:
 //  a  n by n matrix containing the coefficients of
 //     the linear system.
 // ia  row dimension of a in calling function
 //     a(i,j)=a[(i-1)+ia*(j-1)],
 //     ia >= n
 //  b  n by m matrix containing the m right sides
 //     of the equations, on entering.  on returning, b
 //     contains the solutions.  the inverse of a is
 //     returned in b, when inv=1
 // ib  row dimension of b in defining routine
 //     b(i,j)=b[(i-1)+ib*(j-1)]
 //  n  row and column dimension of a.
 //  m  column dimension of b (usually 1)
 //     the program changes m to n when inv=1
 //inv  the inverse of a is calculated
 //     and returned in b when inv=1
 //     b must be large enough to hold the inverse
 //eps  each equation is normalized so that the
 //     coefficients are <= 1 in magnitude.
 //     when a pivot is less than eps the matrix is
 //     considered nearly singular, and ier is set to 1
 //     if a pivot is zero the matrix is singular, and
 //     ier is set to 2. one may set eps=1.e-5 for
 //     single precision, and 1.e-12 for double.
 //     eps does not influence any calculation.
 //     normalization may also prevent exponent overflow.
 //idet compute determinant only if idet = 1
 //      determinants are products of n numbers.
 //      overflow can occur if the elements of the
 //      matrix are large.
 //      So set idet=0 if the determinant is not needed.
 //Output:
 //det  determinant of a.
 //b    m solutions, or the inverse of a, when inv=1
 //Returns:
 // 0  normal return
 // 1  matrix is nearly singular
 // 2  matrix is singular
 //
 //Warning!! the subroutine changes a and b. if they need to be
 //saved, copies must be made before calling the function.
 //This is function is derived from a fortran version of gaussr
 //solves the equation a*c=b for c, where a is a n by n matrix
 //c and b are n row by m column matrices. c is returned as b.
 //Algorithm:  Gaussian elimination with partial pivoting.
 double ab;
 double am;
 double bigest;
 double c;
 //double cdet;
 int i;
 int ier;
 int j;
```

```
int jj;
int k;
int ki;
int kk;
int l;
int ni;
int nn;
int nj;
ier=0;
*det=1.;
if(m <= 0){
 m=1 ;
}
if(inv == 1){
 //set b equal to the identity
 for(i=1;i<=n ;i++){
  for(j=1;j<=n ;j++){
   b[(i-1)+ib*(j-1)]=0.;
   if(i == j){
   b[(i-1)+ib*(j-1)]=1.;
   }
  }
 }
 m=n;
}
//Normalize rows by dividing by largest element
for(i=1;i<=n ;i++){
 bigest=a[(i-1)];
 for(j=2;j<=n ;j++){
  ab=a[(i-1)+ia*(j-1)];
  if(fabs(ab) > fabs(bigest)){
   bigest=ab ;
  }
 }
 if(bigest == 0.0){
  ier=2;
  *det=0.;
  return(ier);
 }
 if(idet){
  *det=(*det)*bigest ;
 }
 for(j=1;j<=n ;j++){
  a[(i-1)+ia*(j-1)] = a[(i-1)+ia*(j-1)]/bigest;
 }
 for(j=1;j<=m ;j++){
  b[(i-1)+ib*(j-1)] = b[(i-1)+ib*(j-1)]/bigest;
 }
}
j=1;
//c     Start of elimination
while(j  <  n) {
 kk=j+1;
 l=j;
 //c      find row l with largest pivot
 for(i=kk;i<=n ;i++){
  if( fabs(a[(i-1)+ia*(j-1)]) > fabs(a[(l-1)+ia*(j-1)]) ){
```

```
   l=i;
  }
 }
 if(fabs(a[(l-1)+ia*(j-1)]) == 0.0){
  ier=2;
  *det=0.;
  return(ier);
 }
 if(fabs( a[(l-1)+ia*(j-1)] )  <=  fabs(eps)){
  ier=1;
 }
 if(l  !=  j){
  //interchange rows l and j
  for(k=1;k<=n ;k++){
   c=a[(l-1)+ia*(k-1)];
   a[(l-1)+ia*(k-1)]=a[(j-1)+ia*(k-1)];
   a[(j-1)+ia*(k-1)]=c;
  }
  for(k=1;k<=m ;k++){
   c=b[(l-1)+ib*(k-1)];
   b[(l-1)+ib*(k-1)]=b[(j-1)+ib*(k-1)];
   b[(j-1)+ib*(k-1)]=c;
  }
  if(idet){
   *det=(*det)*(-1.);
  }
 }
 if(idet){
  *det=(*det)*a[(j-1)+ia*(j-1)];
 }
 //divide row by pivot
  c=a[(j-1)+ia*(j-1)];
 for(k=j;k<=n ;k++){
  a[(j-1)+ia*(k-1)]=a[(j-1)+ia*(k-1)]/c;
 }
 for(k=1;k<=m ;k++){
  b[(j-1)+ib*(k-1)]=b[(j-1)+ib*(k-1)]/c;
 }
 //Add multiple of row j to lower rows
 //to eliminate jth coefficients.
 jj=j+1;
 for(i=jj;i<=n ;i++){
  am=a[(i-1)+ia*(j-1)];
  for(k=1;k<=n ;k++){
   a[(i-1)+ia*(k-1)]= a[(i-1)+ia*(k-1)]- am*a[(j-1)+ia*(k-1)];
  }
  for(k=1;k<=m ;k++){
   b[(i-1)+ib*(k-1)]= b[(i-1)+ib*(k-1)]- am*b[(j-1)+ib*(k-1)];
  }
 }
 j=j+1;
}
//End of elimination
am=a[(n-1)+ia*(n-1)];
if(fabs(am) == 0.0){
 ier=2;
 *det=0.;
```

```
 return(ier);
}
if(fabs(am)  <=  fabs(eps)){
 ier=1;
}
if(idet){
 *det=(*det)*am;
}
//
//Matrix a is now in triangular form.
//We compute the nth component of the solution.
for(k=1;k<=m ;k++){
 b[(n-1)+ib*(k-1)]= b[(n-1)+ib*(k-1)]/am;
}
//Back substitute to compute n-i component
//i=1,2,3,...
nn=n-1;
for(i=1;i<=nn ;i++){
 ni=n-i;
 for(j=1;j<=m ;j++){
  nj=ni+1;
  for(ki=nj;ki<=n ;ki++){
   b[(ni-1)+ib*(j-1)]=
    b[(ni-1)+ib*(j-1)] - a[(ni-1)+ia*(ki-1)]*b[(ki-1)+ib*(j-1)] ;
  }
 }
}
return(ier);
}
```

14 Scientific Programming in C Using Numerical Recipes

14.1 Passing Function Names

14.2 Handling Matrices

15 Scientific Programming in C Using The GNU Library

15.1 Passing Function Names

15.2 Handling Matrices

16 Scientific Programming in C++

16.1 Passing Function Names

Suppose we need to compute a root of some function, or compute the inverse of some function. And suppose that function has some variable parameters. We can use a general root finding algorithm provided we can pass the function definition. We shall use a general root finding class called rootbisection that originally was used in a class for modelling and doing computations on ellipses.

The general approach can be seen by the example case of computing the cube root of a number. Given the number a, we need to find an x such that

$$x^3 = a,$$

which is equivalent to finding a zero of the function

$$f(x) = x^3 - a$$

So consider the following program

```
#include "rootbisection.h"
class  cuberoot : public rootbisection{
 public:
 double number;
 double f(double x){
```

```
  return(x*x*x-number);
 }
 cuberoot() : rootbisection() {}
};
int main(){
  double root;
  cuberoot rb;
  rb.number=3.14159265358979;
  rb.left=0.;
  rb.right=2.;
  root=rb.getroot();
  printf(" The cuberoot of %20.15g is %20.15g \n",rb.number,root);
  printf(" The cube of %20.15g is %20.15g \n",root,pow(root,3));
}
```

This program uses a class that is derived from a class called rootbisection that finds a root of a function by the bisection method. The class rootbisection finds the root of a virtual function called f. So this virtual function must be replaced by a real function called f in our new derived class called cuberoot. We also define a new class member called number, whose cube root is to be found. So our new f function is $f(x) = x^3 - number$. In the main program we define our object of the class rb, set the number element of the class, set the left and right boundaries of our search interval and the call getroot to calculate the root.

The header file for the rootbisection class is rootbisection.h:

```
//==============================================================================
//
//  File name ......: rootbisection.h
//  Author(s) ......: Jim Emery
//  Language .......: C++
//  Started ........: June 2000
//  Last modified ..: June 17,2003
//  Version ........: 1.0
//  Source Path ....: .
//  Operating System: Linux,Windows
//  System .........: Intel
//  Site ...........: ITTC
//                    Electrical Engineering and Computer Science Department,
//                    University of Kansas
//                    Lawrence, KANSAS 66045
//                    U.S.A.
//  Purpose ........: To find a root of a function.
//
//  Note ...........:
//
//==============================================================================
#ifndef _ROOTBISECTIOB_H_
#define _ROOTBISECTION_H_
#include <stdio.h>
```

```
#include <math.h>
#include <stdlib.h>

/**
 * A class for finding the root of a one variable function
 * using the bisection method. The class defines a virtual function f.
 * A real inherited class must be derived from this class where a real
 * function f is defined as in the following example where a cube root
 * class is derived, where f is implemented, and where a constructor  calls
 * the original constructor to give default values to the search interval
 * [a,b], and where default values for the convergence parameters are set:
 * Set printiterations to 1 to get rootbisection.log file.
 *
 * EXAMPLE:
 *     class  cuberoot : public rootbisection{
 *      public:
 *      double number;
 *      double f(double x){
 *       return(x*x*x-number);
 *      }
 *      cuberoot() : rootbisection() {}
 *     };
 *
 *  The root is calculated like this:
 *   cuberoot rb;
 *   rb.number=3.14159265358779;
 *   rb.left=0.;
 *   rb.left=2.;
 *   root=rb.getroot();
 *
 *
 */
class rootbisection{
  public:
  double left;
  double right;
  double rel;
  double ab;
  int itmax;
  int flag;
  int iterations;
  int prn;
  virtual double f(double x) = 0;
  rootbisection(){
   left=0.;
   right=1.;
   rel=1.e-10;
   ab=1.e-10;
   itmax=50;
   prn=0;
  }
  double getroot();
};
#endif // _ROOTBISECTION_H_
```

The computation is contained in rootbisection.cpp:

```
//==============================================================================
//
//  File name ......: rootbisection.cpp
//  Author(s) ......: Jim Emery
//  Language .......: C++
//  Started ........: June 2000
//  Last modified ..: June 17,2003
//  Version ........: 1.0
//  Source Path ....: .
//  Operating System: Linux,Windows
//  System .........: Intel
//  Site ...........: ITTC
//                    Electrical Engineering and Computer Science Department,
//                    University of Kansas
//                    Lawrence, KANSAS 66045
//                    U.S.A.
//  Purpose ........: To find a root of a function.
//
//  Note ........... See example in cuberoot.cpp, or ellipse.cpp
//
//==============================================================================
#include <math.h>
#include <stdio.h>
#include "rootbisection.h"
/**
* Method to find the root by bisection.
*
* Input parameters:
*  left,right interval [left,right] is searched for a root of f(x)=0
*   rel       x is accepted as a root if it differs from the
*             previous iteration by less than rel*abs(x)
*             (thus setting rel=1.e-10 would give about 10 digit accuracy)
*    ab       x is accepted as a root  if  abs(f(x)) is less than ab
* itmax       maximum number of iterations
*             The error after n iterations will be about (b-a)(1/2)^n.
*             For example, if n=50, then (1/2)^n = 8.88e-16
* Output:
*  Returns x, the root of f(x) = 0.
*  Flag values:
*  =1, Convergence by the relative criterion: x_{n-1} - x_{n} < rel * x_{n}
*  =2, Convergence absolute criterion: |f(x_{n})| < ab
*  =3, Error, f(a) and f(b) do not differ in sign.
*  =4, Error, iterations exceed maximum value: itmax
*
*/
double rootbisection::getroot(){
 int ier;
 int i;
 int istop;
 double x1;
 double x2;
 double xx;
 double xp;
 double yy;
 double x;
 FILE* out;
 x1=left;
```

```
 x2=right;
 i=1;
 istop=0;
 yy=f(x1)*f(x2);
 if(yy  >  0.){
  ier=3;
  istop=1;
 }
 xp=x1;
 int count=0;
 while(istop  !=  1) {
  if(prn == 1){
   printf(" count= %d x1=%8.5g x2=%8.5g f1=%8.5g f2=%8.5g\n",count,x1,x2,f(x1),f(x2));
  }
  count++;
  xx=(x1+x2)/2.;
  yy=f(xx)*f(x2);
  if(yy  <=  0.){
   x1=xx;
  }
  else{
   x2=xx;
  }
  if(fabs(xx-xp)  <  rel*fabs(xx)){
   ier=1;
   istop=1;
  }
  else{
   if(fabs(f(xx))  <  ab){
    ier=2;
    istop=1;
   }
   else{
    if(i > itmax){
     ier=4;
     istop=1;
    }
   }
  }
  xp=xx;
  i=i+1;
 }
 x=xx;
 iterations=i;
 flag=ier;
 if(prn == 1){
  printf(" Number of iterations=%d Flag= %d \n",iterations,flag);
 }
 return(x);
}
```

16.2 Handling Matrices

17 Scientific Programming in Basic

17.1 Passing Function Names

17.2 Handling Matrices

18 Scientific Programming in Matlab and Octave

18.1 Introduction

Matlab is a program for doing numerical mathematics. Originally it was developed by Cleve Moler when he was at the University of New Mexico and associated with Sandia Laboratories. It was designed for students in a course in Linear Algebra. It was written with a government grant and so was originally a public domain program. Now it is a very expensive commercial program. Along with matlab comes several *Tool Boxes* such as the Signal Processing Toolbox, the Image Processing Toolbox, and so on.

Both Matlab and Octave are probably best used by editing a script file in an editor. Matlab has an editor, and if you click on an "m" file, the Matlab editor will open it. Then in Matlab one can type the name of the "m" file, to run it, so for example if the file name is test.m, you would type just "text" to run it. When entering commands interactively in Matlab, one must worry about redefining variables. To be safe sometimes one should clear all the variables in the workspace.

Octave is a free GNU program for doing numerical mathematics whose commands are mostly compatible with **Matlab**. So much of a Matlab book can be used to learn Octave.

18.2 Downloading Octave

To download GNU Octave, Google Octave. You will then select

```
http://www.gnuorg/software/octave
Then select download.
You will go to:
http://www.gnuorg/software/octave/download.html
Scroll down to "Windows" and "Octave Forge," which select.
```

```
You will go to
http://octave.sourceforge.net/
Select "Windows Installer" (or " "Octave.app for MacOS X" for macintosh)
Note the location of the saved file, which for Windows is called "Octave-3.0.0-setup.exe"
Click the file to run it. Take all defaults. It will be installed. To run it select
Octave in the program menu, then run it or browse the manual in the html or pdf versions (616 pages).
```

18.3 A Few Command Examples

```
format long, this displays 15 digits on output,
format short, this displays about 5 digits
pwd % prints the current working directory. (a comment is indicated by a percent sign)
cd c:\je\m % change the current directory
scriptname % loads an m-file script called scriptname into the workspace ( do not use the m extension)
testOctave1 % there is a script test program in the c:\je\octave directory
myfunction(a,b) % loads an m-file function, and executes it with parameters a and b
x = -10:0.1:10;
plot (x, sin (x));
lambda=eig(a)
[v,lambda]=eig(a)
diary % diary turns on a log of the commands issued in the session, and stores it in a file called diary
diary off
type scriptname % types a script called scriptname if it has been loaded into the workspace
x=[1,2,3,4,5,6,7,8,9]
y=exp(x)
cd c:\je\m
who % lists the variables that are currently defined
cd \je\octave % changes the working directory to c:\je\Octave
global x % make a variable global so that it can be accessed inside a function
hold on % add the next plot to the current axis
writetofile %  calls the script writetofile.m located in the working directory.
quit % ends the session
```

18.4 Using Octave as a Calculator

Octave and Matlab can be used like a graphing calculator for doing textbook like scientific problems. For example consider finding the current in an alternating current network by combining impedances using complex numbers. Use the argument function to find phase angles. Change the frequency by backing up with the up arrow. Calculate the angle between the voltage and current phasers to calculate power factor. Turn on diary to record your calculation.

18.5 Matlab Scripts, m-Files

A script is a file (an m file, with extension .m) containing Octave commands. When the name of the file is typed at the Octave command line, Octave fetches the file and executes all of the commands in the file just as if they were typed at the Octave command line. For example, here is a script called "writetofile.m" that illustrates the writing of data to files in a couple of ways.

```
x = 0:.1:1;
y = [x; exp(x)]; % the ending semicolon suppresses output
y % this displays the value of the matrix y
fid = fopen('c:\txt\exp1.txt','w');
fprintf(fid,'%6.2f %12.8f\n',y);
fclose(fid);
%
y=y';
fid = fopen('c:\txt\exp2.txt','w');
for i = 1:11
 fprintf(fid,'%6.2f %12.8f\n',y(i,1),y(i,2));
end;
fclose(fid);
```

This was actually a script written for Matlab. It works in Octave, as most basic Matlab commands do. Typing the m-file name, without the m extension, such as by typing "write2file," creates two files in directory c:/txt

Here is a listing of, "/c:/txt/exp1.txt," which is one of the files produced by the script:

```
  0.00   1.00000000
  0.10   1.10517092
  0.20   1.22140276
  0.30   1.34985881
  0.40   1.49182470
  0.50   1.64872127
  0.60   1.82211880
  0.70   2.01375271
  0.80   2.22554093
  0.90   2.45960311
  1.00   2.71828183
```

The file is written with the Unix end of line characters, namely linefeeds (lf). So to convert to the windows end of line characters "crlf," one may use a utility program, like say my **C** program "lf2nl.c".

18.6 Linear Equation Solution

Consider the equations

$$2x + 3y + 4z = 9$$

$$x - y - z = -1$$

$$5x + y - z = 5$$

We can solve these using

```
a=[2,3,4;1,-1,-1;5,1,-1]
b=[9;-1;5]
ai=inverse(a)
c=ai*b
```

Now we computed the inverse of matrix a and then multiplied. This is an expensive calculation, because a solution to a linear set of equations can be solved without computing the inverse and multiplying. So the more efficient solution method is available in Matlab or Octave by using the backwards division operator:

```
c = a\b
```

18.7 Fourier Transform Example

The Fourier transform of the function f is defined as (Goldberg, **The Fourier Transform**)

$$g(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt.$$

By the Fourier integral theorem

$$f(t) = \int_{-\infty}^{\infty} g(\omega) e^{i\omega t} d\omega.$$

Example. Let

$$f(t) = \left\{ \begin{array}{cc} 2e^{-3t} & t \geq 0 \\ 0 & t < 0 \end{array} \right\}$$

Then

$$g(\omega) = \frac{2}{3 + i\omega}$$

Also see foran.tex

Here is a script file to compute the fft in matlab or ocatve:

```
%Example page 229, Mastering Matlab 5, fft
N=128;
t=linspace(0,3,N);
f=2*exp(-3*t);
Ts = t(2) - t(1);
Ws = 2*pi/Ts;
F=fft(f);
Fp=F(1:N/2+1)*Ts;
W=Ws*(0:N/2)/N;
Fa=2./(3+j*W);
plot(W,abs(Fa),W,abs(Fp),'+')
xlabel('Frequency,Rad/s')
ylabel('|F(\omega)|')
title('Figure 21.1; Fourier Transform Approximation')
```

18.8 Reading Data From a File as a Matrix

A data file consisting of numbers, with say m records, each consisting of n numbers, can be read into Octave or Matlab with the script rdmat.m, which is listed here. The numbers on a row, may be separated with blanks, or with a comma. So if data is saved from Excel as a comma delimited list, then it may be read into Octave as a matrix. So for example suppose one has some xy data in a file called a.dat in the Octave working directory (Remember, one can type "pwd" to print your working directory, and can change the working directory using "cd"). To plot this data one might do this:

```
a=rdmat('a.dat')
x=a(:,1)
y=a(:,2)
plot(x,y)
```

Or perhaps the coefficients of the left side and right side of a linear equation are stored in a.dat, and we wish to read in the data and solve the linear equation.

```
m=rdmat('a.dat')
a=m(1:3,1:3)
b=m(1:3,4)
x=a\b % solution of a*x = b
```

Here is the script rdmat.m:

```
%read a matrix from a file
% Example usage: m=rdmat('c:\je\octave\test.txt')
function a=rdmat(fname)
 fid=fopen(fname,'r');
 s1=fgets(fid);
 k=1;
 while(!feof(fid))
  if k == 1
   s=s1;
  else
   s=[s;s1];
  end
  s1=fgets(fid);
  k=k+1;
 end
 a=str2num(s);
endfunction
```

18.9 Solving a Differential Equation Numerically

To solve a differential equation numerically, one must first write it as a system of first order equations, and specify the initial conditions. Here we call lsnode to solve such an equation.

```
%solve differential equation from octave manual,
%where fde is defined in file fde.m
x0 = [1; 2]; % set initial conditions
% select 200 equally spaced time points between 0 and 50
t = linspace(0, 50, 200);
x = lsode ("fde", x0, t);
plot (t, x)
```

The right hand side of the differential equation is defined in a function script called fde.m. Note that the name of the function and the file name must agree.

```
function xdot = fde (x, t)
```

```
 r = 0.25;
 k = 1.4;
 a = 1.5;
 b = 0.16;
 c = 0.9;
 d = 0.8;
 xdot(1) = r*x(1)*(1 - x(1)/k) - a*x(1)*x(2)/(1 + b*x(1));
 xdot(2) = c*a*x(1)*x(2)/(1 + b*x(1)) - d*x(2);
endfunction
```

18.10 Damped One-Dimensional Vibration

The differential equation for damped vibration is

$$m\ddot{x} + c\dot{x} + kx = F(t).$$

The mass is m, the viscous damping constant is c, the stiffness is k, and the applied force is $F(t)$. For free damped vibration the roots of the characteristic equation are

$$-\frac{c}{2m} \pm \sqrt{(c/2m)^2 - k/m}.$$

The natural undamped resonant angular frequency is

$$\omega_n = \sqrt{k/m}.$$

The critical damping constant is

$$c_c = 2m\omega_n.$$

The critical damping ratio is defined to be

$$\zeta = \frac{c}{c_c}.$$

We have

$$c = 2\zeta m\omega_n,$$

so that if we divide the damped vibration equation by the mass m, we can write it in the form

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = \frac{F(t)}{m}.$$

Let us suppose that the force is zero $\zeta < 1$ so that we have a case of decaying oscillation. The equation becomes

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = 0.$$

Solving this equation by substituting in $\exp(pt)$ to get the characteristic polynomial in p, we find that

$$x = x_0 \exp(-\zeta\omega_n t) sin(\omega_d t + \phi),$$

where the angular frequency of damped oscillation is

$$\omega_d = \omega_n\sqrt{1-\zeta^2},$$

and the corresponding period is

$$\tau_d = \frac{2\pi}{\omega_d}.$$

The notation in this section follows *Thomson*: **Theory of Vibration With Applications** Let us consider the case where $x(0) = 0$, so that the phase factor ϕ is zero, we have

$$x = x_0 \exp(-\zeta\omega_n t) sin(\omega_d t).$$

Let us take the natural frequency to be

$$\omega_n = 2\pi,$$

so that the natural period is

$$T_n = 1.$$

Suppose $\zeta = 1/2$, so that ω_d is about 5.4414, and the damped period is about

$$T_d = \frac{2\pi}{\omega_d} = 1.1547\text{sec}.$$

Also suppose $x_0 = 1.$. Here is an octave script called dampedh.m:

```
zeta=1/16;
omega_n=2*pi;
omega_d=omega_n*sqrt(1.-zeta^2);
```

```
n=200;
t1=0.;
t2=3.;
t=linspace(t1,t2,n);
for k= 1:n
 x(k)=exp(-zeta*omega_n*t(k))*sin(omega_d*t(k));
end
plot(t,x)
```

Now let us compute this solution numerically. We want to solve

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = 0.$$

Given the value of x and its first derivative $\dot{(x)}$ at 0, there is a unique solution. The numerical solution should agree closely with

$$x(t) = \exp(-\zeta\omega_n t) sin(\omega_d t).$$

To solve a second order equation numerically, we must convert it to a first order system. So define

$$u_1 = x,$$
$$u_2 = \frac{dx}{dt}.$$

So we get the system

$$\frac{u_1}{dt} = u_2,$$
$$\frac{u_2}{dt} = \frac{d^2x}{dt^2} = -(2\zeta\omega_n u_2 + \omega_n^2 u_1).$$

For the closed form solution computed above, which we want the numerical solution to match, we have

$$x(0) = 0,$$

and the derivative of x is

$$\frac{dx}{dt} = -\zeta\omega_n \exp(-\zeta\omega_n t)\sin(\omega_d t) + \exp(-\zeta\omega_n t)\omega_d \sin(\omega_d t).$$

So

$$\frac{dx(0)}{dt} = \omega_d.$$

So the required initial conditions for our system are

$$u_1(0) = 0,$$

and

$$u_2(0) = \omega_d.$$

A script, called dampedhn.m, for solving the equation numerically, is

```
% dampedhn.m solving a differential equation
% for damped harmonic vibration numerically.
% requires script fdedh.m for the left side of the equation
% See octave.tex
zeta=1/16;
omega_n=2*pi;
omega_d=omega_n*sqrt(1.-zeta^2);
n=200;
t1=0.;
t2=3.;
% set the initial conditions
u0 = [0; omega_d];
% select n equally spaced time points between t1 and t2
t=linspace(t1,t2,n);
u = lsode ("fdedh", u0, t);
```

Before we can run this script we have to provide a definition of the left side of our DE system in the function "fdedh," in an m file with the same name.

```
%fdedh.m leftside of a differential equation for damped harmonic motion
%called by dampedhn.m
function udot = fdedh (u, t)
 zeta=1/16;
 omega_n=2*pi;
 udot(1) = u(2);
 udot(2) = -(2*zeta*omega_n*u(2) + omega_n^2*u(1));
endfunction
```

To compare the values from the closed form solution and the numerical solution we may put them in a two column matrix and list it.

```
x1=x'; %convert x to a column vector
u1=u(:,1) % select the first column of u
for k=1:200 % put x1 and u1 into a 2 column matrix
 m(k,1)=x1(k)
 m(k,2)=u1(k)
end
m % list m
```

18.11 Bibliography for Matlab and Octave

[1]Fausett Laurene V., **Applied Numerical Analysis Using Matlab**, Prentice-Hall, 1999.

[2]Hanselman Duane, and Littlefield Bruce, **The Student Edition of Matlab Version 4 User's Guide**, The Math Works, 1995.

[3]Hanselman Duane, and Littlefield Bruce, **Mastering Matlab 5, A Comprehensive Tutorial and Reference**, Prentice-Hall, 1998.

[4]Eaton John W., Bateman David, Hauberg Soren, **GNU Octave: A high-level interactive language for numerical computations**, Edition 3 for Octave version 3.0.0, July 2007 (PDF document).

19 Scientific Programming in Java

19.1 Passing Function Names

19.2 Handling Matrices

20 Scientific Programming in Symbolic Languages Such as Maple, Mathematica, and Mupad

20.1 Passing Function Names

20.2 Handling Matrices

21 Scientific Programming in Pascal

21.1 Passing Function Names

21.2 Handling Matrices

22 Scientific Programming in Excel

22.1 Passing Function Names

22.2 Handling Matrices

23 Example: Orbit Intersection

Suppose a star S_2 orbits a more massive star S_1, and suppose the orbit of S_1 has major diameter $2a_2$ and minor diameter $2b_2$. Suppose a third star S_3 also orbits S_1 with major diameter $2a_3$ and minor diameter $2b_3$. Suppose the major axis of S_3 makes an angle of θ with the major axis of S_2. Write a program to compute the intersection points of the orbits.

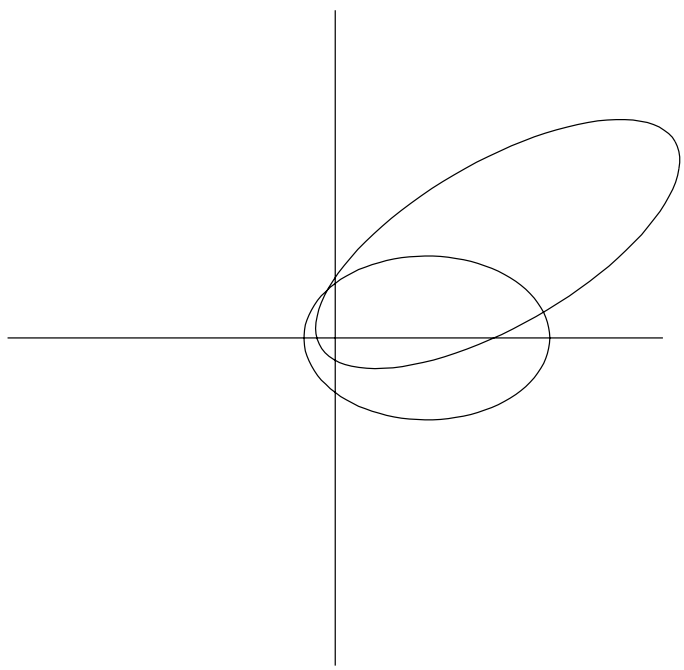


Figure 2: **Problem:** Suppose two stars, S_1 and S_2, orbit a much more massive star S, which is located at the common focus of the stars S_1 and S_2. Find the intersection points of the star orbits. (Hint, let one ellipse have an algebraic representation and the other a parametric representation. Then construct a function so that the zeroes of the function determine the intersection points. Find the zeroes with the bisection method, or by Newton's method.)

24 Example: Boat Analysis.

Suppose a boat hull is defined by piecewise parametric curves. These might be splines, Bezier Curves, or a curve made up of lines and arcs. Such a curve can be treated as a single curve. Consider the vertical center line of a hull section. Given a point on the center line, assume that this point h lies at the water level. Assume the hull is rotated by angle θ. Compute the buoyant force and the center of buoyancy of the hull when submerged and rotated according to the parameters h and θ. To do this we need to compute the area of the submerged hull, which gives the net buoyant force, and the moment of the area, which will determine the center of buoyancy. This can be done by integrating a function with a line integral around the clipped hull. This can be shown by using Green's theorem in the plane. So in this way we may compute functions of h and θ giving the buoyant force and the center of buoyancy. Doing this numerically with a finite number of points n, we can define interpolation methods to define functions for the entire domain. Now using this function study the properties of the hull, with respect to loading, stability, and resistance to capsizing. This would need to be done for many cross sections, since the cross sections will vary along the axis of the boat.

25 Polynomial Evaluation

Let a polynomial have the form

$$p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_n x^n.$$

We shall find an economical way to evaluate the polynomial and its derivatives at a point c. If we divide by $x - c$ we obtain a polynomial $q^0(x)$ and a remainder r_0. That is

$$p(x) = (x - c)q^0(x) + r_0$$

$$= (x - c)(b_1 + b_2 x + b_3 x^2 + ... + b_n x^{n-1}) + b_0$$

Therefore we may compute q^0 and r_0 as follows. We have

$$b_n = a_n,$$

$$b_{n-1} = a_{n-1} + cb_n,$$

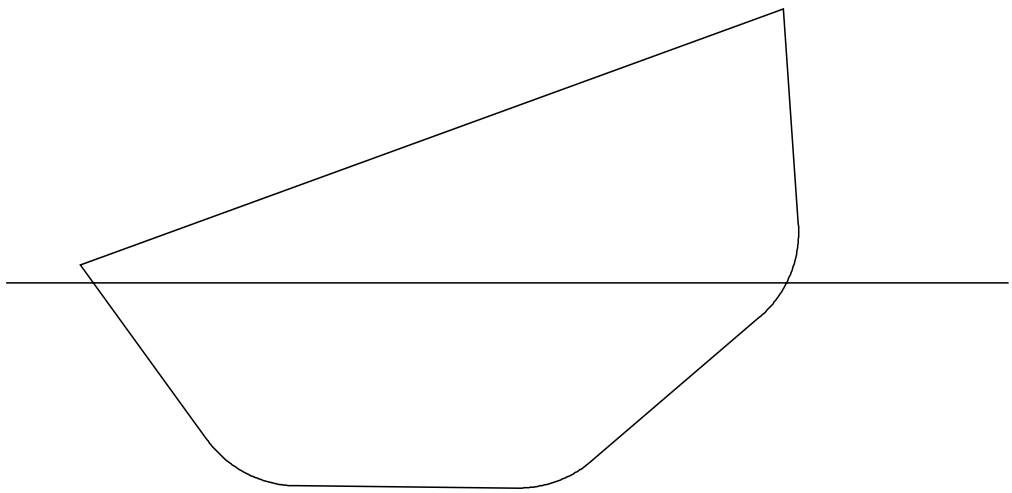


Figure 3: **Boat Stability**. As the boat is tipped to the left, the center of buoyancy lies to the left of the vertical center line of the boat. Assuming that the center of gravity lies on the center line, if the center of gravity is low enough, there will be a torque to level the boat. If the center of gravity is too high, it will lie to the left of the center of buoyancy in this tipped position, and the boat will capsize. This is because the force at the center of buoyancy acts upward and the force at the center of gravity acts downward.

$$b_{n-2} = a_{n-2} + cb_{n-1},$$

$$...$$

$$b_0 = a_0 + cb_1,$$

and

$$r_0 = b_0.$$

Now we repeat this division $n-1$ more times getting

$$q^0(x) = (x-c)q^1(x) + r_1,$$

$$q^1(x) = (x-c)q^2(x) + r_2,$$

$$...$$

$$q^{n-2}1(x) = (x-c)q^{n-1}(x) + r_{n-1}.$$

We have divided $p(x)$ by $x-c$, n times, so that

$$q^{n-1}(x) = a_n.$$

Then

$$\begin{aligned} p(x) &= (x-c)q^0(x) + r_0 \\ &= (x-c)((x-c)q^1(x) + r_1) + r_0 \\ &= (x-c)^2 q^1(x) + r_1(x-c) + r_0 \\ &... \\ &= a_n(x-c)^n + r_{n-1}(x-c)^{n-1} + r_{n-2}(x-c)^{n-2} + ... + r_0. \end{aligned}$$

Differentiating k times for $0 \geq k \geq n$, we have

$$p^{(k)}(c) = k! r_k,$$

where we take $r_n = a_n$. In particular the value of the polynomial at c is

$$p(c) = r_0.$$

The following subroutine does this calculation.

```
c+ polval value of polynomial and derivatives
      subroutine polval(a,n,x,f)
       implicit real*8 (a-h,o-z)
c input:
c  a-polynomial coefficients: p(x)=a(1)+ a(2)x+...+a(n+1)x^n
c  n-degree of polynomial
c  x-evaluation point
c output:
c f-0 thru nth derivatives at x: p(x)=f(1), p'(x)=f(2), p''(x)=f(3) ...
c method: repeated division by (x-c) ( see numanal.tex)
c 11/16/94
      dimension a(*),b(50),f(*)
      k=n+1
      do 10 i=1,k
       b(i)=a(i)
10    continue
      b(k+1) = 0.
      do 30 i=1,n
       if(i .eq. 1)then
        m=1
       else
        m=m*(i-1)
       endif
       do 20 j=k,i,-1
        b(j)=b(j) + x*b(j+1)
20     continue
       f(i) = m*b(i)
30    continue
      f(k)=m*n*a(k)
      return
      end
```

26 Roots of a Cubic Polynomial

First we transform a real root to the unit interval. Let the cubic equation be

$$x^3 + ax^2 + bx + c = 0.$$

Let $\beta =$ -sign(c) and $x' = \beta x$. Then

$$x'^3 + a\beta x'^2 + bx' - |c| = 0.$$

Let $\alpha = \sqrt[3]{|c|}$ and $x'' = x'/\alpha$. Then

$$x''^3 + \frac{a\beta}{\alpha}x''^2 + \frac{b}{\alpha^2}x'' - 1 = 0.$$

Define f(x") by the right side of this equation. We have $f(0) = -1$ and $f(\infty) = \infty$ so there is a positive root. If $f(1) > 0$ then there is a root

between 0 and 1. If $f(1) < 0$ then there is a root greater than 1. In that case let $x''' = 1/x''$ and then

$$x'''^3 - \frac{b}{\alpha^2}x'''^2 - \frac{a\beta}{\alpha}x''' - 1 = 0$$

has a root between 0 and 1. A real root of the original cubic is

$$x_1 = \alpha\beta x_1''$$

or

$$x_1 = \frac{\alpha\beta}{x_1'''}.$$

27 Synthetic Division and Newton's Method

We may write a polynomial

$$p(x) = a_0 + a_1x + ... + a_nx^n$$

in nested form as

$$a_0 + x(a_1 + x(a_2 + x(a_3 + ... + x(a_{n-1} + xa_n))..).$$

To evaluate p at c we compute

$$b_n = a_n$$

$$b_{n-1} = a_{n-1} + cb_n$$

$$.....$$

$$b_k = a_k + cb_{k+1}$$

$$.....$$

$$b_0 = a_0 + cb_1.$$

Then $p(c) = b_0$. Let

$$q(x) = b_1 + b_2x + ... + b_nx^{n-1}.$$

Comparing like terms we see that

$$p(x) = (x - c)q(x) + b_0$$

We have

$$\frac{dp}{dx} = q(x) + (x - c)\frac{dq}{dx}$$

and

$$\frac{dp(c)}{dx} = q(c).$$

Then Newton's method for a root c of p(x) is

$$c_m \to c$$

$$c_{m+1} = c_m - \frac{p(c_m)}{dp(c_m)/dx} = c_m - \frac{b_0}{q(c_m)} = c_m - \frac{b_0}{b_1'}$$

where

$$b_n' = b_n, b_k' = b_k + c_m b_{k+1}, k = 1, ..., n-1.$$

28 Tschebyscheff Approximation

Suppose we are given the cubic polynomial

$$p(x) = x^3 + Ax^2 + Bx - 1$$

defined on the unit interval. We want to find the best quadratic approximation to this cubic in the minimax sense. That is we want to minimize the maximum difference on the interval.

The Tschebyscheff polynomial of degree n is defined by

$$T_n(x) = \cos(n \cos^{-1}(x)) = 2^{n-1}x^n + ...$$

and has the amazing property that of all polynomials of degree n and leading coefficient 2^{n-1}, it has the smallest maximum absolute value on the interval $[-1, 1]$. This value is 1, which is attained in the interval $n+1$ times. All the zeroes of this polynomial lie in $[-1, 1]$. For the proof of this, see *Interpolation and Approximation* by Philip J. Davis (Theorem 3.3.4 p. 62). The idea of the proof is to assume that a second polynomial attains a smaller maximum magnitude in [-1,1]. Subtract it from $T_n(x)$, getting a difference polynomial of degree $n-1$. The $n+1$ extreme points of $T_n(x)$ are attained with alternating sign, that is for example with a positive extreme value, then a negative, then a positive and so on. It is then obvious from the graph of such a function that if one draws a second curve which never attains these extremum values,

then it must intersect the first curve at least n times. These intersection points correspond exactly to zeroes of the difference between the two curves. But the difference is of degree $n-1$ and can not have n zeroes, so there does not exist such a polynomial. This proves the theorem.

Returning to the approximation problem, we transform the domain [-1,1] of

$$T_3(x) = 4x^3 - 3x$$

to the interval [0,1], using the transformation $x \mapsto (x+1)/2$. We find that the Tschebyscheff polynomial becomes

$$32x^3 - 48x^2 + 18x - 1.$$

Dividing by 32 we have

$$x^3 - \frac{3}{2}x^2 + \frac{9}{16}x - \frac{1}{32}.$$

This polynomial has the smallest maximum magnitude of any monic cubic polynomial on [0,1] (monic means that the high order coefficient is 1). The maximum magnitude attained is $\frac{1}{32}$.

Consider the original cubic $p(x)$ written as

$$p(x) = x^3 + q(x)$$

We replace the x^3 term by the negative of the quadratic part of the normalized Tschebyscheff polynomial, getting a quadratic

$$r(x) = \frac{3}{2}x^2 - \frac{9}{16}x + \frac{1}{32} + q(x).$$

Then

$$\|p(x) - r(x)\| = \|x^3 - \frac{3}{2}x^2 + \frac{9}{16}x - \frac{1}{32}\| = \frac{1}{32}.$$

Because $p(0) = -1$ we have $\|p(x)\| \geq 1$ so the relative error is less than 1/32, which is about 3 percent. The explicit quadratic approximation is

$$r(x) = (A + \frac{3}{2})x^2 + (B - \frac{9}{16})x - (\frac{31}{32})$$

29 The Maximum Curvature of a Cubic Polynomial

We will start by locating the extreme points of the cubic. Let

$$f(x) = ax^3 + bx^2 + cx + d.$$

The second derivative is zero at the inflection point, which is

$$\alpha = -\frac{b}{3a}.$$

To eliminate the squared term, let $z = x - \alpha$, then

$$f(x) = h(z) = az^3 + c_z z + d_z,$$

where

$$c_z = c - \frac{b^2}{3a}.$$

The cubic is an odd function about the point (α, d_z). The second derivative is also an odd function, so the curvature of the cubic is an odd function about the point α. The curvature of f is

$$\kappa_f(x) = \frac{f''}{(1 + f'^2)^{3/2}} = \frac{6ax + 2b}{(1 + (3ax^2 + 2bx + c)^2)^{3/2}}.$$

The curvature is preserved by the translation and

$$\kappa(x) = \kappa_h(z) = \frac{h''}{(1 + h'^2)^{3/2}}.$$

κ' is zero at points of extreme curvature. At an extreme point we have

$$0 = \kappa_z'(1 + h'^2)^{5/2} = h'''(1 + h'^2) - 3h''^2 h'.$$

Let $q(z)$ be the right side of this equation. We find that

$$q(z) = Az^4 + Bz^2 + C = 0,$$

where

$$A = -270a^3, B = -72a^2 c_z, C = 6a(1 + c_z^2).$$

Let $w = z^2$, then

$$Aw^2 + Bw + C = 0.$$

The roots are

$$w_1 = \frac{(9c_z^2 + 5)^{1/2} - 2c}{15a}.$$

and

$$w_2 = \frac{(9c_z^2 + 5)^{1/2} + 2c}{-15a}.$$

If $a > 0$ then w_1 is the positive root, while if $a < 0$ then w_2 is the positive root. Let w be the positive root, then the extreme points are

$$x_1 = w^{1/2} - \frac{b}{3a},$$

and

$$x_2 = -w^{1/2} - \frac{b}{3a}.$$

The curvatures at these points are negatives of each other because of the odd symmetry of the curvature about the inflection point. If a is zero then the maximum curvature occurs at the local extremum of the parabola, which is

$$x = -\frac{c}{2b}.$$

If a and b are zero, the curvature is zero everywhere. See FORTRAN subroutine cubicx and the Maple derivation min15 (4/17/91).

30 Ferguson Cubics

A Ferguson cubic takes the form

$$r(u) = r_0(1 - 3u^2 + 2u^3) + r_1(3u^2 - 2u^3) + \dot{r}_0(u - 2u^2 + u^3) + \dot{r}_1(-u^2 + u^3),$$

where u is a parameter in the interval $[0, 1]$. Then

$$r(0) = r_0, r(1) = r_1.$$

The derivative is

$$\dot{r}(u) = r_0(6u^2 - 6u) + r_1(6u - 6u^2) + \dot{r}_0(1 - 4u + 3u^2) + \dot{r}_1(-2u + 3u^2).$$

Then

$$\dot{r}(0) = \dot{r}_0, \dot{r}(1) = \dot{r}_1.$$

The second derivative is

$$\ddot{r}(u) = r_0(-6 + 12u) + r_1(6 - 12u) + \dot{r}_0(-4 + 6u) + \dot{r}_1(-2 + 6u).$$

31 Hermite Interpolation

Given a set of n points $\{p_i\}$ and n tangent angles $\{\theta_i\}$, we shall construct a c_1 piecewise cubic that interpolates these values. From a tangent angle θ we may construct a unit tangent vector

$$\frac{dr}{ds} = (\cos(\theta), \sin(\theta)).$$

For the Ferguson cubic we need

$$\dot{r} = \frac{dr}{du} = \frac{dr}{ds}\frac{ds}{du}.$$

Consider an interpolation point. Let $\Delta_- s$ and $\Delta_+ s$ be the previous and following chord lengths. Let the average chord length be

$$\Delta s = \frac{\Delta_- s + \Delta_+ s}{2}.$$

The u parameter will vary from 0 to 1 between successive points. Therefore

$$\frac{ds}{du} \approx \frac{\Delta s}{1}.$$

We shall approximate $\dot{r}$ by

$$\frac{dr}{ds}\Delta s.$$

In the case of an endpoint, we shall make the obvious modification. Let the curve parameter t take values on the interval $[0, n-1]$. Then if $k \leq t \leq k+1$, the local parameter is

$$u = t - k.$$

This curve interpolates the given values, and is continuous, and has continuous velocity. Also the chord length approximates the arc length, and the

curve should tend to have the same shape as the data. Note that if the speeds at the knots, the $\|\dot{r}\|$, are arbitrary, then the curve may overshoot and may have cusps or loops, and then the chord length may not approximate the arc length, and the shape may not be the same as the data. We may wish to convert the resulting curve to Bezier or B-spline form.

32 On The Determination Of Maximum Curvature Of A Cubic (Dixon)

Cubic splines have found frequent use in computer aided design and manufacturing. As such it is of interest to be concerned as to the nature of its' curvature, in particular where its' maximum curvature occurs. Towards this end, let us consider a general cubic given by:

$$f(x) = ax^3 + bx^2 + cx + d \tag{1}$$

32.1 The Analysis

The curvature of (1) is defined as:

$$\kappa(x) = \frac{f''(x)}{[1 + f'(x)^2]^{\frac{3}{2}}} \tag{2}$$

We elect not to use the absolute value of the second derivative as we are concerned here about the nature or direction of the curvature as well as its' magnitude.

By setting the derivative of (2) equal to zero, we can determine the critical points of (2), hence locating where its' curvature is maximum. After doing this and some algebra we obtain:

$$f'''(x)[1 + f'(x)^2] - 3f''(x)^2 f'(x) = 0 \tag{3}$$

Substituting the expressions for the first, second and third derivatives of (1) into (3), and more algebra we have a quartic equation given by:

$$a[1 + (3ax^2 + 2bx + c)^2] - 2(3ax + b)^2(3ax^2 + 2bx + c) = 0 \tag{4}$$

Let us consider the function of the left hand member of (4) given by:

$$g(x) = a[1 + (3ax^2 + 2bx + c)^2] - 2(3ax + b)^2(3ax^2 + 2bx + c) \tag{5}$$

We will prove two properties: first, that $\tilde{x} = \frac{-b}{3a}$ is a root of $g'(x)$ and second, that (4) has a pair of complex roots for which $\tilde{x}$ is the real part. We have:

$$g'(x) = (3ax + b) \cdot G(x) \tag{6}$$

where,

$$G(x) = -4[a(3ax^2 + 2bx + c) + (3ax + b)^2] \tag{7}$$

This shows the first property that we wanted to prove, that $g'(\tilde{x}) = 0$. This also implies that $(\tilde{x}, g(\tilde{x}))$ is a critical point of $g(x)$. Closer analysis reveals that $\tilde{x}$ is a root of the second derivative of (1), which means that it is a root of (2). And verification of the second property indicates that it is the value of the independent variable where (1) changes its' concavity.

Let us now write $g(x)$ in standard form as follows:

$$g(x) = Ax^4 + Bx^3 + Cx^2 + Dx + E \tag{8}$$

where,

$$A = -45a^3 \tag{9}$$
$$B = -60a^2b \tag{10}$$
$$C = -2a(6ac + 13b^2) \tag{11}$$
$$D = -4b(2ac + b^2) \tag{12}$$
$$E = a + ac^2 - 2b^2c \tag{13}$$

It is well known that cubics are anti symmetrical about their inflection points. Hence, (1) is antisymmetrical about $\tilde{x}$. We have in (8) a quartic that was derived from setting the derivative of (2) equal to zero and simplifying. Therefore, since 'A' determines whether its' tails will be above or below the x-axis, to prove the second property it suffices to show that $A \cdot G(\tilde{x}) < 0$.

Substituting $\tilde{x} = \frac{-b}{3a}$ into (8) we get,after some algebra:

$g(\tilde{x}) = a[1 + (c - \frac{b^2}{3a})^2]$

Thus, $A \cdot g(\tilde{x}) = -45a^4[1 + (c - \frac{b^2}{3a})^2] < 0$

Q.E.D.

Let the complex roots of (8) be given by:

$$\left.\begin{array}{l} r_1 = \tilde{x} + iv \\ r_2 = \tilde{x} - iv \end{array}\right\} \tag{14}$$

Then one of the quadratic factors of (8) is:

$$q_1(x) = (x - r_1)(x - r_2) = x^2 - 2\tilde{x}x + (\tilde{x}^2 + v^2) \tag{15}$$

Let the other quadratic factor be given by:

$$q_2(x) = \alpha x^2 + \beta x + \gamma \tag{16}$$

Then we have:

$$g(x) = q_1(x) \cdot q_2(x) \tag{17}$$

or

$$Ax^4 + Bx^3 + Cx^2 + Dx + E = [x^2 - 2\tilde{x}x + (\tilde{x}^2 + v^2)][\alpha x^2 + \beta x + \gamma] \tag{18}$$

And after some algebra we get:

$$\begin{aligned} A &= \alpha && (19)\\ B &= \beta - 2\tilde{x}\alpha && (20)\\ C &= \gamma - 2\tilde{x}\beta + \alpha(\tilde{x}^2 + v^2) && (21)\\ D &= \beta(\tilde{x}^2 + v^2) - 2\tilde{x}\alpha && (22)\\ E &= \gamma(\tilde{x}^2 + v^2) && (23) \end{aligned}$$

From (19) and (20) we readily get $\alpha = A$ and β, where

$$\beta = B + 2\tilde{x}A \tag{24}$$

And from (23) we get

$$\tilde{x}^2 + v^2 = \frac{E}{\gamma} \tag{25}$$

Making these substitutions in (21) we obtain:

$$\gamma - 2\tilde{x}(B + 2\tilde{x})A + \frac{AE}{\gamma} = C \tag{26}$$

or

$$\gamma^2 - [2\tilde{x}(B + 2\tilde{x}A)]\gamma + AE = C \tag{27}$$

Using the quadratic formula to solve for γ we get:

$$\gamma = \frac{[2\tilde{x}(B + 2\tilde{x}A) + C] \pm \sqrt{[2\tilde{x}(B + 2\tilde{x}A) + C]^2 - 4AE}}{2} \tag{28}$$

Since α is a non zero function of a, we use (21) to solve for v. Which yields:

$$v = \sqrt{\frac{C - \gamma' + 2\tilde{x}(B + 2\tilde{x}A)}{A} - \tilde{x}^2} \tag{29}$$

Where γ' is the γ obtained from (28) using the sign that causes the v in (29) to be real.

Thus, we have the complex roots of (8) given by (14), and since a factor of (8) is the quadratic expression in (15)

$$x^2 - 2\tilde{x}x + (\tilde{x}^2 + v^2) \tag{30}$$

by dividing (30) into (8) we can obtain the quadratic expression given in (16), which is the other factor of (8). The result of carrying out this operation, or by simply using (19),(20) and (21), yields:

$$Ax^2 + (B + 2A\tilde{x})x + \{[C - A(\tilde{x}^2 + v^2)] + 2\tilde{x}(B + 2A\tilde{x})\} \tag{31}$$

The quadratic formula readily gives the roots of (31), which are the real roots of (8) and the values of the independent variable where the cubic (1) attains maximum curvature. That is $(\hat{x}, \kappa(\hat{x}))$ is a point of maximum curvature of (1), where $\hat{x}$ is a root of (31).

33 The Trapezoid Rule For Numerical Integration

The integral of a real function on an interval $[a, b]$ may be approximated by dividing the interval into n subintervals

$$a = x_1 < x_2 < ... < x_n = b,$$

and taking the sum of the trapezoidal areas formed under the curve at each subinterval. Thus we have approximately

$$\int_a^b f(x)dx = \Delta x \left[\frac{f(a)}{2} + \sum_{i=2}^{n-1} f(x_i) + \frac{f(b)}{2} \right],$$

where Δx is the length of the subintervals. The truncation error of the trapezoid rule is relatively large. When we can evaluate the function freely at any point, the trapezoid rule can be combined with Richardson extrapolation to give an accurate algorithm called Romberg integration.

34 The Truncation Error of the Trapezoid Rule

Consider the integral

$$\int_x^{x+h} f(u)du.$$

Let $g' = f$. Then

$$\begin{aligned}\int_x^{x+h} f(u)du &= g(x+h) - g(x)\\ &= g'(x)h + g''(x)h^2/2! + ...\\ &= f(x)h + f'(x)h^2/2! + ...\end{aligned}$$

and

$$f(x+h) \ = f(x) + f'(x)h + f''(x)h^2 2! + ...$$

Solving for $f'(x)$, we get

$$f'(x) = [f(x+h) - f(x) - f''(x)h^2/2! - ...]/h.$$

Substituting for f', we have

$$\int_x^{x+h} f(u)du = f(x)h + (h/2)[f(x+h) - f(x) - f''(x)h^2/2! - ..] + f''(x)h^3/6 + ...$$

$$\begin{aligned}&= (f(x+h) + f(x))h/2 + f''(x)h^3(1/6 - 1/4) + ...\\ &= (f(x+h) + f(x))h/2 - f''(\xi)h^3/12,\end{aligned}$$

where

$$x \leq \xi \leq x + h.$$

35 The Adams-Moulton Formula

We will compute the approximate solution to first order differential equation

$$\frac{dx}{dt} = f(t, x),$$

as a discrete sequence

$$\{(t_i, x_i) : i = 1, 2, 3, 4, ...\}.$$

Suppose $x(t)$ is the actual solution of the differential equation. Using the trapezoid rule

$$x(t_{i+1}) - x(t_i) = \int_{t_i}^{t_{i+1}} f(t, x(t))dt$$

$$= h(f(t_i, x(t_i)) + f(t_{i+1}, x(t_{i+1}))/2 - (h^3/12)x(\xi).$$

We compute the numerical solution sequence iteratively using the implicit equation

$$x_{i+1} = x_i + (1/2)(f(t_i, x_i) + f(t_{i+1}, x_{i+1}))h.$$

Then

$$x_{i+1} - x(t_{i+1}) = x - x(t_i) + (1/2)(f(t_i, x_i) - f(t_i, x(t_i)))h$$

$$+(1/2)(f(t_{i+1}, x_{i+1}) - f(t_{i+1}, x(t_{i+1})))h$$

$$-(h^3/12)x^{(3)}(\xi).$$

$$= 0 + 0 + \frac{\partial f(t_{i+1}, \eta)}{\partial x}(x_{i+1} - x(t_{i+1})) - (h^3/12)x^{(3)}(\xi).$$

Solving this equation

$$x_{i+1} - x(t_{i+1}) = 1/[1 - \frac{h}{2}\frac{\partial f(t_{i+1}, \eta)}{\partial x}]\frac{h^3}{12}x^{(3)}(\xi).$$

Expanding the first factor as a geometric series we have

$$x_{i+1} - x(t_{i+1}) = \frac{h^3}{12}x^{(3)}(\xi).$$

This is an implicit third order method. The implicit equation for x_{i+1} can be solved by fixed point iteration. The equation takes the form $z = g(z)$ and for small enough h,

$$\left|\frac{dg}{dz}\right| \leq 1.$$

So the iteration converges. This formula, which functions as a corrector, may be used in a predictor-corrector method. The predictor is the formula,

$$x_{i+1} = x_{i-1} + 2hf(t_i, x_i),$$

derived from the central difference approximation to the derivative. Expanding $x(t_{i+1})$ and $x(t_{i-1})$ in a Taylor series, and subtracting, we get

$$x(t_{i+1}) - x(t_{i-1}) = 2x'(t_i)h + 2x'''(t_i)h^3/3! + ...$$

Taking $x_{i-1} = x(t_{i-1})$ and $x_i = x(t_i)$ we find

$$x(t_{i+1}) - x_{i+1} = x'''(t_i)h^3/3 + ...$$

So the local truncation error is of order three. The difference of the predictor and corrector gives an estimate of the local truncation error

36 The Euler Method

Consider the differential equation

$$y' = f(x, y)$$

This solution method is the first order approximation

$$x_2 = x_1 + h$$

$$y_2 = y_1 + f(x_1, y_1)h.$$

37 The Taylor Series Method

Consider the differential equation

$$y' = f(x, y)$$

Let us express the solution as a Taylor series

$$y = \sum_{j=0}^{\infty} \frac{y^{(j)}(x_i)h^j}{j!}.$$

We have

$$y''(x) = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} f$$

Let $f_{i,j}$ be the ijth partial of f. Then we may write

$$y'' = f_1 + f_2 f$$

We may continue this and get

$$y''' = f_{11} + 2f_{12}f + f_{22}f^2 + f_1 f_2 + f_2^2 f,$$

and so on. Thus, knowing $f(x, y)$ we may compute the partial derivatives of f to get the derivatives of y, and then substitute these into the Taylor series to get the next step of y. The truncation error then depends on the the number of terms taken.

38 Runge-Kutta Methods

A Runge-Kutta method gives a solution to the differential equation

$$y'(x) = f(x, y)$$

in the form

$$y_{i+1} = y_i + \sum_{j=1}^{n} a_j k_j$$

where

$$k_1 = hf(x_i, y_i)$$

$$k_2 = hf(x_i + p_1 h, y_i + q_{11} k_1)$$

$$k_3 = hf(x_i + p_2 h, y_i + q_{21} k_1 + q_{22} k_2)$$

$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$$

$$k_n = hf(x_i + p_{n-1} h, y_i + q_{n-1,1} k_1 + \dots + q_{n-1,n-1} k_{n-1})$$

We derive these equations by equating terms with the equations given by the Taylor series method.

39 Derivation of a Second Order RungeKutta Method

See University of Missouri, UMKC, Engineering 304 notes.

40 The Fourth Order RungeKutta Method

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = hf$$

$$k_2 = hf(x_i + h/2, y_i + k_1/2)$$

$$k_3 = hf(x_i + h/2, y_i + k_2/2)$$

$$k_4 = hf(x_i + h, y_1 + k_3)$$

The error per step is of order h^5.

A Fortran subroutine for a single step:

