DRAFT

Introductory Lecture Notes in:

MATLAB, MATHEMATICA & Unix

©VICTOR E. SAOUMA

Dept. of Civil Environmental and Architectural Engineering

University of Colorado, Boulder, CO 80309-0428

September 5, 2002

NOTE:

These lecture notes were developed during the Spring Semester of 1996 as handouts for an experimental course proposed by the author.

Whereas numerous excellent books have been written on related subjects, it was the author's objective to synthetise the most important aspects of Engineering Computing into a set of lecture notes.

It is assumed that student would have had a first course in programming, linear algebra, and differential equations.

Because most related books draw their examples from non-Civil Engineering disciplines, (and because of the author's background) most of the examples in these notes are taken from Structural Engineering/Mechanics. Hence, extensive derivation/explanation is provided prior to each problem for those students coming from unrelated disciplines.

Any questions/comments should be forwarded to `saouma@civil.colorado.edu`

Contents

List of Figures

List of Tables

Chapter 1

WEEK I; MATLAB: Basic Introduction

1.1 Background

1.1.1 What is MATLAB?

1 MATLAB is an interactive, matrix-based system for scientific and engineering calculations. Contrarily to programming languages, such as Fortran, C, or Basic you can solve complex numerical problems without actually writing a program.

2 MATLAB is an abbreviation for MATrix LABoratory.

3 MATLAB is the most widely used software package for interactive numeric computation, graphics, data analysis[1].

1.1.2 Availability

4 MATLAB is available in the CAD lab (PC/Windows version), as well as in the Bechtel Laboratory.

5 A relatively inexpensive Student Edition is available from Prentice Hall and can be purchased from the Buffalo Chip.

1.1.3 Accessing MATLAB

6 Once you clicked on the MATLAB icon, the menu shown in Fig. 1.1 will be displayed which may be accomplished through browsing, Fig. 1.1.3.

7 Fig. 1.1.3 illustrates the content of the default directory containing your source code.

[1] Other similar products include IDL, and MATHEMATICA for symbolic computations.

Figure 1.1: Matlab's Main Window

7 Whereas for simple operations, you may directly enter MATLAB commands in the main window and execute them, for long codes it is far simpler to create files containing the MATLAB commands. Those files can then be easily created, edited, and modified.

8 If you are using files (with extension `.m`) to store your code, you must tell MATLAB in which directory those files are stored, this is accomplished by setting the path, Fig. 1.1.3.

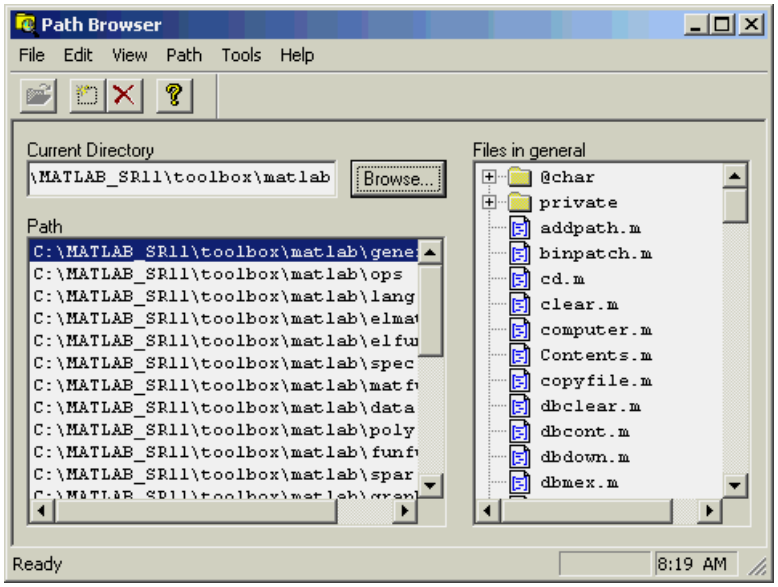


Figure 1.2: Set Path in Matlab

8 Finally, Fig. **??** illustrates a typical working envirnoment, where a file has been selected for editing in one window, and program is executed in the other. Note that upon completion of the program, you can always querry values in the MATLAB main window.

9

1.1.4 Help

10 for help you can either type `help sqrt` (i.e. help command followed by the name of the function, or you may type `lookfor square` (similar to the `man -k` in Unix).

1.1.5 Basic Features

11 When you run MATLAB, there will be one or more windows on your monitor. The command window is the primary one where you interact with MATLAB, and the MATLAB prompt will look like

```
>>
```

with a blinking cursor to the right.

12 Basic commands are shown in Table 1.1.

13 MATLAB has also a few special characters, Table 1.2

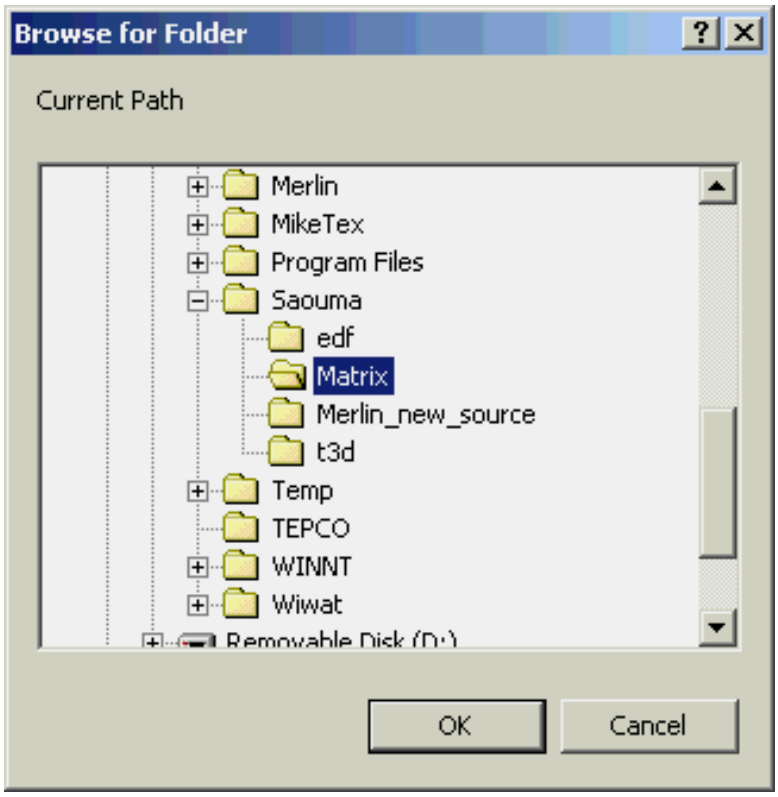


Figure 1.3: Selecting path Directory in Matlab

General	
help	help facility
demo	run demonstrations
who	list variables in memory
what	list M-files on disk
clear	clear workspace
computer	type of computer
^C	local abort
exit	exit MATLAB
quit	same as exit