```
c+ rk4je runge-kutta differential equation step
      subroutine rk4je(y,dydx,n,x,h,yout,derivs)
      implicit real*8(a-h,o-z)
c     modified version of rk4 (numerical recipes)
c  input:
c  y      initial values of vector function
c  dydx   initial values of vector derivative
c  n      number of first order equations
c         y, dydx are n vectors.
c  x      independent variable
c  h      independent variable step
c  output:
c  yout   value of vector function after step h
c  derivs name of external subroutine that
c         calculates vector derivative.
c  Equation that is solved:
c  dy/dx = f(x,y), where f is defined by subroutine derivs
      parameter (nmax=20)
      dimension y(*),dydx(*),yout(*),yt(nmax),dyt(nmax),dym(nmax)
      external derivs
      hh=h*0.5
      h6=h/6.
      xh=x+hh
      do i=1,n
       yt(i)=y(i)+hh*dydx(i)
      enddo
      call derivs(xh,yt,dyt)
      do i=1,n
```

```
        yt(i)=y(i)+hh*dyt(i)
      enddo
      call derivs(xh,yt,dym)
      do i=1,n
       yt(i)=y(i)+h*dym(i)
       dym(i)=dyt(i)+dym(i)
      enddo
      call derivs(x+h,yt,dyt)
      do i=1,n
       yout(i)=y(i)+h6*(dydx(i)+dyt(i)+2.*dym(i))
      enddo
      return
      end
```

A C++ program example:

```
//rk4je.cpp
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
main(){
 void rk4je(double *,double *,int,double,double,
 double *,void (*)(double,double *,double *));
 void der1(double,double *,double *);
 int i,m,n,np,npa,k;
 double y[2],dydx[2],yout[2],x,xend,h;
 //c     starting value of independent variable:
 x=0.;
 #define pi 3.14159265358979
 //c     ending value of independent variable:
 xend=pi/2.;
 //c     approximate step size:
 h=.001;
 //c     number of steps:
 m=(xend-x)/h;
 //c     adjusted step size:
 h=(xend-x)/m;
 //c     approximate number of print points:
 npa=15;
 //c     steps between prints:
 np=m/npa;
 if(np < 1)np=1;
 n=2;
 y[0]=0.;
 y[1]=1.;
 printf("%d %21.14g %21.14g %21.14g\n",0,x,y[0],sin(x));
 for(i=1;i<=m ;i++){
  //c      calculate derivatives of y at x
  der1(x,y,dydx);
  rk4je(y,dydx,n,x,h,yout,der1);
  //c      update values:
  x=x+h;
  y[0]=yout[0];
  y[1]=yout[1];
  k= i % np;
  if((k  ==  0)  ||  (i  ==  m)){
```

```
   printf("%d %21.14g %21.14g %21.14g\n",i,x,y[0],sin(x));
  }
 }
}
//c+ rk4je runge-kutta differential equation step
 void rk4je(double *y,double *dydx,int n,double x,double h,
  double *yout,void (*derivs)(double,double *,double *)){
 //c     modified version of rk4 (numerical recipes)
 //c  input:
 //c  y      initial values of vector function
 //c  dydx   initial values of vector derivative
 //c  n      number of first order equations
 //c         y, dydx are n vectors.
 //c  x      independent variable
 //c  h      independent variable x step
 //c  output:
 //c  yout   value of vector function after step h
 //c  derivs name of external subroutine that
 //c         calculates vector derivative.
 //c  Equation that is solved:
 //c  dy/dx = f(x,y), where f is defined by subroutine derivs
 double hh,h6,xh;
 int i;
 #define nmax 20
 double yt[nmax],dyt[nmax],dym[nmax];
 hh=h*0.5;
 h6=h/6.;
 xh=x+hh;
 for(i=0;i<n;i++){
  yt[i]=y[i]+hh*dydx[i];
 }
 derivs(xh,yt,dyt);
 for(i=0;i<n ;i++){
  yt[i]=y[i]+hh*dyt[i];
 }
 derivs(xh,yt,dym);
 for(i=0;i<n ;i++){
  yt[i]=y[i]+h*dym[i];
  dym[i]=dyt[i]+dym[i];
 }
 derivs(x+h,yt,dyt);
 for(i=0;i<n ;i++){
  yout[i]= y[i]+h6*(dydx[i]+dyt[i]+2.*dym[i]);
 }
}
//c+ der1 derivative values, right side of differential equation system
void der1(double x,double y[],double dy[]){
 //External Functions:
 //c     Second order differential equation
 //c     d^2 w/dx^2 + w = 0,
 //c     Initial conditions: w(0)=0, dw(0)/dx = 1
 //c     Exact solution: w(x) = sin(x)
 //c     To convert to a first order system of two equations
 //c     let new variables y_i equal the derivatives:
 //c     y_1=w, y_2= dw/dx
 //c     Then we get a system of two first order equations:
 //c     d y_1/dx = f_1(x,y) =  y_2
```

```
 //c     d y_2/dx = f_2(x,y) = -y_1
 dy[0]=y[1];
 dy[1]=-y[0];
}
```

41 Richardson Extrapolation

41.1 A Single Extrapolation

Suppose $q > p$ and

$$F(h) = a_0 + a_1 h^p + O(h^q).$$

We want to calculate a_0 as the limit of $F(h)$ as h goes to zero. We have

$$F(2h) = a_0 + a_1 2^p h^p + O(h^q).$$

We may take a linear combination of these two functions to get a function of order q in h. Thus

$$\frac{2^p F(h) - F(2h)}{2^p - 1} = a_0 + O(h^q).$$

Define

$$F_2(h) = \frac{(2^p - 1) + 1)F(h) - F(2h)}{2^p - 1} = F(h) + \frac{F(h) - F(2h)}{2^p - 1}.$$

Then

$$F_2(h) = a_0 + O(h^q).$$

Hence for h small, $F_2(h)$ is an improved approximation to a_0. The technique can be repeated.

41.2 Repeated Richardson Extrapolation

Suppose

$$F(h) = a_0 + a_1 h^{p_1} + a_2 h^{p_2} + a_3 h^{p_3} + ...$$

where

$$p_1 < p_2 < p_3 < ....$$

Define

$$F_1(h) = F(h),$$

and

$$F_{k+1}(h) = F_k(h) + \frac{F_k(h) - F_k(2h)}{2^{p_k} - 1}.$$

Proposition.

$$F_k(h) = a_0 + a_k^{(k)} h^{p_k} + a_{k+1}^{(k)} h^{p_{k+1}} + ..$$

Proof. This is proved by induction. Assuming the result for k, we find that the term involving h^{p_k} in $F_{k+1}(h)$ is

$$a_k^{(k)} h^{p_k} + \frac{a_k^{(k)} h^{p_k} - a_k^{(k)} 2^{p_k} h^{p_k}}{2^{p_k} - 1}$$

$$= a_k^{(k)} h^{p_k} + \frac{a_k^{(k)} h^{p_k}(1 - 2^{p_k})}{2^{p_k} - 1} = 0.$$

Thus

$$a_k^{(k+1)} = 0.$$

This proves the proposition.

42 Common Applications

Two common applications of Richardson extrapolation include numerical differentiation and numerical integration. The central difference approximation for a derivative and the trapezoid approximation to an integral, both with step size h, deliver their values as a limit as h goes to zero, where the nonzero coefficients of the powers of h are even. That is

$$p_1 = 2, p_2 = 4, ..., p_k = 2k.$$

So

$$2^{p_k} = 4^k - 1.$$

The calculations may be displayed in tabular form

$$\begin{array}{cccc} F_1(h) & & & \\ F_1(h/2) & F_2(h/2) & & \\ F_1(h/4) & F_2(h/4) & F_3(h/4) & \\ ... & ... & ... & ... \end{array}$$

The calculation is stopped when sufficient accuracy has been reached as determined by the differences of neighboring values in the table.

43 Solving Differential Equations Using Finite Differences

Consider Laplace's equation in two dimensions:

$$\nabla^2 f = 0.$$

That is

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0.$$

Expanding f as a Taylor series , we have

$$f(x+h,y) = f(x,y) + \frac{\partial f}{\partial x}h + \frac{1}{2!}\frac{\partial^2 f}{\partial x^2}h^2 + ...$$

$$f(x-h,y) = f(x,y) - \frac{\partial f}{\partial x}h + \frac{1}{2!}\frac{\partial^2 f}{\partial x^2}h^2 + ...$$

$$f(x,y+h) = f(x,y) + \frac{\partial f}{\partial y}h + \frac{1}{2!}\frac{\partial^2 f}{\partial y^2}h^2 + ...$$

$$f(x,y-h) = f(x,y) - \frac{\partial f}{\partial y}h + \frac{1}{2!}\frac{\partial^2 f}{\partial y^2}h^2 + ...$$

If we neglect terms in h of order higher than 2, and add these four equations, we get

$$f(x,y) = \frac{1}{4}(f(x+h,y) + f(x-h,y) + f(x,y+h) + f(x,y-h)).$$

Then if we divide the domain into a grid with increment h, f is equal to the average of its four neighbors. We can solve the problem by computing this average at each grid point, over and over, until we get convergence.

44 Central Force Example

See Boyce and DiPrima p65 for related escape velocity problem. Consider the simple differential equation of a one dimensional central force.

$$\frac{dx^2}{dt^2} = -\frac{1}{x^2}.$$

The velocity is

$$v = \frac{dx}{dt}.$$

Then

$$\frac{dv}{dt} = \frac{dv}{dx}\frac{dx}{dt} = v\frac{dv}{dx}.$$

Then

$$v\frac{dv}{dx} = -\frac{1}{x^2}.$$

$$vdv = -\frac{dx}{x^2}.$$

So

$$\frac{v^2}{2} = \frac{1}{x} + c$$

$$c = \frac{v_0^2}{2} - \frac{1}{x_0}.$$

$$v = \sqrt{\frac{2(1+cx)}{x}}$$

$$\frac{dt}{dx} = \frac{1}{v} = \sqrt{\frac{x}{2(1+cx)}}.$$

$$t = \int_{x_0}^{x} \sqrt{\frac{u}{2(1+cu)}}du.$$

See **moon.tex** for Euler numerical solution to related problem.

45 Stiff Equations

As was shown above, a nonlinear equation may be locally approximated by a linear equation. If the eigenvalues of the Jacobian are distinct, then the Jacobian can be diagonalized, and the equations become uncoupled. When the eigenvalues are not distinct, we can work with the Jordan normal form of the matrix. But for purposes of analysis, if the multiple roots of the characteristic equation are changed to give distinct nearly equal roots, we have a good approximation to the original problem, whose jacobian can be diagonalized. Then we are dealing with a one dimensional linear equation. We will show how Euler's method applies to the linear approximation to a first order differential equation. We have

$$x_{n+1} = x_n + hf(t_n, x_n).$$

From Taylor's theorem,

$$x(t_{n+1}) = x(t_n) + x'(t_n)h + x''(\xi)h^2/2,$$

where

$$t_n \leq \xi t_{n+1}.$$

The truncation error is $x''(\xi)h^2/2$. The step size h is chosen to satisfy

$$\epsilon = \|x''(\xi)h^2/2\|,$$

where ϵ is the tolerance for the truncation error. For a system of equations there will be a set of such relations and the largest value satisfying all of them will become the step size h. As the solution continues, the step size will change. Suppose one of the equations of the system has a large negative eigenvalue, then the variable will quickly decay to zero. Then h should increase to solve the system in a reasonable time. But as we shall show there is also a stability requirement that does not allow the step size to be increased. Consider the following example, which is from chemical kinetics. Consider the reactions,

$$A + B \rightarrow C$$

and

$$C + D \rightarrow A,$$

which lead to the differential equations

$$dB/dt = -k_1 AB$$

and

$$dD/dt = -k_2 CD,$$

where

$$A = A_0 - (B - B_0) + (D - D_0)$$

and

$$C = C_0 + (B - B_0) - (D - D - 0).$$

Note that $A + C$ is constant. Suppose

$$A_0 = 2, C_0 = 5, B_0 = .001$$

and

$$D_0 = .0001.$$

Then

$$dB/dt = -(k_1 A_0)B$$

and

$$dD/dt = -(k_2 C_0)D.$$

The solutions are

$$B = B_0 exp(-k_1 A_0 t)$$

and

$$D_0 = D\ exp(-k_2 C_0 t).$$

Suppose $k_1 = 1000$ and $k_2 = 1$. Using Euler's method,

$$B_{n+1} = (1 - 2000h)B_n$$

and

$$D_{n+1} = (1 - 5h)D_n.$$

$B(t)$ goes rapidly to zero and B_n gets very small. But h can not be increased to a reasonable value suitable for the second equation. Because if $\|1 - 2000h\| > 1$, then B_{n+1} will be magnified, and will eventually become arbitrarily large. This is the stiffness problem. The global error is defined to be

$$\delta = x_n - x(t_n).$$

We have

$$x_{n+1} = x_n + ((x_n - g(t_n))J + g'(t_n))h.$$

$$= \delta_n + x(t_n) + ((\delta_n + x(t_n) - g(t_n))J + g'(t_n))h,$$
$$= \delta_n 1 + x(t_n) + \delta_n hJ + x'(t_n)h$$
$$= \delta_n(1 + Jh) + x(t_n) + x'(t_n)h$$
$$\delta_{n+1} = \delta_n(1 + Jh) + (x(t_n) + x'(t_n)h - x(t_{n+1}.$$

The global error is amplified unless either $-1 \leq 1 + Jh \leq 1$ or, $-2 \leq Jh \leq 0$. This controls stability. Consider the backward Euler method,

$$x_{n+1} = x_n + hf(t_{n+1}, x_{n+1}).$$

We find

$$\delta_{n+1} = (1 - hJ)^{-1}\delta_n + (1 - hJ)^{-1}h^n x''(t_n)/2 + O(h^3).$$

The error is damped when $\|1/(1 - hJ)\| \leq 1$, which holds for J in the left half plane. So there is no longer a stability limitation. Methods for stiff equations are implicit. Often fixed point iteration can not be used to solve the implicit equation. Some form of Newton's method is used in this case. The Jacobian must be repeatedly calculated. This can be expensive.

46 Eigenvalues

Given a matrix A, let A^* be the complex conjugate of the transpose. A matrix is hermitian if $A = A^*$. A real hermitian matrix is called a symmetric matrix and then $A = A^T$. The inner product of a complex vector X and a complex vector Y is , $(X, Y) = X^*Y$. The norm or length of a vector X is $\|X\| = \sqrt{(X, X)}$,

$$(X, X) = \bar{x}_1 x_1 + ... + \bar{x}_n x_n = |x_1|^2 + ... + |x_n|^2.$$

Proposition. The eigenvalues of a hermitian matrix are real.
Proof.

$$\lambda(X, X) = (X, \lambda X) = (X, AX) = (A^*X, X) = (AX, X) = (\lambda X, X) = \bar{\lambda}(X, X).$$

Hence $\lambda = \bar{\lambda}$ so λ is real.
Proposition. The eigenvectors corresponding to distinct eigenvalues are linearly independent.

Proposition. If A is Hermitian (symmetric) and eigenvalues λ_1 and λ_2 are not equal, then corresponding eigenvalues are orthogonal. That is

$$(V_1, V_2) = 0.$$

The Spectral Theorem For Real Symmetric Matrices. If A is symmetric then there exists a diagonal matrix D of eigenvalues and an orthogonal matrix V whose columns are eigenvectors so that,

$$AV = VD$$

and

$$V^T AV = D.$$

Note that a matrix is orthogonal if $V^T = V^{-1}$ This is equivalent to the column vectors being orthonormal. An orthogonal transformation preserves the length of vectors.

47 The Jacobi Method For Locating Eigenvalues

Given a symmetric matrix A. Let R be a rotation matrix that transforms the element of A in the pth row and qth column to zero. It is defined by

$$r_{pq} = \sin(\phi), r_{qp} = -\sin(\phi), r_{pp} = \cos(\phi), r_{qq} = \cos(\phi)$$

all other elements on the main diagonal are 1, and all remaining elements of R are zero. The angle ϕ is chosen so that the element of A' in the pth row and qth column is zero, where

$$A' = R^T AR.$$

In what follows let

$$s = \sin(\phi)$$

and

$$c = \cos(\phi).$$

Let $B = AR$, then $A' = R^T B$. Then

$$a'_{pq} = \sum r_{ip} b_{iq} = \sum r_{ip} \sum a_{ij} r_{jq}$$

$$= r_{pp}(a_{pp}r_{pq} + a_{pq}r_{qq}) + r_{qp}(a_{qp}r_{pq} + a_{qq}r_{qq})$$
$$= c(a_{pp}s + a_{pq}c) - s(a_{qp} + a_{qq}c)$$
$$= sc(a_{pp} - a_{qq} + a_{pq}(c^2 - s^2) = 0.$$

Rearranging the last equation,

$$(c^2 - s^2)/(2sc) = (a_{pp} - a_{qq})/(2a_{pq}) = \alpha.$$

Then with α defined by this equation and t defined to be s/c, we get a quadratic equation for t.

$$t^2 + 2\alpha t - 1 = 0.$$

If $\alpha = 0$, then we take the root to be 1, otherwise the root of largest magnitude is,

$$-\alpha - \text{sign}(\alpha)\sqrt{(\alpha^2 + 1)} = -\text{sign}(\alpha)(|\alpha| + \sqrt{\alpha^2 + 1}).$$

Dividing the constant coefficient, which is -1, by this root we get the root of smallest magnitude,

$$\frac{\text{sign}(\alpha)}{(|\alpha| + \sqrt{\alpha^2 + 1})}.$$

The magnitude of the root t is thus less than 1. So the rotation angle, of which t is the tangent, is less than $\pi/4$. The tangent is t, so

$$c = 1/(t^2 + 1)$$

and

$$s = ct.$$

We define a sequence of matrices by, $A_0 = A$, and $A_{n+1} = R_n^T A_n R_n$, where R_n is a rotation matrix that zeroes some off-diagonal element of A . The sequence $\{A_n\}$ converges to a diagonal matrix D. Let $V = R_1 R_2 R_3 .... R_n ...$, then,

$$D = V^T AV.$$

V is an orthogonal matrix because it is the product of orthogonal matrices. Hence $V^T = V^{-1}$ and so $AV = VD$, which shows that the diagonal elements of D are the eigenvalues of A with the columns of V being the corresponding eigenvectors.

To show that the sequence does converge, we define S to be the sum of the squares of the off-diagonal elements of A. Evaluating the components of A', we find

$$a'_{rp} = ca_{rp} - sa_{rp}$$

and

$$a'_{rq} = ca_{rq} + sa_{rp}$$

for r not q or not p. Squaring and adding,

$$(a'_{rp})^2 + (a'_{rq})^2 = (c^2 + s^2)(a_{rp}^2 + a_{rq}^2)$$

$$= (a_{rp}^2 + a_{rq}^2)$$

So the sum of the squares of the off-diagonal elements is invariant, except for a_{pq}^2 and a_{qp}^2, which are replaced by zero. Thus $S' = S - 2a_{pq}^2$. So $\{S_n\}$ is a decreasing sequence bounded by zero. So it converges. We may zero the off-diagonal element of largest magnitude at each step and thus make S_{n+1} decrease by at least a fixed proportion:

$$S_{n+1} \leq \beta S_n \leq \beta^n S_1,$$

where $\beta < 1$. So $\{S_n\}$ converges to zero, which implies that $\{A_n\}$ converges to a diagonal matrix D.

If we carry out this whole calculation in a slightly different way, zeroing the off-diagonal elements in a cyclic order, then we still get convergence to zero. And then we do not need to find the largest element.

Note. The Jacobi method proves the spectral theorem for symmetric matrices.

48 Leverrier's Method for the Characteristic Polynomial

This method uses Newton's identities and the trace of powers of the matrix. Given a polynomial,

$$x^n + c_1 x^{n-1} + ... + c_1 = (x - \alpha_1)(x - \alpha_2)....(x - \alpha_n).$$

Define $s_k = \sum \alpha_i^k$ to be the sum of the kth powers of the roots. The following equations are known as Newton's identities:

$$s_1 + c_1 = 0$$

$$s_2 + c_1 s_1 + 2c_2 = 0$$

$$s_3 + c_1 s_2 + c_2 s_1 + 3c_3 = 0$$

$$.....$$

$$s_n + c_1 s_{n-1} + ... + c_{n-1} s_1 + nc_n = 0$$

For a proof see a classic algebra book such as: **The Theory of Equations**, by J.V. Uspensky. These identities allow the systematic determination of the coefficients of the polynomial from its roots. Definition: The trace of a matrix is the sum of its diagonal elements.
Proposition. Trace(AB) = Trace(BA).
Proof.

$$\text{Trace}(AB) = \sum\sum A_{ik}B_{ki} \sum\sum B_{ki}A_{ik} = \text{Trace}(BA).$$

From this proposition it follows that, + -1

$$\text{Trace}(B^{-1}AB) = \text{Trace}(A).$$

Proposition. Let matrix A have eigenvalues $\lambda_1, ..., \lambda_n$. Then

$$\text{Trace}(A^k) =\sim \lambda_i^k = s_k.$$

The steps of Leverrier's method are: (1) Compute A^k and $s_k = \text{Trace}(A^k)$, (2) Use Newton's identities repeatedly to get the coefficients of the characteristic polynomial.

Faddeev's modification of the method consists in a more efficent calculation method. Refer to the book by James, Smith, and Wohlford.

49 Scientific Typesetting: TeX and Latex

TeX is a typsetting system invented by Don Knuth. It is used widely in Science, Engineering and Mathematics to write papers, books and documents. The system has the capability to add commands. Latex is a widely used system that runs on top of TeX with some simplified and extended commands.

An input file for Latex is an ascii text file with a tex extension as in sample.tex. Tex and Latex are available for most computers free of cost. There are also commercial versions as in PCTeX. MikTex is a free version that runs on Windows. Linux systems come with a version of TeX and Latex. There are versions for MacOS.

50 Downloading, Installing, and Running MikTex on Windows

The file to be downloaded is

```
basic-miktex-2.7.3248.exe
```

This file is 86,335,752 bytes, it was downloaded on 12/1/2008 from

```
http://miktex.org/2.7/Setup.aspx
```

I put the file in a directory that I created called

```
c:\latex
```

I create the tex source files in this directory and run the MikTeX programs from there. But this can be done from any directory. When I ran basic-miktex-2.7.3248.exe by typing its name, everything was installed and the programs were ready to go.

MiKTeX is a program to be run from the command line. The path variable has been altered by the installation to add the binary directory of MiKTeX to the path. This means that the operating system will find the programs from any directory. We will run LaTeX on a source file that is created in a text editor such as notepad.

There is a more gui-like free TeX version for MacOSX called MacTeX, which I downloaded and installed. It produces a pdf file directly. There are probably gui-like Windows versions like this also, that are available. There is a national organization that deals with TeX and LateX and distributes versions. It is called CTAN. They also publish a magazine called Tugboat. I used to subscribe years ago. They have a new distribution called Live TeX , I think. There are live versions of TeX that run directly from a CD or DVD without being installed on the hard drive.

A LaTeX system comes with most Linux distributions.

Latex source is a simple txt file, and as such will always be available and editable. One can not say this about your masterpiece written in Wordstar, WordPerfect, or even Microsoft Word.

51 Creating a Sample LaTeX File

The source for a TeX or LaTeX file is a text file containing content and markup commands. It should have *tex* as the file extension. Notepad may try to put a ".txt" at the end of the saved file name. Beware of this. There are better editors.

Assuming we have created a file called sample.tex, we process it with LaTex with the following command:

```
latex sample.tex
```

If there are errors in the source file, LaTeX will stop at the error and wait for you to correct the error. This is a feature that Don Knuth put into the original TeX to save time in correcting errors because computers were very slow maybe. But it can be maddening. I never do any correcting, but rather type an "x" to escape from LaTeX, while making note of the line number of the error. It is best to use an editor that gives line numbers. I correct the error or errors in the editor , save and then call LaTeX again. This can be done quickly. One should run LaTeX frequently as the material is being written, so that any error that appears, will be in the latest content just written, and so easily found.

Now assuming that the LaTex program runs to completion, three or four files will be created. One of these is the typeset output file, which in this case will be called sample.dvi. This is a device independent file. From this file one creates a printable file. Usually one creates a postscript file, which in the old days could only be printed on a Postscript printer such as the Apple Laser Writer. There is a program called "dvips" that converts the dvi file to a postscript file. The dvips program for MiKTeX runs as follows.

```
dvips sample.dvi
```

It produces a postscript file called "sample.ps." In the old days one would send this file to a postscript printer. One can use the free public program Ghostscript, or a windows gui version Ghostview to view and then print the file to any printer.

Even better, one can convert the program to an Acrobat pdf file. There is such a converter that comes with MiKTeX.

```
ps2pdf sample.ps
```

This creates a pdf file called "sample.pdf," which may be viewed and printed with Acrobat reader. To open the pdf file in Windows from the command line , do this

```
start sample.pdf
```

As an aside, on the Macintosh OSX system, an equivalent action from the command line would be

```
open sample.ps
```

On MacOS the sample.ps file is converted to pdf and displayed in **Presenter**. In presenter one may save the displayed file as sample.pdf. If you happen to have Acrobat Distiller on your windows system, conversion to pdf can be done on the command line like thus:

```
start sample.ps
```

If you are using Notepad as your editor, you should open it outside of the directory you are working in so that it will remain open as you modify your tex source and typeset it from the command line. When you find an error at a given line, you can goto that line in Notepad by selecting the goto item from the Notepad edit menu. Notepad is, of course, not the greatest editor, although cutting and pasting works well.

52 A Listing of the File sample.tex

Below is a listing of the LaTeX file called **sample.tex**. View the final pdf output at:

www.stem2.org/je/sample.pdf

```
\documentstyle[12pt]{article}
\title{A Sample}
\author{James Emery}
\date{12/5/2008}
\begin{document}
\maketitle
\tableofcontents
\section{Preface}
This is a small \LaTeX \, file to be used in demonstrating MikTeX.

\section{Rolles' Theorem}
Let $f$ be a nice function. Suppose $f(a) = f(b) = 0$. Then there is
a number $c$, $a < c < b $ so that $f'(c)=0$.

\noindent{\bf Proof.} There must be a relative maximum or a relative minimum
of $f$ at some point $c$ between $a$ and $ b$.

\section{The Mean Value Theorem}
Let $f$ be a nice function.
There exists a point $c$, $a < c < b $ so that
\[ f'(c) = \frac{f(b) -f(a)}{b -a} . \]

\noindent{\bf Proof.} Let function $g$ be formed by subtracting the straight line
that passes through the points $(a,f(a))$ and $(b,f(b)$. Then
\[ g(x) = f(x) - [ \frac{b-x}{b-a} f(a) + \frac{a-x}{a-b} f(b) ] . \]
Then $g(a) = g(b) = 0 $, so by Rolles's Theorem there is a $c$, $a < c < b$, so
that $g'(c) = 0$. Then
\[ 0 = g'(c) = f'(c) - \frac{f(b) - f(a)}{b-a} . \]
Hence
\[ f'(c) = \frac{f(b) - f(a)}{b-a} . \]

\section{Taylor's Formula}
If we apply Rolles' theorem repeatedly, we get an extension of the mean value theorem,
which is called Taylor's formula. And from that the Taylor series expansion of a function.

\noindent {\bf Theorem} {\it Taylor's Formula} If $f$ is a nice function (has continuous derivatives), then
there is a number $c$, $ a < c < b $, so that
\[ f(b) = f(a) + f^{(1)}(a)(x-a) + f^{(2)} (a) \frac{(x-a)^2}{2!}
+ f^{(3)} (a) \frac{(x-a)^3}{3!} + ...+ \]
\[ f^{(n-1)}(a) \frac{(x-a)^{n-1}}{(n-1)!} + f^{(n)}(c) \frac{(x-a)^{n}}{n!}  \]

\noindent {\bf Proof.} Let $p$ be the polynomial
\[ p(x) = f(a) + f^{(1)}(a)(x-a) + f^{(2)}(a)\frac{(x-a)^2}{2!}+...+
 f^{(n-1)}(a) \frac{(x-a)^{n-1}}{(n-1)!}. \]
Let $M$ be defined by
\[ f(b) = p(b) + \frac{M(b-a)^n}{n!}. \]

Define
\[ g(x) = f(x) -(p(x) + \frac{M(x-a)^n}{n!}) \] .
Then $g(a) = f(a) - f(a) + 0 = 0 $, and $g(b)=0$ by the definition of $M$.
By Rolles' Theorem there is a number $b_1$, $a < b_1 < b,$ so that $g'(b_1) = 0$.
Clearly $g'(a) = 0$, so we may apply Rolles' Theorem to $g'$ and obtain a number
$b_2, a < b_2 < b_1 $, so that $g^{(2)}(b_2) = 0$. Continuing in this way, after $n$
steps, we find that
there is a $c$ so that $a< c < b$ and
```

```
\[ f^{(n)}(c) - M  = g^{(n)}(c) = 0. \]
Therefore  $M = f^{(n)}(c) $ and
\[ f(b) = p(b) + f^{(n)}(c) \frac{(b-a)^n}{n!}, \]
which is Taylor's Formula.

So the Taylor series for $\sin(x)$ is

\[ \sin(x) = \sum_{n=0}^\infty (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} +

and for $\cos(x)$ is

\[ \cos(x) = \sum_{n=0}^\infty (-1)^n \frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + ..

The complex Taylor series for $\exp(z)$ about zero is

\[ \exp(z) = \sum_{n=0}^\infty  \frac{z^n}{n!} = 1 + \frac{z}{1!} + \frac{z^2}{2!} + \frac{z^3}{3!}+ ...\]
So

\[ e^{i \theta} = \exp(i \theta) = 1 + i \frac{\theta}{1!} +  (i)^2 \frac{\theta^2}{2!} + (i)^3 \frac{\theta^3}{3!}+
 (i)^4 \frac{\theta^4}{3!}+ (i)^5 \frac{\theta^5}{5 !}+...\]

\[ = 1 + i \frac{\theta}{1!} -   \frac{\theta^2}{2!} - i \frac{\theta^3}{3!}+  \frac{\theta^4}{4!} + i \frac{\theta^5}{
-   \frac{\theta^6}{6!} - i \frac{\theta^7}{7!} + ...\]
\[ = \cos(\theta) + i \sin(\theta) . \]

Thus if $z= x+ i y$ and
\[ \arg(z) = \theta, \]
Then
\[ z  = | z | e^{i\theta} , \]
and
\[ z_1 z_2 = | z_1 | | z_2 | e^{i(\theta_1 + \theta_2)} . \]

\section{The Laplace Transform}
\[ L(f)(s) = \int_{0}^\infty f(t) e^{-st}dt ,\]
\[ \int_{0}^{\infty} \cos(kt) e^{-st} dt = \frac{s}{s^2 + k^2} ,\]
\[\int_{0}^{\infty} \sin(kt) e^{-st} dt = \frac{k}{s^2 + k^2} \]

\section{Matrix}
If ${\bf A}$ is the matrix
\[ {\bf A} =
\left[
\begin{array}{ccc}
a_{11} & a_{12} & a_{13} \\
a_{21} & a_{22} & a_{23} \\
a_{31} & a_{32} & a_{33}
\end{array}
\right] ,
\]
then its eigenvalues $\lambda$ are the roots of the equation

\[
\left|
\begin{array}{ccc}
{a_{11}- \lambda} & a_{12} & a_{13} \\
a_{21} & {a_{22}- \lambda} & a_{23} \\
a_{31} & a_{32} & {a_{33} - \lambda}
\end{array}
```

```
\right| = 0.
\]

\section{Print It Verbatim}
If we want to include material that is not to be typeset,
we may enclose it between a begin, and an end, verbatim tag.



\noindent For some

\Huge
\[\tau \epsilon \chi \]
\normalsize
is a big thing, but for others
\scriptsize
\[ \tau \epsilon \chi \]
\normalsize
is quite small.


\end{document}
```

53 The Command Line For Running Latex

There may be readers who did not grow up with Unix and DOS and don't know how to use the command line. In earlier versions of Windows there was always an MSDOS icon on the desktop, or in the start menu. However, under Windows NT, Windows 2000, and Windows XP, you might have to create a shortcut to the command line program located at

`%SystemRoot%\system32\cmd.exe`

Alternately click the "Start" button, then the "Run" button then type in "cmd.exe"

On a Macintosh OSX system one gets to the command line by going to the Utilities directory and clicking on Terminal.

Recall some commands:

```
c:  (Go to the c drive on a Windows machine.)
cd \ (Go to the root directory on a Windows machine.)
mkdir latex (Create a directory called latex. )
cd latex (Go to the latex directory.)
type joe.txt (Display a file called joe.tex on a Windows machine.)
cat joe.txt  (Display a file called joe.tex on a Unix, Linux,  or MacOSX machine)
cd ~/ (Go to your home directory on a Unix, Linux, or MacOSX machine)
dir (Display the files in a directory on a Windows machine)
ls -la (Display the files in a directory on a Unix,  Linux, or MacOSX machine)
notepad (Open the notepad editor on a Windows machine)
```

```
open -a TextEdit (Open the TextEdit editor on a MacOSX machine.)

Note: On a windows machine directories are specified with a back slash "\",
but on a Unix,  Linux, or MacOSX machine, directories are specified with a forward slash "/".

Note: The Mac program TextEdit has a word processing mode and a text mode. Be
sure to select "text" under "Format" if you want to create a text or tex file.
```

Note: There are better editors for writing programs and TeX documents then Notepad and TextEdit, on both Windows and Unix/Linux/MacOSX. Winedit is a nice editor for Windows, and has line numbers and so on. I was a shareware program and I don't know if it is still available. There are the standard Unix editors: Vi, Emacs, Pico, Nano, joe and so on. Vi and Emacs, are very powerful, but rather old fashioned and difficult for the novice.

54 A LaTeX Tutorial

LaTeX is a TeX macro that creates simplified commands. It was created by Leslie Lamport. There are many books describing the use of TeX and LaTeX, including a couple by Lamport, and a technical description of TeX by the creator of TeX, Don Knuth. There are probably hundreds of books on LaTeX and TeX. TeX is used by many, if not most of the mathematical and Scientific Journals, and probably a majority of scientific books are written in a version of TeX. Each journal has a certain preferred style, and thus has a required style file.

Here is a very nice short tutorial on using LaTeX is:

`http://www.stem2.org/je/GSWLaTeX.pdf`

55 Placing Figures in LaTeX

There are many systems and macros for placing figures in a TeX and a Latex document. I system I have used for a long time is called **psfig**. This system uses a style file called

`psfig.sty`

This is a macro written in TeX and allows postscript figures to be included in a document as figures. This file does not seem to come with MiKTeX,

but can be found on the internet. It is recommended to be placed in a subdirectory of the LaTex directory called macros. However, I did not see such a subdirectory in MiKTeX, so I placed a copy in the directory from which I am running LaTeX, and this works. To see how to use psfig, refer to my document called **figures.tex,** or **figures.pdf,** or look at one of my tex files that uses psfig. A Postscript file that is to be included needs to have a bounding box statement.

There are many other systems for including graphics figures in a LaTeX document. Bitmap graphics can be included.

56 Placing Figures and Diagrams in A Latex File

57 Installation of PCTeX

The first time PCTex is started, one must run INITex from the menu (to choose between Plain TeX, Latex, or AMSTeX). Choose Latex. When INITex is run, Version 4 will complain that the source is more than 1 year old. Ignore this. Go to settings, select Default settings, select advanced. Under method for rendering DVI, choose Postscript. Go back. Select Postscript. For color depth, select full color.

58 The PostScript Mode in PCTeX

Later versions of PCTeX have a PostScript mode (PCTeX32 V4). This is not the default mode. To change the mode to PostScript we select settings, then advanced, then PostScript. When in PostScript mode, and when the program dvips is selected for viewing, a PostScript file is created from the dvi file. This occurs when one views the dvi file, or when one creates the dvi file by choosing typeset. The PostScript file is viewed using a built in PostScript viewer. This all happens automatically when the typeset button is selected. The default PostScript viewer setting is black and white. So a figure in gray scale, or in color, will usually not look good. The setting can be changed to gray scale, or to color, by selecting Settings, Default Settings, and then PostScript.

A LaTeX file that contains embedded Postscript figures, may be processed with the macro **psfig**. For **psfig** to work in PCTeX, the PostScript file for each figure must have certain characteristics that may differ from those required in older UNIX TeX and **psfig** versions. PCTeX is less forgiving concerning the characteristics of the PostScript file than are the Unix programs that I used in the past. The first character of the PostScript file must not be a space. The file should start with

```
%!PS
```

A bounding box statement must be present in the PostScript file that contains the figures, and should look something like

```
%%BoundingBox: 0 50 450 500
```

If one finds that the figure is not centered properly, then it can be adjusted by changing the bounding box parameters. So in this example 0 50 will be the lower left corner of the PostScript figure, and 450 500 will be the upper right corner. The scale is 72 points to the inch, unless there is a scale change in the file. This all corresponds to how the PostScript file appears on the printed page before it is inserted into the document. So the lower left corner of the PostScript page is 0 0. The figure will be scaled to fit into the space allocated in the document by the psfig call, say in a 3 inch vertical space.

If one has a PostScript file with no bounding box, then a crude way of finding a bounding box would be to print the page and measure a rectangle that encloses the figure on the page, using the scale 72 points to the inch.

59 Saving A Typeset Document as PostScript

When in PostScript mode, and when viewing the typeset output, we will see a PostScript file. This screen display comes from the PCTeX built in PostScript viewer. When we choose to view the dvi file, it will be first converted to PostScript. We can choose **saveas** from the PCTeX menu and save the output as either PostScript or dvi (the dvi will always be saved). Alternately, we can select **Print Setup** and choose a PostScript printer driver and then select the **print to file** option. And then choose **Print**. This will cause the selected PostScript printer driver to be used to create the PostScript file. So the two methods will probably produce somewhat

different files. One might try both methods and compare the result. The print method will work for earlier versions of PCTeX that did not have a PostScript mode. And in fact is a method of getting PostScript output from any Windows application.

60 LaTeX Online Equation Editor

http://www.codecogs.com/components/equationeditor/equationeditor.php

61 Including a PostScript Figure

The following PostScript file **nestedrepeats.ps** is included in a figure in the following way. This Postscript algorithm draws a cubic Bezier curve defined with control points (0,0) (72,72) (72,-72) (288,0). It rotates and repeats this sixteen times. Then it does a reflection and then does the 16 rotations again. This creates a flower like PostScript graphic.

```
%!PS
%%BoundingBox: 0 50 450 500
%%EndComments
%Nested repeats 6-8, from "Learning PostScript"
306 396 translate
2{
16{
0 0 moveto
72 72 72 -72 288 0 curveto
360 16 div rotate
}repeat
-1 1 scale
}repeat
stroke
%eofill
1 setgray
0 0 10 0 360 arc fill
showpage
```

The figure is included with the macro psfig. The following code is embedded in the Latex file.

```
\begin{figure}
%\psfig{figure=/tex/nestedrepeats.ps,height=3in}
\psfig{figure=nestedrepeats.ps,height=5in}
\caption{ A PostScript Graphic.}
\end{figure}
```

The psfig macro must be included in the Latex document. This is done with the following command, which appears as the second line of the tex file.

```
\input{psfig.sty}
```

62 Including A Plot of a function Using psfig Under PCTeX

The PostScript plot file of the function was produced from the programs **plotf.cpp** and **eg2ps.c**. **plotf.cpp** is a program to plot a function and to create an output file in the eg format. **eg2ps.c** converts the eg file to a PostScript file. I might mention that there is a program called **pltax.c** that adds axes and labels to xy data to create an EG function plot. A previous version of the program **eg2ps.c** would output the BoundingBox line without a double percentage sign, and without an ending ":". The Unix version of dvips that I used on an Apollo workstation accepted this. When PCTeX did not, I realized that **eg2ps** was generating incorrect PostScript. This has been corrected so that the start of the PostScript file, is created by eg2ps, is something like:

```
%!PS
%%BoundingBox: 0 50 450 500
%%Creator: eg2ps.c by Jim Emery
%%EndComments
72 300 div 72 300 div scale
100 100 translate
3 setlinewidth
newpath
186 790 moveto
200 846 lineto
214 900 lineto
```

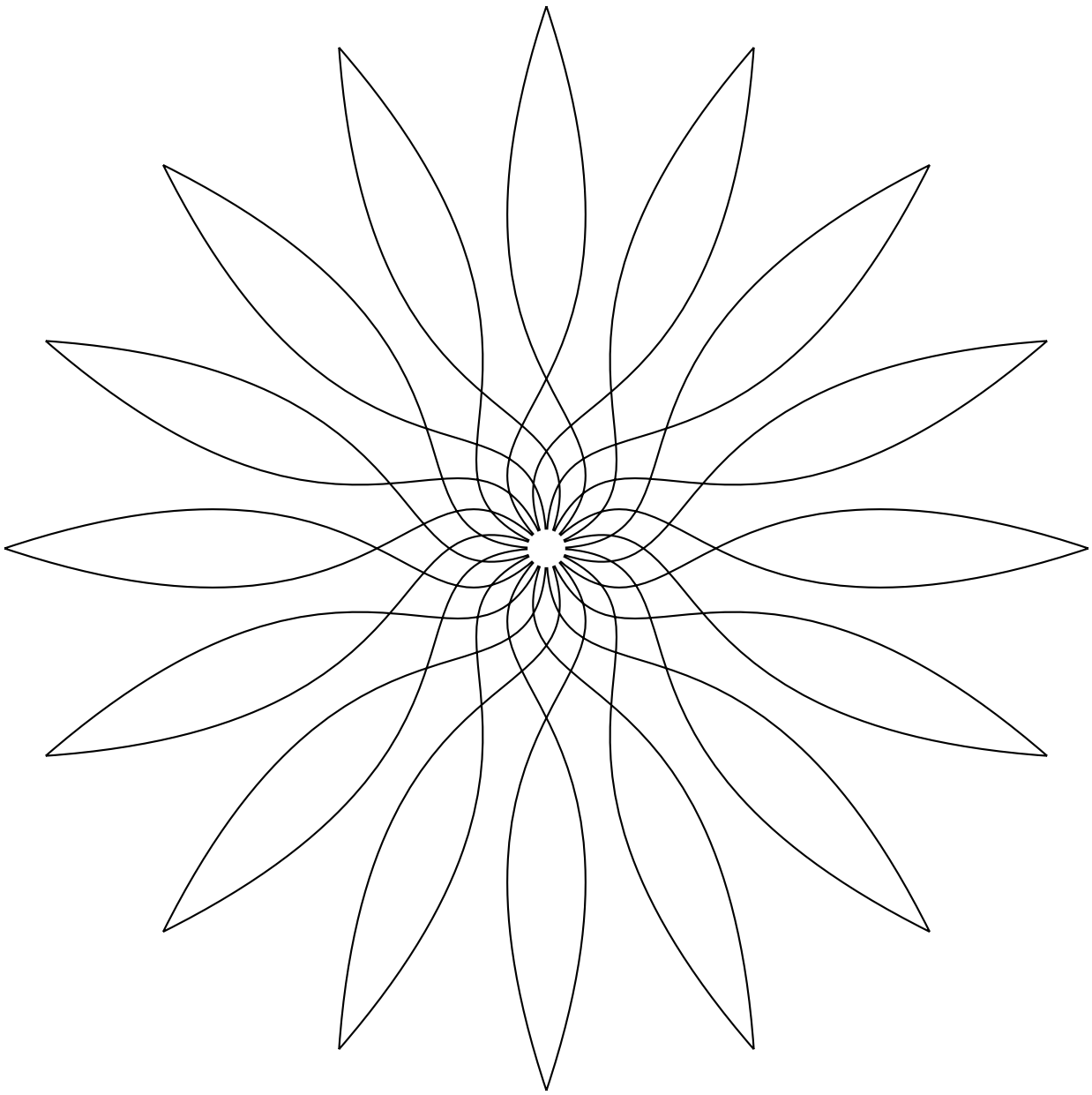


Figure 4: A PostScript Graphic.

```
228 954 lineto
241 1006 lineto
255 1057 lineto
```

NOTE. A PostScript error can occur in some systems if the (control)z left by the editor Kedit, which is an editor that I sometimes use, is at the end of the PostScript file for the figure. In particular, if an included PostScript figure contains a (cntrl)z, then when the LaTeX document, which is typeset as PostScript, is sent to the Hewlett Packard LaserJet 4mp, the printer will stop at the figure because of the (cntrl)z. In the first versions of DOS a (cntrl)z indicated end of file.

Recall that the placement of the figure is controlled by the "Bounding-Box" line of the PostScript file.

The figure is handled by a "figure" macro. This TeX macro is called psfig. A figure is placed as a floating element. It will not be placed by the program in a definite place. This can be a bit frustrating, especially if the figure is placed in the middle of some verbatim text list. There are TeX commands to control this placement. One should consult the TeX manual for information. To use psfig, the beginning of the tex file must contain a line that brings the psfig macro into the TeX document. Here are the first few lines of a file that uses psfig.