Table 1.1: Basic Commands

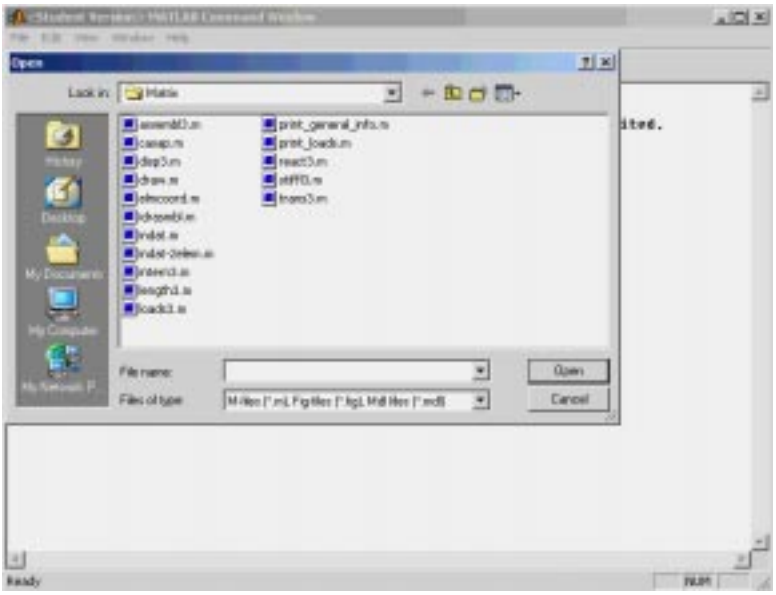


Figure 1.4: Directory Containing Source Code in Matlab

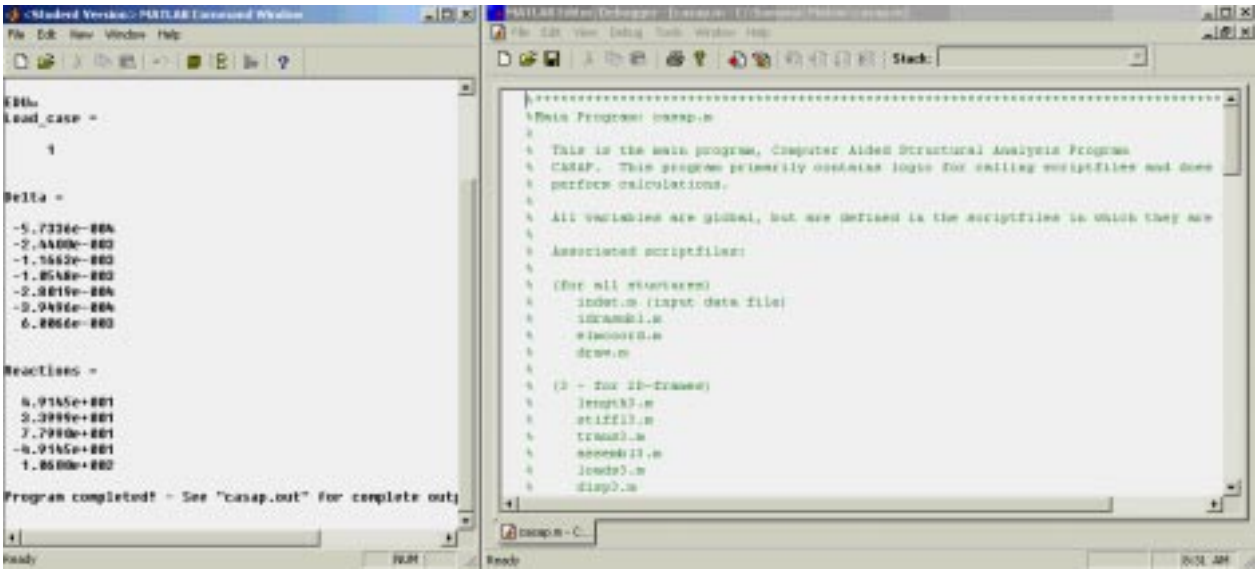


Figure 1.5: Matlab's Operating Environment

Special Characters	
=	assignment statement
[	used to form vectors and matrices
]	see [
(	arithmetic expression precedence
)	see (
.	decimal point
...	continue statement to next line
,	separate subscripts and function arguments
;	end rows, suppress printing
%	comments
:	subscripting, vector generation
!	execute operating system command

Table 1.2: Special Characters

1.1.6 Simple Math

14 To begin with, let us go through a simple example

```
>> 29+6
ans=
   35
>> 21*2+5^2
ans=
   67
>> force=10
force =
        10
>> distance=5
distance=
         5
>> moment=force*distance
moment=
 50
>>
```

Try now the following commands **who, whos, clear, who**.

15 The basic arithmetic operations are

+	addition	7.2+9.6
−	subtraction	11-92
*	multiplication	2.9*4.5
^	power	2.6^0.45
\ or /	left or right division	27/5=5\27

1.1.7 Common Mathematical Functions

16 Common mathematical functions are shown in Table 1.3

1.1.8 Statements, Expressions and Variables

17 MATLAB is an *interprative* language; the expressions you type are interpreted and evaluated. MATLAB statements are usually of the form
variable = expression, or simply
expression

18 Expressions are usually composed from operators, functions, and variable names. Evaluation of the expression produces (most often) a matrix, which is then displayed on the screen and assigned to the variable for future use. If the variable name and = sign are omitted, a variable `ans` (for answer) is automatically created to which the result is assigned.

19 A statement is normally terminated with the carriage return. However, a statement can be continued to the next line with three or more periods followed by a carriage return. On the other hand, several statements can be placed on a single line if separated by commas or semicolons.

```
a=2;b=4;c=-240;deltasq=b^2-4*a*c;delta=sqrt(deltasq)
delta =
    44
x1=(-b+delta)/(2*a)
x1 =
    10
x2=(-b-delta)...
/(2*a)
x2 =
   -12
```

20 If the last character of a statement is a semicolon, the printing is suppressed, but the assignment is carried out. This is essential in suppressing unwanted printing of intermediate results.

21 MATLAB is case-sensitive in the names of commands, functions, and variables. For example, `Apples` is different from `APPLES`.

Elementary Math Functions	
abs	absolute value or complex magnitude
angle	phase angle
sqrt	square root
real	real part
imag	imaginary part
conj	complex conjugate
round	round to nearest integer
fix	round toward zero
floor	round toward $-\infty$
ceil	round toward ∞
sign	signum function
rem	remainder
exp	exponential base e
log	natural logarithm
log10	log base 10
Trigonometric Functions	
sin	sine
cos	cosine
tan	tangent
asin	arcsine
acos	arccosine
atan	arctangent
atan2	four quadrant arctangent
sinh	hyperbolic sine
cosh	hyperbolic cosine
tanh	hyperbolic tangent
asinh	hyperbolic arcsine
acosh	hyperbolic arccosine
atanh	hyperbolic arctangent
Special Functions	
bessel	bessel function
gamma	gamma function
rat	rational approximation
erf	error function
inverf	inverse error function
ellipk	complete elliptic integral of first kind
ellipj	Jacobian elliptic integral

Table 1.3: Common Mathematical Functions

1.1.9 Display Formats

22 All computations in MATLAB are performed in double precision.

23 The format of the displayed output is shown in Table 1.4 Once invoked, the chosen format

format short	default display
format long	16 digits
format short e	5 digits plus exponent
format long e	16 digits plus exponent
format bank	2 decimal digits
format +	positive, negative or zero
format rat	rational approximation
format hex	hexadecimal

Table 1.4: Display Options

remains in effect until changed.

24 The command `format compact` will suppress most blank lines allowing more information to be placed on the screen or page. It is independent of the other format commands.

```
sqrt(pi)
ans =
    1.7725
format long
ans
ans =
   1.77245385090552
format long e
ans
ans =
    1.772453850905516e+000
```

1.1.10 Printing Text and Matrices

25 The `disp` command can be used to display both data and text

```
disp(pi);disp('University of Colorado')
    3.1416
University of Colorado
```

26 Formatted output can be done through the `fprintf` command which provides you with better control on the format. It has two arguments:

1. Text and format specifications which must be enclosed in a single quote. Within the text, the following specifier can be used:

%e	Exponential notation
%f	Fixed point or decimal notation
%g	Whichever is shorter
\n	New line

2. Matrix to be printed

```
v=153.98;

fprintf('Velocity is %f miles an hour',v)

Velocity is 153.980000 miles an hourfprintf('Velocity is %f miles an hour\n',v)

Velocity is 153.980000 miles an hour
fprintf('Velocity is %e miles an hour\n',v)

Velocity is 1.539800e+002 miles an hour
fprintf('Velocity is %g miles an hour\n',v)

Velocity is 153.98 miles an hour
fprintf('Velocity is %8.3f miles an hour\n',v)

Velocity is  153.980 miles an hour
```

27 Text and string operations, Table 1.5

Text and Strings	
abs	convert string to ASCII values
eval	evaluate text macro
num2str	convert number to string
int2str	convert integer to string
setstr	set flag indicating matrix is a string
sprintf	convert number to string
isstr	detect string variables
strcomp	compare string variables
hex2num	convert hex string to number

Table 1.5: Text and String Operations

1.1.11 Variables

28 Variables are case sensitive, can contain up to 19 characters, and must start with a letter.

29 MATLAB has several special variables, Table 1.6.

Special Variables	
ans	answer when expression not assigned
eps	floating point precision
pi	π
i, j	$\sqrt{-1}$
inf	∞
NaN	Not-a-Number
clock	wall clock
date	date
flops	floating point operation count
nargin	number of function input arguments
nargout	number of function output arguments

Table 1.6: Special Variables

1.1.12 MATLAB Workspace

30 When one logs out or exits MATLAB all variables are lost. However, invoking the command `save` before exiting causes all variables to be written to a non-human-readable diskfile named `matlab.mat`. When one later reenters MATLAB, the command `load` will restore the workspace to its former state.

31 To recall previous commands, you can use the cursors keys.

32 MATLAB can execute a sequence of statements stored on diskfiles. Such files are called "M-files" because they must have the file type of `.m` as the last part of their filename.