```
\documentclass{report}
\input{psfig.sty}
\title{Figures in PCTeX}
\author{Jim Emery}
\date{November 21, 2000}
\begin{document}
\maketitle
```

The figure itself is produced with the code:

```
\begin{figure}
\psfig{figure=/tex/fun1.ps,height=3in}
\caption{ A plot of a function.}
\end{figure}
```

Notice that the forward slash is used to delimit directories. This is the convention for Unix systems. The PC uses the back slash. In order to be

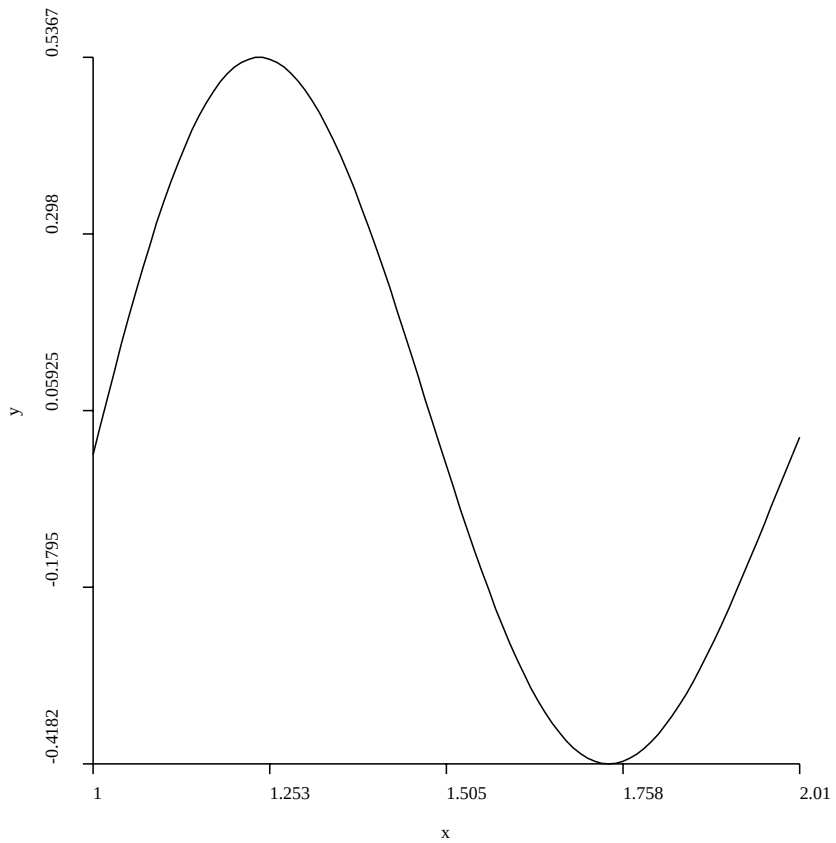


Figure 5: A plot of a function.

consistent the forward slash is used for both systems by psfig. If the file fun1.ps is in the same directory as the source file pcldst.tex, then PCTEX should find the source file without the directory specification.

63 Using Acrobat Distiller to Make a PDF

To print a file containing PostScript figures on a non postscript printer we can use Acrobat Distiller. To do this make the default windows printer some PostScript printer such as the HP Laserjet 6mp. Then select print, which first does a file open, and then gives a printer dialog box, in which the "print to file" option should be checked. The default print file name will have a .prn extension, which should be changed to .ps so that distiller can recognize it as a PostScript file. So suppose a print file called myfile.ps is generated. Then from windows double click this file, or from DOS type start myfile.ps, and Distiller should go to work producing a file called myfile.pdf. Then double click the file myfile.pdf and Acrobat Reader or Acrobat exchange will display the file, from which the file can be printed to any printer.

64 Including a Windows Bitmap Figure With PCTeX

The disadvantage to using a windows bitmap file instead of a PostScript file is usually lower resolution unless the bitmap size is set very high. A bitmap will be very large as compared to a PostScript file, which just contains a program to draw the figure. Sometimes people think that PostScript files are bigger than other files. But this is a misunderstanding of PostScript. Some graphics programs have the ability to output a bitmap file to a PostScript file. But this takes advantage of the ability of PostScript to include a bitmap inside of itself. In fact when this is done, the PostScript file is bigger than the original, because a PostScript file is a text file, and each pixel is represented as two hexidecimal characters. Here is code for a windows bitmap figure example:

```
\begin{figure}
\vskip 6in
\special{bmp:/je/tex/front24.bmp y=6in}
```

```
\caption{Front view of gem.}
\end{figure}
```

It does not need an include file. The TEX command "special" allows placing some content external to TEX in a dvi document. Actually the psfig macro works by using the TeX "special" command. The bmp figures, depending on how they are saved and compressed in Paint, can be rendered as black and white. To make Acrobat convert a bmp to shaded colors, we should save the bmp file as 24 bit color.

65 Another Method of Placement: An Optics Figure

This figure is from the document **optics.tex**. It is placed as described in the PCTeX manual, were the description is very inadequate. One might have much trouble in making **psfig** work with the description given in the PCTeX manual. Notice that there is no begin figure and no end figure. Also there is no caption. Sometimes when using this method there is not enough space to place the figure. I do not recommend this method.

The code for this figure is:

```
\par
\psfig{figure=/je/tex/opticsf1.ps,height=3in}
\par
```

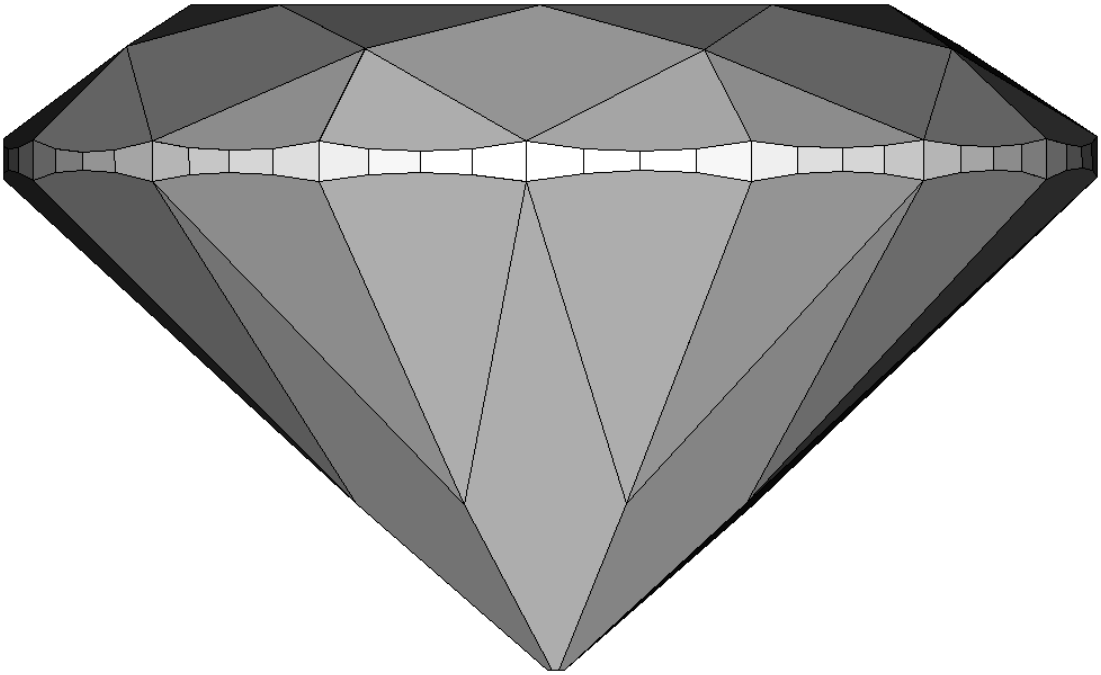
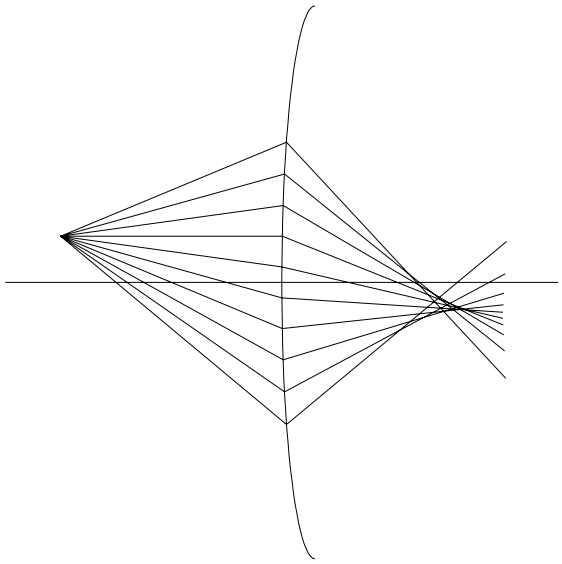


Figure 6: Front view of gem.



66 Making an Acrobat PDF

To make a PDF we need a program that converts PostScript to Acrobat PDF. One such program from Adobe is called Distiller. It comes with Acrobat 4.0. First we must save the typset document in PostScript. We can do this in two ways. In the first way, while we are viewing the dvi file in PostScript mode, we can do a saveas, and choose the PostScript file format.

For earlier versions of PCTeX that did not have a PostScript mode, we can do the following. We do a print to file in PC tex. We select a PostScript printer such as the HP 6MP/PS. We save the print file as p.ps. We do the command "start p.ps", which will call Acrobat Distiller, which will convert to PDF. A PDF file has a built in compression, so it will compress a large bitmap into a much smaller file.

67 A Figure Showing Planck's Black Body Radiation Curves

See the report **The Sea Ice Project**, file name: **nic.tex**. Here is the code for this included PostScript figure:

```
\begin{figure}
\psfig{figure=/je//tex/planck.ps,height=5in}
\caption{ Blackbody radiation curves, energy density per unit wavelength.}
\end{figure}
```

68 Using psfig Under Linux and Unix

The use of psfig under Linux and Unix is quite similar to its use under PCTeX. The required include line may be slightly different.

Under some versions of LaTeX on a Unix system, one does not include the sty file extension in the input psfig line. Some Unix systems may still require no file extension. The TeX "input" command and the "include" command both insert text into the current TeX document. The "include" command apparently will not recognize the sty file extension. So that when LaTeX reads

```
\include{psfig.sty}
```

it searches for the file psfig.sty.tex, which it will not find. That is, "include" always tags on the tex extension. This behavior may have something to do with the new version of LaTeX.

Here is an example of the psfig input line from the document called **plcdst.tex**. Note that under the new version of LaTeX one replaces documentstyle with documentclass.

```
\documentstyle[12pt]{report}
\input{psfig.sty}
\title{Calculating Distance Between Plane Curves}
\author{Jim Emery}
\date{Revised April 28, 1992}
\begin{document}
\maketitle
\tableofcontents
\section{Introduction}
.......................
.......................
\begin{figure}
```

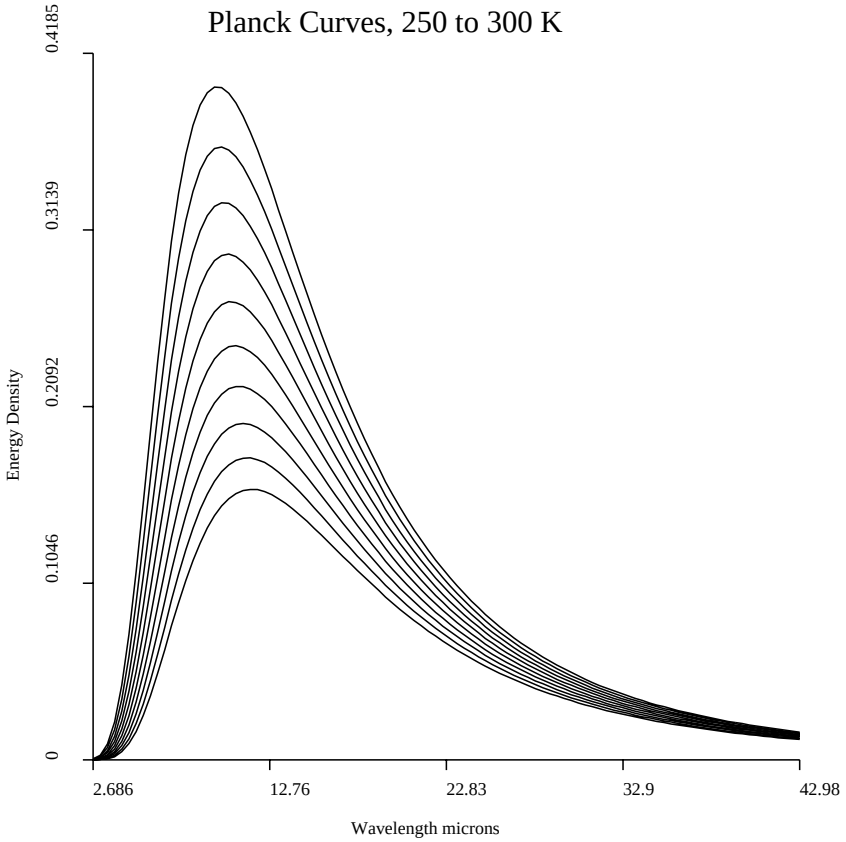


Figure 7: Blackbody radiation curves, energy density per unit wavelength.

```
%vspace{3.0in}
\psfig{figure=plcdstf2.ps,height=3.in}
\caption{The extreme point lies on a segment bisector.}
\end{figure}
.......................
.......................
\end{document}
```

Recall that in the PCTeX case the include line could have been

```
\input{psfig.sty}
```

and so the psfig line could have specified the directory using the Unix forward slash delimiter.

```
\psfig{figure=/je/tex/plcdstf2.ps,height=3.in}
```

The Linux version of psfig will use the psfig.sty form also, but some older Unix versions may not, and may only accept the file name without the sty extension.

The user of TeX, as well as the user of most software, must be an empiricist. He must experiment, he must probe the black box to divine its behavior. That is why they call it Computer Science rather than Computer Logic. The best way to teach students about gravity is to throw them from a cliff. The best way to find high caliber students is to take them to the circus and fire them from a cannon.

69 Generating A PostScript File in Linux and Unix

To generate a PostScript output file from the dvi file generated by Latex, we can use the program **dvips**. Under Linux we type

```
dvips -f myfile.dvi > myfile.ps
```

70 Viewers

Document viewing must be done in X-Windows on a Linux system., First we must start X-Windows with the command: **startx**. A dvi viewer comes on the Linux system that is called **xdvi**. To run it we should type:

```
xdvi myfile.dvi
```

We shall see no figures using **xdvi**, but only blank space where each figure should go. To view the document with figures, we must use a PostScript viewer such as **GhostScript**. To run **GhostScript** on a Linux system, from the command line, we type: **ghostview**. Alternately, we may choose the run menu in the Gnome graphics manager. We type ghostview in the run box. Or we may double click the Ghostscript icon, if we can find it. The PhotoShop clone, which is called Gimp, and which comes with Linux, will also view a PostScript file.

Another method of viewing is to convert the PostScript file to PDF using Acrobat distiller. Then the PDF file may be viewed with Acrobat reader.

A third method is to view the file with a graphics program that supports PostScript. The graphics program CorelDraw8 will display a PostScript file. The PCTeX program itself has a built in PostScript viewer, so files with the ps extension can be opened and viewed in PCTeX. For that to work with a single figure, the PostScript file may need a bounding box declaration. Recall that in a PostScript file there are 72 points to the inch, and the bounding box, which is defined by the lower left corner coordinates and the upper right corner coordinates, should bound the PostScript graphic. PostScript uses a conventional coordinate system with the origin at the lower left corner of the page. PostScript always has a current transformation matrix, and it is applied to all coordinates. This transformation is a combined scaling, rotating, and translating. Initally it will be the identity transformation. But, for example, if the scale factors have been changed, then the 72 points to the inch may no longer apply.

The default PostScript viewing setting is black and white. This can be changed to gray scale or to color, by using the default setting menu.

71 A Figure Created in a Drawing Program

This geodetic coordinates figure was created in CorelDraw. It was exported from CorelDraw as an encapsulated PostScript file (eps). It was saved with the name **ellgen.ps**. When saved from CorelDraw as a PostScript print file, there is some binary data placed at the beginning of the file, and at the end of the file. Although some PostScript viewers, including Ghostscript on Linux, displayed the geodetic figure correctly, PCTeX had a problem with the file. But when the file is exported as an eps file, the exported eps file worked fine.

CorelDraw creates elaborate PostScript dictionaries, so the output PostScript is quite unreadable. Early versions of Adobe Illustrator made more readable PostScript output files. The button "outputtextascurves" was checked in this eps export. This is probably not necessary, but if one does this no external fonts will be needed by the printer or viewer. This PostScript file **ellgen.ps**, which should be about 2k in length, became about 38k in size. This is because of all of the excess general garbage that CorelDraw adds to an eps file.

72 A Figure Created Using EG Graphics

The following illustration of an electromechanical device was created using the EG graphics commands. It was created by running the programs winedit, arcs.cpp, lines.cpp, and wineg.cpp, simultaneously. This constitutes a poor man's CAD system. I used to use the program AutoSketch, save as DXF, and use a program I wrote to convert from DXF to EG. But AutoSketch is a pretty weak program. I suppose I should write my own CAD system. Commercial systems never seem to do exactly what one wants. The file is called **magdevf1.eg**. It was then converted to PostScript using **eg2ps**. This produced the file **magdevf1.ps**. It is quite laborious to create a graphic by manually assembling EG (Emery Graphics) commands. But it is possible. Here is the file **magdevf1.eg**:

```
v-1 1 -1 1
w0 8 3 11
a 4.0614067 7.3728812 1.604951 -3.97 .65 100
a 4.0614067 7.3728812 1.604951  1.118 1.54 100

%a 4.0572182 7.373701 2.0830612 -3.1415926 3.1415926 100
a 4.0572182 7.373701 2.0830612 -3.66 .49 100
a 4.0572182 7.373701 2.0830612 .645 .91 100
```

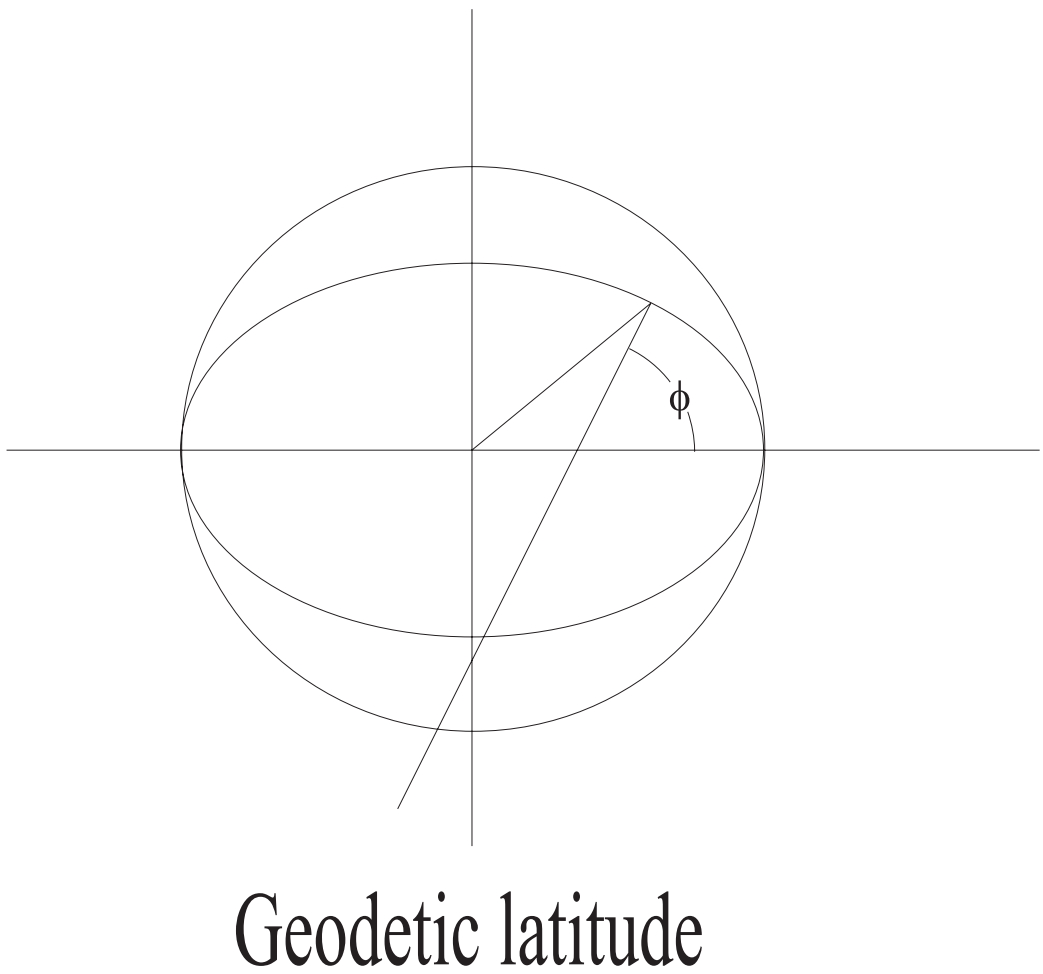


Figure 8: The Geodetic latitude ϕ differs from the spherical coordinate latitude as the sphere is deformed to the prolate ellipsoid.

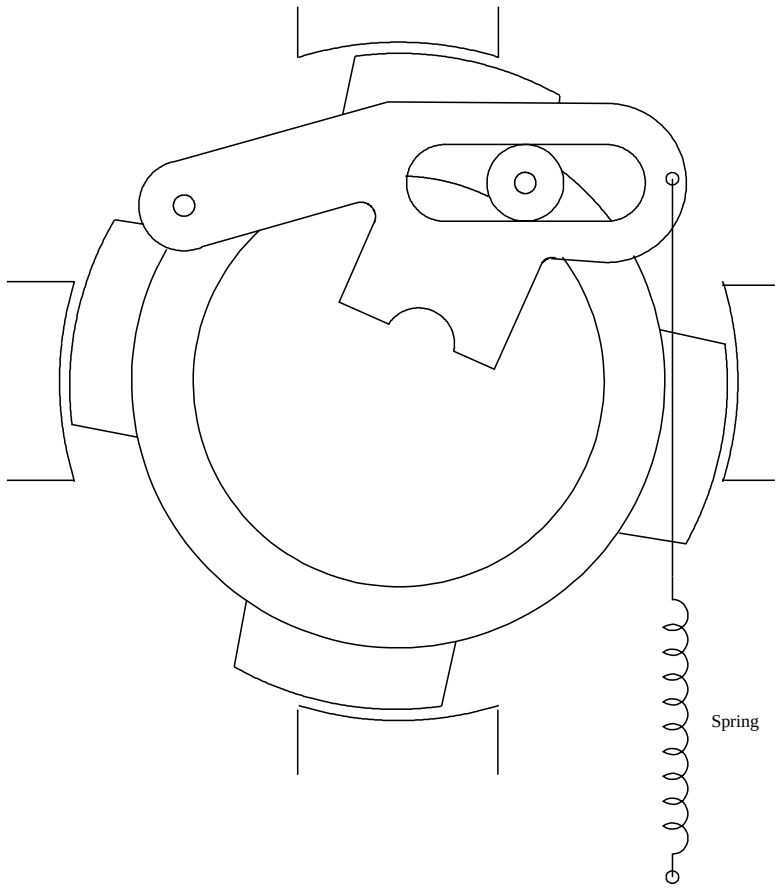


Figure 9: An Electromechanical Device.

```
a 2.38432 8.74626 0.35345971 1.8058527 5.1559178 100
m 2.302 9.09
d 3.975 9.557
m 2.536 8.427
d 3.731 8.767
a 2.38432 8.74626 0.083000015 -3.14159 3.14159 100
a 3.7660643 8.6591609 0.11339655 -0.52831593 1.885167 100
m 3.864 8.602
d 3.598 7.991
d 3.983 7.821
a 4.2167804 7.6665492 0.28019335 2.5577556 -0.20686316 100
m 4.491 7.609
d 4.806 7.466
d 5.179 8.295
%filet
a 5.2595341 8.2440431 0.09530133 1.2343097 2.5774492 100
m5.251 8.334
d             5.7           8.302

%slot
a 5.6876776 8.923197 0.62131922 -1.5509624 1.5300292 100
a 5.6876776 8.923197 0.3 -1.5707 1.5707 100
a 4.422 8.923197 0.3 1.5707 4.7123889 100
m 4.422 8.623197
d 5.6876776 8.623197
m 4.422 9.223197
d 5.6876776 9.223197


a 5.05 8.923197 0.3 -3.1415926 3.145926 100
a 5.05 8.923197 0.083 -3.1415926 3.145926 100

%tab1
m 2.87315.8 5.6615.8
d 2.77515.8 5.14415.8
m 4.50815.8 5.3415.8
d 4.39715.8 4.83615.8
a 4.0392853 7.3771063 2.5661607 -2.0859583 -1.430944 100
m           5.713           9.544
d 3.975 9.557

%tab2
m 5.786 6.188
d           6.303           6.102
a 4.0577535 7.3542586 2.5708527 -0.50876444 0.12038355 100

m            6.61           7.663
d 6.106 7.774


%tab3
a 4.061476 7.3727854 2.5629308 1.0579509 1.7040366 100
m           5.319           9.606
d 5.316 9.547
m           3.721           9.913
d 3.623 9.46
```

```
%tab4

m           1.841           8.634
d           2.073 8.6
a 4.0547086 7.3405925 2.563866 2.6128183 3.2606759 100

m           1.509           7.036
d 2.025 6.938

%stator 1
m       6.5863503        6.6
d       7        6.6
%a 4.0547086 7.3405925 2.65 -.3 .3 100
a 4.0614067 7.3728812 2.65 -.3 .3 100
m       6.5863503        8.123721
d       7        8.123721
%stator2
m        4.838081       9.9
d        4.838081       10.3
%a 4.0547086 7.3405925 2.65 1.2707 1.87079  100
a 4.0614067 7.3728812 2.65 1.2707 1.87079  100
m       3.2715961       9.9
d       3.2715961       10.3
%stator3
m        1.523069       8.15
d        1.       8.15
%a 4.0547086 7.3405925 2.65 2.84159 3.44159 100
a 4.0614067 7.3728812 2.65 2.84159 3.44159 100
m       1.5230648       6.6
d       1.       6.6
%stator4
m       3.2715961       4.8089458
d       3.2715961       4.3
%a 4.0547086 7.3405925 2.65 -1.87079 -1.27079 100
a 4.0614067 7.3728812 2.65 -1.87079 -1.27079 100
m       4.8378532       4.8089558
d       4.8378532       4.3


%spring



z .03
a  6.2000   8.956 .05 -3.14159 3.14159 20
a  6.2000   3.5   .05 -3.14159 3.14159 20
m  6.2000       5.8490
d  6.2000       5.6687
a  6.2000       5.5475       .12124       1.5708      -2.2143     20
a  6.2000       5.3535       .12124       2.2143      -2.2143     20
a  6.2000       5.1595       .12124       2.2143      -2.2143     20
a  6.2000       4.9655       .12124       2.2143      -2.2143     20
a  6.2000       4.7715       .12124       2.2143      -2.2143     20
a  6.2000       4.5775       .12124       2.2143      -2.2143     20
a  6.2000       4.3835       .12124       2.2143      -2.2143     20
a  6.2000       4.1895       .12124       2.2143      -2.2143     20
a  6.2000       3.9955       .12124       2.2143      -2.2143     20
```

```
a  6.2000      3.8015      .12124      2.2143     -1.5708     20
d  6.2000      3.5000
t  6.5031      4.6745      6 Spring
m  6.2000      8.9560
d  6.2000      5.8490
```

73 EG Graphics in Programming.

The best use of EG graphics is to store graphical objects that are created in a programming language. For example, we have several programs to generate surfaces as sets of polygons. The program stores the polygons in EG format. One such program is called **dph.c**. Many such graphics programs can be written. One can create a program to draw an electrical circuit, or to draw other special purpose diagrams. One such program is called **elecschm.ftn**. Creating a general mathematical surface in a commercial drawing program such as Adobe Illustrator is nearly impossible. However, some of the mathematical programs such as Maple, Mathematica, and Matlab can create multidimensional function plots and can export them as PostScript files.

As an example of an EG file written by a program, we present a figure consisting of a triangulated spherical surface. This was created with the program **decomp.ftn**. The function of the program is to find a good way to distribute points uniformly on a sphere. As is well known it is impossible to position n points on a sphere exactly uniformly, except for special cases, namely the vertex positions of the Platonic solids. A report on this is called **subdpoly.tex** The actual drawing is done with one of the programs with a name like **drwt.ftn**, which draws triangles to an EG file.

74 A CorelDraw Figure

This cross section figure was created in CorelDraw and exported as an eps file. The eps extension was changed to ps.

```
\begin{figure}
\psfig{figure=magdevf2.ps,height=5in}
```

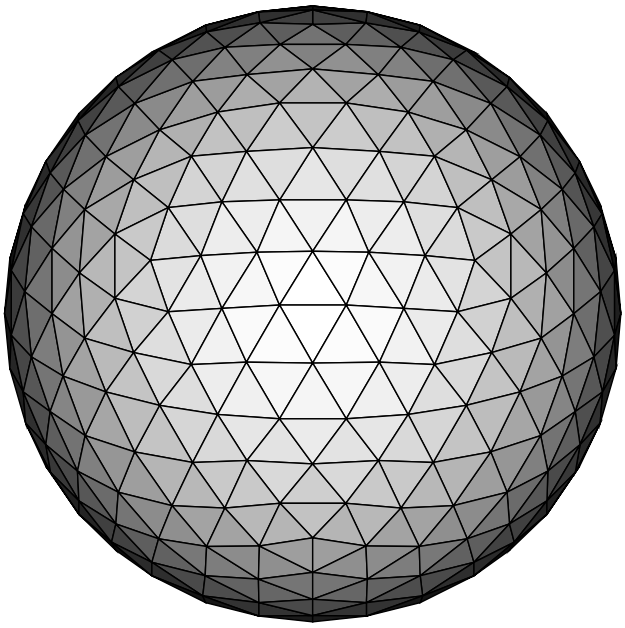


Figure 10: Nearly Uniformly Spaced Points On a Sphere.

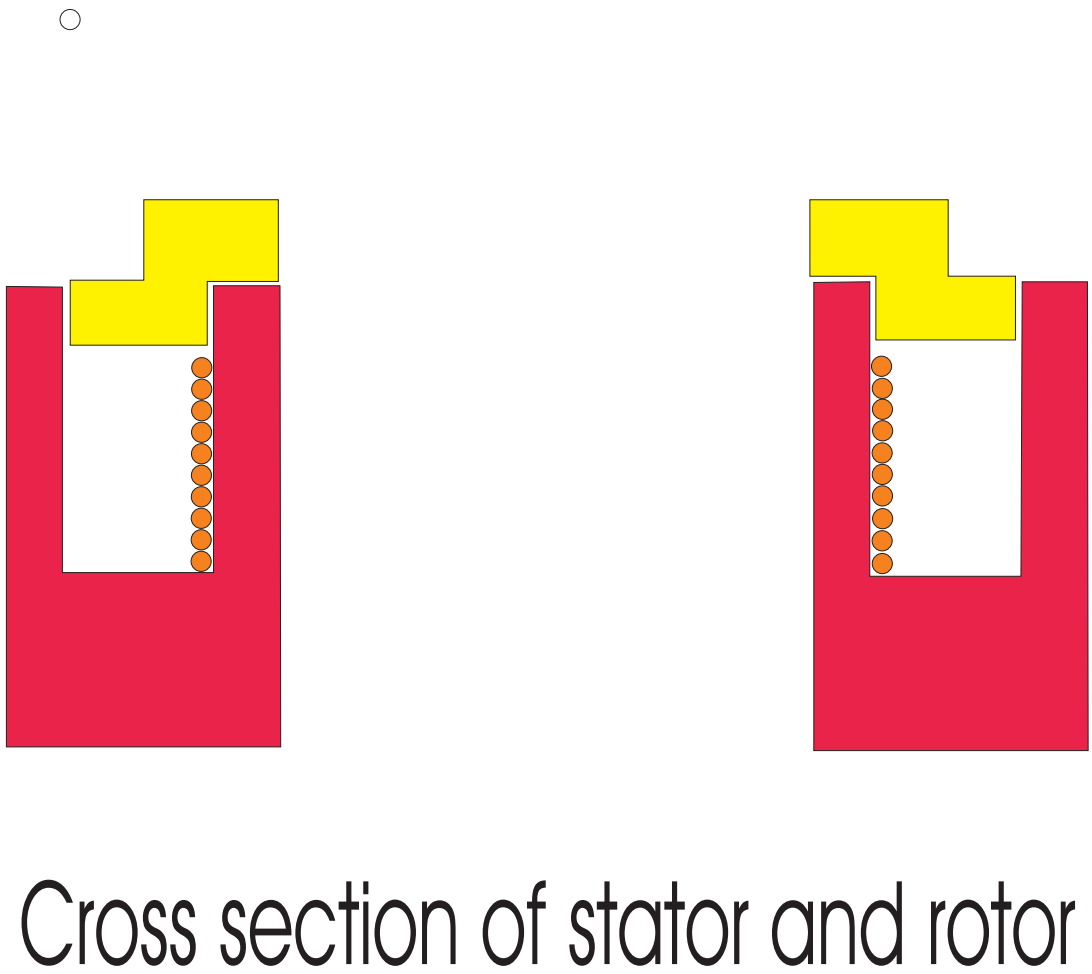


Figure 11: Cross section of rotor and stator showing windings.

```
\caption{ Cross section of rotor and stator showing windings.}
\end{figure}
```

75 A Scanned Figure

The figure titled "Derivation of spring torque" was scanned as a tiff file at a low resolution (150 pixels per inch). Then it was saved as an EPS file in a graphics program. First I tried to do this with PaintShopPro. But PCTeX did not like the EPS file from PaintShopPro. The reason is, I think, that the bitmap in PostScript is stored as a giant string of hexidecimal pairs. This creates a line of huge length. I speculate that PCTex can not handle such a long line. According to the PostScript documentation, newline characters can be embedded in this giant bitmap string without affecting it. The newlines will be ignored. So the string can be broken into normal size lines. When PhotoShop saves a bitmap as an EPS file, it does this line breaking. I imported the tiff file in CorelDraw, but the resolution was poor, so I moved on. Then I tried Adobe PhotoShop. As I saved the file as an EPS file, I set the bitmap header to "none." After I saved the file as EPS, I changed the file extension to ps. This PostScript file, which I saved from PhotoShop, worked fine in PCTeX.

The bad news is that saving the tiff file in EPS format doubles the size of the file. The good news is that it can be imported with **psfig** as a PostScript file. The advantage of this is that both PCTeX and Linux can handle the figure in the same way. We could have imported the scanned file as a bmp file in PCTeX, but then our Latex file would have not worked in Unix and Linux. Actually Linux dvips will still work, it will simply ignore the bitmap figure. Linux version of **dvips** would have ignored the figure because it only handles PostScript figures. Art historians will think that the figure could use a little cleaning and restoring, and I agree. However, one must be careful. Sometimes it is best to leave antiques alone.

```
\begin{figure}
\psfig{figure=magdevf3.ps,height=5in}
\caption{ Derivation of spring torque.}
\end{figure}
```

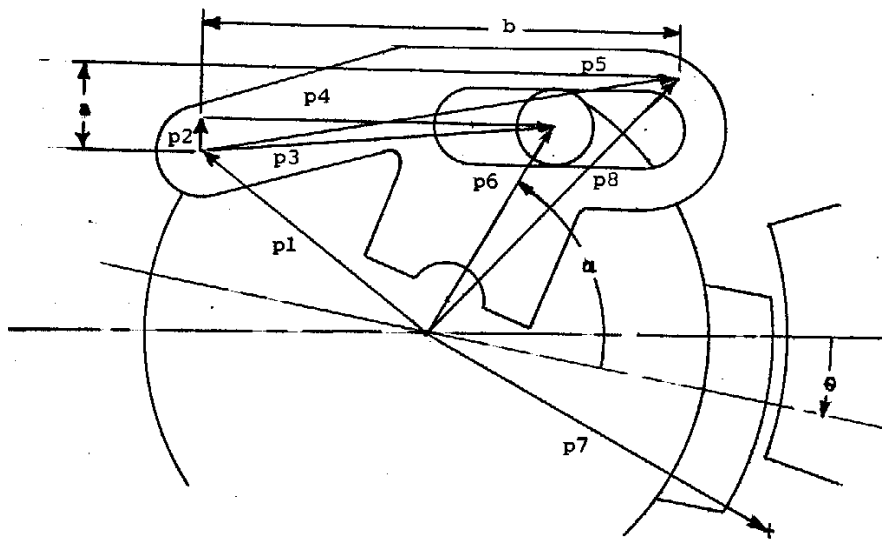


Fig. 2

Figure 12: Derivation of spring torque.

76 A Projective Geometry Figure

A projective figure showing the representation of a circle as a cone of lines was created with a Fortran program called **conef.ftn**. This program generates a set of polygons and curves. The curves are sets of line segments in 3-space. Program **drwpgnl.ftn** converts the polyhedron and the lines to EG graphics using a perspective projection. The EG file is converted to Postscript with **eg2ps.c**. The final file is called **quadricf1.ps**, being a figure in the document called **Conics, Quadrics, and Projective Space**. The source file for the document is **quadric.tex**. The plane with the hole consists of an outer curve and an inner curve. The only real trick is to make sure that the inner and outer curves have opposite orientations, so that the PostScript fill command will determine inside and outside correctly. PostScript uses a winding number calculation for this. **drwpgnl.ftn** only adds PostScript fill commands for polygons, so a setgray, a stroke, and a fill command were added to the EG file by hand. Line segments read by **drwpgnl.ftn** do not necessarily form closed curves and are not necessarily to be filled, so projected curves are not followed by a PostScript **fill** command.

77 Pasting a Figure Created in DesignCAD into MathCAD

DesignCAD Express Version 12, can read AutoCAD files, and can import and export IGES, which later versions of AutoCAD cannot. To past a drawing from DesignCAD to MathCAD, select the drawing with a bounding box using the mouse. Select copy, this puts it in the clipboard. Click somewhere on the MathCAD sheet, and select paste. Also DesignCAD can export a drawing as a windows meta file wmf. This can be put into a microsoft word document then it can be selected and put in the clipboard and pasted to MathCAD. Also a bitmap file may be placed in MathCAD by selecting picture, and somehow specifying the filename? According to the manual anyway. MathCAD documents may be transferred to microsoft word via the clipboard.

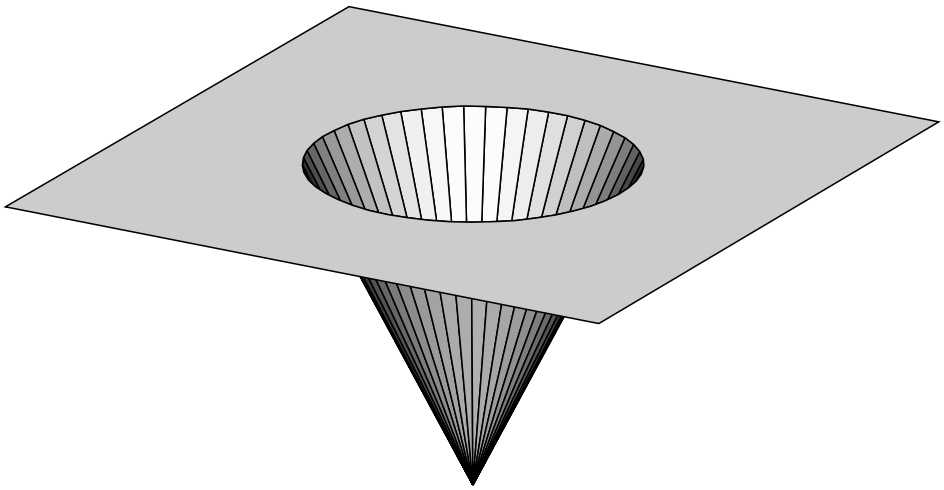


Figure 13: Points of projective 2-space are lines in 3-space. A 3-d linear transformation is a 2-d projective transformation. A rotation of the cone can project the circle to an ellipse, a parabola, or a hyperbola. In the same way the lines through any figure in the plane are a projective space representation of the figure. Any sequence of perspective projections of a figure onto a set of planes can be represented as a sequence of linear transformations in 3-space. The plane shown in the figure is known as the affine plane.

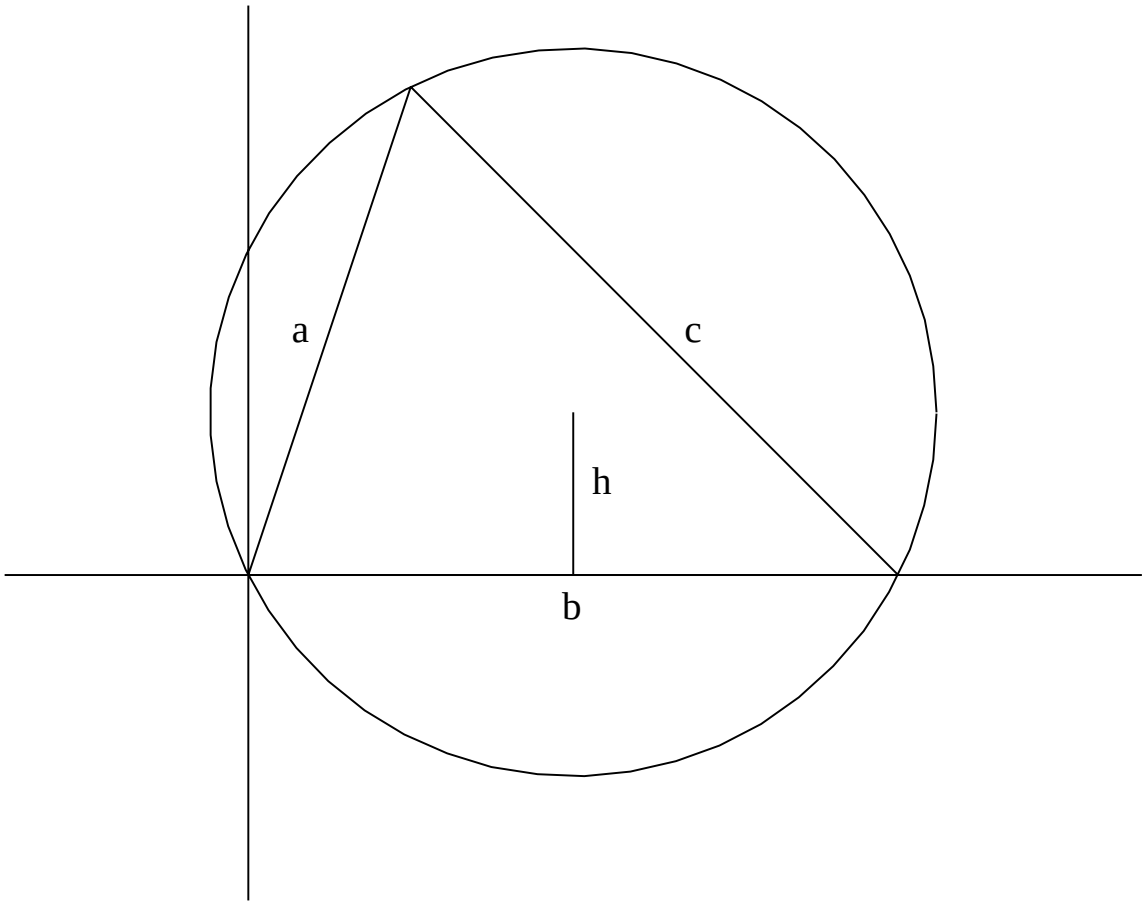


Figure 14: A circle circumscribing A triangle. This is a postscipt figure with text created with **addpstxt.bat**.

78 Creating a Postscript Figure Using AutoCad or DesignCad

To create a postscript figure, follow these steps.

(1) Create the figure in AutoCad, or DesignCad. If created in AutoCad, save as an AutoCad file and read it into DesignCad. Export it as an IGES file.

(2) Use program **iges2eg** to convert the iges file to an eg file.

(3) Use program **pltmerge** to add window and viewport commands to the eg file.

(4) Use program **eg2ps** to convert the eg file to postscript file say p.ps.

(5) Start the postscript file as in s p.ps calling ghostview. Move the curser to a location where one wishes to add letter labels, annotation, et cetera. Note the cursor position. Write down the coordinates in a notebook. Continue for additional text.

(6) Create a text file called tmptxt.txt containing the coordinates of the text from the notebook as in

```
35 245 A
156 346 Note
........
```

(7) Rename the postscript file to **tmp.ps**. Above the last stroke command add

```
:include tmptxt.ps
```

(8) Run the bat file **addpstxt.bat** specifying the output file as say **myfigure.ps**

(9) Load myfigure.ps in Ghostview, and also in a text editor like notepad.

(9) If the symbol font is used, then using notepad, uncomment the symbol font command in the file **myfigure.ps**, and locate it above the first symbol text.

(10) If the text is still not located properly, then find where the text is to be moved using ghostview, and adjust tmp.txt, which also should be in a notepad window for convenience. Rerun addpstxt, click in the Ghostview window again, and the new figure, myfigure.ps will appear. Repeat until the figure is satisfactory.

See the next section for more details about the programs that are being used in this process.

79 Adding Annotation to A Postscript File

The batch file that adds text to a postscript file is called **addpstxt.bat**. It contains the following commands:

```
@echo off
rem addpstxt.bat, Version 2/21/03, Jim Emery
rem adds postscript code to a postscript file for text
if "%1"==""" goto help
if "%2"==""" goto nofontinfo
pstext tmp.txt tmptxt.ps %2
includef tmp.ps %1
goto end
:nofontinfo
pstext tmp.txt tmptxt.ps
includef tmp.ps %1
goto end
:help
echo addpstxt.bat, Version 2/21/2003 Adds postscript code for text annotation.
echo The text is contained in a file called tmp.txt, type pstext for the data
echo structure. The postscript file, to which annotation is to be added, is
echo called tmp.ps, which must exist. File tmp.txt must also exist.
echo The first parameter is the file name of the new postscript file.
echo The second parameter is the fontsize in points, say 12 pts.
echo If the second parameter is not present, the font definition is assumed
echo to be already in the initial file. Also in order for the new postcript
echo code to be added, there must be a :include tmptxt.ps statement in tmp.ps
echo If a second text file is to be added rename the first tmptxt.ps
echo to say tmptxt1.ps and add a :include tmptxt1.ps in tmp.ps
echo Usage: addpstxt newpostscriptfile fontsize
:end
```

The batch file assumes that there is a figure source file called tmp.ps. This represents a figure to which annotation is to be added. This file would typically be created as an eg file and then converted to postscript with eg2ps. The file tmp.txt contains the definition of the text to be added. Each line of tmp.txt would have the postscript x and y coordinates of the text in pts, followed by the text itself. The coordinates would have been found using the curser position displayed when the postscript file is viewed in Ghostscript. The program pstext.c is called to read tmp.txt and to produce a postscript file, which is called tmptxt.ps and which has the new postscript code for the text annotation. In the file tmp.ps one must put a line

```
:include tmptxt.ps
```

so that the new postscript code is added at that point.

The program includef.c is called in the batch file to merge the new code into the original postscript file tmp.ps. An example of a tmp.txt file is

```
160 230 a
260 128 b
305 230 c
271 174 h
```

A finished file displayed here is a figure for a circle circumscribing a triangle:

```
%!PS
%%BoundingBox: 47 17 476 371
%%Creator: eg2ps.c by Jim Emery
%%EndComments
72 300 div 72 300 div scale
100 100 translate
3 setlinewidth
newpath
125 500 moveto
1875 500 lineto
500 -374 moveto
500 0 moveto
500 1375 lineto
stroke
750 1250 moveto
500 500 lineto
1500 500 lineto
750 1250 lineto
stroke
1559 750 moveto
1554 821 lineto
1541 892 lineto
1518 960 lineto
1487 1024 lineto
1448 1084 lineto
1402 1139 lineto
1349 1187 lineto
1290 1228 lineto
1227 1261 lineto
1159 1286 lineto
1090 1302 lineto
1018 1309 lineto
947 1306 lineto
876 1295 lineto
807 1275 lineto
742 1246 lineto
681 1209 lineto
625 1164 lineto
575 1113 lineto
532 1056 lineto
497 993 lineto
470 927 lineto
451 858 lineto
442 787 lineto
442 715 lineto
451 644 lineto
```

```
469 575 lineto
496 508 lineto
531 446 lineto
574 388 lineto
623 337 lineto
679 292 lineto
740 255 lineto
806 226 lineto
874 205 lineto
945 194 lineto
1017 191 lineto
1088 198 lineto
1158 214 lineto
1225 238 lineto
1289 271 lineto
1347 312 lineto
1400 360 lineto
1447 414 lineto
1486 474 lineto
1518 539 lineto
1540 607 lineto
1554 677 lineto
1559 748 lineto
stroke
1000 500 moveto
1000 750 lineto
/Times-Roman findfont
%/Symbol findfont
60 scalefont setfont
 567 858 moveto
 567 858 translate
(a) show
 -567 -858 translate
 983 433 moveto
 983 433 translate
(b) show
 -983 -433 translate
 1171 858 moveto
 1171 858 translate
(c) show
 -1171 -858 translate
 1029 625 moveto
 1029 625 translate
(h) show
 -1029 -625 translate
stroke
showpage
```

The line