33 There are two types of M-files:

Script files are ASCII files which consist of a sequence of normal MATLAB statements. If the file has the filename, say, `hw1.m`, then the MATLAB command `hw1` will cause the statements in the file to be executed. Script files can be edited by a standard editor. Script files can also be used to enter data into a large matrix; in such a file, entry errors can be easily edited out. An M-file can reference other M-files, including referencing itself recursively.

Function files Function files provide extensibility to MATLAB. You can create new functions specific to your problem which will then have the same status as other MATLAB functions.

1.1.13 Saving and Retrieving Data

34 If you want to save the variables in the workspace before you quit, you must use the `save fn` command where `fn` is the file name.

35 To retrieve your data next time, `load fn`.

36 Disk files commands are shown in Table 1.7

Disk Files	
chdir	change current directory
delete	delete file
diary	diary of the session
dir	directory of files on disk
load	load variables from file
save	save variables to file
type	list function or file
what	show M-files on disk
fprintf	write to a file
pack	compact memory via **save**

Table 1.7: Disk Files Commands

1.1.14 Arrays

37 Arrays can be entered in several different ways:

- Entered by an explicit list of elements,
- Generated by built-in statements and functions,
- Created in M-files,
- Loaded from external data files

38 The following session is self explanatory, study it in detail.

```
x=[2. 6. 12.]
x =
     2     6    12

y=[3. 5. 1.]
y =
```

```
     3     5     1
x.*y
ans =
     6    30    12

y'
ans =
     3
     5
     1

x*y'
ans =
    48

x'*y
ans =
     6    10     2
    18    30     6
    36    60    12

2*x
ans =
     4    12    24
x(2)
ans =
     6

x(1:2)
ans =
     2     6

x(3:-1:1)
ans =
    12     6     2

x=(0:0.1:1)
x =
  Columns 1 through 7
         0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000
  Columns 8 through 11
    0.7000    0.8000    0.9000    1.0000
```

```
linspace(0,pi,11)
ans =
  Columns 1 through 7
         0    0.3142    0.6283    0.9425    1.2566    1.5708    1.8850
  Columns 8 through 11
    2.1991    2.5133    2.8274    3.1416

x=[2. 6. 12.]
x =
     2     6    12

x.^2
ans =
     4    36   144

x=[1,2,3]
x =
     1     2     3
```

39 Array operations are shown in Table 1.8

Element by Element Array Operations		
Operation	Algebraic Form	MATLAB
Addition	$a_i + b_i$	$a + b$
Substraction	$a_i - b_i$	$a - b$
Multiplication	$a_i \times b_i$	$a. * b$
Right division	a_i / b_i	$a./b$
Left division	b_i / a_i	$a.\backslash b$
Power	$a_i^{b_i}$	$a.\hat{}b$

Table 1.8: Array Operations

```
x=[6 2 4 8 -2 12];y=[2 6,3, 1 2,10]
y =
     2     6     3     1     2    10
size(y)
ans =
     1     6
x+y
ans =
```

```
     8     8     7     9     0    22
x.*y
ans =
    12    12    12     8    -4   120
x./y
ans =
    3.0000    0.3333    1.3333    8.0000   -1.0000    1.2000
y.\x
ans =
    3.0000    0.3333    1.3333    8.0000   -1.0000    1.2000
```

1.1.15 Graphics

40 One of the major strength of MATLAB are its graphics capabilities. They are numerous, and only few will be explored at this early stage.

41 MATLAB can produce both planar plots and 3-D mesh surface plots. To preview some of these capabilities enter the command `plotdemo`.

42 The `plot` command creates linear x-y plots; if x and y are vectors of the same length, the command `plot(x,y)` opens a graphics window and draws an x-y plot of the elements of x versus the elements of y.

```
x=(0:0.1:10);
y=sin(x).*exp(-x);
plot(x,y)
grid
xlabel('time [s]')
ylabel('Temperature [Deg. F]')
print -deps2 plot1.eps
```

will generate the plot of Fig. 1.6 Note that as new commands are entered, the plot is immediately updated.

43 When in the graphics screen, pressing any key will return you to the command screen while the command `shg` (show graph) will then return you to the current graphics screen. If you are in the Bechtel Lab, then you can have multiple graphics windows.

44 The command `grid` will place grid lines on the current graph.

45 Graphs can be given titles, axes labeled, and text placed within the graph commands which take a string as an argument, Table 1.9.

46 To make multiple plots on a single graph

```
x=0:0.1:2*pi;y1=sin(x);y2=cos(x);plot(x,y1,'-',x,y2,'-.');grid
```

title	graph title
xlabel	x-axis label
ylabel	y-axis label
gtext	interactively-positioned text
text	position text at specified coordinates
Graphs	
plot	linear X-Y plot
loglog	loglog X-Y plot
semilogx	semi-log X-Y plot
semilogy	semi-log X-Y plot
polar	polar plot
bar	bar charts
stairs	stairstep graph
errorbar	add error bars
Graph Annotation	
title	plot title
xlabel	x-axis label
ylabel	y-axis label
grid	draw grid lines
text	arbitrarily position text
gtext	mouse-positioned text
ginput	graphics input
Graph Window Control	
axis	manual axis scaling
hold	hold plot on screen
shg	show graph window
clg	clear graph window
subplot	split graph window

Table 1.9: 2D Graphics Commands

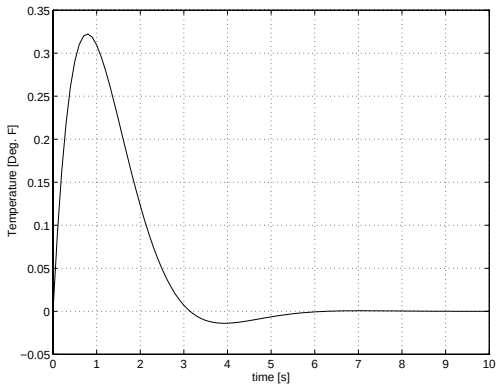


Figure 1.6: Simple Plot

renders a dashed line and dashed dotted line for the first and second graph respectively, resulting in Fig. 1.7

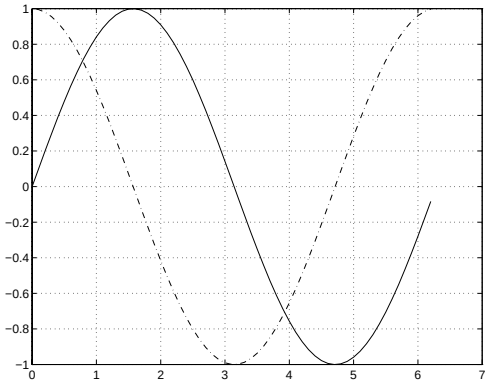


Figure 1.7: Multiple Plots in a Single Graph

47 Line colors can also be separately specified

48 The command `subplot` can be used to partition the screen so that up to four plots can be viewed simultaneously.

1.1.16 Graphics hardcopy

49 A hardcopy of the graphics screen can be most easily obtained with the MATLAB command `print` which will print or save the graph (see example above). The syntax is

```
PRINT [ -ddevice] [ -options ] <filename>
```

where the most common devices are shown in Table 1.10

dps	PostScript for black and white printers
dpsc	PostScript for color printers
dps2	Level 2 PostScript for black and white printers
dpsc2	Level 2 PostScript for color printers
deps	Encapsulated PostScript (EPSF)
depsc	Encapsulated Color PostScript (EPSF)
deps2	Encapsulated Level 2 PostScript (EPSF)
depsc2	Encapsulated Level 2 Color PostScript (EPSF)
dhpgl	HPGL compatible with Hewlett-Packard 7475A plotter
dljetplus	HP LaserJet+
dljet3	HP LaserJet III
dcdeskjet	HP DeskJet 500C with 1 bit/pixel color
dbj10e	Canon BubbleJet BJ10e
dgif8	8-bit color GIF file format

Table 1.10: Printer Devices

1.2 Assignment

1.2.1 Practice

Start by repeating all the examples in this handout.

1.2.2 Work

The force $\mathbf{F}$ which moves a body along a straight line path $\mathbf{S}$ is the scalar product of $\mathbf{F}.\mathbf{S} = W$. In general we may either break down the path into a series of linear segments, or if the path is curvilinear we must perform a line integral $W = \int_S \mathbf{F}.dS$.