```
:include tmptxt.ps
```

in the file tmp.ps becomes

```
/Times-Roman findfont
```

```
%/Symbol findfont
60 scalefont setfont
 567 858 moveto
 567 858 translate
(a) show
 -567 -858 translate
 983 433 moveto
 983 433 translate
(b) show
 -983 -433 translate
 1171 858 moveto
 1171 858 translate
(c) show
 -1171 -858 translate
 1029 625 moveto
 1029 625 translate
(h) show
 -1029 -625 translate
```

in the new figure file. The figure generated by this file is captioned **A Circle Circumscribing A Triangle.**

80 Creating Figures With AutoCad and DesignCad

A figure can be created in AutoCad and saved in some format that can be exported. A good format for saving is the IGES format. However, later versions of AutoCad no longer support the IGESOut command. DesignCad can read the AutoCad file and then export an IGES file. One then may use my program **iges2eg** to convert to an eg file. Then one may convert this file to PostScript using my program eg2ps. Annotation may be added as shown in the previous section.

81 Importing A Figure That Was Generated With A Programming Language Into DesigCAD or AutoCAD

Program plthp creates an hpgl (Hewlett-Packard Pen Plotter Language) file from an eg file. DesignCad will import an hpgl file. Perhaps AutoCAD will also.

82 Least Squares Approximation

82.1 Elementary Formulation

The traditional way of deriving least squares equations is to write the expression for the sum of the squares difference between the given "data" and the approximating function, and then to set the partial derivatives with respect to the coefficients of the approximating function to zero. Let us do this for the case of fitting a straight line to given data. Assume the model $f(x) = ax + b$ and minimize

$$r(a, b) = \sum_{i=1}^{n} (ax_i + b - y_i)^2$$

The conditions for a minimum are

$$\frac{\partial r}{\partial a} = \sum_{i=1}^{n} 2x_i(ax_i + b - y_i) = 0$$

$$\frac{\partial r}{\partial b} = \sum_{i=1}^{n} 2(ax_i + b - y_i) = 0$$

We get a two by two system of equations.

$$a \sum_{i=1}^{n} x_i^2 + b \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i$$

$$a \sum_{i=1}^{n} x_i + b \sum_{i=1}^{n} 1 = \sum_{i=1}^{n} y_i$$

These equations are known as the normal equations of the problem. They have a unique solution if the determinant is not zero, that is if

$$n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2) \neq 0.$$

If the x values are not all equal this follows from the Cauchy-Schwartz inequality applied to the vectors $(1, 1, ...1)$ and $(x_1, x_2, ..., x_n)$. The general problem can be viewed more naturally as being geometric.

82.2 A Geometric View of the Least Squares Problem

The abstract linear least squares problem may be formulated as approximation in a vector space by some element of a subspace. Often this vector space is a space of functions. As examples the subspace could be generated by a bases such as

$$1, x, x^2, x^3, ...,$$

or such as

$$1, \cos(\omega t), \sin(\omega t), \cos(2\omega t), \sin(\omega t), ...$$

The first case would be a polynomial, or power series approximation. And the second would be a Fourier or trigonometric approximation. So consider a vector space V with an inner product of u, with v, written as (u, v). Given a subspace S and an arbitrary element g of V, we are to find the element in S that best approximates g in the norm corresponding to the inner product. The L^2 norm for functions is based on the inner product

$$(f, g) = \int fg,$$

and for sequences is based on the inner product

$$(f, g) = \sum_{i=1}^{n} f_i g_i.$$

This L^2 norm corresponds directly to the "squares" part of the least squares approximation. But the theory carries through for an arbitrary inner product. The norm defined by an inner product is

$$\|f\| = (f, f)^{1/2}.$$

A solution $f \in S$, minimizes

$$(f - g, f - g) = \|f - g\|^2.$$

We will show that the problem is solved as the orthogonal projection of a vector into a subspace. One can think of this as analogous to the simple geometric problem of projecting a vector in space onto a plane. Think of a vector from the origin to a point, and think of a plane through the origin, not containing this vector. The plane is a vector space. A vector in the plane closest to the original vector is obviously the orthogonal projection of

the vector onto the plane. The same thing happens in the general problem, where the plane becomes the subspace. For example the subspace might be the set of all cubic polynomials. And the problem is to best fit the data to a cubic polynomial.

Two vectors are orthogonal, i.e. perpendicular, if their inner product is zero. We require a preliminary theorem to prove the main proposition.
Pythagorean Theorem. If v_1 is orthogonal to v_2, then

$$\|v_1 + v_2\|^2 = \|v_1\|^2 + \|v_2\|^2.$$

Proof.

$$(v_1 + v_2, v_1 + v_2) = (v_1, v_1) + 2(v_1, v_2) + v(_2, v_2) = (v_1, v_1) + (v_2, v_2).$$

Proposition. If $f \in S$ and $(g - f, h) = 0, \forall h \in S$ then f is a solution to the least squares problem.
Proof. Let $s \in S$. We have

$$\|g - s\|^2 = \|(g - f) + (f - s)\|^2 = \|g - f\|^2 + \|f - s\|^2 \geq \|g - f\|^2.$$

By assumption, $g - f$ is orthogonal to the subspace S, and $f - s$ is in S. So the second equality is a consequence of the Pythagorean Theorem.
We have shown that

$$\|g - s\| \geq \|g - f\|, \forall s \in S.$$

so f is the best approximation to g in S and this completes the proof.

Notice that a unique solution always exists because f is the unique orthogonal projection of g into S. For finite subspaces the solution can be formulated as a solution to a set of n linear equations in n unknowns. Let S equal the span of $f_1, .., f_n$. Let the solution be

$$f = c_1 f_1 + c_2 f_2 + .. + c_n f_n.$$

Then the minimum condition is equivalent to

$$(f_i, c_1 f_1 + c_2 f_2 + ..c_n f_n - g) = 0, i = 1, .., n.$$

This is the same as

$$c_1(f_i, f_1) + c_2(f_i, f_2) + ..c_n(f_i, f_n) = (f_i, g), i = 1, .., n.$$

These n linear equations in n unknowns are called the normal equations of the problem. In the usual case, S is a space of discrete functions. These are functions defined on a finite domain. Suppose there are m data values so that the domain is

$$\{p_1, p_2, ..., p_m\}.$$

We identify the function f_i with the vector

$$\begin{bmatrix} f_i(p_1) \\ f_i(p_2) \\ ..... \\ ..... \\ f_i(p_m) \end{bmatrix}$$

f_i is an m dimensional column vector of values of the ith function. We can formulate the minimum conditions with matrices. The inner product is then the transpose of the first vector times the second. We write the transpose of a vector v as v^t. We have $(f_i, f_j) = f_i^t f_j$ Then

$$c_1(f_i, f_1) + c_2(f_i, f_2) + ..c_n(f_i, f_n) = (f_i, g), i = 1, .., n.$$

Thus

$$\begin{bmatrix} f_i^t f_1 & ... & f_i^t f_n \end{bmatrix} \begin{bmatrix} c_1 \\ . \\ . \\ . \\ c_n \end{bmatrix} = f_i^t g$$

If we let A be an m row by n column matrix, whose ith column is f_i, then

$$A = \begin{bmatrix} f_1 & f_2 & ... & f_n \end{bmatrix}$$

Written out

$$A = \begin{bmatrix} f_1(p_1) & f_2(p_1) & ... & f_n(p_1) \\ f_1(p_2) & f_2(p_2) & ... & f_n(p_2) \\ ... & ... & ... & ... \\ f_1(p_m) & f_2(p_m) & ... & f_n(p_m) \end{bmatrix}$$

Also let

$$B = \begin{bmatrix} g(p_1) \\ . \\ . \\ . \\ g(p_m) \end{bmatrix}$$

The normal equations become

$$A^t A \begin{bmatrix} c_1 \\ . \\ . \\ . \\ c_n \end{bmatrix} = A^t B.$$

Note that the original approximation problem in this form is a system of m equations in n unknowns

$$A \begin{bmatrix} c_1 \\ . \\ . \\ . \\ c_n \end{bmatrix} \approx B.$$

Any linear system of this form with $m > n$ can be interpreted as a least squares problem and has an approximate least squares solution. The matrices A and B are a convenient input set to a general linear least squares solver (see the listing of subroutine llsq).

There is always a unique solution to the linear least squares problem. The solution is the orthogonal projection into the subspace. But there will be more than one solution to the normal equations if the given functions spanning the subspace are not linearly independent. The normal equations have a solution, so they are consistent. From the theory of linear equations, if the determinant D of the coefficient matrix of the normal equations is not zero, then there is a unique solution. Then we can solve the equations either by inverting the coefficient matrix, or by gaussian elimination. If D is zero, then there is more than one solution, such solution will involve one or more variables of arbitrary value. Gaussian elimination will fail. The $D = 0$ solution can be computed by using elementary row operations which can be done numerically or with various computer algebra programs. When we are concerned only with the discrete space, it does not matter that there are multiple solutions to the normal equations. Because any set of coefficients gives a linear combination equal to the unique projection into the subspace. The various solutions just give different linear combinations of dependent vectors that equal the same vector. On the other hand if points other than the sample points are in the relevant domain of the functions, then the multiple solutions may give function solutions that are not the same on this extended

domain. To illustrate compare functions f and g where $f(x) = x(x-1)$ is equal to zero on the domain $x = 0$ and $x = 1$, but it is not zero on the extended domain of all real numbers. Let g be the true zero function, $g(x) = 0$. The two functions agree on $\{0, 1\}$, but give different values on an extended domain. Frequently we want to use the least squares solution for interpolation between the given data points, and so the case of multiple solutions to the normal equations does have consequence.

We will show that if f_1,..,f_n are linearly independent then the normal equations have a unique solution. This is obvious because in this case f_1,..,f_n is a basis of S and the unique solution f in S has unique components with respect to this basis. It is also a direct consequence of the following proposition.

Proposition. if f_1,...,f_n are the linearly independent columns of a matrix A, which has $m > n$ rows, then $\det(A^t A)$ is not equal to zero.

Proof. Suppose the determinant is zero. Then there exists $c_1, c_2, .., c_n$, not all zero such that

$$c_1 \begin{bmatrix} (f_1, f_1) \\ (f_2, f_1) \\ \ldots\ldots \\ \ldots\ldots \\ (f_n, f_1) \end{bmatrix} + c_2 \begin{bmatrix} (f_1, f_2) \\ (f_2, f_2) \\ \ldots\ldots \\ \ldots\ldots \\ (f_n, f_2) \end{bmatrix} + .. + c_n \begin{bmatrix} (f_1, f_n) \\ (f_2, f_n) \\ \ldots\ldots \\ \ldots\ldots \\ (f_n, f_n) \end{bmatrix} = 0.$$

Let

$$v = c_1 f_1 + ... + c_n f_n.$$

The first equation shows that $(f_i, v) = 0$, for $i = 1, .., n$. It follows that $(v, v) = 0$. This implies $v = 0$, and so each c_i is zero. This is a contradiction, so the proposition is true.

Example 1. We are to fit the function

$$y = f(x) = a\sin(x) + b\cos(x).$$

to the data

$$\begin{array}{cc} x & y \\ 1.0 & 3.0 \\ 2.5 & 5.6 \\ 3.4 & 7.8 \end{array},$$

Apply the sin function to the x values to get the first column of matrix A and the cos function to get the second column. Let vector B be the y values.

The normal equations are

$$A^t AC = A^t B$$

or in terms of the components

$$\begin{bmatrix} 1.13154358 & 0.22224325 \\ 0.22224325 & 1.86845642 \end{bmatrix} C = \begin{bmatrix} 3.882636366 \\ -10.40652323 \end{bmatrix}$$

The solution is

$$C = \begin{bmatrix} 4.6334245 \\ -6.120705005 \end{bmatrix}$$

So

$$f(x) = 4.6334245 \sin(x) - 6.120705005 \cos(x)$$

The following program does the linear least squares computations.

```
c+ llsq   least squares solution of a*c=b (solving for c)
      subroutine llsq(a,ia,m,n,ws,c,b,ier)
c parameters
c     a-m by n matrix. declared row dimension ia.
c    ws-working storage vector of length m
c     c-vector of size n
c     b-vector of size m
c   ier-return parameter: ier=0 normal return,ier=1 normal
c       equations
c       nearly singular,ier=2 normal equations singular.
c
      dimension a(ia,1),b(1),c(1),ws(1)
c    compute lower elements of jth column  of transpose(a)*a
      do 50 j=1,n
      do 18 i=j,n
      s=0.
      do 15 k=1,m
      s=s+a(k,i)*a(k,j)
 15    continue
 18    ws(i)=s
c
c  compute jth element of right side vector
      s=0.
      do 40 k=1,m
```

```
40    s=s+a(k,j)*b(k)
      c(j)=s
c
c  store lower elements of jth column in a
      do 19 i=j,n
 19    a(i,j)=ws(i)
c
 50    continue
c  fill in upper values
      do 60 i=1,n
      do 60 j=i,n
      a(i,j)=a(j,i)
 60   continue
      ib=1
      mm=1
      eps=1.e-12
      inv=0
c  solve normal equations
      call gausse(a,ia,c,ib,n,mm,inv,eps,det,ier)
      return
      end
```

82.3 Linear Circle Fitting

In some problems our basis functions may not be defined on a subset of the real numbers An example is the case of finding a best fitting circle. We are given a set of points $p_i = (x_i, y_i)$. We have two basic functions defined on the set of points, namely a function that maps the point to its x coordinate and a function that maps the point to its y coordinate. Our space of functions will consist of algebraic combinations of these two basic functions. We may write the general equation of a circle as

$$(x-h)^2 + (y-k)^2 = r^2$$

Expanding this gives

$$-2hx - 2ky + (h^2 + k^2 - r^2) = x^2 + y^2.$$

Our functions are

$$f_1(p_i) = x_i, f_2(p_i) = y_i, f_3(p_i) = 1,$$

and

$$g(p_i) = x_i^2 + y_i^2.$$

The coefficients are

$$c_1 = -2h, c_2 = -2k, c_3 = (h^2 + k^2 - r^2).$$

The radius r and the center (h, k) are determined by the coefficients.

One might wonder what happens when the data points lie on a straight line. Then the geometric problem obviously has no solution, while the least squares problem has a solution, because we have shown that it always has a solution. In this case, the functions we have defined for this problem are not linearly independent. Projection into the spanned subspace gives a minimal function f, but the coefficients of the basis functions are not unique. The solution space is of dimension two. There is an infinite set of coefficients that give the same f. But each set of three coefficients defines a different circle. We are free to choose one of the parameters. This is equivalent to choosing a pair of linearly independent functions. Each time we do this we are solving a different circle problem. For example, if we assume h=0 then we are finding the best fitting circle whose center lies on the y-axis.

It is interesting to discuss the meaning of "best fitting." If all points lie on the circle than this algorithm will find the circle. Further the mapping between data sets and circles is continuous, i. e., if the data points are changed by a small amount, then the fitted circle changes by a small amount. The solution we have described may be adequate, but the problem can be formulated more naturally to minimize the perpendicular distance from the fitted circle to the data. Unfortunately the problem becomes nonlinear. The nonlinear circle fitting problem is to minimize the function

$$f(h, k, r) = \sum_{i=1}^{n} (\|(h, k) - p_i\| - r)^2,$$

where (h, k) are the coordinates of the center, r is the radius, and $\{p_i\}$ is the set of data points. See the section "Nonlinear circle fitting." Here is the linear fitting routine.

```
c+ lsqcir  least squares fit of circle.
      subroutine lsqcir(x,y,n,c,r,ier)
       implicit real*8(a-h,o-z)
c  function        -to calculate the best fitting circle
```

```
c                   to n points.
c  parameters   x,y-arrays containing the coordinates of
c                   the n points.
c                 c-array of dimension 2 containing the
c                   coordinates of the center of the circle.
c                 r-radius of circle.
c               ier-error parameter
c                   =0 ,normal return
c                   =1 ,equation system is singular
c change to real*8 and use gaussr 9/19/91
      dimension x(*),y(*),c(*),a(3,3),b(3)
      z=-(x(1)**2+y(1)**2)
      b(1)=x(1)*z
      b(2)=y(1)*z
      b(3)=z
      a(1,1)=x(1)**2
      a(1 ,2)=x(1)*y(1)
      a(1,3)=x(1)
      a(2,2)=y(1)**2
      a(2,3)=y(1)
      a(3,3)=n
      do 10 i=2,n
      a(1,1)=a(1,1)+x(i)**2
      a(1,2)=a(1,2)+x(i)*y(i)
      a(1,3)=a(1,3)+x(i)
      a(2,2)=a(2,2)+y(i)**2
      a(2,3)=a(2,3)+y(i)
      z=-(x(i)**2+y(i)**2)
      b(1)=b(1)+z*x(i)
      b(2)=b(2)+z*y(i)
   10 b(3)=b(3)+z
      a(2,1)=a(1,2)
      a(3,1)=a(1,3)
      a(3,2)=a(2,3)
      nn=3
      m=1
      inv=-1
      eps=1.e-15
```

```
      ia=3
      ib=3
      idet=0
      call gaussr(a,ia,b,ib,nn,m,inv,eps,idet,det,ier)
      if(ier.gt.0)then
         ier=1
       else
         ier=0
      endif
      c(1)=-b(1)/2.
      c(2)=-b(2)/2.
      r=sqrt(c(1)**2+c(2)**2-b(3))
      return
      end
```

82.4 Nonlinear Circle Fitting

Let $d(p, c)$ be the distance from a point p to a circle c with center (h,k) and radius r.

$$d^2(p, c) = ([(h - x)^2 + (k - y)^2]^{1/2} - r)^2$$

Let v be defined by

$$v(x, y) = [(h - x)^2 + (k - y)^2]^{1/2}$$

Given n points $p_i = (x_i, y_i)$, we minimize

$$f(h, k, r) = \sum_{i=1}^{n} d^2(p_i, c)$$

$$= \sum_{i=1}^{n} (v_i - r)^2.$$

The three necessary equations are

$$\partial f/\partial h = \sum_{i=1}^{n} 2(v_i - r)(h - x_i)/v_i = 0$$

$$\partial f/\partial k = \sum_{i=1}^{n} 2(v_i - r)(k - y_i)/v_i = 0$$

$$\partial f/\partial r = -\sum_{i=1}^{n} 2(v_i - r) = 0$$

We may solve the last equation for r,

$$r = \frac{\sum_{j=1}^{n} v_j}{n}$$

Simplifying we have

$$\sum_{i=1}^{n}(nv_i - \sum_{j=1}^{n} v_j)(h - x_i)/v_i = 0$$

$$\sum_{i=1}^{n}(nv_i - \sum_{j=1}^{n} v_j)(k - y_i)/v_i = 0$$

We may write this as a vector equation

$$\mathbf{F}(h, k) = 0.$$

To solve this equation we may compute the Jacobian $\mathbf{J}$ and use Newton's method

$$\mathbf{X}_{n+1} = \mathbf{X}_n - \mathbf{F}(h, k)\mathbf{J}^{-1},$$

where

$$\mathbf{X} = \begin{bmatrix} h \\ k \end{bmatrix}.$$

The following computer program solves the two variable equation using an IMSL root solver. Starting values are computed form the linear fitting routine LSQCIR.

```
c fitc2_for fit a circle 1-29-90 (nos/ve)
      parameter (n=2,m=5)
      dimension x(n),xguess(n),f(n)
      real xpp(m),ypp(m),c(2)
      external fcn,neqnf
      common /points/ npts,xp(200),yp(200)
      data (xpp(j),j=1,m)/1.,0.,-1.,.0,1./
      data (ypp(j),j=1,m)/9.,1.,0.,-1.,0./
      do 1 i=1,m
      xp(i)=xpp(i)
```

```
      yp(i)=ypp(i)
1     continue
      npts=m
      call lsqcir(xp,yp,m,c,r,ier)
      write(*,*)' linear solution: '
      write(*,*)' center = ',c(1),c(2)
      write(*,*)' radius = ',r
      write(*,*)' ier = ',ier
      call distc(xp,yp,m,c(1),c(2),r,dmn,dmx,dmean)
      write(*,*)' dmn,dmx,dmean = ',dmn,dmx,dmean
      er=.0000001
      itmax=200
      xguess(1)=c(1)
      xguess(2)=c(2)
      call neqnf(fcn,er,n,itmax,xguess,x,fnorm)
c     neqnf is an imsl root finder
      c(1)=x(1)
      c(2)=x(2)
      r=234.
      write(*,*)' nonlinear solution: '
      write(*,*)' center = ',c(1),c(2)
      sumv=0
      do 5 i=1,m
      sumv=sumv+sqrt((c(1)-xpp(i))**2+(c(2)-ypp(i))**2)
5     continue
      r=sumv/m
      write(*,*)' radius = ',r
      call distc(xp,yp,m,c(1),c(2),r,dmn,dmx,dmean)
      write(*,*)' dmn,dmx,dmean = ',dmn,dmx,dmean
      fn=fcn(x,f,n)
      do 20 i=1,n
      write(*,*)' function ',i,f(i)
 20   continue
      end
c
      function fcn(p,f,n)
      common /points/ npts,x(200),y(200)
      real p(n),f(n),h,k,r
```

```
      h=p(1)
      k=p(2)
      write(*,*)' h k = ',h,k
      sumv=0
      do 5 i=1,npts
      sumv=sumv+sqrt((h-x(i))**2+(k-y(i))**2)
5     continue
      s1=0.
      s2=0.
      do 10 i=1,npts
      v=sqrt((h-x(i))**2+(k-y(i))**2)
      s1=s1+(npts*v-sumv)*(h-x(i))/v
      s2=s2+(npts*v-sumv)*(k-y(i))/v
10    continue
      f(1)=s1
      f(2)=s2
      fcn=1.
      return
      end
```

This program computes the distance function.

```
c+ distc  distances between points and circle
    subroutine distc(x,y,n,cx,cy,r,dmn,dmx,rms)
c   input:
c      x,y-points
c        n-number of points
c     cx,cy-center of circle
c  output:
c     dmn-minimum distance from a point to circle
c     dmx-maximum distance from a point to circle
c     rms-square root of the mean square distance
    real x(n),y(n)
    do 10 i=1,n
    d2=(sqrt((x(i)-cx)**2+(y(i)-cy)**2)-r)**2
    d=sqrt(d2)
    if(i.eq.1)then
      sum=d2
      dmn=d
```

```
      dmx=d
    else
      sum=sum+d2
      if(d.lt.dmn)dmn=d
      if(d.gt.dmx)dmx=d
    endif
10    continue
    rms=sqrt(sum/n)
    return
    end
```

82.5 Weighted Least Squares

We can give more importance to certain data points in the fitting process by assigning weights. We can modify our inner product and use the general theory. Define a diagonal n by n matrix W with diagonal elements w_i. Define a new inner product ρ by

$$\rho(f_i, f_j) = f_i^T W f_j.$$

The normal equations become

$$A^T W A x = A^T W b.$$

As an example of this technique we take the exponential fitting problem.

$$y = ae^{bx}$$

This is a nonlinear problem. We take logarithms to get a related linear problem

$$\ln(y_i) = \ln(a) + bx_i.$$

Let

$$\delta_i = \ln(a) + bx_i - \ln(y_i).$$

The linear least squares technique will minimize the sum of squares of the δ_i's. But we want to minimize the sum of squares of the deviations of the original problem

$$d_i = ae^{bx_i} - y_i.$$

We can find an approximate relation between these deviations. Applying the exponential function we have

$$e^{\delta_i} = ae^{bx_i}/y_i.$$

If we replace the exponential function by the first two terms of its Taylor series expansion, we get

$$y_i(1+\delta_i) = ae^{bx_i}.$$

Then

$$y_i\delta_i = ae^{bx_i} - y_i = d_i.$$

We can use $w_i = y_i^2$ and apply weighted least squares to improve the approximate solution to the nonlinear problem.

82.6 Bilinear Interpolation Example

Let the unit square be mapped to a region defined by four corner points $p_{00}, p_{10}, p_{01}, p_{11}$. Define the mapping by

$$q_1 = (1-u)p_{00} + up_{10}$$

$$q_2 = (1-u)p_{01} + up_{11}$$

and

$$p(u,v) = (1-v)q_1 + vq_2.$$

This is a bilinear function. Expanding, the equation becomes

$$p(u,v) = p_{00} + (p_{00}+p_{10})u + (p_{01}-p_{00})v + (p_{00}-p_{10}-p_{01}+p_{11})uv.$$

Now suppose we are given a data set

$$\{(u_i, v_i, b_i) : i = 1, ..., m\},$$

and we are to find a bilinear mapping approximately taking each pair (u_i, v_i) to the vector b_i. We obtain a set of m equations

$$c_1 + c_2u_i + c_3v_i + c_4u_iv_i = b_i,$$

where

$$p_{00} = c_1,$$

$$p_{10} = c_2 - c_1,$$
$$p_{01} = c_3 + c_1,$$
$$p_{11} = c_4 + c_2 + c_3 - c_1.$$

This problem can be solved by least squares. There are three independent systems for the x, y, and z components.

82.7 Fitting Functions With Horizontal Asymptotes, Using the general program lsqgen.

One approach to fitting when there are horizontal asymptotes is to swap the variables so that the asymptotes become vertical. A vertical asymptote is due to a pole. Thus consider the function

$$g(y) = x = \frac{a}{y} + \frac{b}{y-1}.$$

The function as a pole at 0 and a pole at 1. Suppose a and b are 1. Then solving for y we get the inverse

$$f(x) = y = \frac{1}{2}\frac{x + 2 - \sqrt{x^2+4}}{x}.$$

Note that y goes to zero as x goes to positive infinity, and y goes to 1 as x goes to negative infinity.

The technique then is to fit a linear combination of poles using least squares, and then invert the resulting function. This is equivalent to finding symbolically the roots of a polynomial. In the example, the polynomial is of second degree. For higher degree cases one may have to use numerical methods to invert the function. Consider the data which has been switched to give vertical asymptotes:

x	y
0.20000000	7.5000000
0.50000000	4.5000000
2.1000000	1.8000000
3.0000000	1.0000000
4.0000000	-1.0000000
4.5000000	-3.0000000
4.8000000	-5.5000000

We shall try to fit the following linear combination of functions

$$g = a_1 + a_2 x + a_3/x a_4/x^2 + a_5/(x-5) + a_6/(x-5)^2,$$

using **lsqgen.ftn**. We get the following output:

```
% lsqgen
  linear least squares curve fitting
  maximum points=  200
  maximum degree=  19
  enter file name [p.dat]
pp.dat
 7 points

  f(x) =
f(x)= 3.48346534312582     f( 1) +
      -.560314483245716     f( 2) +
      0.932168101605212     f( 3) +
      0.260354762617237E-03 f( 4) +
       2.63951411841873     f( 5) +
      0.268481286458776     f( 6)
             data
      x               y             fit
 0.20000000       7.5000000       7.5005059
 0.50000000       4.5000000       4.4953854
  2.1000000       1.8000000       1.8725003
  3.0000000       1.0000000      0.86063678
  4.0000000      -1.0000000     -0.89576712
  4.5000000      -3.0000000      -3.0358916
  4.8000000      -5.5000000      -5.4973696
sigma =  0.72564817782510E-01
  enter a plot file name [p.gi]

 Enter plot window, default is:
[ 0.20000000       4.8000000      -5.5000000       7.5000000     ]
.1 4.9 -8 8
% pltgpr
```

We can try to invert this equation with Maple. The closed form inversion

is too messy for practical use. Refer to the next section for a listing of **lsqgen.ftn**.

82.8 A General LSQ Program lsqgen.ftn

lsqgen.ftn defines a set of functions in a function subroutine located at the bottom of the listing. To change the LSQ problem, one changes these definitions, and the variable n, which is the number of functions in the linear combination. **lsqgen.ftn** also generates a plotfile (.gi file).

82.9 A Listing of lsqgen.ftn

```
c lsqgen.ftn general linear least squares program 5/5/95
c functions defined in f at bottom of listing
      implicit real*8 (a-h,o-z)
      common /block1/x1,n
      parameter (numpts=200,nfncts=20)
      dimension x(numpts),y(numpts),xt(numpts),yt(numpts)
      dimension a(numpts,nfncts),b(numpts),c(nfncts),ws(numpts)
      dimension ain(5)
      character*20 ans
      character*30 fn,pltfn
      ia=numpts
      zero=0.
      write(*,*)' linear least squares curve fitting'
      write(*,*)' maximum points= ',numpts
      write(*,*)' maximum degree= ',(nfncts-1)
c     cccccccccccccc read data points
      m=0
      write(*,*)' enter file name [p.dat]'
      read(*,'(a)')fn
      if(lenstr(fn).lt.1)then
       fn='p.dat'
      endif
      open(3,file=fn,status='unknown')
10    continue
      call readr(3,ain,nr)
      if(nr.le.0)then
       close(3)
       write(*,*)m,' points'
       if(m .eq. 0)then
        stop
       endif
       go to 20
      endif
      m=m+1
      if(m.gt.numpts)then
       write(*,*)' the file contains more than the allowed '
       write(*,*)' number of points.'
       write(*,*)m,' points read'
       close(3)
```

```
      stop
     endif
     x(m)=ain(1)
     y(m)=ain(2)
     go to 10
20   continue
c    cccccccccccccccc construct and solve normal equations
c    make a dummy call to f to define n
     a(1,1)=f(1,x(1))
     do 40 i=1,m
      b(i)=y(i)
      do 30 j=1,n
       a(i,j)=f(j,x(i))
30    continue
40   continue
     call llsq(a,ia,m,n,ws,c,b,ier)
     if(ier .ne. 0)then
      write(*,*)' llsq has returned an error condition'
      write(*,*)' ier= ',ier
      write(*,*)' normal equations are singular, or nearly singular'
      stop
     endif
     write(*,*)
     write(*,*)' f(x) ='
     do 50 i=1,n
      if(c(i) .ne. zero)then
       if(i .eq. 1)then
        write(*,'(a,g21.15,a,i2,a)')'f(x)=',c(i),' f(',i,') +'
       else
        if(i .ne. n)then
         write(*,'(6x,g21.15,a,i2,a)')c(i),' f(',i,') +'
        else
         write(*,'(6x,g21.15,a,i2,a)')c(i),' f(',i,')'
        endif
       endif
      endif
50   continue
     write(*,*)'            data'
     write(*,*)'     x               y             fit'
     sigma=0.
     do 60 i=1,m
      xx=x(i)
      yy=y(i)
      call flc(c,n,xx,fxx)
      write(*,'(3(g15.8,1x))')xx,yy,fxx
      sigma=sigma+(y(i)-fxx)**2
60   continue
     sigma=sqrt(sigma/m)
     write(*,'(a,g21.14)')'sigma = ',sigma
c    cccccccccccccccccccccccccc plot of data and fitted function
     write(*,*)' enter a plot file name [p.gi]'
     read(*,'(a)')pltfn
     if(lenstr(pltfn).lt.1)then
      pltfn='p.gi'
     endif
     open(2,file=pltfn,status='unknown')
     call plotgi(c,n,x,y,m)
```

```
      end
c+ plotgi plot gi file
      subroutine plotgi(c,n,x,y,m)
       implicit real*8 (a-h,o-z)
       dimension x(*),y(*),c(*)
       dimension ain(4)
       nc=4
       x1=1.0e30
       x2=-x1
       y1=x1
       y2=x2
       do 5 i=1,m
        if(x(i).gt.x2)x2=x(i)
        if(x(i).lt.x1)x1=x(i)
        if(y(i).gt.y2)y2=y(i)
        if(y(i).lt.y1)y1=y(i)
5      continue
c       dx=(x2-x1)/20.
c       x1=x1-dx
c       x2=x2+dx
c       dy=(y2-y1)/20.
c       y1=y1-dy
c       y2=y2+dy
       write(2,'(a)')'v-1 1 -1 1'
       write(*,*)'Enter plot window, default is:'
       write(*,'(a,4(g15.8,1x),a)')'[',x1,x2,y1,y2,']'
       call readr(0,ain,nr)
       if(nr .eq. 4)then
        x1=ain(1)
        x2=ain(2)
        y1=ain(3)
        y2=ain(4)
       endif
       write(2,'(a,4(g12.5,1x))')'w',x1,x2,y1,y2
       do 10 i=1,m
        write(2,'(a,g12.5,1x,g12.5,a)')'s',x(i),y(i),' 1 .01'
10     continue
       nc = mod(nc,5)+3
       write(2,'(a,i2)')'c',nc
       npts=200
       do 30 i=1,npts
        u=(i-1)*(x2-x1)/(npts-1)+x1
        call flc(c,n,u,v)
        if(i .eq. 1)then
         write(2,'(a,g12.5,1x,g12.5)')'m',u,v
        else
         write(2,'(a,g12.5,1x,g12.5)')'d',u,v
        endif
30     continue
      end
c
c+ gausse   solution of linear system, matrix inverse, determinant
      subroutine gausse(a,ia,b,ib,n,m,inv,eps,det,ier)
       implicit real*8 (a-h,o-z)
c  algorithm       -gaussian elimination with partial pivoting.
c  parameters     a-n by n matrix containing the coefficients of
c                   the linear system.
```

```
c                ia-row dimension of a in calling program
c                 b-n by m matrix containing the m right sides
c                   of the equations, on entering.  on returning, b
c                   contains the solutions.  the inverse of a is
c                   returned in b when that option is selected.
c                ib-row dimension of b in calling program
c                 n-row and column dimension of a.
c                 m-column dimension of b.
c               inv-when inv=1 the inverse is calculated
c                   and returned in b
c               eps-each equation is normalized so that the
c                   coefficients are .le. 1 in magnitude.
c                   when a pivot is less than eps the matrix is
c                   considered nearly singular.
c                   if a pivot is zero the matrix is singular.
c               det-determinant of a.
c               ier-return parameter, ier=0 is normal return
c                   ier=1 when matrix is nearly singular
c                   ier=2 when the matrix is singular
c
      logical p
      dimension a(ia,*),b(ib,*)
      zero=0.
      ier=0
      if(inv.ne.1)go to 15
c     set b equal to the identity
      do 10 i=1,n
      do 10 j=1,n
      b(i,j)=0.
      if(i.eq.j)b(i,j)=1.
   10 continue
      m=n
15    continue
c normalize rows
      do 20 i=1,n
      bigest=abs(a(i,1))
      do 16 j=2,n
      ab=abs(a(i,j))
      if(ab.gt.bigest)bigest=ab
16    continue
      p=bigest.eq.zero
      if(p)ier=2
      if(p)det=0.
      if(p)return
      do 18 j=1,n
18    a(i,j)=a(i,j)/bigest
      do 19 j=1,m
19    b(i,j)=b(i,j)/bigest
20    continue
      det=1.
      j=1
 30   kk=j+1
      l=j
      do 32 i=kk,n
   32 if(abs(a(i,j)).gt.abs(a(l,j)))l=i
      p=abs(a(l,j)).eq.zero
      if(p)ier=2
```

```
      if(p)det=0.
      if(p)return
      p=abs(a(l,j)).le.eps
      if(p)ier=1
 34   if(l.eq.j)go to 40
c     interchange rows
      do 37 k=1,n
      c=a(l,k)
      a(l,k)=a(j,k)
   37 a(j,k)=c
      do 39 k=1,m
      c=b(l,k)
      b(l,k)=b(j,k)
   39 b(j,k)=c
      det=det*(-1.)
   40 det=det*a(j,j)
c     divide row by pivot
      c=a(j,j)
      do 50 k=j,n
   50 a(j,k)=a(j,k)/c
      do 55 k=1,m
   55 b(j,k)=b(j,k)/c
c     add multiple of row j
      jj=j+1
      do 70 i=jj,n
      am=a(i,j)
      do 60 k=1,n
   60 a(i,k)=a(i,k)-am*a(j,k)
      do 70 k=1,m
   70 b(i,k)=b(i,k)-am*b(j,k)
      j=j+1
      if(j.ne.n)go to 30
      am=a(n,n)
      p=abs(am).eq.zero
      if(p)ier=2
      if(p)det=0.
      if(p)return
      p=abs(am).le.eps
      if(p)ier=1
 73   det=det*am
      do 80 k=1,m
   80 b(n,k)=b(n,k)/am
c     back substitution
      nn=n-1
      do 100 i=1,nn
      ni=n-i
      do 100 j=1,m
      nj=ni+1
      do 100 ki=nj,n
  100 b(ni,j)=b(ni,j)-a(ni,ki)*b(ki,j)
      return
      end
c
c+ llsq   least squares solution of a*c=b (solving for c)
      subroutine llsq(a,ia,m,n,ws,c,b,ier)
       implicit real*8 (a-h,o-z)
c parameters
```

```
c     a-m by n matrix. declared row dimension ia.
c    ws-working storage vector of length m
c     c-vector of size n
c     b-vector of size m
c   ier-return parameter: ier=0 normal return,ier=1 normal equations
c       nearly singular,ier=2 normal equations singular.
c
      dimension a(ia,*),b(*),c(*),ws(*)
c    compute lower elements of jth column  of transpose(a)*a
      do 50 j=1,n
      do 18 i=j,n
      s=0.
      do 15 k=1,m
      s=s+a(k,i)*a(k,j)
 15    continue
 18    ws(i)=s
c
c  compute jth element of right side vector
      s=0.
      do 40 k=1,m
40    s=s+a(k,j)*b(k)
      c(j)=s
c
c  store lower elements of jth column in a
      do 19 i=j,n
 19    a(i,j)=ws(i)
c
 50    continue
c  fill in upper values
      do 60 i=1,n
      do 60 j=i,n
      a(i,j)=a(j,i)
 60   continue
      ib=1
      mm=1
      eps=1.e-12
      inv=0
c  solve normal equations
       mprnt=0
      if(mprnt.eq.1)then
         write(*,*)'  coefficients of normal equations a= '
         do 500 i=1,n
           write(*,'(7(1x,g11.4))')(a(i,j),j=1,n)
500      continue
        read(*,*)
         write(*,*)' b= '
         do 510 i=1,n
           write(*,'(1(1x,g11.4))')c(i)
510      continue
        read(*,*)
      endif
      call gausse(a,ia,c,ib,n,mm,inv,eps,det,ier)
      return
      end
c+ readr read a row of floating point numbers
      subroutine readr(nf, a, nr)
      implicit real*8(a-h,o-z)
```

```
c   numbers are separated by spaces
c   examples of valid numbers are:
c   12.13 34 45e4 4.78e-6 4e2,5.6D-23,10000.d015
c  nf=file number, 0 for standard input file
c   a=array of returned numbers
c  nr=number of values in returned array,
c     or 0 for empty or blank line,
c     or -1 for end of file on unit nf.
c requires functions val and length
      dimension a(*)
      character*200  b
      character*200  c
      character*1 d
        c=' '
      if(nf.eq.0)then
         read(*,'(a)',end=99)b
      else
         read(nf,'(a)',end=99)b
      endif
      nr=0
      l=lenstr(b)
      if(l.ge.200)then
        write(*,*)' error in readr subroutine '
        write(*,*)' record is too long '
      endif
      do 1 i=1,l
       d=b(i:i)
       if (d.ne.' ') then
        k=lenstr(c)
        if (k.gt.0)then
           c=c(1:k)//d
        else
           c=d
        endif
       endif
       if( (d.eq.' ').or.(i.eq.l)) then
        if (c.ne.' ') then
          nr=nr+1
           call valsub(c,a(nr),ier)
          c=' '
        endif
       endif
1     continue
      return
99    nr=-1
      return
      end
c
c+ lenstr   nonblank length of string
      function lenstr(s)
c  length of the substring of s obtained by deleting all
c  trailing blanks from s.  thus the length of a string
c  containing only blanks will be 0.
      character   s*(*)
      lenstr=0
      n=len(s)
      do 10 i=n,1,-1
```

```
      if(s(i:i) .ne. ' ')then
         lenstr=i
         return
      endif
10    continue
      return
      end
c+ valsub converts string to floating point number (r*8)
      subroutine valsub(s,v,ier)
      implicit real*8(a-h,o-z)
c     examples of valid strings are: 12.13 34 45e4 4.78e-6 4E2
c     the string is checked for valid characters,
c     but the string can still be invalid.
c      s-string
c      v-returned value
c      ier- 0 normal
c           1 if invalid character found, v returned 0
c
      logical p
      character s*(*),c*50,t*50,ch*15
      character z*1
      data ch/'1234567890+-.eE'/
      v=0.
      ier=1
      l=lenstr(s)
      if(l.eq.0)return
      p=.true.
      do 10 i=1,l
      z=s(i:i)
      if((z.eq.'D').or.(z.eq.'d'))then
          s(i:i)='e'
      endif
      p=p.and.(index(ch,s(i:i)).ne.0)
 10   continue
      if(.not.p)return
      n=index(s,'.')
      if(n.eq.0)then
          n=index(s,'e')
          if(n.eq.0)n=index(s,'E')
          if(n.eq.0)n=index(s,'d')
          if(n.eq.0)n=index(s,'D')
          if(n.eq.0)then
             s=s(1:l)//'.'
          else
            t=s(n:l)
            s=s(1:(n-1))//'.'//t
          endif
          l=l+1
      endif
      write(c,'(a30)')s(1:l)
      read(c,'(g30.23)')v
      ier=0
      return
      end
c+ str  floating point number to string
      subroutine str(x,s)
      implicit real*8(a-h,o-z)
```

```
      character s*25,c*25,b*25,e*25
       zero=0.
      if(x.eq.zero)then
         s='0'
         return
      endif
      write(c,'(g11.4)')x
      read(c,'(a25)')b
      l=lenstr(b)
      do 10 i=1,l
      n1=i
      if(b(i:i).ne.' ')go to 20
 10   continue
 20   continue
       if(b(n1:n1).eq.'0')n1=n1+1
       b=b(n1:l)
       l=l+1-n1
       k=index(b,'E')
       if(k.gt.0)e=b(k:l)
       if(k.gt.0)then
         s=b(1:(k-1))
         k1=index(b,'E+0')
         if(k1.gt.0)then
          e='E'//b((k1+3):l)
         else
          k1=index(b,'E+')
          if(k1.gt.0)e='E'//b((k1+2):l)
         endif
         k1=index(b,'E-0')
         if(k1.gt.0)e='E-'//b((k1+3):l)
         l=k-1
       else
           s=b
       endif
       j=index(s,'.')
       n2=l
      if(j.ne.0)then
               do 30 i=1,l
                n2=l+1-i
                if(s(n2:n2).ne.'0')go to 40
30              continue
       endif
40              continue
        s=s(1:n2)
        if(s(n2:n2).eq.'.')then
           s=s(1:(n2-1))
           n2=n2-1
         endif
        if(k.gt.0)s=s(1:n2)//e
      return
      end
c+ flc  value of linear  combination function: flc(x) = c_{i} f_i(x)
      subroutine flc(c,n,x,fx)
       implicit real*8 (a-h,o-z)
       dimension c(*)
       fx=0.
       do 10 i=1,n
```

```
        fx = fx + c(i)*f(i,x)
10     continue
       return
      end
c+ f definition of the ith fitting function: f_i(x)
      function f(i,x)
       implicit real*8 (a-h,o-z)
       common /block1/x1,n
c      define parmeters x1 = pole
        x1 = 5.
c      define number of functions n, main program will make a dummy call
c       to obtain n
       n=6
      zero=0.
       if(i .eq. 1)then
        f=1.
       endif
       if(i .eq. 2)then
        f=x
       endif
       if(i .eq. 3)then
        if(x .ne. zero)then
         f=1./x
        endif
       endif
       if(i .eq. 4)then
        if(x .ne. zero)then
         f=(1./x)**2
        endif
       endif
       if(i .eq. 5)then
        r=x-x1
        if(r .ne. zero)then
         f=(1./r)
        endif
       endif
       if(i .eq. 6)then
        r=x-x1
        if(r .ne. zero)then
         f=(1./r)**2
        endif
       endif
       return
      end
```

83 References for LaTeX and Postscript

The book **The Latex Graphics Companion** by Michel Goossens, Sebastian Rahtz, and Frank Mittelbach, Addison-Wesley, 1997, explains the use of various graphics programs with Latex and PostScript. It shows how to use several programs that create figures for Latex. Unfortunately it does not

say much about psfig, but concentrates on a similar macro package called PSTricks. There may be documentation for psfig that can be obtained from the internet. Only later versions of PCTeX support the use of psfig and PostScript. To view the output of PCTeX that contains psfig pictures, PCTeX must be operated in PostScript mode. The disadvantage of this mode is that it gives somewhat slower typesetting. See the PCTeX manual for details of using embedding graphics figures.. A version of PCTeX that supports PostScript mode is: PCTeX32 version 4.0. The manual for this version of PCTeX gives information on the PostScript mode.

There are many books on PostScript, including the Red, Green, and Blue books from Adobe. The Red book is now in its third edition. There is a document by Jim Emery containing material about EG graphics. It is called "Computer Graphics" (graphics.tex). There are several programs for viewing and printing EG graphics files. One for Windows 95-98 is called wineg.cpp. Utility programs mentioned in this document can be obtained from the author.

84 The Matrix Singular Value Decomposition

The singular value decomposition of an m by n real matrix A is

$$A = UDV^T,$$

where D is an m by n diagonal matrix and U and V are orthogonal matrices. The diagonal elements of D are called singular values.

The L_2 norm of a matrix is defined by

$$\|A\| = \sup \frac{\|Ax\|}{\|x\|}.$$

Proposition.(Golub and Van Loan p71) Let A be a real m by n matrix. There exists an m by m orthogonal matrix U, a n by n orthogonal matrix V, and an m by n diagonal matrix D, with diagonal elements

$$\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq .... \geq \sigma_p \geq 0,$$

so that

$$U^T AV = D.$$

The diagonal elements $\{\sigma_1, ..., \sigma_p\}$ are called the singular values of A.
Proof. If A is zero let U and V be the identity matrices and let $D = 0$. If A is a nonzero m by 1 vector, then let

$$x = A/\|A\|.$$

Choose U_1 so that

$$U = \begin{bmatrix} x & U_1 \end{bmatrix}$$

is orthogonal. Let V be the one dimensional identity. Then

$$U^T AV = \begin{bmatrix} x^T \\ U_1^T \end{bmatrix} A = \begin{bmatrix} \|A\| \\ 0 \\ ... \\ 0 \end{bmatrix}.$$

We set $\sigma_1 = \|A\|$.

If A is a 1 by n vector then we may apply the previous case to A^T, and take the transpose of the result.

Finally suppose A is an m by n matrix. There exists an x so that $\|x\| = 1$ and $\|A\| = \|Ax\|$. Let

$$y = \frac{Ax}{\|A\|}.$$

Let $\sigma_1 = \|A\|$. Then $\|y\| = 1$ and $Ax = \sigma_1 y$. Let U_1 and V_1 be orthogonal matrices so that

$$U = \begin{bmatrix} y & U_1 \end{bmatrix}$$

and

$$V = \begin{bmatrix} x & V_1 \end{bmatrix}$$

are orthogonal. Then

$$U^T AV = \begin{bmatrix} y^T \\ U_1^T \end{bmatrix} AV$$

$$= \begin{bmatrix} y^T \\ U_1^T \end{bmatrix} \begin{bmatrix} Ax & AV_1 \end{bmatrix}$$

$$= \begin{bmatrix} y^T \\ U_1^T \end{bmatrix} \begin{bmatrix} \sigma_1 y & AV_1 \end{bmatrix}$$

$$= \begin{bmatrix} \sigma_1 & y^T A U_1 \\ \sigma_1 U_1^T y & U_1^T A V_1 \end{bmatrix}$$

$$= \begin{bmatrix} \sigma_1 & w^T \\ 0 & B \end{bmatrix}.$$

An orthogonal matrix preserves length, for example

$$\|Uz\|^2 = (Uz)^T Uz = z^T U^T U z = z^T z = \|z\|^2.$$

Hence an orthogonal matrix has norm equal to one. We allows use to compute

$$\|A_1\| = \|U^T A V\| \leq \|U^T\| \|A\| \|V\| = \|A\|.$$

Solving for A and using a similar argument, we have

$$\|A\| \leq \|A_1\|,$$

So

$$\|A\| = \|A_1\|.$$

Let

$$C = A_1 \begin{bmatrix} \sigma_1 \\ w \end{bmatrix}$$

$$= \begin{bmatrix} \sigma_1 & w^T \\ 0 & B \end{bmatrix} \begin{bmatrix} \sigma_1 \\ w \end{bmatrix}$$

$$= \begin{bmatrix} \sigma_1^2 + \|w\|^2 \\ Bw \end{bmatrix}.$$

Then

$$\sigma_1^2 + \|w\|^2 = \|C \begin{bmatrix} 1 \\ 0 \\ .. \\ 0 \end{bmatrix} \|$$

$$\leq \|C\|.$$

Then

$$\sigma_1 = \|A\| = \|A_1\| \geq \frac{\|C\|}{\sqrt{\sigma_1^2 + \|w\|^2}} = \sqrt{\sigma_1^2 + \|w\|^2},$$

which implies that $w = 0$. Thus

$$U^T AV = \begin{bmatrix} \sigma_1 & 0 \\ 0 & B \end{bmatrix}.$$

Now we shall apply an induction assumption, namely that B has a singular value decomposition. We write

$$U_2^T BV_2 = D_2,$$

where D_2 has diagonal elements

$$\sigma_2 \geq \sigma_3 \geq .... \geq \sigma_p \geq 0.$$

Then

$$\begin{bmatrix} 1 & 0 \\ 0 & U_2^T \end{bmatrix} U^T AV \begin{bmatrix} 1 & 0 \\ 0 & V_2 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & U_2^T \end{bmatrix} A_1 \begin{bmatrix} 1 & 0 \\ 0 & V_2 \end{bmatrix}$$

$$= \begin{bmatrix} \sigma_1 & 0 \\ 0 & D_2 \end{bmatrix}.$$

Let U be replaced by the orthogonal matrix

$$U \begin{bmatrix} 1 & 0 \\ 0 & U_2 \end{bmatrix}$$

and V by the orthogonal matrix

$$V \begin{bmatrix} 1 & 0 \\ 0 & V_2 \end{bmatrix}.$$

Then we have

$$U^T AV = D.$$

It remains to show that $\sigma_1 \geq \sigma_2$. We have

$$\sigma_1 = \|A\| = \|U^T AV\| = \|D\| \geq \|Dz\| = \sigma_2,$$

where

$$z = \begin{bmatrix} 0 \\ 1 \\ 0 \\ .. \\ 0 \end{bmatrix}.$$

This completes the proof.

In frequent applications of the **svd**, there are more rows than columns of A. When $m \geq n$, there is a *compact* form of the **svd**. For example, suppose

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} = UDV^T = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \end{bmatrix} V^T.$$

The last column of U and the last row of D have no function and can be deleted. Thus

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} = U_c S V^T = \begin{bmatrix} u_1 & u_2 \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} V^T.$$

If $m \geq n$ then the **svd** may be written in *compact* form as

$$A = USV^T,$$

where S is an n by n matrix obtained from D by deleting the $m - n$ bottom rows, which are zero. U is an m by n matrix, which is obtained by deleting the last $m - n$ columns of the full version of U. V is the same n by n matrix in both full and *compact* versions. The full version may be restored by expanding U to a m by m orthogonal matrix and adding zero rows to S to produce D.

There is a similar *compact* form when $m < n$. Then V becomes non-square.

84.1 Eigenvalues

Let A have singular value decomposition

$$A = USV^T.$$

Then the eigenvalues of

$$M = A^T A,$$

are the squares of the singular values.

We have

$$A^T A = VSU^T USV^T = VS^2 V^T.$$

Or

$$A^T AV = VS^2.$$

Or

$$A^T A \begin{bmatrix} V_1 & V_2 & ...V_n \end{bmatrix} = \begin{bmatrix} \sigma_1^2 V_1 & \sigma_2^2 V_2 & ...\sigma_n^2 V_n \end{bmatrix}.$$

So the columns of V are the eigenvectors of $A^T A$.

84.2 Range and Kernel

We shall assume that $m \geq n$ and use the *compact* form of the **SVD**. Let A be a m row by n column matrix. A Singular Value Decomposition (SVD) of A consists of matrices U, S, and V so that

$$A = USV^T.$$

S is an n by n diagonal matrix. U is an m by n matrix, which has orthonormal columns. V is n by n orthogonal matrix. Let u_j be the jth column vector of U, s_j the jth diagonal element of S, and v_j the jth column vector of V.

Proposition. The range of A is the span R of

$$\{u_j : j = 1..n, s_j \neq 0\}).$$

Proof. Let

$$y = SV^T x = \begin{bmatrix} y_1 \\ . \\ . \\ y_n \end{bmatrix}.$$

So clearly the range of A is a subset of R. On the other hand, if s_i is not zero, then because V is orthogonal, V^T is its inverse. So the equation

$$V^T x_i = \frac{1}{s_i} \begin{bmatrix} \delta_{1i} \\ \delta_{2i} \\ . \\ . \\ \delta_{ni} \end{bmatrix},$$

has a unique solution. The solution maps to u_i. Thus R is a subset of the range of A.

Proposition. The kernel of A is the span K of the set

$$\{v_j : j = 1..n, s_j = 0\}).$$

Proof. The rank of U is n, so the kernel of A is the kernel of SV^T. Thus we shall consider the kernel of the latter transformation from $\Re^n$ to $\Re^n$. Let x be an arbitrary vector in the domain. The $v_j, j = 1..n$ are basis of the n dimensional space. So we may find constants so that

$$x = \alpha_1 v_1 + ... + \alpha_n v_n.$$

Then

$$y = SV^T x = S \begin{bmatrix} (v_1, x) \\ (v_2, x) \\ . \\ . \\ (v_n, x) \end{bmatrix} = \begin{bmatrix} s_1(v_1, x) \\ s_2(v_2, x) \\ . \\ . \\ s_n(v_n, x) \end{bmatrix} = \begin{bmatrix} s_1\alpha_1(v_1, v_1) \\ s_2\alpha_2(v_2, v_2) \\ . \\ . \\ s_n\alpha_n(v_n, v_n) \end{bmatrix}.$$

Thus $y = 0 \Leftrightarrow s_i\alpha_i = 0, \forall i \Leftrightarrow s_i = 0$ or $\alpha_i = 0, \forall i$. This proves the proposition.

Note. The singular value decomposition displays the kernel of a linear transformation. In particular if λ is an eigenvalue of A, then the corresponding eigenvector or eigenvectors form the kernel of

$$A - \lambda I,$$

and so the eigenspace can be computed using the SVD.

84.3 The Psuedoinverse

Define S^p to be a diagonal matrix in the following way. If $s_i \neq 0$ let the ith diangonal element of S^p be $1/s_i$, otherwise let it be zero. The psuedoinverse of the matrix A is defined as

$$A^P = VS^pU^T.$$

If A is square then both U and V are square orthogonal matrices whose inverses are their transposes. They are isometries. If A is a square nonsingular matrix, then the psuedoinverse is the inverse.

Suppose A is square, but singular. Consider the equation

$$Ax = b.$$

This equation may not have a solution.
Proposition. Suppose that

$$Ax = b$$

has a solution. Then $x_1 = A^p b$ is a solution of smallest magnitude.
Proof. (see 1/18/89, and 2/20/91) Let

$$b = \sum_{i=1}^{n} \alpha_i u_i = U\alpha.$$

then

$$Ax_1 = AA^p b =$$

$$USV^T VS^p U^T b = USS^p U^T U\alpha = USS^p \alpha = U\alpha = b.$$

We have

$$SS^p \alpha = \alpha,$$

because $s_i = 0$ if and only if $\alpha_i = 0$.

To prove the minimum property, suppose $x_2 = x^* + x_1$ is a second solution. Then $x^* \in \mathrm{Ker}(A)$. Suppose

$$x^* = \sum_{i=1}^{n} \beta_i v_i = V\beta.$$

We have

$$V^T x^* = \beta,$$

and β_i is not zero iff s_i is zero, which shows that β is perpendicular to the product of S^p with any vector. We have

$$\|x_2\| = \|x_1 + x^*\| = \|V(S^p U^T + V^T x^*)\|$$

$$= \|S^p U^T + V^T x^*\| = \|S^p U^T + \beta\|.$$

The third equality holds because V is an isometry. We know that $S^p U^T$ is perpendicular to β, so we can apply the Pythagorean theorem.

$$\|x_2\|^2 = \|S^p U^T\|^2 + \|\beta\|^2 \geq \|S^p U^T\|^2$$

$$= \|VS^pU^T\|^2 = \|x_1\|^2.$$

This completes the proof.

Proposition. Suppose that A is an n by n matrix and that the equation

$$Ax = b$$

does not have a solution. Let $x = A^p b$, then

$$\|Ax - b\|$$

is a minimum.

Proof.

To prove the minimum property, suppose x_2 is a second vector in the domain of A. Let $Ax_2 = Ax_1 + b^*$. Let

$$b^* = \sum_{i=1}^{n} \beta_i u_i = U\beta.$$

We have

$$U^T b^* = U^T U\beta = \beta.$$

β_i is not zero iff s_i is zero, which shows that β is perpendicular to the product of S^p with any vector. We have

$$\|Ax_2 - b\| = \|Ax_1 - b + b^*\|$$

$$= \|USV^TVS^pU^Tb - b + b^*\|$$

$$= \|USS^pU^Tb - b + b^*\| = \|U(SS^pU^Tb - U^Tb + U^Tb^*)\|$$

$$= \|(SS^p - I)U^Tb + U^Tb^*\| = \|(SS^p - I)U^Tb + \beta\|$$

From the theorem on the range of A, β_i is zero if s_i is zero. But if the ith element on the diagonal of $SS^p - I$ is not zero, then s_i is zero, and so β_i is zero. Thus $(SS^p - I)U^Tb$ is perpendicular to β, so we can apply the Pythagorean theorem.

$$\|Ax_2 - b\|^2 = \|(SS^p - I)U^Tb\|^2 + \|\beta\|^2 \geq \|(SS^p - I)U^Tb\|^2$$

$$= \|U(SS^p - I)U^Tb\|^2$$

$$= \|Ax_1 - b\|^2.$$

This completes the proof.

We can extend this result to the m by n case.

Proposition. Suppose that A is an m by n matrix, with $m > n$, and suppose that the equation

$$Ax = b$$

does not have a solution. Let $x = A^p b$, then

$$\|Ax - b\|$$

is a minimum.

Proof. Extend A, U, S, and V to m by m matrices. Let

$$\bar{A} = \begin{bmatrix} A & 0 \end{bmatrix},$$

$$\bar{U} = \begin{bmatrix} U & U_2 \end{bmatrix},$$

$$\bar{S} = \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix},$$

and

$$\bar{V}^T = \begin{bmatrix} V^T & 0 \\ 0 & V_2^T \end{bmatrix},$$

so that $\bar{V}$ and $\bar{U}$ are orthogonal. Then

$$\bar{A} = \bar{U}\bar{S}\bar{V}^T = \begin{bmatrix} U & U_2 \end{bmatrix} \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V^T & 0 \\ 0 & V_2^T \end{bmatrix} =$$

$$\begin{bmatrix} U & U_2 \end{bmatrix} \begin{bmatrix} V^T & 0 \\ 0 & V_2^T \end{bmatrix} = \begin{bmatrix} USV^T & 0 \end{bmatrix} = \begin{bmatrix} A & 0 \end{bmatrix}.$$

Let

$$\begin{aligned} \bar{x} &= \bar{V}\bar{S}^p\bar{U}^T b \\ &= \begin{bmatrix} V & 0 \\ 0 & V_2 \end{bmatrix} \begin{bmatrix} S^p & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} U^T \\ U_2^T \end{bmatrix} b \\ &= \begin{bmatrix} VS^pU^T \\ 0 \end{bmatrix} b \\ &= \begin{bmatrix} VS^pU^Tb \\ 0 \end{bmatrix}. \end{aligned}$$

Then

$$\|\bar{A}\bar{x} - \bar{b}\|$$

is minimized. But for all

$$\bar{y} = \begin{bmatrix} y \\ y_2 \end{bmatrix},$$

$$\bar{A}\bar{y} = Ay.$$

So $x = VS^pU^Tb$ minimizes

$$\|Ax - b\|.$$

This completes the proof.

84.4 Application to Fitting a Straight Line in Space

(See the NIST Algorithm Testing System) Given: n points p_i, a point on a line a, and a vector t in the direction of the line, where

$$a = \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

Let

$$q_i = p_i - a.$$

The square of the distance from the point p_i to the line is

$$d_i^2 = \|q_i\|^2 - \frac{(q_i, t)^2}{\|t\|^2}$$

$$= q_i^T q_i - \frac{t^T q_i q_i^T t}{t^T t}.$$

The sum of the square distances of the n points is

$$S(a,t) = \sum_{i=1}^{n} [q_i^T q_i - \frac{t^T q_i q_i^T t}{t^T t}].$$

$$= \sum_{i=1}^{n} q_i^T q_i - \frac{t^T M t}{t^T t},$$

where

$$M = \sum_{i=1}^{n} q_i q_i^T$$

$$= \begin{bmatrix} q_1 & q_2 & \dots & q_n \end{bmatrix} \begin{bmatrix} q_1^T \\ q_2^T \\ \dots \\ q_n^T \end{bmatrix}$$

$$= H^T H.$$

M is symmetric. H will be used later. We may write S in a different form:

$$S(a,t) = \sum_{i=1}^{n} [q_i^T q_i - \frac{q_i^T t t^T q_i}{t^T t}]$$

$$= \sum_{i=1}^{n} (q_i, \frac{I - tt^T}{t^T t} q_i)$$

$$= \sum_{i=1}^{n} (q_i, Bq_i),$$

where

$$B = I - \frac{tt^T}{t^T t}.$$

B is a symmetric matrix.

Differentiating with respect to x

$$\frac{dS}{dx} = \sum_{i=1}^{n} [(\frac{dq_i}{dx}, Bq_i) + (q_i, B\frac{dq_i}{dx})]$$

$$= \sum_{i=1}^{n} 2(B\frac{dq_i}{dx}, q_i).$$

We have

$$\frac{dq_i}{dx} = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}.$$

Let B_1 be the first column of B, which is also the first row of B (because B is symmetric). Then

$$B\frac{dq_i}{dx} = -B_1.$$

So we get

$$\frac{dS}{dx} = -\sum_{i=1}^{n} 2(B_1, q_i)$$

$$= -\sum_{i=1}^{n} 2B_1^T q_i$$

$$= -2B_1^T \sum_{i=1}^{n} q_i.$$

Similarly

$$\frac{dS}{dy} = -2B_2^T \sum_{i=1}^{n} q_i,$$

and

$$\frac{dS}{dz} = -2B_3^T \sum_{i=1}^{n} q_i.$$

These three scalar equations combine to the vector equation

$$\nabla S = -2B \sum_{i=1}^{n} q_i.$$

The minimum occurs where

$$\nabla S = 0,$$

A solution is

$$\sum_{i=1}^{n} q_i = 0,$$

or

$$a = \frac{\sum_{i=1}^{n} p_i}{n}.$$

That is, a is the centroid of the data points.

S is minimized when

$$\frac{t^T M t}{t^T t}$$

is maximized. This is a quadratic form evaluated at a unit vector. We have

$$Q(t/\|t\|) = \frac{t^T M t}{t^T t}.$$

The maximum value, taken over all unit vectors, occurs at the largest eigenvalue of M.

From above

$$M = H^T H,$$

where H is the data matrix,

$$H = \begin{bmatrix} q_1^T \\ q_2^T \\ \dots \\ q_n^T \end{bmatrix}.$$

We find the singular value decomposition of H,

$$H = USV^T.$$

In a previous section we showed that the largest eigenvalue of M is σ_1^2 and the first column V, which we write as V_1, is the corresponding eigenvector and hence the direction of the line. That is

$$t = V_1.$$

Here is an example of the calculation given as a matlab program:

```
% lsql.m matlab program to compute least squares 9/10/92
% line in space.
diary
% data points (x,y,z):
h=[1    .99   1;
   2    1.9  2.1;
   3.1  2.99 3.01;
   3.99 4.1  3.97];
% n is the number of data points
n=4;
% compute centroid of data points
a=[0 0 0 ];
for i=1:n;
a=a+h(i,:);
end;
a=a/n;
% subtract centroid from data
for i=1:n;
h(i,:)=h(i,:)-a;
end;
% find singular value decomposition of data matrix h
```

```
[u,s,v]=svd(h)
% the centroid is on the lsq line
point=a
% the line direction is the eigenvector corresponding to
% the largest eigenvalue of h'*h
t=v(:,1);
direction=t
% compute the rms distance to the line
d2=-s(1,1)^2;
for i=1:n;
d2=d2+h(i,:)*h(i,:)';
end;
rmsdistancetoline=sqrt(abs(d2))
```

Here is the output when the program is run:

```
u =
   -0.6705    0.5456   -0.0526    0.5000
   -0.2275   -0.6926    0.4674    0.5000
    0.2304   -0.2519   -0.7959    0.5000
    0.6676    0.3989    0.3811    0.5000
s =
    3.9143         0         0
         0    0.1384         0
         0         0    0.0816
         0         0         0
v =
    0.5754   -0.2086   -0.7908
    0.5952    0.7699    0.2300
    0.5609   -0.6030    0.5672
point =
    2.5225    2.4950    2.5200
direction =
    0.5754
    0.5952
    0.5609
rmsdistancetoline =

    0.1606
```

```
863 flops.
```

84.5 Linear Least Squares Using SVD

The **svd** improves the roundoff error over that of the direct normal equation solution.

85 Solving Partial Differential Equations

Partial differential equations can be classed as linear and nonlinear, and further as elliptic, parabolic, or hyperbolic. Solution methods are vastly different for different types of equations. Elliptic equations are the the most simple. And the most simple of the elliptic equations is Laplace's equation. Laplace's equation is very common in physics. We shall discuss Laplace's equation in the next sections.

86 An Introduction to Laplace's Equation in Physics

86.1 Introduction

The general theory of Laplace's equation is called potential theory.

86.2 Gravity

The force on a particle of mass m by a particle of mass M which is located at the origin, is

$$F = -G\frac{nM\hat{r}}{r^2}$$

G is the gravitational constant, r is the distance between the particles and $\hat{r}$ is a unit vector directed from M to m. The mass M establishes a force field, which we call H. We write the force field as

$$H = -G\frac{M\hat{r}}{r^2}.$$

When there is a distribution of mass particles M_i, the force field is the vector sum of the fields of each particle. The force on a particle of mass m by a gravitational field H is

$$F = mH.$$

86.3 Gauss's Law

Referring to the figure, by similarity of the triangles, the area element ds' and the projection of ds, which is $ds\cos(\theta)$, have the ratio

$$\frac{\hat{r}\cdot ds}{ds'} = r^2.$$

So

$$\frac{\hat{r}\cdot ds}{r^2} = ds'.$$

Since ds' is an area element on the unit sphere S', we have

$$\int_S \frac{\hat{r}\cdot ds}{r^2}\int_{S'} ds' = 4\pi.$$

The surface boundary of a volume A, is written ∂A. Therefore

$$\int_{\partial A} H\cdot ds = -\int_{\partial A}\frac{GM\hat{r}\cdot ds}{r^2} = -GM4\pi.$$

For the case of a distributed mass, with density function ρ, each mass element $dm = \rho dv$ produces a contribution to the field dH, where dv is the volume element. Then

$$\int_{\partial A} dH\cdot ds = -G\rho 4\pi dv.$$

When we integrate, we get

$$\int_{\partial A} H\cdot ds = -4\pi G\int_A \rho dv.$$

This is the integral form of Gauss's law. It says that the surface integral of the field is equal to $-4\pi G$ times the amount of mass inside the surface. In the next section we shall introduce the concept of the divergence, $\nabla\cdot H$, of a vector field. Then the differential form of Gauss's law is

$$\nabla\cdot H = -4G\pi\rho.$$

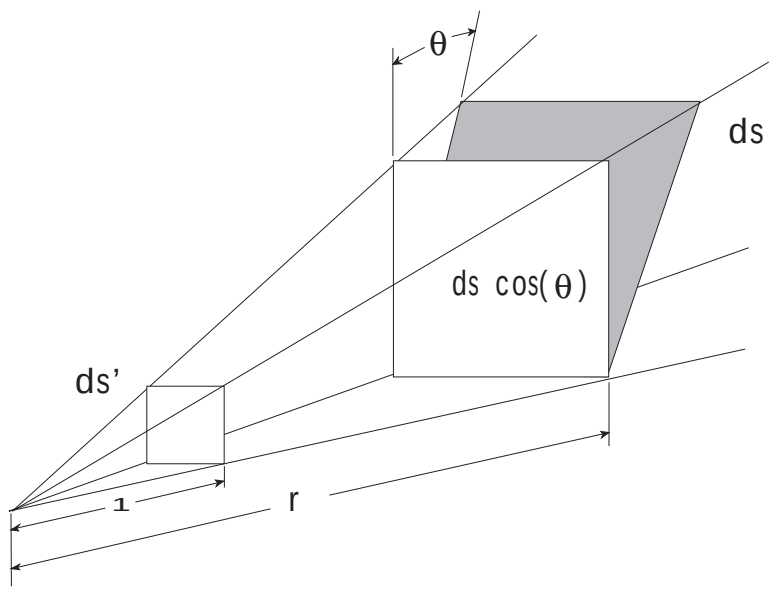


Figure 15: Proof of Gauss' Law.

86.4 Curl, Divergence, Gradient

The curl of a vector field **F** in cartesian coordinates is

$$\nabla \times F = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ F_x & F_y & F_z \end{vmatrix}$$

$$\nabla \times \mathbf{A} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \partial/\partial x & \partial/\partial y & \partial/\partial z \\ A_x & A_y & A_z \end{vmatrix}$$

$$= \left(\frac{\partial A_z}{\partial y} - \frac{\partial A_y}{\partial z}\right)\mathbf{i}$$

$$- \left(\frac{\partial A_z}{\partial x} - \frac{\partial A_x}{\partial z}\right)\mathbf{j}$$

$$\left(\frac{\partial A_y}{\partial x} - \frac{\partial A_x}{\partial y}\right)\mathbf{k}.$$

Divergence theorem. Stokes theorem. Directional derivative.

86.5 Potential

$$H = -\nabla V$$

$$\nabla \cdot H = -4\pi G\rho$$

Substituting get Laplaces equation

$$\nabla^2 V = 0.$$

86.6 The Heat Equation

Let the heat flux vector be Φ, and the temperature T. The Fourier equation is

$$\Phi = -k\nabla T.$$

Φ has units like calories per second per square meter. We shall use the first law of thermodynamics

$$\Delta Q = \Delta W + \Delta u.$$

Let c be the specific heat with units like calories per Kg per degree. So if m is the mass, the change in internal energy Δu for a change in temperature ΔT is $c\Delta T m$. Assuming zero work, the heat flow into the volume is

$$\Delta Q = c\Delta T m.$$

The continuity equation, which says that the rate of increase of internal energy (increase in heat) is equal to the flow of heat through the surface, is

$$\nabla \cdot \Phi = -c\rho\frac{\partial T}{\partial t} = -\frac{\partial \psi}{\partial t}.$$

Substituting for Φ, we have the heat equation

$$\nabla \cdot k\nabla T = c\rho\frac{\partial T}{\partial t}.$$

When k is constant, under steady state conditions,

$$\frac{\partial T}{\partial t} = 0,$$

this reduces to Laplace's equation

$$\nabla^2 T = 0.$$

86.7 Electric Potential

The electric field has zero curl, hence there exists a potential function ϕ such that

$$E = -\nabla\phi.$$

In places where there is no charge we have

$$\nabla \cdot E = 0,$$

Therefore the potential satisfies Laplace's equation

$$\nabla^2\phi = \nabla \cdot \nabla\phi = -\nabla \cdot E = 0.$$

86.8 Electric Current Flow

For electric current flow, we have Ohm's law,

$$J = \sigma E = \sigma\nabla\phi.$$

This is analogous to the Fourier law for heat conduction. Also there is a continuity equation for charge conservation, that is the change in charge in a volume is equal to the rate of flow of charge through the bounding surface. This also is completely analogous to the flow of heat and so again the potential ϕ satisfies Laplaces equation. The reciprocal of the conductivity σ is the resistivity, which has units like ohm-meters.

86.9 Diffusion Equation

The equation for diffusion is the same as the heat equation so again we get Laplace's equation in the steady state.

86.10 Analytic and Harmonic Functions

An analytic function satisfies the Cauchy-Riemann equations. Differentiating these two equations we find that the both the real and imaginary parts of the function satisfy Laplace's equation in two dimensions. Such functions are a major source for finding solutions to two dimensional boundary value problems.

86.11 Viscosity

Consider a fluid in a state of laminar flow. Let the rate of change of shear angle be

$$\frac{d\alpha}{dt}.$$

The coefficient of viscosity is defined as

$$\eta = \frac{\sigma_s}{d\alpha/dt},$$

where σ_s is the shear stress. The unit of viscosity is the Poise, which is one dyne second per square cm.

Suppose two nested coaxial cylinders are separated by a fluid. Let the outer cylinder be fixed and the inner cylinder rotated by a torque τ. Let the inner cylinder have radius r_1 and the outer cylinder radius r_2. Let the cylinder be rotating at angular velocity ω. Let the cylinders have height h. Then

$$\tau = r_1 F = r_1 2\pi r_1 h \sigma_s.$$

Now

$$\frac{d\alpha}{dt} = \frac{v}{r_2 - r_1} = \frac{r_1 \omega}{r_2 - r_1}.$$

Then

$$\tau = 2\pi r_1^2 h \eta \frac{d\alpha}{dt} = 2\pi r_1^3 h \eta \frac{\omega}{r_2 - r_1}.$$

If τ and ω are measured, then this equation may be solved for the viscosity. At about 20 degrees C, the viscosity of Benzene is .65 centipoise, and that of mercury is 1.55 cp. At 0 degrees the viscosity of water is 1.79 cp, and at 100 degrees it is .28 cp. The viscosity of heavy oil at 15 degrees is 660 cp. The viscosity of air is 171 μp at ..

86.12 Fluid Mechanics

We shall derive Euler's Equation (Applied Hydro and Aeromechanics). Let the velocity of a fluid be v. It depends upon the time t, and the distance along the streamline s. We write

$$v = e(s, t).$$

Let $s(t)$ be the position of a specific fluid particle. Then its velocity is

$$v = f(t) = e(s(t), t).$$

Its acceleration is

$$\frac{df}{dt} = \frac{\partial e}{\partial s}\frac{ds}{dt} + \frac{\partial e}{\partial t} = \frac{\partial e}{\partial s}v + \frac{\partial e}{\partial t}.$$

Consider a small portion of a stream tube of length δs. The acceleration times the mass is equal to the net force.

$$\frac{df}{dt}\rho dA\delta s = g\rho dA\delta s\sin(\theta) + dA(p - (p + \frac{\partial p}{ds}\delta s),$$

where θ is the angle of incline of the stream tube. Then

$$\frac{\partial e}{\partial s}v + \frac{\partial e}{\partial t} = g\sin(\alpha) - \frac{1}{\rho}\frac{\partial p}{ds}.$$

This is the equation of Euler.

Under steady flow

$$\frac{\partial e}{\partial t} = 0$$

So

$$v\frac{\partial e}{\partial s} = dh - \frac{\partial p}{\partial s}\frac{1}{\rho}.$$

Integrating with time held constant, we find

$$v^2/2 + gh + p/\rho = C,$$

where C is a constant. This is Bernoulli's equation.

There are two approaches to observing fluid flow. One may watch the flow as it passes a fixed position. This is the Eulerian approach. Alternately one may follow a fluid particle, which is the Lagrangean approach. Thus Euler sits on the shore and watches Lagrange go by in a boat. In the Eulerian approach, the velocity at which the fluid flows past a fixed point x at time t is given by a function $e(x, t)$. Let us now look at the problem from the Lagrangian viewpoint. Suppose $(x(t), y(t), z(t))$ is the position of a moving fluid particle. Then the velocity of the particle is

$$v = f(t) = e(x(t), y(t), z(t), t).$$

The acceleration of the particle is

$$\frac{dv}{dt} = \frac{df}{dt} = \frac{\partial e}{\partial t} + \frac{\partial e}{\partial x}\frac{dx}{dt} + \frac{\partial e}{\partial y}\frac{dy}{dt} + \frac{\partial e}{\partial z}\frac{dz}{dt}$$

$$= \frac{\partial e}{\partial t} + v \cdot \nabla v.$$

This is called the material derivative of the velocity. The acceleration of the flow is

$$\frac{\partial e}{\partial t}.$$

The introduction of viscosity leads to the Navier-Stokes Equation. See Lass, "Elements of Pure and Applied Mathematics," for a mathematical treatment of the Navier-Stokes Equation.

Viscosity is a consequence of momentum transfer. Consider two trains passing one another at differing velocities. Suppose suitcases are thrown repeatedly back and forth between the trains. As a suitcase is passed from the faster train to the slower, the slower train experiences an impulse and a force and tends to increase its speed. When the opposite happens the fast train experiences a retarding force. No net transfer of mass occurs. Momentum is being exchanged. Viscosity may be considered also as the rate of change of shear strain. The coefficient of viscosity is a proportionality factor. The Navier-Stokes equation is:

$$\frac{\partial w}{\partial t} + \nabla w \cdot w = G - \frac{1}{\rho}\nabla p + \frac{\mu}{\rho}\nabla^2 w,$$

where G is a body force, such as gravity (See Prandtl and Tietjens, "Hydro and Aeromechanics"). The term

$$\nabla w \cdot w = \nabla v_x \cdot v + \nabla v_y \cdot v + \nabla v_z \cdot v,$$

is nonlinear. The Navier-Stokes equation is a nonlinear partial differential equation, for which there is no general existence theory.

86.13 Velocity Potential and Stream Function

Let

$$w = ui + vj$$

be a two dimensional velocity field. $w(x, y, t)$ is the velocity of a fluid particle at position (x, y) and time t. Suppose the fluid is incompressible, so that its density ρ is constant. The continuity equation is

$$\nabla(\rho w) = \frac{\partial \rho}{\partial t} = 0.$$

Hence

$$\nabla(\rho w) = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0.$$

Now consider the differential form

$$-v dx + u dy.$$

We have just shown that

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0.$$

So that

$$\frac{\partial u}{\partial x} = -\frac{\partial v}{\partial y}.$$

which means that the differential form is closed and so exact. It is exact so it is the differential of some function f so that

$$df = \frac{\partial f}{\partial x} dx + \frac{\partial f}{\partial y} dy = -v dx + u dy.$$

Thus

$$\frac{\partial f}{\partial x} = -v,$$

and

$$\frac{\partial f}{\partial y} = u.$$

We have shown this to be a consequence of the assumed incompressibility. We shall further restrict the fluid to be irrotational, which means it has zero curl. These two assumptions will lead to Laplace's equation. The curl of a vector field A, is defined as

$$\nabla \times A = \begin{vmatrix} i & j & k \\ \partial/\partial x & \partial/\partial y & \partial/\partial z \\ A_x & A_y & A_x \end{vmatrix}.$$

In two dimensional flow, the z component of velocity is zero, so the curl is

$$\nabla \times w = \left(\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \right) k.$$

Then the assumption of zero curl forces

$$\frac{\partial v}{\partial x} = \frac{\partial u}{\partial y}.$$

Combining this with the consequence of incompressibility,

$$\frac{\partial f}{\partial x} = -v,$$

and

$$\frac{\partial f}{\partial y} = u,$$

we get Laplace's equation

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0.$$

Let us consider why $\nabla \times w$ is called the curl. Stokes theorem is

$$\int_{\partial S} w \cdot d\ell = \int_S (\nabla \times w) \cdot ds,$$

where S is a surface element, and ∂S is the boundary curve of the surface element. Because we have assumed that $\nabla \times w = 0$, the line integral around the surface element is zero, that is

$$\int_{\partial S} w \cdot d\ell = 0.$$

If the fluid were to curl around some point, then the line integral following this circular flow would certainly not be zero. Hence $\nabla \times w = 0$ does imply that such curling of the fluid does not occur. Now given the harmonic function f, one can find a conjugate harmonic function g such that

$$f + ig,$$

is an analytic function. By the Cauchy-Riemann equations

$$w = -\nabla g.$$

Hence g is a velocity potential, because its negative gradient is the velocity. Now ∇f is perpendicular to w, so the lines of constant f are parallel to the velocity w. That is the lines are stream lines. Hence f is called the stream function. If we know the stream function on the boundary of a flow region, then f is a solution to the Laplace boundary value problem. From f, we may compute the conjugate function g, and then the velocity as the negative gradient of g. Assuming steady state flow, we may apply Bernoulli's equation and from the velocity deduce the pressure. In the case of an airfoil we may then deduce the lift by integrating the pressure.

86.14 Solution Methods

There are several ways to solve Laplace's equation and other partial differential equations of mathematical physics. Among the methods are separation of variables, variational methods, and numerical methods such as the finite difference method, and the finite element method.

86.15 Separation of Variables

It is frequently possible to write a solution as a product of functions, each function depending on only one variable. Then one finds ordinary differential equations for each function of the product. Then one solves the problem by finding solutions that fits the boundary conditions of the problem.

86.16 The Finite Difference Method

By using central difference approximations to the derivatives, one may reduce the problem to a finite difference equation on a grid.

$$\frac{\partial V}{\partial x} = \frac{V(x_n, y_m) - V(x_{n-1}, y_m)}{h},$$

and

$$\frac{\partial^2 V}{\partial x^2} = \left(\frac{\partial V(x_{n+1}, y_m)}{\partial x} - \frac{\partial V(x_{n-1}, y_m)}{\partial x} \right) \Big/ h.$$

Adding a similar expression for

$$\frac{\partial^2 V}{\partial x^2},$$

we get

$$\nabla^2 V = \frac{V(x_{n+1}, y_m) + V(x_{n-1}, y_m) + V(x_n, y_{m-1}) + V(x_n, y_{m+1}) - 4V(x_n, y_m)}{h^2}.$$

Hence setting

$$\nabla^2 V = 0,$$

and solving , we obtain

$$V(x_n, y_m) = \frac{V(x_{n+1}, y_m) + V(x_{n-1}, y_m) + V(x_n, y_{m-1}) + V(x_n, y_{m+1})}{4}.$$

This gives a sparse system of linear equations for the variables $V(x_n, y_m)$. The system is usually solved by iteration.

86.17 Variational Methods and the Finite Element Method

86.18 References for Laplace's Equation

87 A Review of Vector Analysis

87.1 Introduction

Vector Analysis is a classical subject dealing with those aspects of vectors which have application in applied mathematics and Physics. The Physicist J. Willard Gibbs is considered the founder of Vector Analysis. Linear Algebra is the study of finite dimensional vector spaces and has some relation to Vector Analysis. But Vector Analysis is Analysis and could have been called Vector Calculus.

87.2 The Inner Product

We shall prove the law of cosines. Suppose we have three points

$$p_0 = (0, 0), p_1 = (b, 0), p_2 = (x, y) = (a\cos(\theta), a\sin(\theta)).$$

These points form a triangle with sides p_0p_2, p_0p_1, p_2p_1. These sides have lengths a, b, c. The angle between side p_0p_1 and side p_0p_2 is θ. We have

$$c^2 = (x - b)^2 + y^2$$

$$= (x-b)^2 + a^2 - x^2$$
$$= x^2 - 2xb + b^2 + a^2 - x^2$$
$$= a^2 + b^2 - 2xb = a^2 + b^2 - 2ab\cos(\theta).$$

Thus we have the law of cosines, namely the square of the side opposite an angle of a triangle, is equal to the sum of the squares of the adjacent sides, minus two times the product of the sides and the cosine of the angle. That is,

$$c^2 = a^2 + b^2 - 2ab\cos(\theta).$$

The inner product (dot product) of two vectors, A and B, is defined as

$$A \cdot B = a_1b_1 + a_2b_2 + a_3b_3.$$

Then the dot product of a vector with itself is the square of its length. That is,

$$A \cdot A = a_1a_1 + a_2a_2 + a_3a_3 = \|A\|^2.$$

Let

$$C = B - A.$$

Then

$$\|C\|^2 = (B-A)\cdot(B-A)$$
$$= B\cdot B - B\cdot A - A\cdot B + A\cdot A$$
$$= \|B\|^2 - 2A\cdot B + \|A\|^2.$$

From which it follows that

$$2A\cdot B = \|A\| + \|B\|^2 - \|C\|^2.$$

But the right hand side is, by the law of cosines,

$$2\|A\|\|B\|\cos(\theta),$$

where θ is the angle between vectors A and B. Hence

$$A\cdot B = \|A\|\|B\|\cos(\theta).$$

Thus if the the dot product is zero, then the cosine is zero, and so the angle between the vectors is plus or minus $\pi/2$, and the vectors are perpendicular.

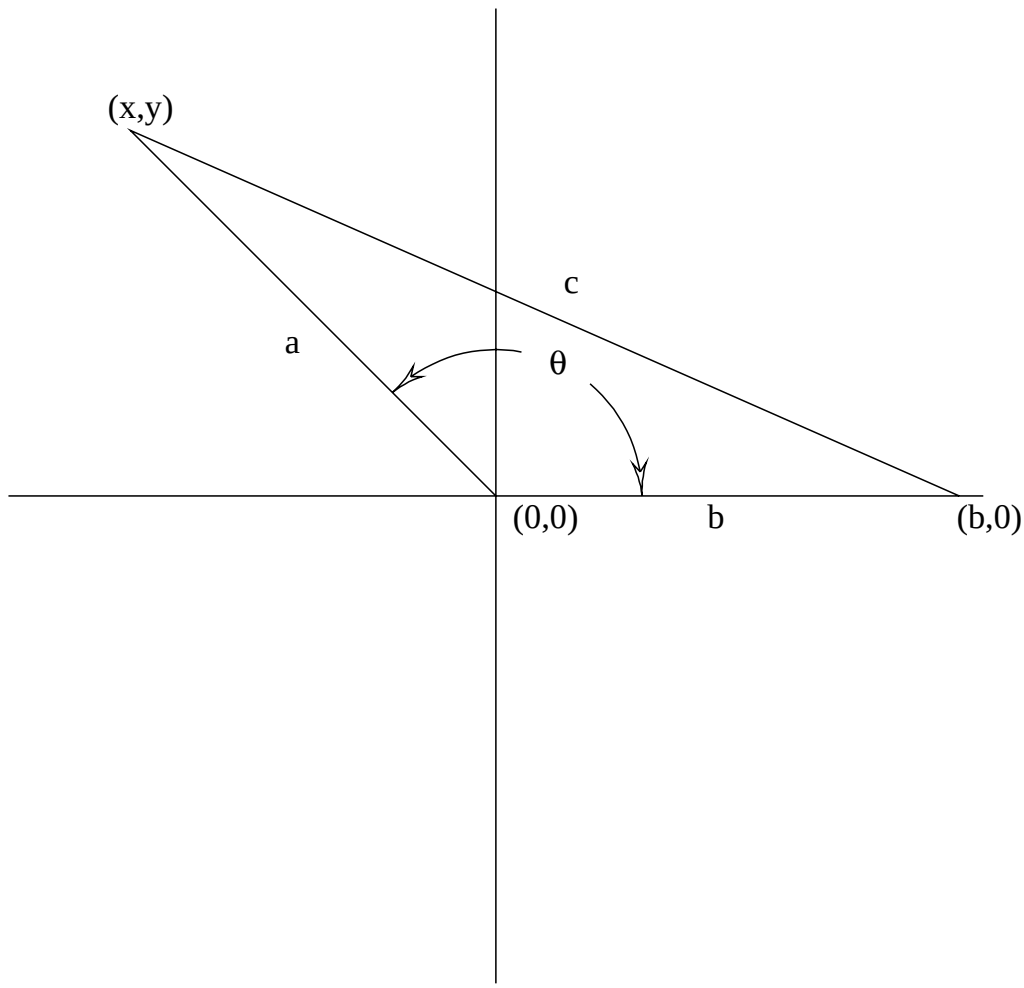


Figure 16: Derivation of the law of cosines. $x = \cos(\theta)$, $y = \sin(\theta)$. Computing c^2, we find that $c^2 = a^2 + b^2 - 2ab\cos(\theta)$.

87.3 The Vector Product

The vector product of two vectors A and B, (the cross product), is defined to be

$$A \times B = (a_2b_3 - a_3b_2)i + (a_3b_1 - a_1b_3)j + (a_1b_2 - a_2b_1)k,$$

where i, j, k are the unit coordinate vectors. This may be written as a determinant with i, j, k in the first row, the components of A in the second, and the components of B in the third row.

$$\mathbf{A} \times \mathbf{B} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

When the rows of a determinant are interchanged, the sign of the determinant changes, hence

$$A \times B = -B \times A.$$

Then

$$A \times A = -A \times A.$$

But this can be true only if

$$A \times A = 0.$$

We have shown that the vector product of any two parallel vectors is zero.

Given three vectors A, B, C, we see that

$$A \cdot (B \times C),$$

is given as the determinant that has rows A, B, and C. By interchanging these rows twice, we see that

$$A \cdot (B \times C) = (A \times B) \cdot C.$$

That is, in the scalar triple product, the dot and the cross may be interchanged. Now using this result, we see that

$$(A \times B) \cdot B = A \cdot (B \times B) = A \cdot 0 = 0.$$

Then $A \times B$ is perpendicular to B. Similarly it is perpendicular to A. Therefore we have shown that the vector product of two vectors is perpendicular

to each of them. This establishes the direction of the vector product, except possibly for sign. One may further establish the right hand rule. The direction of $A \times B$ is given by the right hand rule: Curl the fingers of your right hand from A to B, then $A \times B$ is in the direction of your thumb. One may verify directly that if V is a vector in the upper xy half plane that

$$i \times V$$

points in the positive z direction. This verifies the right hand rule in this case. One may also show the invariance of the cross product to a rigid motion, which establishes the right hand rule in general.

By direct computation one may verify that the vector triple product satisfies

$$A \times (B \times C) = B(A \cdot C) - C(A \cdot B).$$

This is the "Back Minus Cab Rule". We have established the direction of the cross product, now we shall find its magnitude. Let

$$C = A \times B.$$

Then

$$\begin{aligned}
\|C\|^2 &= C \cdot C \\
&= (A \times B) \cdot C \\
&= A \cdot (B \times C) \\
&= A \cdot (B \times (A \times B)) \\
&= A \cdot (A(B \cdot B) - B(B \cdot A) \\
&= (A \cdot A)(B \cdot B) - (A \cdot B)^2 \\
&= \|A\|^2\|B\|^2(1 - \cos^2(\theta)) = \|A\|^2\|B\|^2 \sin^2(\theta).
\end{aligned}$$

The magnitude of the cross product is the product of the lengths of the vectors, times the sine of the angle between them,

$$\|A \times B\| = \|A\|\|B\| \sin(\theta).$$

Example The equation of a plane. Let the plane have a unit normal vector N. Let $P = (x, y, z)$ be a point on the plane. Let d be the distance

from the origin to the plane. Then d is equal to the length of P times the cosine of the angle between P and the normal N. Hence

$$d = P \cdot N.$$

Therefore the equation of the plane is

$$P \cdot N - d = xn_1 + yn_2 + zn_3 - d = 0.$$

Suppose we are given three points P_1, P_2, P_3 and we wish to find the equation of the plane passing through these points. The normal to the plane is perpendicular to each of $P_2 - P_1$ and $P_3 - P_1$. Therefore

$$N = \frac{(P_2 - P_1) \times (P_3 - P_1)}{\|(P_2 - P_1) \times (P_3 - P_1)\|}$$

Also d is equal to the inner product of N with any one of the three points. For example

$$d = P_1 \cdot N.$$

Then the equation of the plane is

$$P \cdot N - P_1 \cdot N = xn_1 + yn_2 + zn_3 - d = 0.$$

87.4 Heron's Formula for the Area of a Triangle

Let T be the area of a triangle with sides given by vectors A, B, and C, and corresponding side lengths a, b and c. The area is one half of the magnitude of the cross product of the vectors A and B. That is,

$$2T = \|A \times B\|.$$

So

$$4T^2 = a^2b^2 \sin^2(\theta) = a^2b^2(1 - \cos^2(\theta)) = a^2b^2 - \|A \cdot B\|^2.$$

Also

$$c^2 = \|C\|^2 = \|A - B\|^2 = (A - B) \cdot (A - B) = a^2 - 2A \cdot B + b^2.$$

Then

$$\|A \cdot B\|^2 = \frac{(c^2 - (a^2 + b^2))^2}{4}.$$

Substituting this into the equation that we found above, namely

$$4T^2 = a^2b^2 - \|A \cdot B\|,$$

we get

$$\begin{aligned}
16T^2 &= 4a^2b^2 - (a^2 + b^2 - c^2)^2 \\
&= [2ab - (a^2 + b^2 - c^2)][2ab + (a^2 + b^2 - c^2)] \\
&= [c^2 - (a - b)^2][(a + b)^2 - c^2] \\
&= [c - (a - b)][c + (a + b)][a + b - c][a + b + c] \\
&= [c + b - a][c + a - b][a + b - c][a + b + c] \\
&= [a + b + c - 2a][a + b + c + -2b][a + b + c - 2c][a + b + c].
\end{aligned}$$

Dividing each product on the right by 2, we get

$$T^2 = (s - a)(s - b)(s - c)s,$$

where

$$s = \frac{a + b + c}{2},$$

is the half perimeter of the triangle. Taking the square root, we get Heron's formula,

$$T = \sqrt{(s - a)(s - b)(s - c)s}.$$

This derivation is suggested in a problem in Apostol's Calculus.

87.5 Curl, Divergence, Gradient

The curl of a vector field $\mathbf{A}$ in cartesian coordinates is

$$\begin{aligned}
\nabla \times \mathbf{A} &= \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \partial/\partial x & \partial/\partial y & \partial/\partial z \\ A_x & A_y & A_z \end{vmatrix} \\
&= \left(\frac{\partial A_z}{\partial y} - \frac{\partial A_y}{\partial z}\right)\mathbf{i} \\
&\quad - \left(\frac{\partial A_z}{\partial x} - \frac{\partial A_x}{\partial z}\right)\mathbf{j}
\end{aligned}$$

$$\left(\frac{\partial A_y}{\partial x} - \frac{\partial A_x}{\partial y}\right)\mathbf{k}.$$

The divergence of a vector field $\mathbf{A}$ in cartesian coordinates is

$$\nabla \cdot \mathbf{A} = \frac{\partial A_x}{\partial x} + \frac{\partial A_y}{\partial y} + \frac{\partial A_z}{\partial z}$$

If a surface S has bounding curve ∂S, Stokes theorem is

$$\int_S \nabla \times \mathbf{A} \cdot \mathbf{n} ds = \int_{\partial S} \mathbf{A} \cdot d\mathbf{r},$$

which allows a surface integral to be evaluated as a line integral around the boundary of the surface. The surface normal is $\mathbf{n}$.

The divergence theorem allows a volume integral to be evaluated as a surface integral. Let V be a volume and ∂V be it enclosing surface. Then

$$\int_V \nabla \cdot \mathbf{A} dv = \int_{\partial V} \mathbf{A} \cdot \mathbf{n} ds.$$

Green's Theorem in the Plane.

If S is an area in the x, y plane then a special case of Stokes Theorem gives

$$\begin{aligned}
&\int_S \left(\frac{\partial A_y}{\partial x} - \frac{\partial A_x}{\partial y}\right) ds \\
&= \int_{\partial S} \mathbf{A} \cdot d\mathbf{r} \\
&= \int_{\partial S} (A_x dx + A_y dy) \\
&= \int \mathbf{r} \cdot d\mathbf{r},
\end{aligned}$$

where

$$\mathbf{r}(t) = A_x(t)\mathbf{i} + A_y(t)\mathbf{j}$$

is the boundary curve bounding this area A. This is called Green's Theorem in the Plane.

The Area. As an application of Green's Theorem we can find the area enclosed by a curve by evaluating a line integral around the curve. So let $A_x = -y/2$ and $A_y = x/2$, then

$$\int_S ds =$$

$$\int_S \left(\frac{\partial A_y}{\partial x} - \frac{\partial A_x}{\partial y} \right) ds$$

$$= \int_{\partial S} (A_x dx + A_y dy).$$

$$= (1/2) \int_{\partial S} (-y dx + x dy).$$

Area Example. Let the region S be bounded by the curve

$$\mathbf{r} = r\cos(t)\mathbf{i} + r\sin(t)\mathbf{j},$$

for

$$0 \leq t \leq 2\pi.$$

Then

$$dx = -r\sin(t)dt$$

$$dy = r\cos(t)dt.$$

Then the area α is

$$\alpha = (1/2) \int_{\partial S} (-y dx + x dy).$$

$$= \frac{r^2}{2} \int_0^{2\pi} (\sin^2(t) + \cos^2(t))dt = \frac{r^2}{2} 2\pi = r^2\pi.$$

The Center of Mass . Now let us find the x coordinate of the center of mass of a region S bounded by the curve $\mathbf{r}(t)$. Let

$$A_x = 0,$$

and

$$A_y = \frac{x^2}{2}.$$

Then

$$\int_S x ds$$

$$= \int_S \left(\frac{\partial A_y}{\partial x} - \frac{\partial A_x}{\partial y} \right) ds$$

$$= \int_{\partial S} (A_x dx + A_y dy)$$

$$= \int_{\partial S} \frac{x^2}{2} dy.$$

So the center of mass x coordinate is

$$x_{cm} = \frac{1}{\alpha} \int_{\partial S} \frac{x^2}{2} dy,$$

where α is the area of region S. Similarly using

$$A_x = -\frac{y^2}{2},$$

and

$$A_y = 0,$$

we find

$$y_{cm} = -\frac{1}{\alpha} \int_{\partial S} \frac{y^2}{2} dx.$$

A Center of Mass Example . Let the area be the right half circle of radius r. Let the area be bounded by the curve

$$\mathbf{r} = r\cos(t)\mathbf{i} + r\sin(t)\mathbf{j},$$

for

$$-\pi/2 \leq t \leq 2\pi,$$

and by the straight line from $(0, r)$ to $(0, -r)$.

Now

$$dy = r\cos(t)dt,$$

So

$$\int_{\partial S} \frac{x^2}{2} dy$$

$$= \int_{-\pi/2}^{\pi/2} \frac{r^3}{2} \cos^3(t)dt + \int_{-r}^{r} 0dy = \frac{2}{3}r^3.$$

Hence the x coordinate of the center of mass is

$$x_{cg} = \frac{(2r^3/3)}{\pi r^2/2} = \frac{4r}{3\pi}$$

The Area Moment of Inertia . Letting

$$A_x = 0$$

and

$$A_y = x^3/3$$

We get for the moment of inertia about the y axis

$$I_y = \int_S x^2 ds$$

$$= \int_S \frac{\partial A_y}{\partial x} - \frac{\partial A_x}{\partial y} ds$$

$$= \int_{\partial S} A_x dx + A_y dy$$

$$= \int_{\partial S} (x^3/3) dy$$

Moment of Inertia Example . Let the area be a circle of radius r. Let the area be bounded by the curve

$$\mathbf{r} = r\cos(t)\mathbf{i} + r\sin(t)\mathbf{j},$$

for

$$0 \leq t \leq 2\pi.$$

Then

$$x = (r\cos(t))^3$$

and

$$dy = rcos(t)dt$$

So

$$I_y = r^4 \int_0^{2\pi} \cos^4(t)/3dt$$

$$= \frac{\pi r^4}{4}.$$

By the parallel axis theorem, the moment of inertia about an axis through the center of mass may be obtained from the moment of inertia about a parallel axis at a distance d from the center of mass axis.

These formulas allow us to compute areas, centers of mass, and moments of inertia for areas bounded by piecewise defined curves.

87.6 The Parallel Axis Theorem

The moment of inertia about an axis that does not pass through the center of mass of a body is equal to the moment of inertia about a parallel axis that passes through the center of gravity plus d^2M where d is the distance between the two parallel axes, and M is the mass of the body. Without loss of generality we shall assume that the parallel axes are parallel to the x axis and that the center of gravity of the body lies at $(0, y_0, z_0)$, so that $\sqrt{y_0^2 + z_0^2} = d$. We introduce a center of mass coordinate system

$$x' = x, y' = y - y_0, z' = z - z_0$$

Let the moment of inertia of the body about its center of mass be

$$I_{xxcm} = \int\int (y'^2 + z'^2) dm$$

Then the moment of inertia in the unprimed system is

$$I_{xx} = \int\int (y^2 + z^2) dm$$

$$= \int\int ((y' + y_0)^2 + (z' + z_0)^2) dm$$

$$= I_{xxcm} + \int\int 2y'y_0 dm + \int\int 2z'z_0 dm + \int\int (y_0^2 + z_0^2) dm$$

$$= I_{xxcm} + 0 + 0 + d^2M = I_{xxcm} + d^2M$$

87.7 Curvilinear Coordinates

Let u_1, u_2, u_3 be a system of curvilinear orthogonal coordinates in Euclidean three space. Let t be a tangent vector (such as a velocity). Then it may be written as a linear combination of basis vectors, which we shall define. We have

$$t = c^1 \partial/\partial u_1 + c^2 \partial/\partial u_2 + c^3 \partial/\partial u_3,$$

where the differential operators play the role of basis vectors. Let $<, >$ be the Euclidean inner product. We may consider $\partial/\partial u_i$ to be the tangent vector to the ith coordinate curve. Thus if $C_i(u_i)$ is the ith coordinate curve, then we identify the differential operator

$$\partial/\partial u_i$$

with the curve tangent vector

$$dC_i/du_i.$$

(Actually this is a special case of the natural isomorphism between curve tangent vectors and the linear functionals defined by the curves, which are called derivations. Thus if $\alpha(u)$ is a curve and f a function, then

$$\frac{df(\alpha)}{dt}(t_0),$$

maps f to a real number. And this gives the same value for any other curve that has the same tangent.)

Example: Let x_1, x_2, x_3 be the usual Euclidean coordinates. Let the first coordinate curve be

$$C_1(x_1) = x_1\mathbf{i} + x_2\mathbf{j} + x_3\mathbf{k},$$

where x_2 and x_3 are held fixed. Then

$$\partial/\partial x_1 = \partial C_1/\partial x_1 = \mathbf{i}$$

Example: Spherical coordinates r, θ, ϕ. We have

$$x = r\sin(\theta)\cos(\phi)$$

$$y = r\sin(\theta)\sin(\phi)$$

$$z = r\cos(\theta)$$

Define coordinate curve $C(r)$ by holding θ and ϕ fixed. We find that

$$\partial/\partial r = \sin(\theta)\cos(\phi)\mathbf{i} + \sin(\theta)\sin(\phi)\mathbf{j} + \cos(\theta)\mathbf{k}$$

This is a unit vector. In general coordinate tangent vectors are not unit vectors. We wish to work with unit vectors, so we define unit vectors $\mathbf{a}_i$ in the direction of $\partial/\partial u_i$. Then the unit vectors in spherical coordinates are

$$\mathbf{a}_r = \sin(\theta)\cos(\phi)\mathbf{i} + \sin(\theta)\sin(\phi)\mathbf{j} + \cos(\theta)\mathbf{k}$$

$$\mathbf{a}_\theta = \cos(\theta)\cos(\phi)\mathbf{i} + \cos(\theta)\sin(\phi)\mathbf{j} - \sin(\theta)\mathbf{k}$$

$$\mathbf{a}_\phi = -\sin(\phi)\mathbf{i} + \cos(\phi)\mathbf{j}$$

The length of the tangent vector t is

$$ds^2 =< t, t >= \sum_{i=1}^{3}\sum_{j=1}^{3} c^i c^j < \partial/\partial u_i, \partial/\partial u_j >$$

Assuming an orthogonal system

$$ds^2 =< t, t >= \sum_{i=1}^{3} (c^i)^2 < \partial/\partial u_i, \partial/\partial u_i > .$$

Define dual vectors $d\mathbf{u}_i$ by

$$d\mathbf{u}_i(\partial/\partial u_j) = \delta^i_j$$

(δ^i_j equals 1 if $i = j$ and zero otherwise). Then $d\mathbf{u}_i(t) = c^i$. In the old days, they did not carefully distinguish a function from its value, and wrote $du_i = d\mathbf{u}_i(t)$. Then $c_i = du_i$, so

$$t = du_1\partial/\partial u_1 + du_2\partial/\partial u_2 + du_3\partial/\partial u_3.$$

They then succumbed to the temptation to substitute dt for t. And if the length of t is written as ds, then one gets

$$ds^2 =< t, t >= \sum_{i=1}^{3} du_i^2 < \partial/\partial u_i, \partial/\partial u_i > .$$

This is old notation, which was devised because of imperfect understanding. It is still widely used and is somewhat intuitive, but it is found to be confusing when examined closely. Define

$$g_{ij} =< \partial/\partial u_i, \partial/\partial u_j > .$$

and

$$h_i^2 = g_{ii},$$

which is the square of the length of the coordinate tangent vectors.

Then for orthogonal coordinates the length element becomes

$$ds^2 =< t, t >= \sum_{i=1}^{3} h_i^2 du_i^2.$$

For spherical coordinates r, θ and ϕ

$$h_1 = 1, h_2 = r, h_3 = r\sin(\theta)$$

The spherical volume element is $dv = h_1 h_2 h_3 dr d\theta d\phi = r^2 \sin(\theta) dr d\theta d\phi$. For cylindrical coordinates r, θ and z ,

$$h_1 = 1, h_2 = r, h_3 = 1.$$

The volume element is $dv = r dr d\theta dz$.

Example: divergence in spherical coordinates. We use the fundamental definition:

$$div\mathbf{F} = lim_{V\to 0}\frac{1}{V}\int_S \mathbf{F}\cdot\mathbf{n}da$$

where the volume element is a small nearly cubical element with edges along coordinate curves and of lengths $h_1 dr$, $h_2 d\theta$, and $h_3 d\phi$. We find

$$div\mathbf{F} = \frac{1}{r^2}\frac{\partial(F_r r^2)}{\partial r} + \frac{1}{r\sin(\theta)}\frac{\partial(F_\theta \sin(\theta))}{\partial\theta} + \frac{1}{r\sin(\theta)}\frac{\partial(F_\phi)}{\partial\phi}$$

Proposition. $\mathbf{a}_i = h_i \nabla u_i$.

Proof. ∇u_i is in the u_i direction, say $\nabla u_i = \alpha \mathbf{a}_i$. We have $ds^2 = h_i^2 du_i^2$. Thus $du_i/ds = 1/h_i$. But the directional derivative is

$$du_i/ds = \nabla u_i \cdot \mathbf{a}_i = \alpha \mathbf{a}_i \cdot \mathbf{a}_i = \alpha$$

Thus $\mathbf{a}_i = h_i \nabla u_i$.

Gradient. The gradient in curvilinear orthogonal coordinates is

$$\nabla f = \frac{1}{h_1}\frac{\partial f}{\partial u_1}\mathbf{a}_1 + \frac{1}{h_2}\frac{\partial f}{\partial u_2}\mathbf{a}_2 + \frac{1}{h_3}\frac{\partial f}{\partial u_3}\mathbf{a}_3$$

This follows by differentiating:

$$\nabla f = (\frac{\partial f}{\partial u_1}\frac{\partial u_1}{\partial x} + \frac{\partial f}{\partial u_2}\frac{\partial u_2}{\partial x} + \frac{\partial f}{\partial u_3}\frac{\partial u_3}{\partial x})\mathbf{i}$$

$$+(\frac{\partial f}{\partial u_1}\frac{\partial u_1}{\partial y}+\frac{\partial f}{\partial u_2}\frac{\partial u_2}{\partial y}+\frac{\partial f}{\partial u_3}\frac{\partial u_3}{\partial y})\mathbf{j}$$
$$+(\frac{\partial f}{\partial u_1}\frac{\partial u_1}{\partial z}+\frac{\partial f}{\partial u_2}\frac{\partial u_2}{\partial z}+\frac{\partial f}{\partial u_3}\frac{\partial u_3}{\partial z})\mathbf{k}$$
$$=\frac{\partial f}{\partial u_1}\nabla u_1+\frac{\partial f}{\partial u_2}\nabla u_2+\frac{\partial f}{\partial u_3}\nabla u_3$$
$$=\frac{1}{h_1}\frac{\partial f}{\partial u_1}\mathbf{a}_1+\frac{1}{h_2}\frac{\partial f}{\partial u_2}\mathbf{a}_2+\frac{1}{h_3}\frac{\partial f}{\partial u_3}\mathbf{a}_3.$$

Divergence. The divergence is

$$\nabla\cdot\mathbf{F}=\frac{1}{h_1h_2h_3}(\frac{\partial(h_2h_3F_1)}{\partial u_1}+\frac{\partial(h_1h_3F_2)}{\partial u_2}+\frac{\partial(h_1h_2F_3)}{\partial u_3}).$$

We prove this as follows. Let

$$\mathbf{F}=F_1\mathbf{a}_1+F_2\mathbf{a}_2+F_3\mathbf{a}_3.$$

Using facts such as

$$\mathbf{a}_1=\mathbf{a}_2\times\mathbf{a}_3=h_2h_3\nabla u_2\times\nabla u_3,$$

we find

$$\nabla\cdot\mathbf{F}=\nabla(F_xh_2h_3)\cdot(\nabla u_2\times\nabla u_3)+F_xh_2h_3\nabla\cdot\nabla u_2\times\nabla u_3+...$$
$$=(\frac{\partial(F_1h_2h_3)}{\partial u_1}\nabla u_1+\frac{\partial(F_1h_2h_3)}{\partial u_2}\nabla u_2+\frac{\partial(F_1h_2h_3)}{\partial u_3}\nabla u_3)\cdot\nabla u_2\times\nabla u_3$$
$$+F_1h_2h_3\nabla\cdot\nabla u_2\times\nabla u_3+...$$

$$=(\frac{\partial(F_1h_2h_3)}{\partial u_1}\nabla u_1\cdot\nabla u_2\times\nabla u_3+0+0$$
$$+F_1h_2h_3\nabla\cdot\nabla u_2\times\nabla u_3+...$$
$$=\frac{1}{h_1h_2h_3}\frac{\partial(F_1h_2h_3)}{\partial u_1}+\frac{1}{h_1h_2h_3}\frac{\partial(F_2h_1h_3)}{\partial u_2}+\frac{1}{h_1h_2h_3}\frac{\partial(F_3h_1h_2)}{\partial u_3}.$$

The " + ... " stands for similar terms involving F_2 and F_3. We have used

$$\nabla u_1\cdot\nabla u_2\times\nabla u_3=\frac{1}{h_1h_2h_3},$$

and

$$\nabla \cdot \nabla u_2 \times \nabla u_3 = 0.$$

The latter result follows from the identity involving divergence of a cross product and the fact that the curl of a divergence is zero.

Laplacian. The Laplacian is the divergence of the gradient. Thus

$$\nabla^2 f = \nabla \cdot \nabla f =$$

$$\frac{1}{h_1 h_2 h_3}\left(\frac{\partial((h_2 h_3/h_1)\partial f/\partial u_1)}{\partial u_1} + \frac{\partial((h_1 h_3/h_2)\partial f/\partial u_2)}{\partial u_2} + \frac{\partial((h_1 h_2/h_3)\partial f/\partial u_3)}{\partial u_3}\right).$$

Curl. The Curl is

$$\nabla \times \mathbf{F} =$$

$$\frac{\mathbf{a}_1}{h_2 h_3}\left(\frac{\partial(h_3 F_3)}{\partial u_2} - \frac{\partial(h_2 F_2)}{\partial u_3}\right) +$$

$$\frac{\mathbf{a}_2}{h_1 h_3}\left(\frac{\partial(h_1 F_1)}{\partial u_3} - \frac{\partial(h_3 F_3)}{\partial u_1}\right) +$$

$$\frac{\mathbf{a}_3}{h_1 h_2}\left(\frac{\partial(h_2 F_2)}{\partial u_1} - \frac{\partial(h_1 F_1)}{\partial u_2}\right)$$

We shall prove this as follows. Let

$$\mathbf{f} = f_1 h_1 \nabla u_1 + f_2 h_2 \nabla u_2 + f_2 h_2 \nabla u_3.$$

Then because the curl of a gradient is zero, we have

$$\nabla \times \mathbf{f} = \nabla f_1 h_1 \times \nabla u_1 + \nabla f_2 h_2 \times \nabla u_2 + \nabla f_3 h_3 \times \nabla u_3.$$

Now

$$\nabla f_1 h_1 \times \nabla u_1 =$$

$$\left(\frac{\partial(f_1 h_1)}{\partial u_1}\nabla u_1 + \frac{\partial(f_1 h_1)}{\partial u_2}\nabla u_2 + \frac{\partial(f_1 h_1)}{\partial u_3}\nabla u_3\right) \times \nabla u_1,$$

and similar expressions for the other terms. We have $\nabla u_1 \times \nabla u_1 = 0$ and

$$\nabla u_2 \times \nabla u_1 = -\frac{\mathbf{a}_1 \times \mathbf{a}_2}{h_1 h_2} = -\frac{\mathbf{a}_3}{h_1 h_2},$$

and so on. The result follows by making such substitutions.

88 High Performance Computing and Supercomputing

Names and items in the history of supercomputing: Alan Turing, John Von Neumann, Seymor Cray, World War II Decrypting the German Enigma Machine, Cryptography, The Colossus Machine, Bletchley Park, UK, The IBM Stretch Computer, Seymor Cray and CDC (Control Data Corporation), the CDC 6000, Cray Research, Vector Machines, Pipeline computing, Parallel Processing, Massively Parallel Procesing Clusters, Linux, Beowolf. From one Operation Per Second in 1938, to 1.759 PetaFLOPS of the Cray Jaguar at Oak Ridge, fastest computer as of November 2009. FLOP (FLoating point Operations Per Second), TFLOPS = TerraFLOPS = 10^{12}, PFLOPS = PetaFlops = 10^{15}. Next will come exa, zetta, and yotta. The standard algorithmic test is the **Linpack** benchmark. Current supercomputers usually run the Linux operating system and use Fortran compilers, or C compilers. Formerly supercomputers ran specialized operating systems. Early Cray designed supercomputers used a 60 bit word, so that a floating point operation was a double precision operation. The assembler language for the CDC 6600 was called compass, and I wrote some programs in it. Computers in the 60's had magnetic core memory, so every bit had a small magnetic donut, which would take up almost as much space as a modern single processor.

The 24th International Conference on Supercomputing will be held June 1-4, 2010 in Epochal Tsukuba, Tsukuba, Japan.

88.1 History