Given a force $\mathbf{F}$ with components (2,4,6), i.e. $\mathbf{F} = 2\mathbf{i} + 4\mathbf{j} + 6\mathbf{k}$, and the following points

Point	Coordinates		
	X	Y	Z
O	0	0	0
A	5	0	0
B	0	7	0
C	0	0	11
D	-200	-500	+1000
E	6	8	12

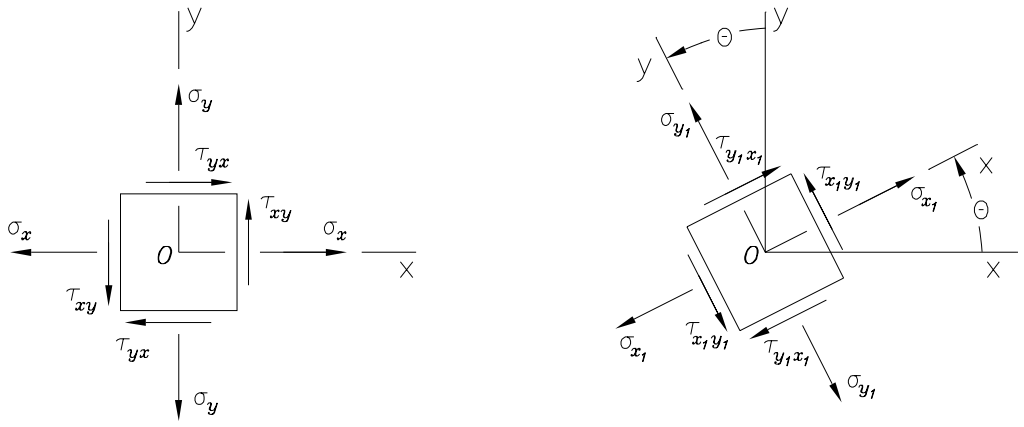
Determine the work for each of the following paths

Case	Path
1	O-A-B-C-E
2	O-C-E
3	O-D-E
4	O-E

Discuss your results.

1.2.3 Stresses

If an infinitesimal element is subjected to the cartesian stresses shown below,



it can be shown that the stresses along any other orientation are given by:

$$\sigma_{x1} = \frac{\sigma_x + \sigma_y}{2} + \frac{\sigma_x - \sigma_y}{2}\cos 2\theta + \tau_{xy}\sin 2\theta \quad \text{(1.1-a)}$$

$$\tau_{x_1y_1} = -\frac{\sigma_x - \sigma_y}{2}\sin 2\theta + \tau_{xy}\cos 2\theta \quad \text{(1.1-b)}$$

and the principal stresses are given by:

$$\sigma_{1,2} = \frac{\sigma_x + \sigma_y}{2} \pm \sqrt{\left(\frac{\sigma_x - \sigma_y}{2}\right)^2 + \tau_{xy}^2} \quad (1.2)$$

For $\sigma_x = 12,300$ psi, $\sigma_y = -4,200$ psi and $\tau_{xy} = -4,700$ psi,

1. Determine the stresses when the element is rotated by +30 degrees (counter-clockwise),
2. Determine the principal stresses
3. Plot the variation of all three stresses in terms of θ

1.2.4 Measurement Errors, [1]

Let us consider an instrument with a scale graduated from 0-1000 (such as 0 to 1000 volts). If the instrument is guaranteed as belonging to the *3% class*, the maximum error is ±3% of the full-scale deflection, in this case ±30. Hence, if the meter reads 500, the true value can be anywhere in the range 470-530. The corresponding relative error equals $\frac{30}{500}100 = 6\%$ Hence, it is clear that the full scale deflection of the measuring instrument should not be much higher than the range of expected values. One frequently used rule of thumb is to havethe expected value between 1/2 and 2/3 of the full scale deflection.

For this instrument

1. Calculate and display the per cent error for measured values in increments of 100.
2. Plot the percent error, against the measured values, in the interval 0-1000. Use intervals of 10.

1.2.5 Statistical Analysis; Part I

1.2.5.1 Elements of Statistics

50 Elementary statistics formulaes will be reviewed, as they are needed to properly understand structural reliability.

51 When a set of N values x_i is clustered around a particular one, then it may be useful to characterize the set by a few numbers that are related to its *moments* (the sums of integer powers of the values):

Mean: estimates the value around which the data clusters.

$$\boxed{\mu = \frac{1}{N}\sum_{i=1}^{N} x_i} \tag{1.3}$$

it is the arithmetic average of all the data points.