```
Year     Supercomputer     Peak speed   (Rmax) Location
1938 Zuse Z1 1 OPS Konrad Zuse, Berlin, Germany
1941 Zuse Z3 20 OPS Konrad Zuse, Berlin, Germany
1943 Colossus 1 5 kOPS Post Office Research Station, Bletchley Park, UK
1944 Colossus 2 (Single Processor) 25 kOPS Post Office Research Station, Bletchley Park, UK
1946 Colossus 2 (Parallel Processor) 50 kOPS Post Office Research Station, Bletchley Park, UK
1946 UPenn ENIAC 5 kOPS Department of War Aberdeen Proving Ground, Maryland, USA
1954 IBM NORC 67 kOPS Department of Defense U.S. Naval Proving Ground, Dahlgren, Virginia, USA
1956 MIT TX-0 83 kOPS Massachusetts Inst. of Technology, Lexington, Massachusetts, USA
1958 IBM AN/FSQ-7 400 kOPS 25 U.S. Air Force sites across the continental
     USA and 1 site in Canada (52 computers)
1960 UNIVAC LARC 250 kFLOPS Atomic Energy Commission (AEC) Lawrence Livermore National
     Laboratory, California, USA
1961 IBM 7030 "Stretch" 1.2 MFLOPS AEC-Los Alamos National Laboratory, New Mexico, USA
1964 CDC 6600 3 MFLOPS AEC-Lawrence Livermore National Laboratory, California, USA
1969 CDC 7600 36 MFLOPS
1974 CDC STAR-100 100 MFLOPS
```

1975 Burroughs ILLIAC IV 150 MFLOPS NASA Ames Research Center, California, USA
1976 Cray-1 250 MFLOPS Energy Research and Development Administration (ERDA)
Los Alamos National Laboratory, New Mexico, USA(80+ sold worldwide)
1981 CDC Cyber 205 400 MFLOPS (~40 systems worldwide)
1983 Cray X-MP/4 941 MFLOPS U.S. Department of Energy (DoE) Los Alamos National Laboratory;
Lawrence Livermore National Laboratory; Battelle; Boeing
1984 M-13 2.4 GFLOPS Scientific Research Institute of Computer Complexes, Moscow, USSR
1985 Cray-2/8 3.9 GFLOPS DoE-Lawrence Livermore National Laboratory, California, USA
1989 ETA10-G/8 10.3 GFLOPS Florida State University, Florida, USA
1990 NEC SX-3/44R 23.2 GFLOPS NEC Fuchu Plant, Fuchu,_Tokyo, Japan
1993 Thinking Machines CM-5/1024 59.7 GFLOPS DoE-Los Alamos National Laboratory;
National Security Agency
Fujitsu Numerical Wind Tunnel 124.50 GFLOPS National Aerospace Laboratory, Tokyo, Japan
Intel Paragon XP/S 140 143.40 GFLOPS DoE-Sandia National Laboratories, New Mexico, USA
1994 Fujitsu Numerical Wind Tunnel 170.40 GFLOPS National Aerospace Laboratory, Tokyo, Japan
1996 Hitachi SR2201/1024 220.4 GFLOPS University of Tokyo, Japan
Hitachi/Tsukuba CP-PACS/2048 368.2 GFLOPS Center for Computational Physics,
University of Tsukuba, Tsukuba, Japan
1997 Intel ASCI Red/9152 1.338 TFLOPS DoE-Sandia National Laboratories, New Mexico, USA
1999 Intel ASCI Red/9632 2.3796 TFLOPS
2000 IBM ASCI White 7.226 TFLOPS DoE-Lawrence Livermore National Laboratory, California, USA
2002 NEC Earth Simulator 35.86 TFLOPS Earth Simulator Center, Yokohama, Japan
2004 IBM Blue Gene/L 70.72 TFLOPS DoE/IBM Rochester, Minnesota, USA
2005 136.8 TFLOPS DoE/U.S. National Nuclear Security Administration,
Lawrence Livermore National Laboratory, California, USA 280.6 TFLOPS
2007 478.2 TFLOPS
2008 IBM Roadrunner 1.026 PFLOPS DoE-Los Alamos National Laboratory, New Mexico, USA 1.105 PFLOPS
2009 Cray Jaguar 1.759 PFLOPS DoE-Oak Ridge National Laboratory, Tennessee, USA

89 The Kalman Filter

See Norman Morrison, "Introduction to Sequential Smoothing and Prediction" McGraw-Hill, 1969.

Connection to the Phase-Locked-Loop circuit electronic device.

Other references:

Brown92 Brown, R. G. and P. Y. C. Hwang. 1992. Introduction to Random Signals and Applied Kalman Filtering, Second Edition, John Wiley and Sons, Inc.
Gelb74 Gelb, A. 1974. Applied Optimal Estimation, MIT Press, Cambridge, MA.
Grewal93 Grewal, Mohinder S., and Angus P. Andrews (1993). Kalman Filtering Theory and Practice. Upper Saddle River, NJ USA, Prentice Hall.
Jacobs93 Jacobs, O. L. R. 1993. Introduction to Control Theory, 2nd Edition. Oxfor University Press.
Julier96 Julier, Simon and Jeffrey Uhlman. A General Method of Approximating Nonlinear Transformations of Probability Distributions, Robotics Research Group, Department of Engineering Science, University of Oxford [cited 14 November 1995]. Available from http://www.robots.ox.ac.uk/~siju/

work/publications/Unscented.zip.
Also see: A New Approach for Filtering Nonlinear Systems by S. J. Julier, J. K. Uhlmann, and H. F. Durrant-Whyte, Proceedings of the 1995 American Control Conference, Seattle, Washington, Pages:1628-1632. Available from http://www.robots.ox.ac.uk/~siju/work/publications/ACC95_pr.zip.
Also see Simon Julier's home page at http://www.robots.ox.ac.uk/~siju/.
Kalman60 Kalman, R. E. 1960. A New Approach to Linear Filtering and Prediction Problems, Transaction of the ASMEJournal of Basic Engineering, pp. 35-45 (March 1960).
Lewis86 Lewis, Richard. 1986. Optimal Estimation with an Introduction to Stochasti Control Theory, John Wiley and Sons, Inc.
Maybeck79 Maybeck, Peter S. 1979. Stochastic Models, Estimation, and Control, Volu 1, Academic Press, Inc.
Sorenson70 Sorenson, H. W. 1970. Least-Squares estimation: from Gauss to Kalman, IEEE Spectrum, vol. 7, pp. 63-68, July 1970.

90 Elementary Physics Formulas

90.1 Simple Motion

Constant velocity case:

$$d = vt,$$

where $d = x - x_0$.
Constant acceleration case:

$$v = at$$

$$x = x_0 + v_0 t + \frac{a}{2}t^2$$

$$x = \frac{v_0 + v}{2}t \quad \text{(mnemonic: think average velocity)}$$

$$ax = \frac{v^2 - v_0^2}{2} \quad \text{(mnemonic: think of energy)}$$

These formulas do not apply to general motion.

90.2 Vectors

If $\mathbf{w} = \mathbf{u} + \mathbf{v}$ then the sum of the components of $\mathbf{u}$ and $\mathbf{v}$ are the components of $\mathbf{w}$.

$$w_x = u_x + v_x, w_y = u_y + v_y.$$

The vector magnitude is

$$w = \sqrt{w_x^2 + w_y^2}.$$

Vector components are

$$w_x = w\cos(\theta),$$

and

$$w_y = w\sin(\theta),$$

where θ is the vector angle.

90.3 Parabolic Trajectory of a Projectile

A particle moves in a parabolic trajectory when there is uniform acceleration in y, with $a_y = -g$, and a constant velocity in x,

$$v_x = constant.$$

90.4 Centripetal Acceleration

$$a = v^2/r = (r\omega)^2/r = r\omega^2$$

90.5 Newton's Laws of Motion

$$\mathbf{v} \text{ is constant if } \mathbf{f} = 0 \quad \text{(First law)},$$

$$\mathbf{F} = m\mathbf{a} \quad \text{(Second law)},$$

$$\mathbf{F}_{12} = -\mathbf{F}_{21} \quad \text{(Third law)}.$$

90.6 Gravity

Weight is the force of gravity. On the earth's surface

$$w = mg$$

where $g = 9.81$ m/s^2. Newton's law of universal gravitation is

$$f = \frac{GmM}{r^2}$$

90.7 Friction

$$F_{friction} = \mu N$$

Friction dissipates energy.

90.8 Momentum

$$\mathbf{p} = m\mathbf{v}$$

Conservation of momentum in collisions is a consequence of Newton's third law. Conservation of momentum:

$$\mathbf{p}_{initial} = \mathbf{p}_{final}$$

In two dimensions this gives two equations and so allows two unknowns to be found.

90.9 Energy

Kinetic:

$$E_k = \frac{1}{2}mv^2$$

Potential: (gravity on earth's surface)

$$E_p = mgh.$$

Potential: (spring, f= -kx)

$$E_p = \frac{1}{2}kx^2$$

In a perfectly elastic collision kinetic energy is conserved. This adds one equation to the two momentum equations. Three unknowns can be determined. Work is the energy generated by applying a force along a distance.

$$w = fd\cos(\theta)$$

θ is the angle between the force and the path.
Power is the rate of doing work.

$$p = \frac{\Delta w}{\Delta t}$$

90.10 Elasticity, Stress-Strain Relations

Linear:

$$\text{Stress:} \quad \sigma = \frac{\text{normal force}}{\text{area}}$$

$$\text{Strain:} \quad \epsilon = \frac{\text{change in length}}{\text{original length}} = \frac{\Delta l}{l_0}$$

Hooke's Law:

$$\sigma = E\epsilon \quad (E = \text{Young's modulus}).$$

Bulk:

$$\sigma = -B\epsilon_b.$$

Bulk strain:

$$\epsilon_b = \frac{\Delta V}{V_0}$$

Shear:

$$\sigma_s = G\epsilon_s.$$

Shear strain:

$$\epsilon_s = \frac{\Delta y}{\ell} = \tan(\text{shear angle})$$

$$\text{Shear stress:} \quad \sigma = \frac{\text{tangential force}}{\text{area}}$$

90.11 Torque

In the two dimensional case, given a force **F** applied at point **B**, the torque τ, about point **A** can be written in three ways as

$$\tau = rF\sin(\theta) = xF_y - yF_x = Fd \tag{32}$$

where **r** is the vector from **A** to **B** with components x and y, θ is the angle between **r** and **F**, and d is the distance from **A** to the line of action of **F**. A counterclockwise torque is positive. In general

$$\tau = \mathbf{r} \times \mathbf{f}.$$

90.12 Equilibrium

A rigid body is in equilibrium if the sum all external forces (x and y components) is zero, and the sum of all external torques about any point is zero.

The x-component of the center of gravity of n bodies is defined by

$$xm = x_1m_1 + x_2m_2 + ... + x_nm_n \tag{33}$$

where x_i and m_i is the mass and position of the ith body, and m is the total mass.

90.13 Rotation about a fixed axis

The angular velocity and acceleration of a body rotating about a fixed axis is defined by

$$\omega = Lim_{\Delta t \to 0}\frac{\Delta\theta}{\Delta t} \tag{34}$$

$$\alpha = Lim_{\Delta t \to 0}\frac{\Delta\omega}{\Delta t} \tag{35}$$

The tangential distance d, speed v, and acceleration a at distance r from the axis, are related to the angular variables by

$$d = r\theta, v = r\omega, a = r\alpha \tag{36}$$

The equations for constant velocity motion, and the equations for constant acceleration motion apply by substituting the angular variables.

The moment of inertia of a system of n bodies is defined by

$$I = r_1^2 m_1 + r_2^2 m_2 + ... + r_n^2 m_n \tag{37}$$

Newton's equation of motion becomes

$$\tau = I\alpha \tag{38}$$

Angular momentum like linear momentum is conserved. The angular momentum is

$$L = I\omega \tag{39}$$

Rotational kinetic energy is

$$E_{rot} = \frac{I}{2}\omega^2 \tag{40}$$

90.14 Simple Harmonic Motion

Simple harmonic motion is the result of a force that is proportional to the negative of the displacement ($F \propto -x$). A solution (when the maximum displacement occurs at time t=0) is

$$x = A\cos(\omega t) \tag{41}$$

The motion repeats when $\omega T = 2\pi$. The time t= T is called the period. We have

$$T = \frac{2\pi}{\omega}, f = \frac{1}{T} = \frac{\omega}{2\pi} \tag{42}$$

A is called the amplitude and equals the maximum displacement. f is the frequency and ω is the angular frequency.

$$v = -\omega A\sin(\omega t) \tag{43}$$

This result follows from the fact that the slope of a tangent line to the cos curve equals the negative of the sin.

$$a = -\omega^2 A\cos(\omega t) = -\omega^2 x \tag{44}$$

This result follows from the fact that the slope of a tangent line to the sin curve equals the cos.

Spring example:

$$F = -kx \Rightarrow a = -\frac{k}{m}x = -\omega^2 x \Rightarrow f = \frac{\omega}{2\pi} = \frac{1}{2\pi}(k/m)^{1/2} \tag{45}$$

Pendulum example:

$$F = -\frac{mg}{l}x \Rightarrow a = -\frac{g}{l}x = -\omega^2 x \Rightarrow f = \frac{\omega}{2\pi} = \frac{1}{2\pi}(g/l)^{1/2} \tag{46}$$

90.15 Special Relativity

Let v be the relative velocity in the x direction between two coordinate frames, an unprimed frame and a primed frame. Let c be the velocity of light, $c = 3 \times 10^8$ m/s. Define

$$\gamma = \frac{1}{\sqrt{1 - v^2/c^2}}.$$

As v gets close to c, γ gets large. At ordinary everyday velocities γ equals one. Lorentz Transformations:

$$x' = \gamma(x - vt)$$

$$y' = y$$

$$z' = z$$

$$t' = \gamma(t - \frac{xv}{c^2})$$

Time dilation:

$$\Delta t' = \gamma \Delta t$$

Length contraction:

$$\Delta x' = \frac{\Delta x}{\gamma}.$$

Let u be the velocity of a body in the unprimed frame and u' the velocity in the primed frame. Velocity addition:

$$u' = \frac{u - v}{1 - uv/c^2}.$$

Momentum:

$$p = \gamma m_0 v.$$

Energy:

$$E = \gamma m_0 c^2.$$

Events that are simultaneous in one frame may not be simultaneous in another frame. It is possible for A grandfather to become younger than a grandson.

90.16 Fluids, Forces and Motion

Density:

$$\rho = \text{ mass/volume.}$$

Specific gravity

$$S_g = \rho/\rho_{H_2O}.$$

Pressure: $P = F/A$, Pa = 1 n/m^2, 1 atm is nearly 10^5 Pa. Torr = (pressure of 1 mm of mercury). Hydrostatic pressure:

$$P = \rho g h$$

The buoyant force on an object placed in a fluid is equal to the weight of the displaced fluid. Bernoulli's equation (frictionless flow in a pipe):

$$P + \frac{1}{2}\rho v^2 + \rho g h = \text{constant}$$

90.17 Heat and Thermodynamics

Temperature:

$$T_{Kelvin} \approx 273^\circ + T_{centigrade}$$

Ideal gas law:

$$PV = nRT$$

n = number of kmoles, R is the gas constant.

$$R = 8.314\text{KJ/(Kmole-Kelvin) (mnemonic: } \pi = 3.14)$$

A Kmole contains $N_A = 6.02 \times 10^{26}$ molecules. Temperature expansion:

$$\frac{\Delta \ell}{\ell} = \alpha_\ell \Delta T$$

$$\frac{\Delta V}{V} = \alpha_V \Delta T$$

Heat transfer: Let R be the flow of heat.

$$R_{conduction} = \frac{KA\Delta T}{\ell}$$

$$R_{radiation} = e\sigma A T^4$$

$$K = \text{thermal conductivity}$$

$$e = \text{emissivity, } 0 \leq e \leq 1$$

$$\sigma = \text{Stefan-Boltzmann constant}$$

Translational kinetic energy of a molecule:

$$\varepsilon_k = \frac{1}{2}mv^2 = \frac{3}{2}kT$$

$$\text{Boltzmann's constant } = k = R/N_A$$

This shows that the mean velocity of a molecule is proportional to the absolute temperature and that the internal energy of an ideal gas is a function of the absolute temperature only. The specific heat capacity at constant volume of an ideal gas is

$$C_V^M = \frac{f}{2}R,$$

where f is the number of degrees of freedom. For a monatomic gas $f = 3$.
Conservation of energy, the first law of thermodynamics:

$$\Delta U = Q - W$$

The second law of thermodynamics: (Kelvin) A cyclic heat engine can not convert heat Q from a reservoir at temperature T entirely into work.
Efficiency of a heat engine:

$$\eta = \frac{Q_{input} - Q_{output}}{Q_{input}} = \frac{W}{Q_{input}}$$

No engine can be more efficient than a reversible engine. For A reversible engine

$$\frac{Q_{hot}}{Q_{cold}} = \frac{T_{hot}}{T_{cold}}$$

Thus for any engine

$$\eta \leq 1 - \frac{T_{cold}}{T_{hot}}$$

Entropy:

$$\Delta S = \frac{\Delta Q_{reversible}}{T}$$

$$S = k \ln(W)$$

W is the thermodynamic probability. It is the number of possible ways a system can exist in a given macroscopic state. Entropy is a measure of disorder. A perfect crystal at absolute zero has zero entropy, for there is only one way that it can exist with all the molecules perfectly positioned with zero energy and zero velocity. The importance of entropy: Enthalpy H, for ordinary chemical reactions, is a measure of internal energy given by

$$H = U - PV.$$

The Gibbs free energy,

$$G = H - TS,$$

is a chemical potential energy. Chemical reactions occur in such a way as to minimize G. Entropy determines the direction of chemical reactions and thus the evolution of life. The entropy of the universe steadily increases.

91 Electrical Circuits

91.1 Maxwell's Equations

Recall that if we have a vector field

$$\mathbf{C} = C_x\mathbf{i} + C_y\mathbf{j} + C_z\mathbf{k},$$

then the curl of $\mathbf{C}$ is

$$\nabla \times \mathbf{C} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \\ C_x & C_y & C_z \end{vmatrix}$$

and the divergence is

$$\nabla \cdot \mathbf{C} = \frac{\partial C_x}{\partial x} + \frac{\partial C_y}{\partial y} + \frac{\partial C_z}{\partial z}.$$

The Maxwell Equations in MKS form are

$$\nabla \times \mathbf{H} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t},$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t},$$

$$\nabla \cdot \mathbf{D} = \rho,$$

$$\nabla \cdot \mathbf{B} = 0.$$

The first equation is Ampere's Law, with the Maxwellian addition of the displacement current term

$$\frac{\partial \mathbf{D}}{\partial t}.$$

Ampere's law says that each portion of current flow produces a magnetic field. If there is no changing field $\mathbf{D}$, which is a modification of the $\mathbf{E}$ field caused by the presence of electrically polarized materials, then the first Maxwell equation becomes

$$\nabla \times \mathbf{H} = \mathbf{J}.$$

The vector field $\mathbf{D}$ is defined by

$$\mathbf{D} = \epsilon_0 \mathbf{E} + \mathbf{P},$$

where $\mathbf{P}$ is the electric dipole moment per unit volume in a dielectric material. The number ϵ_0 is called the permitivity of free space.

Applying Stoke's Theorem we have

$$\int_C \mathbf{H} \cdot d\mathbf{R} = \int_S \nabla \times \mathbf{H} \cdot d\mathbf{S} = \int_S \mathbf{J} \cdot d\mathbf{S} = i.$$

That is, the line integral of the magnetic intensity $\mathbf{H}$ around a path C equals the amount of curremt i flowing through the surface S that is bounded by C, which is Ampere's law. Let us remark about the displacement term. Now if there were a capacitor placed in our wire, there is current flowing through the wire, but no actual charge flow between the capacitor plates. Hence, if we let our surface S pass between the capacitor plates then there would be a zero J, and thus zero i. And so our line integral of $\mathbf{H}$ around the magnetic circuit would be zero. So depending on where we place our surface we get zero or not zero. That is why the displacement term must be added to the first maxwell equation.

The second equation is Faraday's law of induction. Using Stoke's theorem we have

$$\int_C \mathbf{E} \cdot d\mathbf{R} = \int_S \nabla \times \mathbf{E} \cdot d\mathbf{S} = -\int_S \frac{\partial \mathbf{B}}{\partial t} \cdot d\mathbf{S} = -\frac{\partial \Phi}{\partial t}.$$

That is, the electric potential (MMF) around a circuit C is equal to the rate of magnetic flux change through the circuit.

If the material is soft iron and essentially linear with little hystereses we may write

$$\mathbf{B} = \mu \mathbf{H},$$

where μ is a constant called the permeability. Such material forms a linear magnetic circuit. From the fourth law,

$$\nabla \cdot \mathbf{B} = 0.$$

The divergence of the $\mathbf{B}$ field is zero. Using the divergence theorem to convert a volume integral to a an integral on the bounding surface, we have

$$0 = \int_V \nabla \cdot \mathbf{B} dV = \int_S \mathbf{B} \cdot d\mathbf{S},$$

which means that every flux line entering a volume, leaves the volume. Thus there are no sources of magnetic flux lines and so they form continuous loops of flux.

91.2 The Clamp Meter

The clamp meter for measuring current is an annular ring of soft iron that closes around the wire. The clamp meter measures the alternating current in the wire. The line integral of the H field along a path located in the soft iron is equal to the amount of current that passes through the ring of soft iron. This defines a B field along each path, and hence the total flux in the magnetic circuit. This magnetic flux is proportional to the current i. A coil is wound around the magnetic path and the induced voltage is given by Faraday's law as the negative derivative of the flux with respect to time, times the number of turns in the coil. Thus the induced voltage in the coil is proportional to the current in the wire. Alternating current is sinusoidal, as is the derivative of the flux induced by the current. So the induced voltage in the sensing coil is also sinusoidal.

91.3 Inductance and Mutual Inductance

Suppose we have n circuits linked by magnetic flux. Then the voltage induced in circuit i according to Faraday's law is given by

$$\xi_i = \sum_{j=1}^{n} \frac{d\Phi_{ij}}{dt},$$

where Φ_{ij} is the flux linking circuit i due to current in circuit j. We can write

$$\frac{d\Phi_{ij}}{dt} = \frac{d\Phi_{ij}}{dI_j}\frac{dI_j}{dt} = M_{ij}\frac{dI_j}{dt},$$

where I_j is the current in circuit j, and

$$M_{ij} = \frac{d\Phi_{ij}}{dI},$$

is called the mutual inductance.

Example. Suppose we have a torroid and two windings around the torroid. Then according to ampere's law, the mean value of the magnetic induction field magnitude B is

$$B = \frac{\mu_0 N_1 I}{\ell},$$

where N_1 is the number of turns in winding 1, and ℓ is the mean length of the path around the inside of the torroid. If we assume that B is approximately constant throughout the cross section of the torroid, then the flux in the torroid caused by current I_1 is

$$\Phi_1 = \frac{\mu_0 N_1 I A}{\ell},$$

where A is the cross sectional area of the torroid. In such a torroid made of soft iron all of the flux is confined to the interior of the torroid, and the voltage induced in circuit 2 is

$$\xi_2 = \frac{d\Phi_1}{dt} = N_2 \frac{d}{dI_1}\left(\frac{\mu_0 N_1 I A}{\ell}\right)\frac{dI_1}{dt}$$

$$= N_2 \frac{\mu_0 N_1 A}{\ell}\frac{dI_1}{dt}.$$

$$= M_{21}\frac{dI_1}{dt}.$$

So the mutual inductance is

$$M_{21} = N_2 N_1 \frac{\mu_0 A}{\ell}$$

A single winding induces a voltage in itself with mutual inductance

$$M_{11} = N_1 N_1 \frac{\mu_0 A}{\ell}.$$

Such an inductance is called the self inductance, which is written as

$$L_1 = M_{11}.$$

For an inductor L the voltage induced across the inductor is given as

$$\xi = L \frac{dI}{dt}.$$

This is the voltage drop across an inductor in a circuit where a current I flows. In the case of the torroid with a linear soft iron core, we see that the inductance is essentially constant and independent of the current I, and proportional to the square of the number turns N^2. This sort of relationship holds for most practical inductors.

91.4 Magnetic Circuits

A continuous tube of flux Φ forms a magnetic circuit. Let the circuit pass through a coil containing N turns and current i. For a path around the circuit

$$Ni = \oint \mathbf{H} \cdot d\mathbf{r} = \sum H_i L_i = \sum \frac{L_i \phi}{\mu_i A_i} = \Phi \sum \Re_i.$$

This equation is an approximation. $\mathbf{H}_i$ is an assumed constant value of $\mathbf{H}$ in the ith piece of the circuit. The reluctance of the ith piece is $\Re_i$. L is the length of the piece and A is the cross sectional area. The magnetomotive force mmf is Ni. We have

$$mmf = \Phi \Re.$$

A large flux path may be treated as a set of parallel paths and the net reluctance can be computed by a technique similar to that of computing parallel resistances.

91.5 Steady State Alternating Currents

Consider the RLC circuit with a voltage source.

$$L\frac{di}{dt} + Ri + \frac{q}{C} = v.$$

Differentiating

$$L\frac{d^2i}{dt^2} + R\frac{di}{dt} + \frac{i}{C} = \frac{dv}{dt}.$$

Let

$$i = I_0 \exp j\omega t,$$

and

$$v = V_0 \exp j\omega t,$$

where I_0 and V_0 may be complex numbers to allow i and v to be out of phase. Then

$$[-\omega^2 L + Rj\omega + \frac{1}{C}]I_0 \exp j\omega t = V_0 j\omega \exp j\omega t.$$

Then

$$[-\omega^2 L + Rj\omega + \frac{1}{C}]I_0 = V_0 j\omega.$$

Dividing by $j\omega$

$$[-\frac{\omega^2 L}{j\omega} + R + \frac{1}{Cj\omega}]I_0 = V_0.$$

Then

$$[R + (\omega L - \frac{1}{\omega C})j]I_0 = V_0.$$

Then

$$I_0 = \frac{V_0}{Z},$$

where

$$Z = R + (\omega L - \frac{1}{\omega C})j$$

is the impedance.

The admittance is the reciprocal of the impedance. Notice that as ω varies from 0 to ∞, Z is a line in the complex plane parallel to the imaginary axis, and it does not pass through zero. An inversion mapping, $Z \to 1/Z$, being a special case of a linear fractional transformation (also called a projective transformation or a bilinear mapping), maps circles to circles (where a

straight line is a special circle of infinite radius). The line maps to a circle on the complex sphere. It passes through the north pole, but not the south pole. On the sphere the mapping $Z \to 1/Z$ corresponds to a 180 degree rotation of the sphere about a horizontal axis so that the image of the circle on the sphere no longer passes through the north pole. Hence the plane image must be a true circle. So the admittance lies on a circle provided the impedance does not pass through zero, that is, provided R is not zero. Details about such mappings are contained in many books on complex analysis. See for example: **Elements of the Theory of Functions** by K. Knopf.

In the case of a circuit network we have

$$Mi = \frac{dv}{dt},$$

where M is a matrix with second order differential operator terms, i is a current vector and v is a voltage vector. The same technique applies. The steady state solution is found by inverting the impedance matrix.

91.6 RLC Transient Solution

The equation for a RLC circuit with a voltage source is

$$L\frac{di}{dt} + Ri + \frac{q}{C} = v.$$

Differentiating we obtain a second order differential equation with constant coefficients

$$L\frac{d^2i}{dt^2} + R\frac{di}{dt} + \frac{i}{C} = \frac{dv}{dt}.$$

The general solution consists of a linear combination of two linearly independent solutions of the homogeneous equation and any particular solution. Given a harmonic voltage

$$v = V_0 \exp j\omega t,$$

a complex particular solution is

$$i_c = I_0 \exp j\omega t,$$

where

$$I_0 = V_0/Z,$$

and where I_0, V_0, and the impedance

$$Z = R + j(\omega L - 1/(\omega C))$$

are complex numbers. Either the real part or the imaginary part of this complex solution may be taken as a particular solution i_p. It remains to find two linearly independent solutions of the homogeneous equation

$$L\frac{d^2 i}{dt^2} i + R\frac{di}{dt} + \frac{i}{C} = 0.$$

Substituting $i = \exp(jmt)$ into this equation, we get a polynomial in m, which has roots

$$-\frac{R}{2L} \pm \sqrt{(R/2L)^2 - 1/LC}.$$

The natural undamped (R=0) frequency is

$$\omega_n = \sqrt{1/LC}.$$

The critical damping resistance is the value of R that makes the expression

$$(R/2L)^2 - 1/LC$$

zero. Therefore the critical damping resistance is

$$R_c = 2L\omega_n.$$

The critical damping ratio is

$$\zeta = \frac{R}{R_c}.$$

Then we have

$$R = 2\zeta L\omega_n,$$

and

$$\frac{R}{2L} = \zeta\omega_n.$$

The system is underdamped if $\zeta < 1$, overdamped if $\zeta > 1$, and critically damped if $\zeta = 1$. Suppose the equation is underdamped. Then two independent solutions of the homogeneous equation are

$$i_1 = \exp(-\zeta\omega_n t)\sin(\omega_d t),$$

and

$$i_2 = \exp(-\zeta\omega_n t)\cos(\omega_d t),$$

where the angular frequency of damped oscillation is

$$\omega_d = \omega_n\sqrt{1-\zeta^2}.$$

We may take as particular solution $i_p = I_0 \exp(j(\omega t + \phi))$ where I_0 is real.

We have

$$i = A_1 i_1 + A_2 i_2 + i_p$$

and

$$\frac{di}{dt} = A_1\frac{di_1}{dt} + A_2\frac{di_2}{dt} + \frac{di_p}{dt}.$$

We wish to find the constants A_1 and A_2 appearing in these two equations.

Suppose we have the initial conditions

$$i(0) = \alpha_1$$

and

$$\frac{di}{dt}(0) = \alpha_2.$$

Then we can solve the equations for the constants A_1 and A_2 in terms of α_1 and α_2, and thus determine the unique solution $i(t)$.

We have

$$i(0) = A_2 + I_0\exp(j\phi)$$

and

$$\frac{di}{dt}(0) = A_1\omega_d - A_2\zeta\omega_n + jI_0\omega\exp(j\phi)$$

Using the real parts we have

$$A_2 + I_0\cos(\phi) = \alpha_1$$

and

$$A_1\omega_d - A_2\zeta\omega_n - I_0\omega\sin(\phi) = \alpha_2.$$

So

$$A_2 = \alpha_1 - I_0\cos(\phi)$$

and

$$A_1 = \frac{\alpha_2 + A_2\zeta\omega_n + I_0\omega\sin(\phi)}{\omega_d}$$

The real solution is

$$i(t) = A_1\exp(-\zeta\omega_n t)\sin(\omega_d t) + A_2\exp(-\zeta\omega_n t)\cos(\omega_d t) + I_0\cos(\omega t + \phi).$$

The program **rlc.ftn** plots this solution.

91.7 A Piezioelectric Transducer Equivalent Circuit

(See also: Emery **Transducer Equivalent Circuits**). In the the RLC circuit described above, suppose we introduce a second capacitor C_2 connected in parallel across the voltage source. The source is assumed active and able to supply a constant voltage for any load up to its rated capacity. The series resistance of the shunt capacitor is assumed so small that a change in voltage induces an essentially instantaneous change in the charge on the capacitor so that we can ignore the transient response. Then if i_2 is the current thru the capacitor, we have

$$\frac{i_2}{C_2} = \frac{dv}{dt}.$$

If $v = V_0 \exp(j(\omega t + \phi))$, then we have

$$i_2 = jV_0\omega C_2 \exp(j(\omega t + \phi)).$$

The real part is

$$i_2 = -V_0\omega C_2 \sin(\omega t + \phi).$$

The source current i is the sum of the currents through the parallel branches

$$i = i_1 + i_2,$$

where i_1 is the RLC current from the previous section. Thus

$$i(t) = A_1 \exp(-\zeta\omega_n t)\sin(\omega_d t) + A_2 \exp(-\zeta\omega_n t)\cos(\omega_d t) + I_0 \cos(\omega t + \phi) - V_0\omega \sin(\omega t + \phi).$$

We compute the constants A_1 and A_2 from the initial conditions.

Suppose the initial conditions are

$$i(0) = \alpha_1$$

and

$$\frac{di}{dt}(0) = \alpha_2.$$

Then we have

$$i(0) = A_2 + I_0 \cos(\phi) - V_0\omega \sin(\phi)$$

and

$$\frac{di}{dt}(0) = A_1\omega_d - A_2\zeta\omega_n - I_0\omega \sin(\phi) - V_0\omega^2 \cos(\phi).$$

So

$$A_2 = \alpha_1 - I_0 \cos(\phi) + V_0 \omega \sin(\phi)$$

and

$$A_1 = \frac{\alpha_2 + A_2 \zeta \omega_n + I_0 \omega \sin(\phi) + V_0 \omega^2 \cos(\phi)}{\omega_d}$$

The program **teqcir.ftn** plots this solution.

91.8 The Ideal Transformer

Two simplified models of a real transformer are the ideal transformer and the perfect transformer. The ideal transformer is characterized by the turns ratio n. The ideal transformer has the simple defining properties relating the input and output voltages and currents:

$$v_1 = n v_2$$

$$i_2 = -n i_1.$$

See Balabanian and Bickart, page 42.

91.9 The Perfect Transformer

The perfect transformer is characterized by a coupling ratio $k = 1$, where

$$k^2 = \frac{M^2}{L_1 L_2}.$$

If $k = 1$ then $M = \sqrt{L_1 L_2}$, so that

$$\frac{v_1}{v_2} = \frac{L_1 di_1/dt + M di_2/dt}{M di_1/dt + L_2 di_2/dt}$$

$$= \frac{\sqrt{L_1}}{\sqrt{L_2}} \left[\frac{\sqrt{L_1} di_1/dt + \sqrt{L_2} di_2/dt}{\sqrt{L_1} di_1/dt + \sqrt{L_2} di_2/dt} \right]$$

$$= \frac{\sqrt{L_1}}{\sqrt{L_2}}.$$

If the inductance of a coil is proportional to the square of the number of turns then

$$\frac{v_1}{v_2} = n,$$

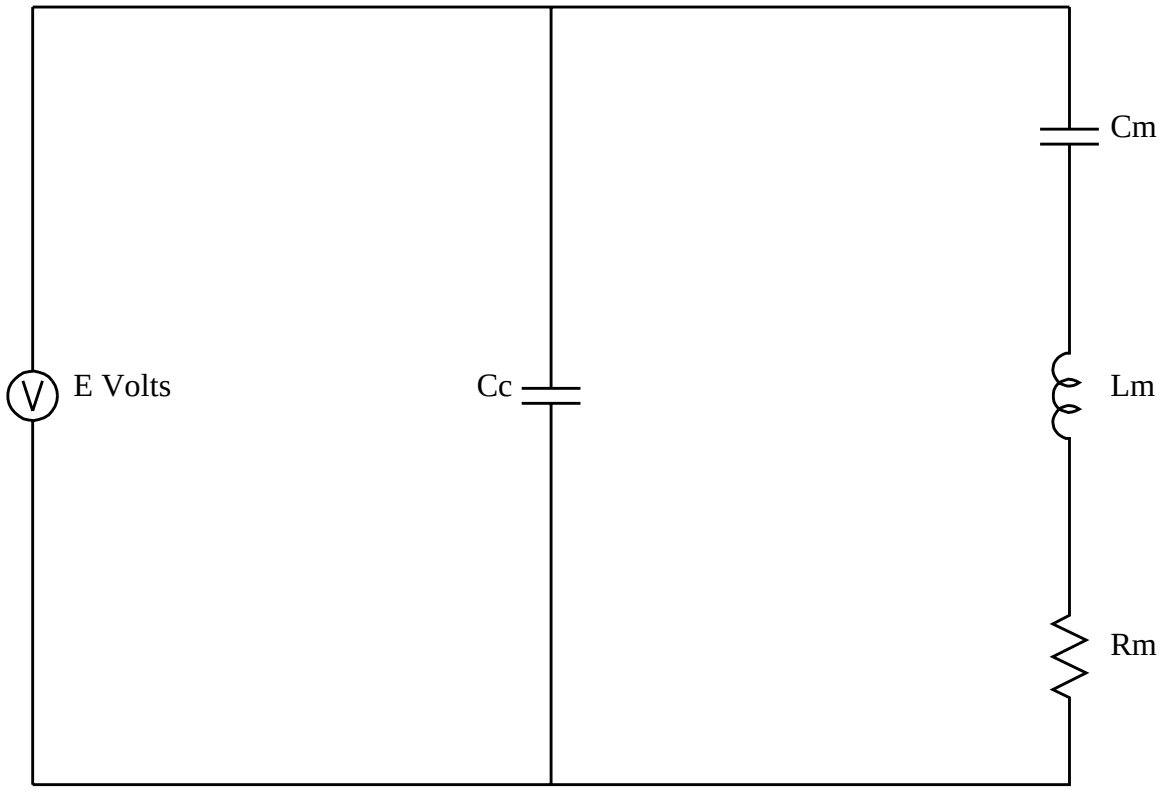


Figure 17: Transducer equivalent circuit, with motional capacitor C_m, and clamped capacitor C_c.

where n is the turns ratio. The flux through a coil is proportional to the current in each turn, hence to the current times the number of turns. The induced voltage is proportional to the number of turns and to the rate of change of flux. Hence the voltage is proportional to the rate of change of current times the square of the number of turns. Hence the inductance is proportional to the square of the number of turns. Also see the inductance of two inductances in series, (Reitz and Millford). If L_1 and L_2 go to infinity while n is constant, then the perfect transformer becomes an ideal transformer. See Balabanian and Bickart pp. 43-45.

91.10 Transformer Equivalent Circuits

See notes 1-17-86 and see Millman and Taub, **Pulse and Digital Circuits**, pp 253-257.

91.11 Thevenin's Theorem

Thevenin's theorem is presented in almost all circuit analysis books: *Any two terminals of a network may be replaced by a network consisting of an equivalent series resistor and an equivalent voltage source*

A real proof of the theorem is not often given. For a good proof of the theorem, for the general case, see **Linear Network Theory**, Clifford Ferris, p256.

91.12 Resonance of an RLC Circuit and The Quality Factor

Reference: Vincent Del Torro, **Principles of Electrical Engineering**, pp. 206-214. Given a series RLC circuit, for resonance we have

$$I = \frac{V}{R + j(\omega L - 1/\omega C)},$$

and

$$\omega L - 1/\omega C = 0.$$

$$\omega = \frac{1}{\sqrt{LC}}.$$

Let

$$I = I_0 \cos(\omega t).$$

The energy in the inductor is

$$e_L = \frac{1}{2} I^2 L = \frac{1}{2} I_0^2 \cos^2(\omega t).$$

The energy stored in the capacitor is

$$\begin{aligned} e_C = \frac{1}{2} C V^2 &= \frac{1}{2} C (I \frac{-j}{\omega C})^2 \\ &= \frac{1}{2} C \frac{I_0^2 \sin^2(\omega t)}{C^2 \omega^2} \\ &= \frac{1}{2} L I_0^2 \sin^2(\omega t). \end{aligned}$$

Hence

$$e_L + e_C = \frac{1}{2} L I_0^2.$$

Let ω_1 and ω_2 be the half-power points, where the power is one half the power at the resonant point ω_0. The magnitude of the impedance at the half power point must be $\sqrt{2}R$. Hence

$$\omega_1 L - \frac{1}{\omega_1 C} = R$$

Then

$$\omega_1 = -\alpha + \sqrt{\alpha^2 + \omega_0^2},$$

where

$$\alpha = \frac{R}{2L}.$$

And

$$\omega_2 = \alpha + \sqrt{\alpha^2 + \omega_0^2}.$$

The bandwidth is

$$\omega_{bw} = \omega_2 - \omega_1 = 2\alpha = \frac{R}{L}.$$

The quality factor is defined as

$$Q_0 = \frac{\omega_0}{\omega_{bw}} = \frac{\omega_0 L}{R} = \frac{X_l}{R}.$$

Then

$$Q_0 = \frac{\omega_0 L}{R} = \frac{\omega_0 L I_0^2/2}{R I_0^2/2}$$
$$= 2\pi \frac{L I_0^2/2}{(R I_0^2/2)/f_0},$$

where f_0 is the resonant frequency. Then the quality factor is 2π times the stored energy, divided by the energy dissipated per cycle.

We may write an alternate expression for Q_0. Since the resonant frequency is

$$\omega_0 = \frac{1}{\sqrt{LC}},$$

we have

$$Q_0 = \frac{\omega_0 L}{R} = \frac{1}{R}\sqrt{\frac{L}{C}}.$$

91.13 Quality Factor at Nonresonance

Let the current in a series RLC circuit be

$$i(t) = I_m e^{j\omega t}.$$

The voltage across the capacitor is

$$v(t) = i\frac{-j}{\omega C} = \frac{I_m}{\omega C}(\sin(\omega t) - j\cos(\omega t))$$

The sum of the energy stored in the inductor and capacitor at time t is

$$W = \frac{1}{2}L\Re(i)^2 + \frac{1}{2}C\Re(v)^2$$

$$= \frac{1}{2}L I_m^2 \cos^2(\omega t) + \frac{1}{2}\frac{I_m^2}{\omega^2 C}\sin^2(\omega t).$$

Integrating over period

$$T = \frac{2\pi}{\omega},$$

we get the average stored energy. The average of each of $\cos^2(\omega t)$, and $\sin^2(\omega t)$, is $1/2$, so that

$$W_{ave} = \frac{1}{2}I_m^2\frac{1}{2}(L + \frac{1}{\omega^2 C})$$

$$= I_{rms}^2 \frac{1}{2}(L + \frac{1}{\omega^2 C})$$

The energy dissipated per cycle in the resistor is

$$W_d = I_{rms}^2 RT = I_{rms}^2 R \frac{2\pi}{\omega}.$$

Then the quality factor at frequency ω is

$$Q(\omega) = \frac{2\pi W_{ave}}{W_d} = \frac{1}{2R}(\omega L + \frac{1}{C\omega}).$$

If we differentiate Q with respect to ω and set the derivative equal to zero, we get

$$L = \frac{1}{\omega^2 C}.$$

So that Q has a local extremum at the resonant frequency

$$\omega = \frac{1}{\sqrt{LC}}.$$

This is a minimum, since Q goes to infinity as ω goes to infinity, and as ω goes to zero.

At Resonance

$$\omega = \omega_r = \frac{1}{\sqrt{LC}},$$

so that

$$Q(\omega_r) = \frac{1}{2R}(\omega_r L + \frac{\omega_r}{C\omega_r^2}).$$

$$= \frac{\omega_r L}{R}.$$

Example. Consider the example of a piezoelectric equivalent circuit from **Mentesana**:

$$Z_r = 7.5\text{ohm}, f_r = 69.624\text{KHz}, Z_a = 450.7\text{ohm}, f_a = 69.942\text{KHz}.$$

Then the parameters for the equivalent mechanical RLC circuit are

$$R = 7.5\text{ohm}.$$

$$L = .0146H.$$

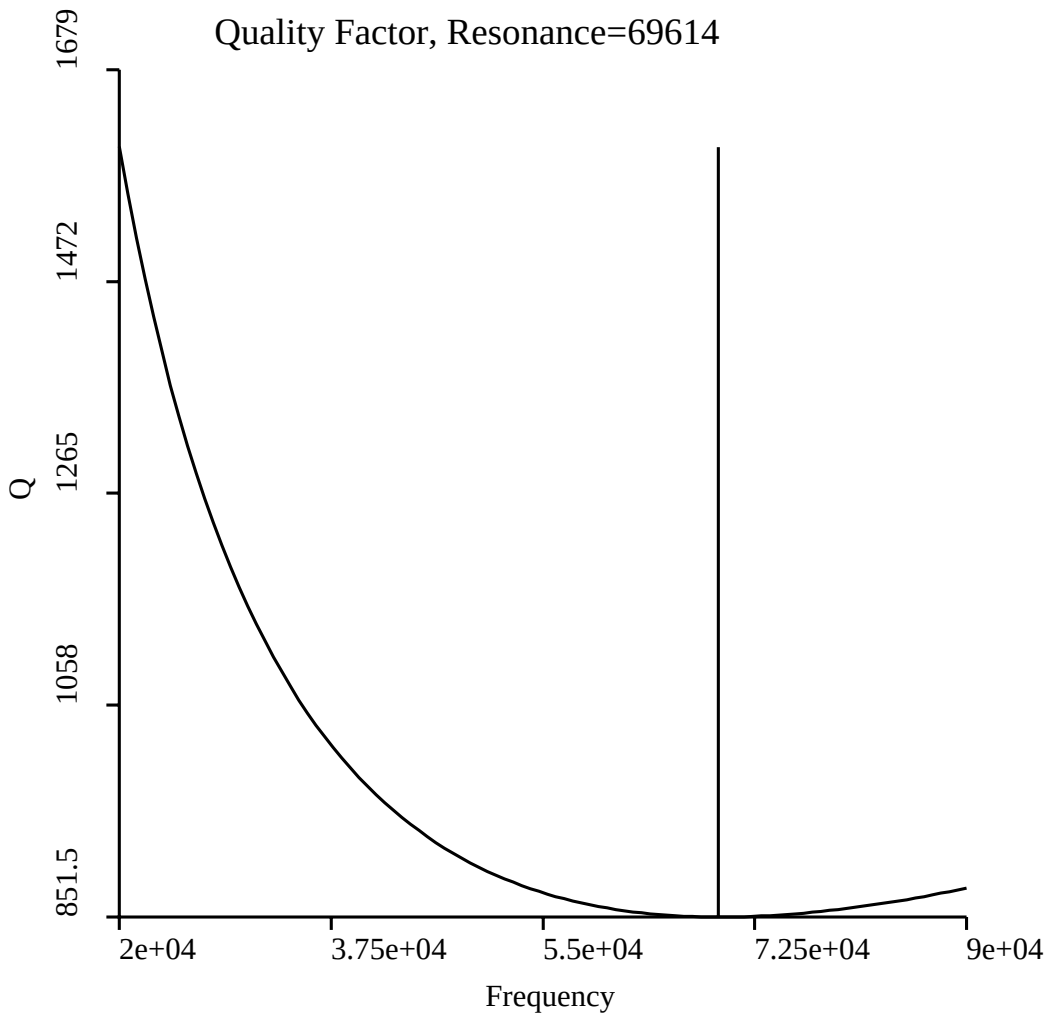


Figure 18: Plot of quality factor as a function of frequency for RLC circuit.

$$C_1 = 3.58 \times 10^{-10} F$$

The so called clamping capacitance is

$$C_2 = 3.91 \times 10^{-8} F$$

```
c qf.ftn print quality factor as a function of frequency
      implicit real*8 (a-h,o-z)
      real*8 l
      f1=20.0e3
      f2=90.0e3
      pi=4.*atan(1.d0)
      n=100
      do i=1,n
       f=(i-1)*(f2-f1)/(n-1) +f1
       omega=f*2.*pi
       q=qf(omega)
       write(*,'(2(g15.8,1x))')f,q
      enddo
      r=7.5
      l=.0146
      c=3.58e-10
      omegar=sqrt(1./(l*c))
      fr=omegar/(2.*pi)
      write(*,'(a,g15.8)')'Resonance frequency = ',fr
      q=omegar*l/r
      write(*,'(a,g15.8)')'Resonance quality factor = ',q
      end
c
      function qf(omega)
      implicit real*8 (a-h,o-z)
      real*8 l
      r=7.5
      l=.0146
      c=3.58e-10
      qf=(omega*l + 1./(c*omega))/(2.*r)
      return
      end
```

91.14 General Networks

To solve general networks we write down the equations for all the loops in the network. We give a name to the current flowing in each loop. Also we may need the equations for the currents flowwing into each node point of the circuit graph. In the case of steady alternating currents, we will get a set of linear algebraic equations with complex coefficients. Which are easily solved using linear algebra.

For transient solutions we will get a similar set of differential equations that are usually solved by approximate numerical techniques.

91.15 A Spice Direct Current Example

Grab some Spice examples for this and the next two sections.

91.16 A Spice Alternating Current Example

91.17 A Spice Transient Example

91.18 Bibliography For Electrical Circuits

[1]Reitz John, Milford Frederick, **Foundations Of Electromagnetic Theory**.
[2]Mentesana C P, **Optical Stronglink Development And Piezoelectric Motor Evaluation**, AlliedSignal KCD, KCP-613-4573, May 92.
[3]Balabanian And Bickart, **Linear Network Theory: Analysis, Properties, Design And Synthesis**, 1981, Linda Hall TK454.2 B36.
[4]Millman Jacob, Taub Herbert, **Pulse And Digital Circuits**, 1956
[5]Ferris Clifford, **Linear Network Theory.**
[6]Del Torro Vincent, **Principles Of Electrical Engineering**, Prentice-Hall, 2nd ed, 1972.

92 What is a Vector Space and What is Linear Algebra?

92.1 Vectors and Linear Algebra

Roughly speaking A Vector Space is a set of objects that can be added together and can be multiplied by a scalar. The scalars are usually real or complex number. More formally the objects of a vector space form an additive group, actually a commutative additive group, known as an Abelian group. By the way, you don't have to live in the suburbs, and go "both ways," to be commutative. Technically, the scalars need only form a field. A field has an addition and a multiplication and in particular must have a multiplicative inverse. Thus the integers are not suitable, because given an arbitrary integer such as say 3, there is no integer multiplied by 3 that gives 1, that is 1/3 is not an integer. The integers do not have multiplicative inverses, and so are not a field.

The ordinary vectors in 3-space having x,y,z components can be added together and multiplied by a scalar. They form a vector space. So we may write unit vectors

$$\mathbf{i} = (1, 0, 0), \mathbf{j} = (0, 1, 0), \mathbf{k} = (0, 0, 1),$$

which are a basis of this space.

But also functions can form a vector space. So consider all polynomials of degree 3 or less. Adding two such polynomials together gives a new polynomial also of degree three or less. And we may multiply such a polynomial by a scaler. A basis is a set of elements in the vector space so that every other element may be written as a linear combination of these elements. So in our case of the polynomials, consider the four polynomials $1, x, x^2, x^3$, which we might call v_1, v_2, v_3, v_4. If V is our vector space, then for each $v \in V$ there exists numbers a_1, a_2, a_3, a_4 so that

$$v = a_1 v_1 + a_2 v_2 + a_3 v_3 + a_4 v_4$$

and no one of the $1, x, x^2, x^3$ can be written as a linear combination of the others. That is the $1, x, x^2, x^3$ are called linearly independent. This is equivalent to the following: if $\{v_1, v_2, ..., v_n\}$ are such that if

$$a_1 v_1 + a_2 v_2 + ... + a_n v_n = 0$$

implies that every $a_i = 0$ then $\{v_1, v_2, ..., v_n\}$ are called linearly independent. Otherwise they are called linearly dependent. A set $B = \{v_1, v_2, ..., v_n\}$ is said to span the space V, if every $v \in V$ can be written as a linear combination of elements of B. If B both spans the space V and is a linearly independent set , then it is called a bases. One can show that every bases has the same number of elements. This number is called the dimension of the vector space. Linear Algebra is the study of finite dimensional vector spaces. So for example the set of all continuous functions on the real line form a vector space, but clearly this space does not have a finite bases, so is not a finite dimensional vector space.

A linear transformation L from one vector space V to another U is a mapping that satisfies:

$$L(av_1 + bv_2) = aL(v_1) + aL(v_1)$$

is called a linear transformation. The name comes from the fact that for points of the geometric plane and a mapping that takes straight lines to straight lines is called linear because straight lines go to straight lines. And this property may be characterized by the linear transformation conditions. given above. So we may talk about solutions to a differential equation being linearly independent, so such solutions form a vector space.

A linear transformation is determined by how it maps each base vector.

Given a basis $B = \{b_1, b_2, ..., b_n\}$ of a n-dimensional vector space V , then every element of V

can be written as

$$v = \sum_{i=1}^{n} a_i b_i$$

The n-tuple $a = (a_1, a_2, ..., a_n)$ is called the coordinate vector of v. If T is a linear transformation and $T(v) = u$, and $c = (c_1, c_2, ..., c_n)$ is the coordinate vector of u, then we can show that there is a matrixM_T determined by T so that

$$c = M_T a$$

That is, a linear transformation has a corresponding matrix, and the transformation can by obtained by multiplying matrices. Further if T_1 and T_2 are two transformations, then the composition transformation

$$T_2(T_1(v))$$

has matrix

$$M_{T_1} M_{T_2}.$$

That is composition is the same as matrix multiplication.

The set of real numbers $\mathbf{R}$ is a vector space over itself. A linear transformation taking a vector space to the reals is a special transformation called a linear functional. The name comes about because an integral applied to a space of functions may take each function to a real number. Such an integral is a linear transformation taking functions to numbers. So it is called a functional, a function of functions.

93 Finite Dimensional Vector Spaces

Linear algebra is the study of finite dimensional vector spaces and linear transformations. A vector space is a quadruple $(V, F, +, *)$. V is a set of vectors, F is a field of scalars, $+$ is the operation of vector addition, and $*$ is the operation of scalar multiplication. We usually do not write the multiplication operator, that is, we write $\alpha * v$ as αv. Let $\alpha, \beta \in F$ and $u, v, w \in V$. The following axioms are satisfied:

1. $u + v = v + u$
2. $u + (v + w) = (u + v) + w$
3. There is a zero element so that $0 \in V$ so that $u + 0 = u$.
4. For each $u \in V$, there is an inverse element -u so that $u + (-u) = 0$
5. $\alpha(u + v) = \alpha u + \alpha v$.
6. $(\alpha + \beta)u = \alpha u + \beta u$.
7. $(\alpha\beta)u = \alpha(\beta u)$.
8. $1u = u$

A finite set of vectors $v_1, v_2, ...v_n$ is linearly independent if

$$\alpha_1 v_1 + \alpha_2 v_2 + ... + \alpha_n v_2 = 0$$

implies that each α_i is zero. Otherwise the set is called linearly dependent. A subset S of V is a subspace of V if the sum of any two elements in S is in

S and the scalar product of any element in F with any element in S is in S. That is S is closed under addition and scalar multiplication. The subspace spanned by vectors

$$v_1, v_2, ...v_n$$

is

$$S = \{\alpha_1 v_1 + \alpha_2 v_2 + ... + \alpha_n v_n : \alpha_i \in F\}$$

Theorem The nonzero vectors $v_1, v_2, ...v_n$ are linearly dependent if and only if some one of them is a linear combination of the preceeding ones.
Proof. Suppose v_k can be written as a linear combination of $v_1, ...v_{k-1}$. Then we have a linear combination of $v_1, ...v_k$ set equal to zero with $\alpha_k = -1$, so that these vectors are linearly dependent. Conversely, suppose $v_1, ...v_n$ are dependent. Then we can find a set of α_i so that

$$\alpha_1 v_1 + \alpha_2 v_2 + ... + \alpha_n v_n = 0,$$

and at least one of them is not zero. Let k be the largest index for which α_i is not zero, then dividing by α_k we find that v_k is a linear combination of the preceeding vectors.
Corollary. Any finite set of vectors contains a linear independent subset that spans the same space.
Theorem. Let vectors $v_1, v_2, ..., v_n$ span V. Suppose the vectors $u_1, u_2, ..., u_k$ are linearly independent. Then $n \geq k$.
Proof. The set $u_1, v_1, v_2, ..., v_n$ is linearly dependent and spans V, so some v_j is dependent on its predecessors. Then the set $u_1, v_1, v_2, ..., v_{j-1}, v_{j+1}, ..., v_n$ spans V, and is dependent. We may continue this, adding a u_i while removing a v_j and still having a set that spans V and is dependent. This can be continued until the u_i are exhausted, otherwise the v_j would be exhausted first and some subset of the $u_1, u_2, ..., u_k$ would then be dependent, which is not possible. Therefore there are more v_j than u_i, which forces

$$n \geq k.$$

Definition A *basis* of a vector space V is a set of linearly indepencent vectors that spans V. A vector space is finite dimensional if it has a finite basis.
Theorem Suppose a vector space has a finite basis

$$A = \{v_1, v_2, ..., v_n\}.$$

Then any other basis also has n elements.

Proof Let $B = \{u_1, u_2, u_3, ...\}$, which is possibly infinite, be a second basis of V. By the previous theorem $n \geq k$ for any subset

$$u_1, u_2, ..., u_k$$

of B. It follows that

$$B = \{u_1, u_2, u_3, ..., u_m\},$$

for some m, and that $n \geq m$. Reversing the role of A and B, we apply the previous theorem again to get $m \geq n$, which proves the corollary. We conclude that the dimension of a finite dimensional vector space can be well defined as the number of elements in any basis. Any vector $v \in V$ can be represented as a linear combination of the basis elements:

$$v = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + ... + \alpha v_n v_n.$$

This representation is unique, because if we subtract two different representations we would get a representation of the zero vector as a linear combination of the basis vectors with at least one nonzero coefficent, which contradicts that the vectors are linearly independent.

The scalar coefficients are called the coordinates of v, and form a element in the cartesian n-product of the scalar field F. These n-tuples of scalars themselves form an n dimensional vector space and are isomorphic to the original vector space. Let U and V be two vector spaces. A function

$$T : U \to V$$

is called a linear transformation if

1. For $u_1, u_2 \in U, T(u_1 + u_2) = T(u_1) + T(u_2)$.
2. For $\alpha \in F, u \in U, T(\alpha u) = \alpha T(u)$.

A linear transformation from U to itself, is called a linear operator. Associated with every linear transformation is a matrix. Let

$$\{u_1, u_2, u_3, ..., u_n\}$$

be a bases of U and let

$$\{v_1, v_2, ..., v_m\}$$

a bases of V. For each u_j, $T(u_j)$ is in V, so that it may be written as a linear combination of the basis vectors, we have

$$T(u_j) = \sum_{i=1}^{m} a_{ij} v_i.$$

Now let $u \in U$ and let its coordinates be $x_1, ..., x_n$. Vector u is represented by the coordinate vector

$$\begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix}$$

We have

$$\begin{aligned} T(u) = T(\sum_{j=1}^{n} x_j u_j) &= \sum_{j=1}^{n} x_j T(u_j) \\ &= \sum_{j=1}^{n} x_j \sum_{i=1}^{m} a_{ij} v_i \\ &= \sum_{i=1}^{m} (\sum_{j=1}^{n} a_{ij} x_j) v_i \\ &= \sum_{i=1}^{m} y_i v_i, \end{aligned}$$

where the $y_1, y_2, ..., y_m$ are the components of the vector $T(u)$ in vector space V. We have shown that the coordinate vector x of u is mapped to the coordinate vector y of $T(u)$ by matrix multiplication,

$$\begin{bmatrix} y_1 \\ y_2 \\ ... \\ y_n \end{bmatrix} = \begin{bmatrix} a_{11} & ..... & a_{1n} \\ a_{21} & ..... & a_{2n} \\ ... & ..... & ... \\ a_{m1} & ..... & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{bmatrix}$$

Now suppose we have two linear transformations

$$T : U \to V,$$

and

$$S : V \to W.$$

The composite transformation is

$$ST : U \to W,$$

defined by $ST(u) = S(T(u))$.
Theorem. If A is the matrix of linear transformation S, and B is the matrix of linear transformation T, then the matrix multiplication product AB is the matrix of ST.

93.1 Determinants

A determinant is a multilinear functional defined on a square matrix. A linear functional is a mapping from a vector space to a field of scalars. A multilinear functional is a function defined on a cartesian product of the vector space. The column vectors of a matrix may be considered to be vectors of the vector space V. The set of n column vectors constitute a point in the cartesian product. The functional is linear in the sense that

$$f(v_1, v_2, ..., v_k + v_k', ..., v_n) = f(v_1, v_2, ..., v_k, ..., v_n) + f(v_1, v_2, ..., v_k', ..., v_n),$$

and

$$f(v_1, v_2, ..., \alpha v_k, ..., v_n) = \alpha f(v_1, v_2, ..., v_k, ..., v_n).$$

A multilinear functional is alternating if interchanging a pair of variables changes the sign of the function.
Definition. A determinant $D(A)$ of an n dimensional square matrix A is the unique alternating multilinear functional defined on the n column vectors of A, which takes value 1 on the identity matrix.
Properties.

1. If two column vectors of a matrix are identical, then the determinant is zero. This follows because interchanging the columns changes the sign of the determinant, but the new matrix has not changed, so the value of the determinant is the same. The determinant must be zero.

2. Adding a multiple of one column to a second does not change the value of the determinant. This is clear from

$$\begin{aligned} &D(v_1, ..., v_i, ..., v_j + \alpha v_i, ..., v_n) \\ &= D(v_1, ..., v_i, ..., v_j, ..., v_n) + \alpha D(v_1, ..., v_i, ..., v_i, ..., v_n) \\ &= D(v_1, ..., v_i, ..., v_j, ..., v_n) + 0. \end{aligned}$$

Example To compute the determinant of

$$\begin{bmatrix} 2 & 2 \\ 3 & 4 \end{bmatrix}$$

subtract the first column from the second, then three times the second from the first, getting

$$\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

So $D(A) = 2D(I) = 2$, where I is the identity matrix. Once we have a matrix in diagonal form, we see from its definition as a multilinear functional, that the determinant is equal to the product of each multiplier of each column times the determinant of the identity. That is, the value equals the product of the diagonal elements.

Example *Cramers's Rule* Suppose we have a system of n equations in n unknowns $x_1, x_2, ..., x_n$ written in the form

$$x_1 v_1 + x_2 v_2 + ..., x_n v_n = v.$$

We have

$$D(v, v_2, v_3, ..., v_n) = D(x_1 v_1 + x_2 v_2 + ... + x_n v_n, v_2, ..., v_n)$$

$$= x_1 D(v_1, v_2, v_3, ..., v_n).$$

So that unknown x_1 is given by

$$x_1 = \frac{D(v, v_2, v_3, ..., v_n)}{D(v_1, v_2, v_3, ..., v_n)}.$$

There is clearly a similar expression for each of $x_2, ...x_n$.

To compute a determinant we can perform permutations on the columns and add scalar multiples of columns to other columns, to put the matrix into diagonal form. Once in diagonal form, (or triangular form) the determinant equals the product of the diagonal elements.

There is an alternate definition of the determinant involving permutations. Let a be a n by n matrix. Consider the sum

$$\sum_{\sigma} s(\sigma) a_{1\sigma(1)} a_{2\sigma(2)} ... a_{n\sigma(n)}$$

where σ is a permutation of the integers $1, 2, 3, 4, ..., n$ and $s(\sigma)$ is the sign of the permutation. The sign of the identitiy permutation is one, and interchanging a pair of elements, a transposition, changes the sigh of the permuntation. Notice that this is a multilinear functional, and is alternating, and further equals one on the identity matrix. Therefore it is the unique such functional, and so is equal to the determinant.

Properties.

1. $D(A^T) = D(T)$.
2. $D(AB) = D(A)D(B)$.
3. Expansion by minors about row i. Let A_{ij} be the matrix obtained from A by deleting row i and column j. Then

$$D(A) = \sum_{j=1}^{n} (-1)^{i+j} a_{i,j} D(A_{ij}).$$

93.2 Permutations

See the paper **Group Theory** by James Emery (Document groups.tex).

93.3 Kernal and Range

Let U be a n dimensional vector space. Let T be a linear transformation from U to a vector space V. Let $K(T)$ be the kernel of T and $R(T)$ the range of T. Then

$$dim(K(T)) + dim(R(T)) = n$$

Let $u_1, u_2, ..., u_p$ be a basis of the kermal of T. It can be extended to be a full basis of U. Suppose V is also n dimensional. Let A be the matrix of T with respect to this basis. Then the first column of A must be all zeroes, so that $D(A)$ is zero. Let U have a second basis and let S be the linear transformation mapping the first bases to the second. Let B be the matrix of S. Then B has an inverse B^{-1} and

$$D(B)D(B^{-1}) = D(BB^{-1}) = D(I) = 1.$$

Then $D(B)$ is not zero. The matrix of T with respect to the second basis is $D(AB) = D(A)D(B)$, so that in general the determinant of the matrix of T with respect to bases of U and V is zero if and only if the kernal of T is not zero, and this is true if and only if T has an inverse.

93.4 Inner Product Spaces

Let V be a vector space with an inner product (u, v) . A basis can be used to construct an orthonormal basis (Gramm-Scmidt orthonormalization). An inner product space is also a normed linear space with the norm

$$\|u\| = (u, u)^{1/2}.$$

The Cauchy-Schwartz inequality is

$$(u, v)^2 \leq \|u\|^2 \|v\|^2$$

The triangle inequality is

$$\|u + v\| \leq \|u\|^2 + \|v\|^2.$$

93.5 Quadratic Forms

A symmetric matrix defines a quadratic form. Every quadratic form can be diagonalized.

93.6 Canonical Forms

Jordon Normal form, Rational Normal form, Triangular form, LU decomposition.

93.7 Upper Triangular Form

For every matrix M, there is a a matrix T so that

$$T^{-1}MT$$

is upper triangular. One may make an eigenvector of M the first column of T, to get a partitioned matrix where the first column has zeroes below the first element of the first column. Then one may apply induction to the remaining lower dimensional submatrix. See Richard Bellman **The Stability Theory of Differential Equations** Dover, for a simple proof.

93.8 Isometries, Rotations, Orthogonal Matrices

An isometry is a transformation that preserves length.

$$||T(v)|| = ||v||.$$

Let T be a linear transformation that is an isometry. Let T have a matrix representation M in some orthogonal basis. Then we shall show that M is an orthogonal matrix. This means that the norm of any column is one, and that any two columns are orthogonal.

We have

$$Mv \cdot Mv = v \cdot v$$

That is, if v is a column vector, then

$$(Mv)^T(Mv) = v^T M^T M v = v^T v.$$

Let v_i be the column vector whose ith element is one, and all other elements are zero. Then letting $P = M^T M$, we see that $P_{ii} = 1$. Letting $v = v_i + v_j$, for i not j, we find that $P_{ij} = -P_{ji}$. But P is symmetric. So $P_{ij} = 0$. Thus P is the identity matrix. So we have shown that the inverse of an orthogonal matrix is its transpose. This also implies that its row vectors are orthonormal. In two space or three space, a rotation matrix, representing a rotation about a fixed axis, is clearly an isometry. Hence it is orthogonal. Because

$$\det(M) = \det(M^T),$$

the determinant must be 1, or -1. Clearly any real eigenvalue of an isometry must be 1 or -1. This follows from the value of the determinant. An orthogonal matrix whose determinant is 1, called proper. A proper orthogonal transformation represents a rotation. A proper orthogonal matrix defines three Euler angles, which explicitly shows that the matrix represents a rotation. See Nobel, **Applied Linear Algebra.** Also for the determination of the rotation axis of the matrix, which is an eigenvector, see the discussion in the paper on rotation matrices contained in an issue of the IEEE Journal on Computational Geometry and Graphics. Also see the computer codes in the Emery Fortran and C++ libraries.

93.9 Rotation Matrices

See the paper **Rotations** by James Emery (Document rotation.tex).

93.10 Exponentials of Matrices and Operators in a Banach Algebra

See See the paper **Rotations** by James Emery (Document rotation.tex).

93.11 Eigenvalues

93.12 Unitary Transformations

93.13 Transpose, Trace, Self-Adjoint Operators

The trace of a matrix is the sum of the diagonal elements. The trace has the following property. If A and B are square matrices then

$$trace(AB) = trace(BA).$$

This follows because

$$\begin{aligned} trace(AB) &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}b_{ji} \\ &= \sum_{j=1}^{n}\sum_{i=1}^{n} a_{ij}b_{ji} \\ &= \sum_{j=1}^{n}\sum_{i=1}^{n} b_{ji}a_{ij} \\ &= trace(BA). \end{aligned}$$

If

$$M' = PMP^{-1},$$

then

$$trace(M') = trace(PMP^{-1}) = trace(P^{-1}PM) = trace(M).$$

93.14 The Spectral Theorem

93.15 Tensors

A tensor is a multilinear functional.

93.16 Projection Operators

93.17 Linear Algebra Bibliography

94 Projective Space

The concept of projective space arises from the theory of perspective projection. Formally, a projective space of dimension n is the set of one dimensional subspaces of a vector space of dimension $n+1$. A set of one dimensional subspaces is essentially the set of lines that pass through the origin. So intuitively consider a two dimensional picture or painting that represents a three dimensional scene. Such a painting will show parallel lines converging to a distant point called a vanishing point. So suppose we are viewing a three dimensional scene, and suppose we concentrate on a point in the scene, say the corner of a house. If we imagine a straight line from our eye to the point, and we are making a painting of the house, then we locate the image of this house corner point on our canvas where this line from our eye to the point intersects the canvas. So a point on our two dimensional canvas corresponds to this line in three space. So this is the essence of the concept of projective space: a point in our n dimensional space corresponds to a line or one dimensional subspace in a $n+1$ dimensional space.

94.1 Computations With Two Dimensional Straight Lines, Homogeneous Coordinates

We will introduce some of the ideas of projective space by discussing straight lines. Consider the equation of a line

$$a_1x + a_2y + a_3 = 0.$$

Let us introduce a third coordinate w, and define a vector p with coordinates

$$p_1 = xw, p_2 = yw, p_3 = w.$$

Then the equation can be written in terms of p by multiplying the equation by w, and using the dot product. The equation becomes

$$A \cdot p = 0,$$

where A is the vector

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

and

$$p = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} xw \\ yw \\ w \end{bmatrix}.$$

The components of p are called homogeneous coordinates. They are determined up to a scalar multiple. That is, if we multiply all three coordinates by the same number, then they are still homogeneous coordinates of the point. If $p_3 = w$ and w is not zero, we may divide the coordinates by w, which makes the third coordinate 1. Often we will scale the coordinates so that $w = 1$, although this is not necessary.

Let us find the line A that passes through the points p_1 and p_2. Since the points must lie on the line, we have two equations that must be satisfied:

$$A \cdot p_1 = 0,$$

and

$$A \cdot p_2 = 0.$$

A solution to these equations is $A = p_1 \times p_2$. It is a solution because the cross product of two vectors is perpendicular to each of the vectors.

We will introduce the concept of a point at infinity by consider the equations of two parallel lines. Let the parallel lines have equations

$$a_1 x + a_2 y + a_3 = 0$$

and

$$a_1 x + a_2 y + b_3 = 0,$$

where a_3 and b_3 are not equal. Let us write these equations with the homogeneous coordinate vector x.

$$a_1 x_1 + a_2 x_2 + a_3 x_3 = 0$$

$$a_1 x_1 + a_2 x_2 + b_3 x_3 = 0.$$

These lines do not intersect at any finite point. Subtracting one from the other, we have $(a_3 - b_3)x_3 = 0$, which implies $x3 = 0$. This indicates that

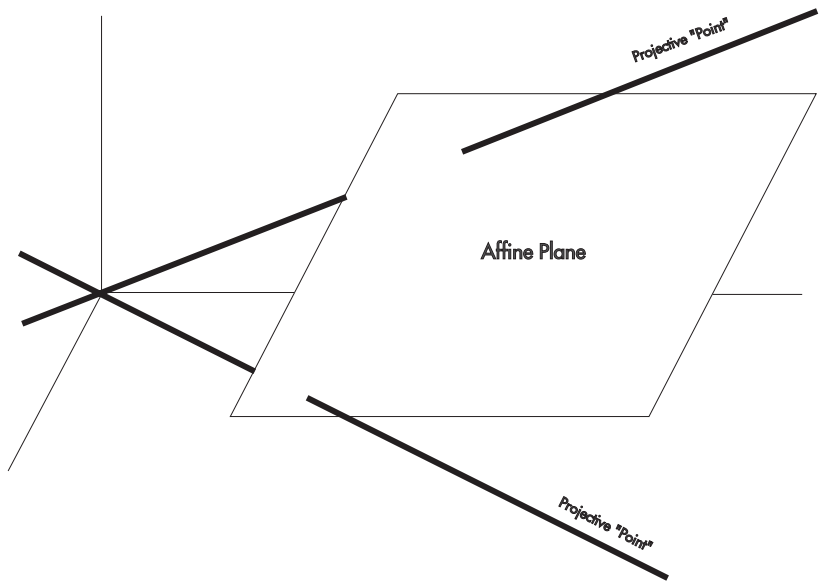


Figure 19: The Affine Plane and Projective Space.

when the third coordinate of a point is 0, the point is at infinity. Let us consider again the line $A = p_1 \times p_2$, which passes through p_1 and p_2. If we subtract p_1 from p_2, we obtain a point $p_1 - p_2$,which is on A, and which is a point at infinity. This follows because

$$A \cdot (p_1 - p_2) = p_1 \times p_2 \cdot (p_1 - p_2) = 0 - 0 = 0,$$

which shows that the point is on A. In doing the subtraction we assume that p_1 and p_2 have been scaled to have the same third coordinate, so that $p_1 - p_2$ has a zero third coordinate, and so is at infinity. Note that the vector $p_1 - p_2$ has the direction of A.

94.2 A Geometric Motivation For Projective Geometry

We have given an algebraic motivation for projective geometry. Next we will give a geometric motivation, and then we will give a general definition of projective space.

Suppose we have a cone with vertex at the origin. Let there be a plane $z = 1$. The intersection of the cone with the plane is a curve, which depending

on the orientation of the cone, will be either an ellipse, a parabola, or a hyperbola. We may identify the lines on the cone with points on the plane. The plane is called the affine plane. The parabola and hyperbola have points at infinity. An ellipse (a circle is a special case of an ellipse) does not have infinite points. In the figure, which shows projective points and the affine plane, one may think of the two dark lines passing through the origin as lying on the cone. They pierce the affine plane as shown. If either of these lines were to move so as to become parallel to the plane, the corresponding intersection point would go to infinity. This means that points at infinity on the plane correspond to lines that are parallel to the affine plane. A parabola results when only one line on the cone does not intersect the affine plane $z = 1$. This occurs when the cone is resting on the $z = 0$ plane. If the $z = 0$ plane intersects the cone rather than being tangent as in the case of the parabola, then there will be two intersection lines, that is, two points at infinity. This is the case of the hyperbola. If no lines on the cone are in the $z = 0$ plane then we get the case of an ellipse. We can represent all of the points on a conic section by lines through the origin, even the infinite ones. Two dimensional projective space is a set of lines through the origin. Lines through the origin are really 1-dimensional subspaces, that is they are the collection of all the vectors that point in the direction of such a line. Homogeneous coordinates are the coordinates of any vector in the direction of the line. Multiplying them by a scaler gives the coordinates of another vector in the direction of the line. We have used the $z = 1$ plane as the affine plane and this is convenient, but we could have used any other plane that does not pass through origin. Those points that we have called, points at infinity, depend on which plane we select. That is, if we choose a different affine plane, then different points are the points at infinity. Namely those lines through the origin that are parallel to the new plane. So points at infinity are not really aspects of projective space itself.

We can use these techniques to do certain simple calculations in an efficient manner. For example, suppose we want the perpendicular bisector of the segment joining two points p_1 and p_2. The perpendicular bisector of the line segment joining the points is

$$M = (p_1 - p_2)^{\perp} \times (p_1 + p_2)/2,$$

where $\perp$ denotes a vector rotated by 90 degrees. M is a line through the point at infinity $p_1 - p_2$, and through the midpoint of the line segment. The

midpoint is $(p_1 + p_2)/2$. If a vector has coordinates $(x, y, 0)$ then the vector rotated by 90 degrees has coordinates $(-y, x, 0)$.

For a second example, suppose again we have two points p_1 and p_2. They define a line A. Suppose we have a point d not on A. Let N be the line parallel to A and through d. Then

$$N = d \times (p_1 - p_2).$$

94.3 Higher Dimensional Projective Space

Higher dimensional projective spaces is defined in the same way as the two dimensional projective space treated here. An n-dimensional projective space is the set of all one- dimensional subspaces (lines through the origin) in some $n+1$ dimensional vector space. We have used three dimensional real euclidean space as the vector space, in which to define our two-dimensional projective space. So a point in two dimensional real projective space is the set all 3d vectors having a fixed direction.

Rational polynomials come about as affine representations of homogeneous polynomials. The canonical conic is

$$r = (t^2, t, 1) = (x, y, z),$$

This is a parabola:

$$x = t^2 = y^2.$$

A general conic is then

$$r' = P(r) = (a_1t^2 + a_2t + a_3, b_1t^2 + b_2t + b_3, c_1t^2 + c_2t + c_3),$$

where P is a projective transformation. The transformations we need are special projective transformations. We need only a rotation and a scaling. Suppose we have a conic section curve. It is the intersection of some cone and the affine plane. We rotate the cone so that its axis is parallel to the z-axis. The new intersection curve is a circle. We scale the x and y coordinates so that the circle is the unit circle (radius = 1). We have shown that any conic section can be mapped to the unit circle by a projective transformation. This can be reversed and a unit circle mapped to any conic. Finally in two such steps any conic can be mapped to any other conic. In the process finite points can be mapped to infinite points and vice versa. A projective transformation is any nonsingular linear transformation in the underlying vector

space considered to operate on the projective points (the one-dimensional subspaces). A special projective transformation that takes finite points to finite points and infinite points to infinite points is called an affine transformation. All points that are finite are related. Affine is derived from words meaning "related to". Thus the affine plane is the set of related or finite points. A projective transformation being essentially a linear transformation can be represented by a matrix. The mapping of a homogeneous coordinate vector can be accomplished by matrix multiplication. Mathematicians tend to use column vectors for points with transformation of points corresponding to multiplication on the left. This is done because it corresponds to the traditional way of writing functions. Computer scientists being somewhat ignorant prefer row vectors (they take up less space on the printed page). Row vectors are established in computer graphics representation of homogeneous coordinates. The programs here listed use column vectors.

94.4 Barycentric Coordinates For Triangular Elements In 2-Space

(also see the sections on barycentric coordinates in graphics.tex) Let α_1, α_2, and α_3 be points in a two dimensional projective space. Let α be an arbitrary point in the space. Then α is a one dimensional subspace of some three dimensional vector space. We wish to assign coordinates to points of the projective space. Suppose p_1, p_2, p_3 are representative vectors of the three basic projective points α_1, α_2, and α_3 respectively. We suppose that the vectors are linearly independent. Let p be in α. Then p has a set of coordinates with respect to the three linearly independent vectors. However, these coordinates will vary depending upon how we select p_1, p_2, p_3 and p from the respective one dimensional subspaces. However, suppose we identify a fourth projective point, α_u, which we take to have unit coordinates: $\lambda_1 = 1, \lambda_2 = 1$, and $\lambda_3 = 1$. Now because a projective point is a one dimensional subspace, any constant multiple of these coordinates will have to represent the same projective point. That is, for example, the unit point will also have coordinates, $\lambda_1 = 5, \lambda_2 = 5, \lambda_3 = 5$. The unit point forces us to select a certain set of basis vectors. These are vectors p_1, p_2, p_3 so that if $p_u \in \alpha_u$, then

$$p_u = cp_1 + cp_2 + cp_3,$$

for some number c. Then p_1, p_2, p_3 are well defined, (except that they could all be scaled by the same number). Then the coordinates of any α are well defined. That is, if $p \in \alpha$, then

$$p = \lambda_1 p_1 + \lambda_2 p_2 + \lambda_3 p_3.$$

And so these are the well defined projective coordinates defined by the projective space reference points α_1, α_2, α_3 and the unit point α_u.

We shall introduce a special projective coordinate system. Let p_1, p_2, and p_3 be the vertices of a nondegenerate triangle. The barycentric coordinate system is a projective coordinate system with reference points $[p_1], [p_2], [p_3]$ and unit point $[(p_1+p_2+p_3)/3]$. $(p_1+p_2+p_3)/3$ is the center of gravity (the barycenter) of the triangle. If p is an arbitrary point, then

$$p = \lambda_1 p_1 + \lambda_2 p_2 + \lambda_3 p_3$$

We may assume that the coordinates are scaled so that

$$\lambda_1 + \lambda_2 + \lambda_3 = 1.$$

The barycentric coordinates are also vector space coordinates in the three dimensional vector space with reference vectors p_1, p_2, p_3. Here we shall think of them in that way. We will deduce some relationships for the coordinates that hold for permutations of the basis vectors p_1, p_2, p_3. So let σ be a permutation of 1,2,3. (see fecrrnt.tex) for a continuation of this material.

94.5 Barycentric Coordinates In Euclidean N-Space

Suppose we are given an n-simplex with vertices $v_0, v_1, v_2, ..., v_n$. The barycentric coordinates of a point p sum to one. If the coordinates satisfy

$$0 < \lambda_i < 1,$$

then the point is an interior point of the simplex. If any coordinate is negative, then the point is exterior to the simplex. If

$$0 \leq \lambda_i \leq 1,$$

then the point is in the interior or on the boundary of the simplex. In the case

$$0 \leq \lambda_i \leq 1,$$

when a coordinate $\lambda_j = 0$, the point is on the boundary of the simplex opposite the vertex p_j.

To find the barycentric coordinates we may select an arbitrary vertex, say p_n, and solve the linear system

$$\sum_{i=0}^{n-1} \lambda_i (p_i - p_n) = p - p_n,$$

for $\lambda_0, ..., \lambda_{n-1}$. Since the barycentric coordinates sum to 1, this also determines λ_n.

Let us apply this to the problem of determining that a point is in a triangle of the plane. Suppose we are given the triangle vertices

$$p_1 = (1, 2),$$

$$p_2 = (1, 3),$$

$$p_3 = (2, 3).$$

and wish to determine if $p = (1.5, 2.6)$ is in the triangle. Our linear system is

$$\begin{bmatrix} (1-2) & (1-2) \\ (2-3) & (3-3) \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} (1.5-2) \\ (2.6-3) \end{bmatrix}.$$

The solution is

$$\lambda_1 = .1, \lambda_2 = .4$$

Then we compute $\lambda_3 = .5$. Therefore, because all coordinates are between 0 and 1, the point is in the triangle.

The general computation to determine an interior point, requires 11 additions or subtractions, 6 multiplications, 2 divisions, and 3 comparisons. The computation may be done as follows.

Let $a_{11} = x_1 - x_3$,$a_{21} = y_1 - y_3$,$a_{12} = x_2 - x_3$,$a_{22} = y_2 - y_3$, and $b_1 = x - x_3$,$b_2 = y - y_3$. Then letting D be the determinant

$$D = a_{11}a_{22} - a_{21}a_{12},$$

we have

$$\lambda_1 = \frac{b_1 a_{22} - b_2 a_{12}}{D},$$

$$\lambda_2 = \frac{a_{11}b_2 - a_{21}b_1}{D},$$

and

$$\lambda_3 = 1 - \lambda_1 - \lambda_2$$

See the C procedure **bary2**.

94.6 Barycentric Coordinates Relative To A Triangle In Space

Let us now consider the problem of computing the barycentric coordinates of a point P with respect to a triangle in space with vertices $P_1, P_2 P_3$. Let

$$A_1 = P_1 - P_3,$$

$$A_2 = P_2 - P_3,$$

$$A = P - P_3.$$

We shall set up a system of orthogonal vectors U_1, U_2, U_3 Let

$$B_3 = A_1 \times A_2,$$

$$B_2 = B_3 \times A_1.$$

Let

$$U_1 = \frac{A_1}{\|A_1\|},$$

$$U_2 = \frac{B_2}{\|B_2\|},$$

$$U_3 = \frac{B_3}{\|B_3\|} = U_1 \times U_2.$$

Using the "Back Minus Cab" rule, we have

$$B_2 = -(A_1 \cdot A_2)A_1 + (A_1 \cdot A_1)A_2.$$

Let P' be the projection of P to the plane of the triangle (which is P if P is already in that plane). Then we have

$$P' = (A \cdot U_1)U_1 + (A \cdot U_2)U_2$$

$$P_1 = (A_1 \cdot U_1)U_1 + (A_1 \cdot U_2)U_2$$
$$P_2 = (A_2 \cdot U_1)U_1 + (A_2 \cdot U_2)U_2$$

Then we have the two dimensional case given above where the points P, P_1, P_2, P_3 have coordinates with respect to U_1, U_2

$$x = A \cdot U_1$$

$$y = A \cdot U_2$$

$$x_1 = A_1 \cdot U_1$$

$$y_1 = A_1 \cdot U_2$$

$$x_2 = A_2 \cdot U_1$$

$$y_2 = A_2 \cdot U_2$$

and

$$x_3 = 0$$
$$y_3 = 0$$

The Fortran subroutine **baryt** implements this calculation.

94.7 Barycentric Coordinates Relative To A Line Segment In Space

Given a line segment with endpoints p_1 and p_2, and a point q, we wish to compute barycentric coordinates of the normal projection q' of q onto the line through p_1 and p_2. That is

$$q' = \lambda_1 p_1 + \lambda_2 p_2,$$

where

$$\lambda_1 + \lambda_2 = 1.$$

Let

$$u = \frac{p_2 - p_1}{\|p_2 - p_1\|}.$$

We have

$$q' = p_1 + ((q - p_1) \cdot u)u = p_1 + \lambda_2(p_2 - p_1) = (1 - \lambda_2)p_1 + \lambda_2 p_2.$$

So that

$$\lambda_2 = \frac{(q - p_1) \cdot (p_2 - p_1)}{\|p_2 - p_1\|^2},$$

and

$$\lambda_1 = 1 - \lambda_2.$$

As an example of the use of this calculation, suppose one wishes to compute the intersection of two line segments. Specifically in the plane one may use cross products to obtain an intersection point q of the two lines containing the segments. Then one may compute the barycentric coordinates with respect to each of the line segments to check that all such coordinates are nonnegative, which means that the intersection point is common to both segments.

94.8 Projective Space, Conic Curves, Quadric Surfaces, and Quadric Solids

Quadric solids are solids bounded by quadric surfaces. Quadric solids provide a rich collection of sets for constructing models of 3-dimensional objects. A large set of quadric solids may be built by combining primitive quadric solids, such as spheres, ellipsoids, blocks, cones, wedges, and cylinders. Many physical objects are combinations of quadric shapes. There are 17 types of quadric surfaces, including such less common surfaces as the parabolic hyperboloid.

The world is full of objects built from rectangular blocks, circular cylinders, and spheres. These objects are quadric solids. Polyhedra are quite common in nature. Minerals and crystals often take a polyhedral form. Some viruses have the shape of a polyhedron. Polyhedra are quadric solids becuase they can be built from planar half spaces, and planes are quadric surfaces (pairs of planes). Essentially all objects can be approximated, to an arbitrary accuracy with polyhedra. More specifically, their surfaces may be approximated with triangles.

But a polyhedron is locally flat and has no local curvature. So the planar faces of a polyhedron can not model a local curved shape exactly. An advantage of using quadric surfaces is that they can locally approximate curvature

and shape [O'Neil p.202]. An object can be represented by a relatively small number of primitive quadric solids. If we regard atoms as tiny balls, then in a sense, all objects are built from spheres. But Avogadro's number 6.03×10^{23} is very large, so modeling with spheres is somewhat impractical. However, in the field of scientific visualization we do model with rectangular cells defined by very large numbers of lattice points. When modeling simple manufactured parts, we are interested in finding exact models. Quadric solids can serve well in such modeling.

Quadric solids by themselves do not give a full practical set of primitives for modeling. For example, surfaces of revolution are ubiquitous. But most surfaces of revolution are not quadric surfaces. Further a large number of quadrics are frequently required to get an acceptable approximation to a surface of revolution. The torus is a surface of revolution that is not a quadric surface. A torus could be approximated by a set of cylinders. But the differential geometric properties would not be approximated. Parametric free form surfaces, such as spline surfaces, are widely used in manufacturing, but are not quadric surfaces.

Computations with quadric solids are simple. The calculation of a normal direction, at a point on a quadric surface, is accomplished by a matrix multiplication. The intersection points of a straight line with a primitive quadric solid require only the solution of a quadratic equation. The transformed equation of a translated or rotated quadric surface is easily found.

With the advent of the computer age, there has been a renaissance of calculation. Many well known algorithms and calculating methods, while theoretically powerful, were formerly of little interest. They were two burdensome for the human computer. In the past techniques were emphasized that could be carried out easily by hand. But now those calculations, which were too lengthy for hand calculation, are proper for computer modeling and computer graphics. Frequently the novice in geometric modeling and graphics does not use the proper tools. Naturally he uses the calculations taught him in the schools and colleges, which are often only the small amount of analytic geometry found in the calculus course, and deals, for example, with the simultaneous solution of a system of equations in a fixed coordinate system. A vector approach is superior for many calculations. The vector methods tend to be independent of the coordinate system. Those with scientific education are usually familiar with vectors. But they are frequently unaware of some very useful mathematics. This is the mathematics of projective geometry, projective space, and specifically the idea of homogeneous coordinates.

Projective space is very important in certain advanced areas of mathematics. It is an important aspect of the field of algebraic geometry. But even among mathematicians, knowledge of projective and algebraic geometry is not universal.

Often the course in projective geometry, which is offered in American universities and colleges, is designed for future teachers of high school mathematics. This course, or a course called modern geometry, is taught to show the future teacher of Euclidean geometry, that there are many geometries, and and that not all of the geometries are Euclidean. Niether the course on projective geometry, nor the books designed for it, emphasize the beautiful simplifications and generalizations of calculation that come from treating points as belonging to projective space. The few pages discussing projective space in Birkhoff and MacLane (**Survey of Modern Algebra**), are more illuminating than the numerous pages of the elementary books on projective geometry. This is because the connection to linear algebra and vector spaces is not exploited in these books.

The purpose of this report is, first to describe a modeling system which uses quadric solids as primitive objects, and second to present some elementary ideas of projective geometry which are useful for computing. These latter ideas, although elementary, are not readily available. The quadric solid modeling system is implemented in a computer program. An original version was called **Quadric**. A later version is called **QS90.FTN**. It produces views of 3-dimensional objects, calculates volumes, calculates moments, and calculates inertia tensors.

The idea of representing objects as primitive solids was quite fashionable in the 1980's. Now most solid modeling systems are surface modeling systems. But primitive solid modeling still has some interest. Quadric solids are obvious candidates for the primitives. There are several papers dealing with quadric surfaces [Leven].

The modeling system described here represents objects as Boolean combinations of quadric half space primitives. An image of the model is made by scanning its surface and calculating an illumination intensity at each surface point. The 2-dimensional picture plane is scanned to construct intensity values. A surface normal is calculated at the point on the surface, where the line through the picture point pierces the object. The intensity value is determined from the inner product of the normal at the first pierce point with an illumination vector. Since the primitives are quadric solids, this can be done easily.

A quadric surface is the set of real zeros of a quadratic form in four dimensions. Such a set of zeros is called a variety in algebraic geometry. A quadratic form is a polynomial in which each term is of the second degree. The set of points were the form takes nonpositive values is call a lower half space. An upper half space is the set where the form takes nonnegative values. A quadric form defines two quadric half spaces. Examples of half spaces are the points inside an ellipsoid, and the points outside two parallel planes.

It is easy to find the intersection points of a quadric surface with a straight line. The parameters of the intersection points on the line will be the roots of a quadratic equation. Surface normals are also easily calculated.

The objects are built using the standard set operators, union, intersection and complementation. Some modeling schemes, for example the **PADL** system, which was developed at the university of Rochester, uses regularized set operators. These operators guarantee that the resulting objects are topologically closed, and homogeneously three dimensional. Using standard operators one may get a set that is not closed. Because we have only an approximate model of the surface and do not explicitely calculate the boundary, the use of regularized operations does not seem appropriate. For intersection points which are candidates to be on the boundary, We construct a small line segment in the direction of the normal that contains the point. If one end of the segment is inside, and the other outside, the point is a boundary point of the object.

94.9 Overview

We shall start with two-dimensional projective space. This space is easy to visualize. Projective 2-space is the set of lines in 3-dimensional space which pass through the origin. Projective 2-space is the set of one dimensional subspaces of a 3-dimensional vector space. A plane that does not pass through the origin is given a special role. It is called the affine plane. A coordinate system is usually selected so that the affine plane is the plane $z = 1$. The idea of projective space can be illustrated by considering the intersection of a cone and a plane. The intersection curve is called a conic section. See Fig. 1.

When the cone is rotated, the conic section is transformed to a new conic section. A circle may go into a hyperbola, a parabola, or an ellipse. The lines on the cone are points of projective 2-space. The intersection of these

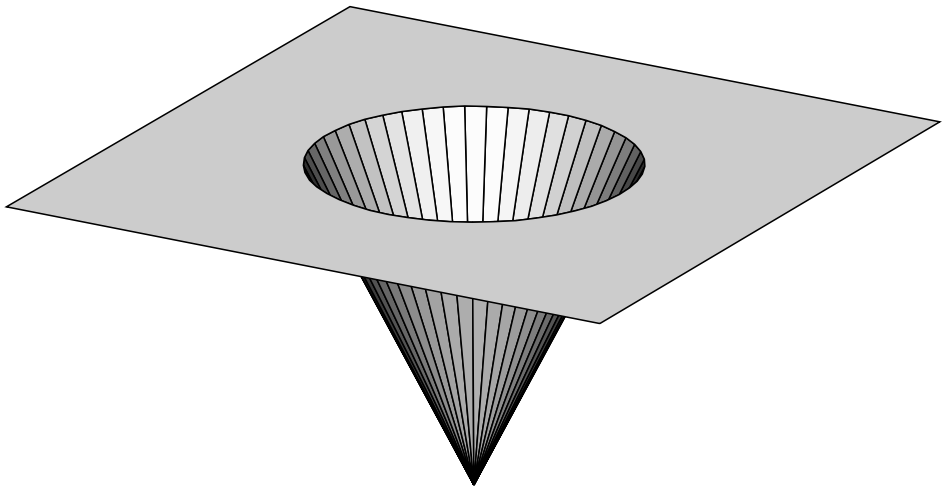


Figure 20: Points of projective 2-space are lines in 3-space. A 3-d linear transformation is a 2-d projective transformation. A rotation of the cone can project the circle to an ellipse, a parabola, or a hyperbola. In the same way the lines through any figure in the plane are a projective space representation of the figure. Any sequence of perspective projections of a figure onto a set of planes can be represented as a sequence of linear transformations in 3-space. The plane shown in the figure is known as the affine plane.

lines with the affine plane are the representation of the points in affine space. However the projective points (i.e. lines) and the affine points are to be considered the same objects, namely 2-dimensional points. The components of any vector which lies in the subspace representing a projective point, are called the homogeneous coordinates of the point. Homogeneous coordinates are determined only up to a scalar multiple. A line in 3-space which is parallel to the affine plane meets it at infinity. The projective point in the direction of this line is called a point at infinity. Tts z coordinate will be zero. The property of being a point at infinity depends on the particular affine plane selected.

In Fig. 1, if after the cone is rotated to produce an ellipse in the plane, we fix the plane to the rotated cone and then rotate the cone back to its original position, then we see that we have a perspective projection from the circle on one plane to an ellipse on a second plane. And we see that such a perspective projection is essentially the same as the 3d linear transformation, namely the original rotation of the cone. Again suppose a figure in the plane is translated in the plane with the lines joining the points of the figure following the translation. This is clearly a linear transformation of the lines or more precisely of the vectors in the direction of the lines. The plane figure would be translated the same if we were to translate the origin or cone vertex with the translation of the plane figure. So we see that the shifting of the point of perspective projection is equivalent to a linear transformation of the 3-d space. So suppose we project a figure from one plane to a second, and then from the second figure to a third figure on a third plane from a second perspective point. Then we see from what we have said before that this is equivalent to a set of linear transformations of the fixed 3-d space, namely rotations, translations and possibly a uniform scaling. So the study of these sequences of perspective projections, which were the original subject of projective geometry may be studied by considering linear transformations of one dimensional subspaces of a 3 dimensional vector space, or of a n dimension vector space in general. One can illustrate this equivalence easily in the one dimensional case by drawing sequences of perspective projections of points on lines, and then showing how the planes can be lined up parallel by rotations and translations so that there is a single perspective projection point, projecting points onto a stack of parallel planes, which by scaling can be made to coincide.

Now consider a 2-dimensional line. We may view it as a line lying in the affine plane. The points of this line in projective space, will be the locus of

lines which pass through it and the origin. It is a plane in 3-space. Because any two planes with a point in common must coincide or meet in a line, it follows that any 2-dimensional lines in projctive space meet in a point. Two parallel lines meet at a point at infinity. Notice that 2-dimensional projective space contains more points than affine space, that is it contains the points at infinity. In two dimensional space an affine transformation is the sum of a linear transformation and a translation. If the linear transformation is orthogonal (i.e. a rotation) then the affine transformation is a rigid motion or a congruence. Now any nonsingular linear transformation maps the set of one dimensional subspaces onto itself. So we may consider such a mapping as taking 2-dimensional projective points to projective points. Such a mapping is thus called a projective transformation. These transformations distinguish euclidean, affine, and projective geometry . This is the thesis of Felix Klein. This is the essence of his famous Erlangen program. He defined a geometry to be the study of the invariants of a group of transformations. Thus the concept of angle is a property of euclidean space because angle is invariant under a rigid motion. Parallelism belongs to affine geometry because it is preserved under an affine transformation. Being a conic section is a property of projective geometry, because it is preserved under a projective transformation. Being an ellipse, however, is not a projective property. There is a projective transformation that takes an ellipse to a different type of conic section.

We shall now discuss one of the major reasons for using homogeneous coordinates for calculations. An affine transformation is a special case of a projective transformation. Since a projective transformation is in essence a linear transformation in a higher dimensional space, it can be represented by a matrix, and the transformation of a point may be accomplished with matrix multiplication. An affine transformation is represented as a matrix multiplication. There many advantages to using homogeneous coordinates. Quadrics are represented homogeneously as quadratic forms, and can be represented as symmetric matrices. Perspective views are projections from projective 3-space to projective 2-space. A three dimensional vanishing point, that is a point at infinity, may be projected to a finite two dimensional point, i.e. the vanishing point of converging railroad tracks. The intersection point of two lines can be calculated without considering slopes, or without considering parallelism. That is, the intersection algorithm does not need to consider exceptional cases, such as the case of an infinite slope. The intersection of two parallel lines, which gives a point at infinity, does not

require a special case.

The idea of the polar of a point with respect to a quadric plays a large role in the theory of quadrics. A quadratic form is represented by a symmetric matrix. There is an associated bilinear form, which is a bilinear functional. This is A function from a vector product space to a scalar field. A real bilinear functional is a mapping from the cartesian product of vector spaces to the real numbers, which is linear in each variable. For a fixed point or pole p, the polar is the set of points such that the bilinear form corresponding to the quadric vanishes on any pair of points consisting of p and a point from the set. In two dimensions the polar is a line, in three dimensions it is a plane. If p is on the surface of the quadric, the polar is the tangent plane. This gives the normal vector. If p is at infinity, the polar is a diameter of the quadric.

The cross ratio is an important invariant in projective geometry. It is ratio of distances between four collinear points. The cross ratio is used to prove several properties of projective space.

We shall require the intersection of a line and a quadric. A line is defined by any two distinct points, including the case of points at infinity. The intersection points are determined by the roots of a quadratic equation. There are several cases: (1) There may be two intersection points, (2) There may be no real intersection points, (3) There may be a tangent point, and (4) The line may be a ruling. A ruling is a straight line that lies entirely in the surface of the, quadric. For example, bilinear interpolation on a rectangle produces a ruled quadric surface.

Conic arcs have been used in various applications of geometry. For example, they are used in the **APT** program, which generates machine tool paths. A conic arc is simply a portion of a conic section. A projective transformation exists that takes four points to four points. Thus if we desire a given conic arc defined by four points, we construct the transformation that takes a given parabola, which is situated so that its points are a function of one parameter, to the desired arc. Then We have a parametric representation of the arc. One advantage is that sines and cosines do not have to be calculated for the points of a circular arc.

Surface normals can be considered points at infinity. A vector field is a mapping that assigns a vector to a given point. The surface normal is a vector field defined on the surface. But this field is really only a direction field and a direction is essentially a point at infinity. Thus we may consider a surface normal as a point at infinity.

Three dimensional projective space is defined in the same way as 2-dimensional space. It consists of all 1-dimensional subspaces of a 4-dimensional vector space. The affine plane becomes a 3-dimensional hyperplane. Since we can not climb into 4-dimensional space,we can not visualize projective space from outside as we did in the 2-dimensional case.

Objects are constructed from quadric half-space primitives. There are only two bounded primitives, namely the sphere and the ellipsoid. We create bounded objects by suitably combining the primitives. To each set corresponds a characteristic function which takes a value of true or false (or 1 and 0) on each point of space. Its value at a point x is true if and only if x is in the set. The mapping which takes a set to its characteristic function, union to "and", intersection to "or", and complementation to "not", is a Boolean algebra isomorphism from sets to the algebra of propositional functions. The latter algebra is represented in computer programming languages. Thus in our program objects are represented as logical expressions. Actually algebraic expressions are binary trees [Knuth], so we may consider our object a binary tree with primitives at the terminal nodes and operators at the other nodes.

We determine the characteristic function of a primitive from the corresponding quadratic form. Then the characteristic function of the object comes directly from the definition of the object as a logical expression. Using topology, we may show that a point is on the surface of the object only if it is on the surface of some primitive, and the normal to the surface at a point on the surface is equal to the the normal of some primitive. If the surface point lies on more than one primitive, then either the primitives are tangent, or they are not. In the former case they have the same normal, in the later case the point lies on the intersection curve and hence is on the boundary between regions where the normal belongs to one, or to the other primitive. Eeither can be assigned. Usually when a point lies on several primitives, it is not important how the normal is assigned since the intersection of the primitives will be of measure zero and so is negligible in any view or integration. a slight problem is the specification of the primitives.

Although all primitives are specified by symmetric 4 by 4 matrices, it may be difficult in practice to determine the coefficients from their ordinary characterization. The primitives are most easily specified by entities such as radius, center, major axis, and orientation. We know the matrices of certain well situated primitives. For example, we know the matrix of the sphere of radius one that has its center at the origin. We may transform a simple

primitive to obtain the matrix of a more complex primitive. Transformations employed are rotation, scaling, and translation. For example, by scaling, the unit sphere can be made an ellipse. It then can be rotated and translated. the new matrix of the transformed quadric can be obtained by multiplying the original matrix on the left and the right by matrices related to the transformations.

The motivation for constructing the modeling system was the desire for a system that would compute the volume, center of gravity and the inertia tensor. Since the characteristic function is available, a Monte Carlo integration method is suggested. This technique consists in assigning a probability measure to a bounded space containing the object, and then obtaining the value of the integral as the expectation of some random variable. For example, the volume integral is defined as the expectation of the characteristic function. The expectation is estimated from a random sample of the random variable. In the case of the volume, the variance of the characteristic function is quite high. Thus a large number of samples is required so that the sample variance, which is a measure of the error of the estimated integral, is small. Stratified sampling will often decrease the sampling variance. In stratified sampling one breaks up the domain into smaller sub domains so that the variance is smaller in each subdomain. For example, if a small block is chosen completely inside the object then the variance of the characteristic function is zero. Likewise the variance of the characteristic function completely outside the function is zero. Only those blocks which contain boundary points contribute to the variance. Stratified sampling should decrease the overall variance and so fewer sample points should be required for the error to be within a given tolerance. When only one point is sampled from each subdomain we have essentially a Riemann sum for the integral.

in general, the amount of work required to evaluate an integral numerically is proportional to the number of function evaluations required in one dimension raised to a power equal to the dimension of the space. This is true at least when the multiple integral is computed by iterated integration. Hence if 100 function evaluations are required in one dimension, 1,000,000 evaluations are required in three dimensions, so it pays to reduce the dimension.

By applying the divergence theorem the integral over a volume can usually be reduced to the integral over a surface; so we need a technique for calculating surface integrals. To integrate on a surface we must have some model of the surface. In this modeling system we do not have such an explicit

model. We have a multiple surface patch defined by projection. When the object is projected to two dimensions, to every point in the two dimensional region corresponds a set of points where the projection line pierces the object. This defines a multiple valued function.

A manifold is the mathematical generalization of a surface. It is a space together with a set of coordinate systems or surface patches. For a two dimensional manifold, a surface patch is a 1-1 mapping from the manifold to an open set in 2-dimensional euclidian space. A complete set of patches that covers the manifold is called an atlas. The projections from several projection points of an object will be a set of patches that completely cover the surface of the object. However, when we include all pierce points, these mappings are not 1-1, so are not legitimate coordinate systems. Still we can do integration on these multilayer patches, because integration is loosely an adding up of all infinitesimal surface elements, and we may add the elements in any order. We can add the elements corresponding to pierce points of a given projection line all at once. There is a difficulty, the set of patches that define a manifold are not disjoint, so that when we add the results of integrating each patch, we may well be adding a given surface element many times. We need a partition of unity. A partition of unity is a set of weighting functions applied to each patch so that the sum of the weighted patch integrals is the whole surface integral [**Spivak**]. When our surface patches derive from projections in the x, y, and z directions a natural partition of unity is the set of squares of the components of the surface normal vectors. The individual layers of each view must be considered as separate patches. So we have reduced our task to the calculation of a two dimensional integration over a rectangular domain. the integrand will in general be discontinuous and we must choose a suitable integration algorithm. Gaussian quadrature is not suitable because the integrand will be discontinuous. The generalized trapezoid method will work, but may require more calculation and give less accuracy than necessary.

Because the moment of inertia about any axis can be obtained from the inertia tensor, it is economical to compute the full inertia tensor rather than just the moments of inertia. There are two different tensors that are called the inertia tensor. One definition is given in mathematics books, and another in some physics books. In order to clarify what is being calculated, I have included material concerning angular momentum, angular velocity, moment of inertia, and their relation to the inertia tensor. In the next section I will begin a more detailed treatment of the material.

94.10 Projective Space and the Conic Sections

Projective geometry is related to the theory of perspective. Perspective was developed by the Italian painters during the renaissance. To explain perspective, consider the eye as the origin in a 3-dimensional coordinate system. Let a ray go out from the eye. Suppose this ray impinges on some object at a point. In some sense, the ray represents the point. Let a picture plane be interposed between the eye and the object. Then the point where the ray intersects the picture plane is the projection of the object point to its picture image. Through this process, three dimensional space is projected to two dimensional space. The concern of the artist is to accurately represent the object as it appears to the eye. The artist is concerned with the appearance of parallel lines and vanishing points. Vanishing points are the images of points and lines that are infinitely far away. Such points are called points at infinity. Spatial points at infinity will be discussed in the section on perspective. Here we confine ourselves to the two dimensional case. the artist Albrect Durer, who dabbled a bit in geometry (see Panofsky) ,constructed a device for recording the perspective image of an object in the picture plane by recording the position on the plane of a chord stretched from a fixed point to a point on the object. This device appears in one of Durer's works and may be familiar to the reader. The idea of projective geometry is to consider, the chord, or more accurately the mathematical ray, as representing a two dimensional point.

projective space is formally defined as the set of 1-dimensional subspaces of a vector space. it can be illustrated by considering those plane curves which are called conic sections. Figure 1 shows a cone with its axis in the z direction, and with a 90 degree vertex angle.

This cone meets the plane $z = 1$ in a circle. A point on the surface of the cone satisfies the equation

$$x^2 + y^2 - z^2 = 0$$

The circle in the plane has the equation

$$x^2 + y^2 - 1 = 0$$

The first equation can be written as

$$\begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 0$$

$$P^T AP = 0$$

where

$$P = \begin{bmatrix} x \\ y \\ x \end{bmatrix}$$

and

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

The function $Q(P) = P^t AP$ is a quadratic form. The cone is the set

$$S(Q) = \{P : Q(P) = 0\}.$$

Let L be a nonsingular linear transformation with matrix B.

Then L maps the set $S(Q)$ to

$$L(S(Q)) = \{P : Q(L^{-1}P) = 0\}$$

$$= \{P : (B^{-1}P)^T A(B^{-1}P) = 0\}$$

$$= \{P : P^T(B^{-1})^T A(B^{-1}P) = 0\}$$

$$= \{P : (B^{-1}P)^T A(B^{-1}P) = 0\}$$

$$= \{P : L(Q)P = 0\}.$$

where $L(Q)(P) = Q(L^{-1}(p))$. $L(Q)$ is a quadratic form with symmetric matrix

$$(B^{-1})^T AB^{-1}.$$

Let L be the transformation of a rotation about the x-axes by an angle a. Let $\cos(a) = c$ and $\sin(a) = s$. Then the matrix of L is

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{bmatrix}$$

and $B^{-1} = B^T =$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix}$$

Then $L(Q)$ has matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c & \\ 0 & -s & c \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & \\ 0 & -s & c \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & -s & -c \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c^2 - s^2 & -2cs \\ 0 & -2cs & s^2 - c^2 \end{bmatrix}$$

When the angle a is 45 degrees, the matrix of $L(Q)$ is

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

The equation of the rotated cone is

$$\begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} x \\ -z \\ -y \end{bmatrix}.$$

$$= x^2 - 2yz = 0.$$

The equation of its intersection with the plane $z = 1$ is

$$x^2 - 2y = 0.$$

or

$$y = \frac{x^2}{2}.$$

This is the equation of a parabola.

When the angle a is 90 degrees, the equation of the cone becomes

$$\begin{bmatrix} x & y & \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & -1 & 0 & y \\ 0 & 0 & 1 & z \end{bmatrix}$$

$$= x - y + z = 0.$$

The equation of its intersection with the plane $z = 1$ is

$$y - x = 1.$$

This is the equation of a hyperbola. the plane that cuts the cone is called the affine plane. for each point in the plane there is a corresponding line which passes through the point and through the origin. The set of lines that pass through the origin is called projective space. Each line is considered a point of the space. Those points in projective space (i.e. lines in 3-dimensional Euclidean space) that are parallel to the affine plane are called points at infinity, because they meet the plane only at infinity. There are more projective points than affine points. The coordinates of any vector in the direction of a line which passes through the origin are the homogeneous coordinates of the projective point represented by the line. Because the coordinates of any scalar multiple of the vector are also homogeneous coordinates of the projective point, homogeneous coordinates are determined only up to a scalar multiple. So $(1, 2, 3)$ and $(2, 4, 6)$ are homogeneous coordinates of the same point. A point at infinity has coordinates of the form $(x, y, 0)$. If a point with coordinates (x, y, z) is not an ideal point (e.g. a point at infinity), then its affine coordinates are $(x/z, y/z)$.

One of the properties of projective space is that every pair of distinct lines meet in exactly one point. By a line we mean the locus of points that satisfy an homogeneous equation of the form

$$ax + by + cz = 0.$$

The corresponding affine equation is

$$ax + by + c = 0.$$

Then lines which are parallel in the affine plane meet at a point at infinity. For example, the parallel lines with affine equations

$$ax + by + c = 0.$$

and

$$ax + by + d = 0.$$

meet at the point, which has coordinates $(-b, a, 0)$. Affine rotations and translations are represented as linear transformations in projective space. A rotation has matrix

$$R = \begin{bmatrix} \cos(a) & \sin(a) & 0 \\ -\sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Thus

$$\begin{bmatrix} \cos(a) & \sin(a) & 0 & x \\ -\sin(a) & \cos(a) & 0 & y \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

$$= \begin{bmatrix} \cos(a)x + \sin(a)y = -\sin(a)x + \cos(a)y1 \end{bmatrix}.$$

The following calculation represents an affine translation

$$Tp = \begin{bmatrix} 1 & 0 & a & x \\ 0 & 1 & b & y \\ 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x + a \\ y + b \\ 1 \end{bmatrix}.$$

The product of the two matrices above then both rotates and translates. This shows how affine transformations can be calculated using homogeneous coordinates.

A projective point is a set of all scalar multiples of a vector. It is a one dimensional subspace of a vector space. In general, projective n-space is the set of 1-dimensional subspaces contained in a vector space of dimension $n + 1$. The nonsingular linear transformations on the vector space map the set of 1-dimensional subspaces onto themselves. These linear mappings are called projective transformations when they are applied to projective space. Considering the previous example, it is easy to see that an affine transformation is a special case of a projective transformation.

We now introduce quadratic forms. Let A be a symmetric matrix and let p be a coordinate vector.

$$p = \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

Then $P^T AP$ is a quadratic form. For example, if

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{bmatrix}.$$

Then

$$P^T AP = x + y + z + 4xy + 6xz + 10yz.$$

All quadratic forms can be obtained in this way. Corresponding to each quadratic form is an associated bilinear form or tensor of rank two. It is a function of two vector variables and is linear in each. It is given as

$$B(P, Q) = P^T AQ.$$

A conic section S is the set of zeroes of a quadratic form in 2-dimensional space. Thus

$$S = \{vP : P^T AP = 0\}.$$

Given a point Q, the polar of Q with respect to a quadric represented by a matrix A is the following set.

$$\{P : B(P, Q) = 0\}.$$

Since the equation is linear, the polar of a point is a line in the two dimensional case, and a plane in the three dimensional case. Directly from the definition it is seen that P is on the polar of Q if and only if Q is on the polar of P. Properties of the polar can be obtained by using the concept of the cross ratio of four collinear points.

94.11 Application to the Projection of an Image to a Screen

When an image is projected to a screen as in a slide projector or a computer video projector, if the screen is not normal to the projection direction, what

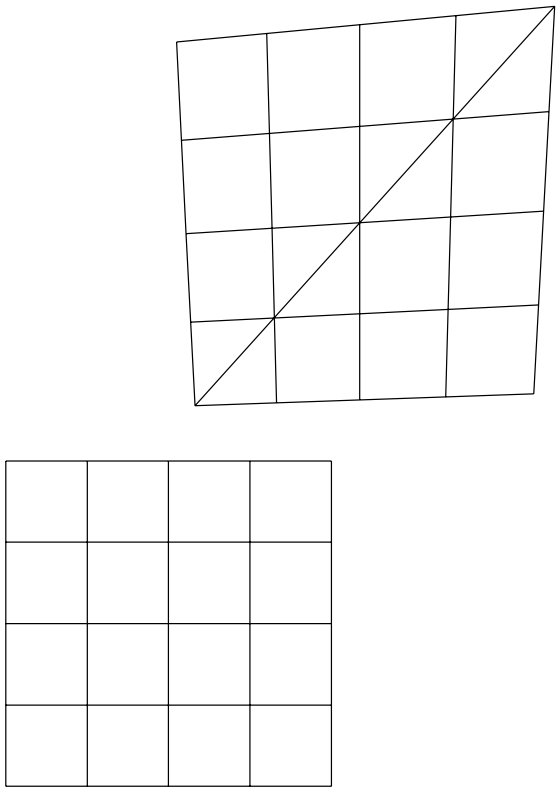


Figure 21: Correcting a distorted screen image with a projective transformation defined as the unique projective transformation taking four points to four points. The figure was created with program **prj2sceeen.ftn** . The diagonal is drawn to illustrate that the transformation is linear in the sense of taking all straight lines to straight lines.

should be a rectangle, will be a somewhat distorted quadrilateral. If a video is taken of this image, and if pixel locations need to be identified, then a transformation taking this distorted image back to a rectangular image is required. Because the distortion is caused by the screen being being rotated relative to the projector, this distortion is caused by a projective transformation. To map back one requires the inverse projective transformation. This can be calculated from the corner points of the distorted image, because it turns out that there is a unique projective transformation taking four points in the plane to a second four points in the plane, provided no three points are colinear.

94.12 A Projective Transformation That Takes Four Points To Four Points

Now we shall show that there is a unique projective transformation mapping four points in the plane to four points in the plane.

Proposition. Let A, B, C, and D be distinct points in the plane, no three of which are collinear. Let A', B', C', and D', also be distinct points in the plane, no three of which are collinear. Then there is a unique projective transformation taking the first four points to the second four points.

Proof. Consider the points in two dimensional projective as having a three dimensional homogeneous coordinate representation, with the third coordinate being 1. So for example

$$A = \begin{bmatrix} A_x \\ A_y \\ 1 \end{bmatrix}$$

First there are numbers a, b, c so that

$$D = aA + bB + cC$$

or

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} A & B & C \end{bmatrix}^{-1} D.$$

This is true because A, B, and C are a basis of the R^3 vector space. Similarly there are numbers a', b', c' so that

$$D' = a'A + b'B + c'C.$$

None of a, b, or c are zero, because no three of the given points are collinear. We wish to find a matrix M so that

$$MA = A'$$

$$MB = B'$$

$$MC = C'$$

Because homogeneous coordinate representations do not change when multiplied by a scalar, we can just as well look for a matrix such that

$$MA = \alpha A'$$

$$MB = \beta B'$$

$$MC = \gamma C',$$

where α, β and γ are yet unspecified numbers. Joining the column vectors together A, B, C together to form a three by three matrix we may write out requirement as

$$M[ABC] = [\alpha A' \beta B' \gamma C'].$$

Then

$$M = [\alpha A' \beta B' \gamma C'][ABC]^{-1}.$$

So we define a matrix M as a product of three by three matrices, formed from the point column vectors.

$$M = \begin{bmatrix} \alpha A' & \beta B' & \gamma C' \end{bmatrix} \begin{bmatrix} A & B & C \end{bmatrix}^{-1}.$$

Then the projective transformation defined by M maps the first three unprimed points to the first three primed points. We have also

$$MD = aMA + bMB + cMC$$

$$= a\alpha A' + b\beta B' + c\gamma C'$$

This equals

$$a'A' + b'B' + c'C' = D',$$

if we take

$$\alpha = \frac{a'}{a}$$

$$\beta = \frac{b'}{b}$$

and

$$\gamma = \frac{c'}{c}$$

Notice that none of a, b, c are zero because A, B, C are linearly independent.

It remains to show that the projective transformation defined by M is unique. We have

$$MA = a"A', MB = b"B', MC = c"C',$$

and

$$MD = D'.$$

Suppose there is a matrix N and numbers $a_N b_N, c_N$, and d_N, so that

$$NA = a_N A', NB = b_N B', NC = c_N C',$$

and

$$ND = d_N D'.$$

Then ND is given by

$$ND = N(aA + bB + cC)$$

$$= aa_N A' + bb_N B' + cc_N C'.$$

And ND is also given by

$$ND = d_N D'.$$

Equating these two expressions for ND, and dividing by d_N, we get

$$D' = (aa_N/d_N)A' + (bb_N/d_N)B' + (cc_N/d_N)C'.$$

But

$$D' = a'A' + b'B' + c'C',$$

and because A', B', C' are linearly independent, we must have

$$aa_N/d_N = a',$$

$$bb_N/d_N = b',$$

and

$$cc_N/d_N = c'.$$

Then

$$a = d_N(a'/a_N) = d_N a",$$

$$b = d_N b",$$

and

$$c = d_N c".$$

So matrix N is a scalar multiple of matrix M,

$$N = d_N M.$$

Therefore N and M represent the same projective transformation.

Here is a MatLab script to create this projective transformation for some example points.

```
% prjtrans.m, A projective transformation taking quadrilateral a,b,c,d
%to quadrilateral ap,bp,cp,dp
%Reference: quadric.pdf "Conics, Quadrics, and Projective Space"
%
a=[20;20;1]
b=[60;30;1]
c=[70;60;1]
d=[25;50;1]
m1=[a';b';c']
m1=m1'
m1i=inv(m1)
e=m1i*d
ap=[1;1;1]
bp=[100;1;1]
cp=[100;100;1]
dp=[1;100;1]
m2=[ap';bp';cp']
m2=m2'
m2i=inv(m2)
f=m2i*dp
g1=f(1)/e(1)
g2=f(2)/e(2)
g3=f(3)/e(3)
ap2=g1*ap
bp2=g2*bp
cp2=g3*cp
m3=[ap2';bp2';cp2']
m3=m3'
p=m3*m1i
pa=p*a
pb=p*b
pc=p*c
pd=p*d
pa=(1/pa(3))*pa
pb=(1/pb(3))*pb
pc=(1/pc(3))*pc
pd=(1/pd(3))*pd
```

When this script is run we get the following output:

```
>> prjtrans

a =

    20
    20
     1


b =

    60
    30
     1


c =

    70
    60
     1


d =

    25
    50
     1


m1 =

    20    20     1
    60    30     1
    70    60     1


m1 =

    20    60    70
    20    30    60
     1     1     1


m1i =

   -0.0273    0.0091    1.3636
    0.0364   -0.0455    0.1818
   -0.0091    0.0364   -0.5455


e =

    1.1364
   -1.1818
```

```
    1.0455


ap =

     1
     1
     1


bp =

   100
     1
     1


cp =

   100
   100
     1


dp =

     1
   100
     1


m2 =

     1     1     1
   100     1     1
   100   100     1


m2 =

     1   100   100
     1     1   100
     1     1     1


m2i =

   -0.0101         0    1.0101
    0.0101   -0.0101         0
         0    0.0101   -0.0101


f =

     1
    -1
     1
```

```
g1 =

    0.8800


g2 =

    0.8462


g3 =

    0.9565


ap2 =

    0.8800
    0.8800
    0.8800


bp2 =

   84.6154
    0.8462
    0.8462


cp2 =

   95.6522
   95.6522
    0.9565


m3 =

    0.8800    0.8800    0.8800
   84.6154    0.8462    0.8462
   95.6522   95.6522    0.9565


m3 =

    0.8800   84.6154   95.6522
    0.8800    0.8462   95.6522
    0.8800    0.8462    0.9565


p =

    2.1834   -0.3599  -35.5893
   -0.8628    3.4478  -50.8201
   -0.0019    0.0043    0.8321
```

```
pa =

    0.8800
    0.8800
    0.8800


pb =

   84.6154
    0.8462
    0.8462


pc =

   95.6522
   95.6522
    0.9565


pd =

    1.0000
  100.0000
    1.0000


pa =

    1.0000
    1.0000
    1.0000


pb =

  100.0000
    1.0000
    1.0000


pc =

  100.0000
  100.0000
    1.0000


pd =

    1.0000
  100.0000
    1.0000
```

So the required projective transformation is

$$p = \begin{bmatrix} 2.1834 & -0.3599 & -35.5893 \\ -0.8628 & 3.4478 & -50.8201 \\ -0.0019 & 0.0043 & 0.8321 \end{bmatrix}$$

After projecting point a to pa, we need to divide the homogenious projective coordinates by the third coordinate of pa in order to get the affine form with the third coordinate equal to 1, and so on.

94.13 A Screen Projection Program Example

This program maps a specified quadrilateral to a second quadrilateral using the unique projective transformation taking four points to four points. There are two linear equation solutions and one matrix inversion, which are done by the Gaussian elimination subroutine called **gausse**. The program draws the quadrilaterals to the file **p.eg**. This file was converted to a postscript file using program **eg2ps.c**. See the figure labelled, "Correcting a distorted screen image ..."

```
c  prj2screen.ftn projection to screen 1/2/10  (from ptrn4.ftn)
      implicit real*8(a-h,o-z)
      dimension a1(3),b1(3),c1(3),d1(3),a2(3),b2(3),c2(3),d2(3)
      dimension t(3,3)
      dimension rx(3,3)
      dimension ry(3,3)
      dimension r(3,3)
c     four points
      data a1/1.d0,1.d0,1.d0/
      data b1/-1.d0,1.d0,1.d0/
      data c1/-1.d0,-1.d0,1.d0/
      data d1/1.d0,-1.d0,1.d0/

      nf=8
      open(nf,file='p.eg',status='unknown')
      s=.15
      do k=1,2
        a1(k)=s*a1(k)
        b1(k)=s*b1(k)
        c1(k)=s*c1(k)
        d1(k)=s*d1(k)
      end do
      it=3
      ir=3
      m=3
      n=3
      l=1
      n2=3
```

```
one=1.
pi=3.14159265358979
angx=-20.0d0*pi/180.0d0
call rot3x(angx,rx)
angy=10.0d0*pi/180.0d0
call rot3y(angy,ry)
call matm(ry,3,m,n,rx,3,3,r,3)
call matm(r,ir,m,n,a1,3,l,a2,3)
call matm(r,ir,m,n,b1,3,l,b2,3)
call matm(r,ir,m,n,c1,3,l,c2,3)
call matm(r,ir,m,n,d1,3,l,d2,3)
write(*,'(a,3(1x,g15.8))')' a2=',(a2(j),j=1,3)
write(*,'(a,3(1x,g15.8))')' b2=',(b2(j),j=1,3)
write(*,'(a,3(1x,g15.8))')' c2=',(c2(j),j=1,3)
write(*,'(a,3(1x,g15.8))')' d2=',(d2(j),j=1,3)
do k=1,3
 a2(k)=a2(k)/a2(3)
 b2(k)=b2(k)/b2(3)
 c2(k)=c2(k)/c2(3)
 d2(k)=d2(k)/d2(3)
enddo
write(*,'(a,3(1x,g15.8))')' a2=',(a2(j),j=1,3)
write(*,'(a,3(1x,g15.8))')' b2=',(b2(j),j=1,3)
write(*,'(a,3(1x,g15.8))')' c2=',(c2(j),j=1,3)
write(*,'(a,3(1x,g15.8))')' d2=',(d2(j),j=1,3)
call ptrn4(a1,b1,c1,d1,a2,b2,c2,d2,it,t)

n1=5
do i=1,n1
 u=(i-1)*one/(n1-1)
 x1=(1.-u)*a1(1)+u*b1(1)
 y1=(1.-u)*a1(2)+u*b1(2)
 x2=(1.-u)*d1(1)+u*c1(1)
 y2=(1.-u)*d1(2)+u*c1(2)
 call xmove(nf,x1,y1)
 call xdraw(nf,x2,y2)
enddo
do i=1,n1
 u=(i-1)*one/(n1-1)
 x1=(1.-u)*a1(1)+u*d1(1)
 y1=(1.-u)*a1(2)+u*d1(2)
 x2=(1.-u)*b1(1)+u*c1(1)
 y2=(1.-u)*b1(2)+u*c1(2)
 call xmove(nf,x1,y1)
 call xdraw(nf,x2,y2)
end do

idrw=0
if(idrw .eq. 1)then
 do i=1,n1
  u=(i-1)*one/(n1-1)
  x1=(1.-u)*a2(1)+u*b2(1)
  y1=(1.-u)*a2(2)+u*b2(2)
  x2=(1.-u)*d2(1)+u*c2(1)
  y2=(1.-u)*d2(2)+u*c2(2)
  call xmove(nf,x1,y1)
  call xdraw(nf,x2,y2)
```

```
       enddo
       do i=1,n1
        u=(i-1)*one/(n1-1)
        x1=(1.-u)*a2(1)+u*d2(1)
        y1=(1.-u)*a2(2)+u*d2(2)
        x2=(1.-u)*b2(1)+u*c2(2)
        y2=(1.-u)*b2(2)+u*c2(2)
        call xmove(nf,x1,y1)
        call xdraw(nf,x2,y2)
       end do
      end if

      call xmove(nf,a2(1),a2(2))
      call xdraw(nf,c2(1),c2(2))

      idrw=1
      if(idrw .eq. 1)then
       do i=1,n1
        u=(i-1)*one/(n1-1)
        x1=(1.-u)*a1(1)+u*b1(1)
        y1=(1.-u)*a1(2)+u*b1(2)
        x2=(1.-u)*d1(1)+u*c1(1)
        y2=(1.-u)*d1(2)+u*c1(2)
        call tr(t,it,x1,y1,tx1,ty1)
        call tr(t,it,x2,y2,tx2,ty2)
        x3=(x1+x2)/2.
        y3=(y1+y2)/2.
        call tr(t,it,x3,y3,tx3,ty3)
        call xmove(nf,tx1,ty1)
        call xdraw(nf,tx3,ty3)
        call xdraw(nf,tx2,ty2)
       enddo

       do i=1,n1
        u=(i-1)*one/(n1-1)
        x1=(1.-u)*a1(1)+u*d1(1)
        y1=(1.-u)*a1(2)+u*d1(2)
        x2=(1.-u)*b1(1)+u*c1(1)
        y2=(1.-u)*b1(2)+u*c1(2)
        call tr(t,it,x1,y1,tx1,ty1)
        call tr(t,it,x2,y2,tx2,ty2)
        call xmove(nf,tx1,ty1)
        call xdraw(nf,tx2,ty2)
       enddo
      end if
      end
c+ tr projective transformation
      subroutine tr(t,it,x,y,tx,ty)
c     input:
c         t   projective 3 by 3 matrix from ptrn4
c       x,y   coordinates of point
c     output:
c     tx,ty   coordinates of transformed point
      implicit real*8(a-h,o-z)
      dimension t(it,*)
      zero=0.
      tx=t(1,1)*x+t(1,2)*y+t(1,3)
```

```
      ty=t(2,1)*x+t(2,2)*y+t(2,3)
      w=t(3,1)*x+t(3,2)*y+t(3,3)
      if(w .ne. zero)then
       tx=tx/w
       ty=ty/w
      endif
      return
      end
c+ ptrn4    projective transformation taking four points to four points
      subroutine ptrn4(a1,b1,c1,d1,a2,b2,c2,d2,it,t)
      implicit real*8(a-h,o-z)
c input:
c  a1,b1,c1,d1 homogeneous coordinates of first four points
c              e.g. a1(1),a1(2),a1(3),
c              x=a1(1)/a1(3), y=a1(2)/a1(3)
c              usually on input a1(3)= 1
c  a2,b2,c2,d2 homogeneous coordinates of second four points
c           it row dimension of matrix t in calling program
c output:
c            t three by three projective transformation matrix
c
c              if p is a point,
c              where  p^T=(x,y,1),
c              and q = t*p, then
c              the transformed point is
c              x' = q(1)/q(3), y' = q(2)/q(3)
c
      dimension a1(*),b1(*),c1(*),d1(*),a2(*),b2(*),c2(*),d2(*)
      dimension t(3,*)
      dimension a(3,3),b(3,3),x1(3),x2(3),x3(3)
      zero=0.
      ia=3
      ib=3
      ix=3
      do i=1,3
       a(i,1)=a1(i)
       a(i,2)=b1(i)
       a(i,3)=c1(i)
       x1(i)=d1(i)
  end do
      m=1
      n=3
      l=3
      inv=0
      eps=1.e-12
      call gaussr(a,ia,x1,ix,n,m,inv,eps,idet,det,ier)
      do  i=1,3
      a(i,1)=a2(i)
      a(i,2)=b2(i)
      a(i,3)=c2(i)
      x2(i)=d2(i)
  enddo

      call gaussr(a,ia,x2,ix,n,m,inv,eps,idet,det,ier)
      do i=1,3
       if(x1(i) .ne. zero)then
        x3(i)=x2(i)/x1(i)
```

```
      else
       write(*,*)' division error in ptrn4 '
      endif
  end do

      do  i=1,3
       a(i,1)=a1(i)
       a(i,2)=b1(i)
       a(i,3)=c1(i)
      enddo
      inv=1
      m=3
      call gaussr(a,ia,b,ib,n,m,inv,eps,idet,det,ier)
      do i=1,3
       a(i,1)=x3(1)*a2(i)
       a(i,2)=x3(2)*b2(i)
       a(i,3)=x3(3)*c2(i)
  end do

      call matm(a,ia,m,n,b,ib,l,t,it)
      return
      end
c+ gaussr solution real linear eqs., matrix inv., determinant (real*8)
      subroutine gaussr(a,ia,b,ib,n,m,inv,eps,idet,det,ier)
      implicit real*8(a-h,o-z)
c  solves the equation a*c=b for c, where a is an n by n matrix
c  c and b are n row by m column matrices. c is returned as b.
c  algorithm       -gaussian elimination with partial pivoting.
c  parameters     a-n by n matrix containing the coefficients of
c                   the linear system.
c                ia-row dimension of a in defining routine
c                   e.g. in the routine where a is defined,
c                   a might be dimensioned as:
c                   dimension a(nr,nc)
c                   then ia must be set to nr.  we may have n < ia, but
c                   must not have n > ia, or n*n > nr*nc.
c                   ia is needed for proper addressing of matrix a.
c                   fortran stores by column first: a(i,j)=a(i+(j-1)*ia))
c                 b-n by m matrix containing the m right sides
c                   of the equations, on entering.  on returning, b
c                   contains the solutions.  the inverse of a is
c                   returned in b when inv=1
c                ib-row dimension of b in defining routine
c                 n-row and column dimension of a.
c                 m-column dimension of b (usually 1)
c                   the program changes m to n when inv=1
c               inv-the inverse of a is calculated
c                   and returned in b when inv=1
c                   b must be large enough to hold the inverse
c               eps-each equation is normalized so that the
c                   coefficients are <= 1 in magnitude.
c                   when a pivot is less than eps the matrix is
c                   considered nearly singular, and ier is set to 1
c                   if a pivot is zero the matrix is singular, and
c                   ier is set to 2. one may set eps=1.e-5 for
c                   single precision, and 1.e-12 for double.
c                   eps does not effect any calculation.
```

```
c                   normalization may also prevent exponent overflow.
c              idet-compute determinant only if idet = 1
c                    determinants are products of n numbers.
c                    overflow can occur if the elements of the
c                    matrix have large exponents.
c                    set idet=0 if the determinant is not needed.
c               det-determinant of a.
c               ier-return parameter,
c               ier=0  normal return
c               ier=1  matrix is nearly singular
c               ier=2  matrix is singular
c
c  warning!! the subroutine changes a and b. if they need to be
c  saved, copies must be made before calling the subroutine.
c  the subroutine can be converted to different number type
c  by uncommenting the appropriate implicit statement.
c     implicit complex(a-h,o-z)
c     implicit complex*16(a-h,o-z)
      logical cdet
      dimension a(ia,*),b(ib,*)
      cdet=idet .eq. 1
      zero=0.
      ier=0
      det=1.
      if(m.le.0)m=1
      if(inv.eq.1)then
c      set b equal to the identity
       do 10 i=1,n
        do 10 j=1,n
         b(i,j)=0.
         if(i.eq.j)b(i,j)=1.
10     continue
       m=n
      endif
c normalize rows
      do 20 i=1,n
       bigest=a(i,1)
       do 16 j=2,n
        ab=a(i,j)
        if(abs(ab).gt.abs(bigest))bigest=ab
16     continue
       if(bigest.eq.zero)then
        ier=2
        det=0.
        return
       endif
       if(cdet)det=det*bigest
       do 18 j=1,n
        a(i,j)=a(i,j)/bigest
18     continue
       do 19 j=1,m
        b(i,j)=b(i,j)/bigest
19     continue
20    continue
c
      j=1
30    continue
```

```
c do while j<n *********
      kk=j+1
      l=j
c  find row l with largest pivot
      do 32 i=kk,n
       if(abs(a(i,j)).gt.abs(a(l,j)))l=i
32    continue
      if(abs(a(l,j)).eq.zero)then
        ier=2
        det=0.
        return
      endif
      if(abs(a(l,j)).le.abs(eps))ier=1
      if(l.ne.j)then
c  interchange rows l and j
       do 37 k=1,n
        c=a(l,k)
        a(l,k)=a(j,k)
        a(j,k)=c
37     continue
       do 39 k=1,m
        c=b(l,k)
        b(l,k)=b(j,k)
        b(j,k)=c
39     continue
       if(cdet)det=det*(-1.)
      endif
      if(cdet)det=det*a(j,j)
c  divide row by pivot
      c=a(j,j)
      do 50 k=j,n
       a(j,k)=a(j,k)/c
50    continue
      do 55 k=1,m
       b(j,k)=b(j,k)/c
55    continue
c     add multiple of row j to lower rows
c     to eliminate jth coefficients
      jj=j+1
      do 80 i=jj,n
       am=a(i,j)
       do 60 k=1,n
        a(i,k)=a(i,k)-am*a(j,k)
60     continue
       do 70 k=1,m
        b(i,k)=b(i,k)-am*b(j,k)
70     continue
80    continue
      j=j+1
      if(j.ne.n)go to 30
c end while ******
      am=a(n,n)
      if(abs(am).eq.zero)then
       ier=2
       det=0.
       return
      endif
```

```
      if(abs(am).le.abs(eps))ier=1
      if(cdet)det=det*am
c a is now in triangular form
c compute nth component of solution
      do 90 k=1,m
       b(n,k)=b(n,k)/am
90    continue
c     back substitute to compute n-i component
c     i=1,2,3,...
      nn=n-1
      do 120 i=1,nn
       ni=n-i
       do 110 j=1,m
        nj=ni+1
        do 100 ki=nj,n
         b(ni,j)=b(ni,j)-a(ni,ki)*b(ki,j)
100     continue
110    continue
120   continue
      return
      end
c+ matm  matrix multiplication
      subroutine matm(a,ia,m,n,b,ib,l,c,ic)
        implicit real*8(a-h,o-z)
c arguments
c  a-matrix
c ia-row dimension of a in calling program
c  m-number of rows of a
c  n-number of columns of a
c  b-matrix
c ib-row dimension of b in calling program
c  l-number of columns of b
c  c-product matrix: c=a*b
c ic-row dimension of c in calling program
c
      dimension a(ia,*),b(ib,*),c(ic,*)
c     c=a*b
      do 10 i=1,m
      do 10 j=1,l
      c(i,j)=0.
      do 5 k=1,n
5     c(i,j)=c(i,j)+a(i,k)*b(k,j)
10    continue
      return
      end


c+ xdraw  draw parameters to external plot file
      subroutine xdraw(nfile,x,y)
      implicit real*8(a-h,o-z)
      character s*25,t*80
      t='d'
      n=2
      call str(x,s)
      l=lenstr(s)
      t(n:80)=s
      n=n+l+1
```

```
      call str(y,s)
      l=lenstr(s)
      t(n:80)=s
      write(nfile,'(a)')t(1:(n+l-1))
      return
      end
c
c+ xmove  move parameters to external plot file
      subroutine xmove(nfile,x,y)
      implicit real*8(a-h,o-z)
      character s*25,t*80
      t='m'
      n=2
      call str(x,s)
      l=lenstr(s)
      t(n:80)=s
      n=n+l+1
      call str(y,s)
      l=lenstr(s)
      t(n:80)=s
      write(nfile,'(a)')t(1:(n+l-1))
      return
      end
c+ lenstr   nonblank length of string
      function lenstr(s)
c  length of the substring of s obtained by deleting all
c  trailing blanks from s.  thus the length of a string
c  containing only blanks will be 0.
      character   s*(*)
      lenstr=0
      n=len(s)
      do 10 i=n,1,-1
      if(s(i:i) .ne. ' ')then
         lenstr=i
         return
      endif
10    continue
      return
      end
c+ str    floating point number to string
      subroutine str(x,s)
      implicit real*8(a-h,o-z)
      character s*25,c*25,b*25,e*25
       zero=0.
      if(x.eq.zero)then
         s='0'
         return
      endif
      write(c,'(g11.4)')x
      read(c,'(a25)')b
      l=lenstr(b)
      do 10 i=1,l
      n1=i
      if(b(i:i).ne.' ')go to 20
 10   continue
 20   continue
       if(b(n1:n1).eq.'0')n1=n1+1
```

```
       b=b(n1:l)
       l=l+1-n1
       k=index(b,'E')
       if(k.gt.0)e=b(k:l)
       if(k.gt.0)then
         s=b(1:(k-1))
         k1=index(b,'E+0')
         if(k1.gt.0)then
          e='E'//b((k1+3):l)
         else
          k1=index(b,'E+')
          if(k1.gt.0)e='E'//b((k1+2):l)
         endif
         k1=index(b,'E-0')
         if(k1.gt.0)e='E-'//b((k1+3):l)
         l=k-1
       else
          s=b
       endif
       j=index(s,'.')
       n2=l
      if(j.ne.0)then
              do 30 i=1,l
               n2=l+1-i
               if(s(n2:n2).ne.'0')go to 40
30             continue
       endif
40             continue
        s=s(1:n2)
        if(s(n2:n2).eq.'.')then
           s=s(1:(n2-1))
           n2=n2-1
         endif
        if(k.gt.0)s=s(1:n2)//e
      return
      end
c+ rot3x    3d rotation about the x axis.
      subroutine rot3x(ang,r)
      implicit real*8(a-h,o-z)
c  r is the 3 by 3 matrix of rotation about the unit x
c  vector by ang radians.  the direction of rotation
c  is determined by the right hand rule:  with the thumb
c  of the right hand pointing in the direction of the
c  vector the fingers determine the positive direction
c  of rotation.
c     the vectors are column vectors.
      dimension r(3,3)
      c=cos(ang)
      s=sin(ang)
      do  i=1,3
       do  j=1,3
        r(i,j)=0.
       enddo
      enddo
      r(1,1)=1.
      r(2,2)=c
      r(2,3)=-s
```

```
      r(3,2)=s
      r(3,3)=c
      return
      end
c+ rot3y rotation about y axis.
      subroutine rot3y(ang,r)
      implicit real*8(a-h,o-z)
c
c  r is the 3 by 3 matrix of rotation about the unit y
c  vector by ang radians.  the direction of rotation
c  is determined by the right hand rule:  with the thumb
c  of the right hand pointing in the direction of the
c  vector the fingers determine the positive direction
c  of rotation.
c
      dimension r(3,*)
      c=cos(ang)
      s=sin(ang)
      do  i=1,3
       do  j=1,3
        r(i,j)=0.
        enddo
       enddo
      r(1,1)=c
      r(2,2)=1.
      r(1,3)=s
      r(3,1)=-s
      r(3,3)=c
      return
      end
```

94.14 Projective Space Bibliography

[1]Aslander Louis And Mackenzie Robert, **Introduction To Differential Manifolds**, Dover N.y 1977.

[2]Barnhill R E, Riesenfeld R F (Editors), **Computer Aided Geometric Design**, Academic Press 1974.

[3]Birkhoff Garrett And Maclane Saunders, **A Survey Of Modern Algebra Revised Edition**, Macmillan 1953.

[4]Brunk H D, **Introduction To Mathematical Statistics**, Blaisdell 2nd Ed. 1965.

[5]Courant R, Robbins H, **What Is Mathematics?**, Oxford University Press 1947.

[6]Eisen Martin, **Introduction To Mathematical Probability Theory**, Prentice-hall Englewood Cliffs N.j. 1969.

[7]Faux I.d. And Pratt M.j., **Computational Geometry For Design And Manufacture**, Halsted New York 1979.

[8]Goldstein Herbert, **Classical Mechanics**, Addison-wesley, 1950.
[9]Halmos Paul R., **Introduction To Hilbert Space**, Chelsea New York 2nd Ed. 1957.
[10]Hodge W V D, Pedoe D, **Methods Of Algebraic Geometry**, Cambridge U. Press 1968.
[11]Knuth Donald, **The Art Of Computer Programming**, Vol. 1 Addison-wesley 2nd Ed.1973..
[12]Lass Harry, **Elements Of Pure And Applied Mathematics**, Mcgraw-hill 1957..
[13]Leven Joshua, **A Parametric Algorithm For Drawing Pictures Of Solid Objects Composed Of Quadric Surfaces**, Comm. Acm V. 19 No. 10 P555 Oct 1976.
[14]Mcconnell Albert J, **Applications Of Tensor Calculus**, Dover 1931.
[15]Olmsted John M H, **Solid Analytic Geometry**, Appleton-century-crofts New York 1947..
[16]O'neill Barrett, **Elementary Differential Geometry**, Academic Press New York 1966.
[17]Panofsky Erwin, **Durer As A Mathematician**, In The World Of Mathematics, Vol. 1 James Newman Ed., Simon And Schuster New York 1956.
[18]Requicha A A G, Voelker H B, **Constructive Solid Geometry**, Technical Memorandum 25,, Production Automation Project The University Of, Rochester Rochester N. Y.1977.
[19]Sasaki Tateaki, **Multidimensional Monte Carlo Integration Based On Factorized Approximation Functions**, Siam Jour. Num. Analysis Vol. 15 No. 5 Oct. 1978.
[20]Semple J G, Kneebone G T, **Algebraic Projective Geometry**, Oxford 1952.
[21]Smart E H, **A First Course In Projective Geometry**, Macmillan 1913.
[22]Spivak Michael, **Calculus On Manifolds**, W.a Benjamin, New York 1965.
[23]Struik Dirk J, **Analytic And Projective Geometry**, Addison-wesley Cambridge Mass. 1953.
[24]Willmore T J, **An Introduction To Differential Geometry**, Oxford U. Press 1959.
[25]Yakowitz S, Krimmel J E , Szidarouszky F, **Weighted Monte Carlo Integration**, Siam Jour. Num. Anal. Vol. 15, No. 6 Dec. 1978.

95 General Bibliography

[1]Mamoru Kurata, **Numerical Analysis of Semiconductor Devices**, tk7871.85 .k83, Lexington Books.
[2]Ortega J M j, **Numerical Analysis (a Second Course)**, emery, qa297.o78, 1972.
[3]Unknown, **Numerical Methods for Stiff Equations**, qa371 .m58.
[4]Ortega J. M. , Rheinboldt W. C., **Studies in Numerical Analysis 2**.
[5]Thompson J. F., Wasi, Mastin, **Numerical Grid Generation**, north holland 1985.
[6]Press William H, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, **Numerical Recipes. The Art of Scientific Computing, 3rd Edition**, 2007, ISBN 0-521-88068-8. (C++ code) , Cambridge University Press .
[7]Fausett Laurene V. **Applied Numerical Analysis, Using Matlab** . Prentice-Hall, 1999.
[8]Akin Ed (J. E. Akin), **Object-Oriented Programming Via Fortran 90/95**, Cambridge University Press, 2003.
[9]DeVries Paul L., **A First Course in Computational Physics**, 1994, John Wiley.
[10]Ortega James M., Grimshaw Andrew S., **An Introduction to C++ and Numerical Methods**, Oxford University Press, 1999
[11]Crandall Richard E, **Topics in Advanced Scientific Computation**, 1996, Telos Springer.
[12]Van Loan Charles F., **Introduction to Scientific Computing**, (Matlab), 2000, Prentice-Hall.
[13]Landau Rubin H, Paez Manuel J, **Computational Physics, Problem Solving With Computers**, 1997, Prentice-Hall