Expected Value: If data are not available, an expected value is assigned based on *experience and judgment*, and $E(x) = \mu_x$.

Both the mean and the expected values are termed first moment, and they correspond to the centroid of a probability density distribution.

$$E(x) = \mu_x \begin{cases} = \int_{-\infty}^{\infty} x f(x) dx & \text{Continuous systems} \\ = \sum_{i=1}^{N} x f(x) & \text{Discrete systems} \end{cases} \tag{1.4}$$

Median: of a sorted series ($x_{i-1} < x_i < x_{i+1}$) is defined as:

$$x_{med} = \begin{cases} x_{\frac{N+1}{2}} & N \text{ odd} \\ \frac{1}{2}(x_{\frac{N}{2}} + x_{\frac{N}{2}+1}) & N \text{ even} \end{cases} \tag{1.5}$$

Variance: is an indication of the "width" of the cluster:

$$\boxed{\begin{aligned} \sigma^2 &= \frac{1}{N-1}\Sigma_{i=1}^{N}(x_i-\mu)^2 \quad (1.6) \\ &\text{or} \\ \sigma^2 &= \frac{1}{N}\Sigma_{i=1}^{N}(x_i-\mu)^2 \quad (1.7) \end{aligned}}$$

Note that if N is less than 10, it is more appropriate to use the second equation, otherwise use the first one.

Standard Deviation: is defined as the square root of the Variance

$$\boxed{\sigma = \sqrt{\sigma^2}} \tag{1.8}$$

Coefficient of Variation: is the standard deviation normalized with respect to the mean:

$$\boxed{\mathrm{V} = \frac{\sigma}{\mu}} \tag{1.9}$$

When insufficient data are available to accurately compute the moments, the coefficient of variation is often estimated on the basis of experience.

Covariance: Pairs of random variables may be correlated or independent. If correlated, then the likelihood of y depends on the likelihood of x. Thus, covariance σ_{xy} measures the combined effect of how two variables vary together.

$$\boxed{\sigma_{xy} = \frac{1}{N}\sum_{i=1}^{N}(x_i-\mu_x)(y_i-\mu_y)} \tag{1.10}$$

Correlation Coefficient: ρ_{xy} is a nondimentional measure of the degree of correlation

$$\boxed{\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x\sigma_y}} \tag{1.11}$$

A correlation coefficient of 1.0 or -1.0 indicates a perfect linear correlation. A positive value indicates that the variables either increase or decrease together, a negative one indicates that one value increases while the other decreases. A zero value indicates that there is no linear correlation between the variables.

Skewness: characterizes the degree of asymmetry of a distribution around its mean. It is defined in a non-dimensional value. A positive one signifies a distribution with an asymmetric tail extending out toward more positive x

$$\text{Skew} = \frac{1}{N}\Sigma_{i=1}^{N}\left[\frac{x_i-\mu}{\sigma}\right]^3 \tag{1.12}$$

Kurtosis: is a nondimensional quantity which measures the "flatness" or "peakedness" of a distribution. It is normalized with respect to the curvature of a normal distribution. Hence a negative value would result from a distribution resembling a loaf of bread, while a positive one would be induced by a sharp peak:

$$\text{Kurt} = \frac{1}{N}\Sigma_{i=1}^{N}\left[\frac{x_i-\mu}{\sigma}\right]^4 - 3 \tag{1.13}$$

the -3 term makes the value zero for a normal distribution.

52 The expected value (or mean), standard deviation and coefficient of variation are interdependent: knowing any two, we can determine the third.

53 Distribution of variables can be mathematically represented.

54 A Uniform distribution implies that any value between x_{min} and x_{max} is equaly likely to occur.

55 The general normal (or Gauss) distribution is given by, Fig. 1.8:

$$\boxed{\phi(x) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2}} \tag{1.14}$$

56 A normal distribution $N(\mu,\sigma^2)$ can be normalized by defining

$$\boxed{y = \frac{x-\mu}{\sigma}} \tag{1.15}$$

and y would have a distribution $N(0,1)$:

$$\boxed{\phi(y) = \frac{1}{\sqrt{2\pi}}e^{-\frac{y^2}{2}}} \tag{1.16}$$

57 The normal distribution has been found to be an excellent approximation to a large class of distributions, and has some very desirable mathematical properties:

1. $f(x)$ is symmetric with respect to the mean μ.

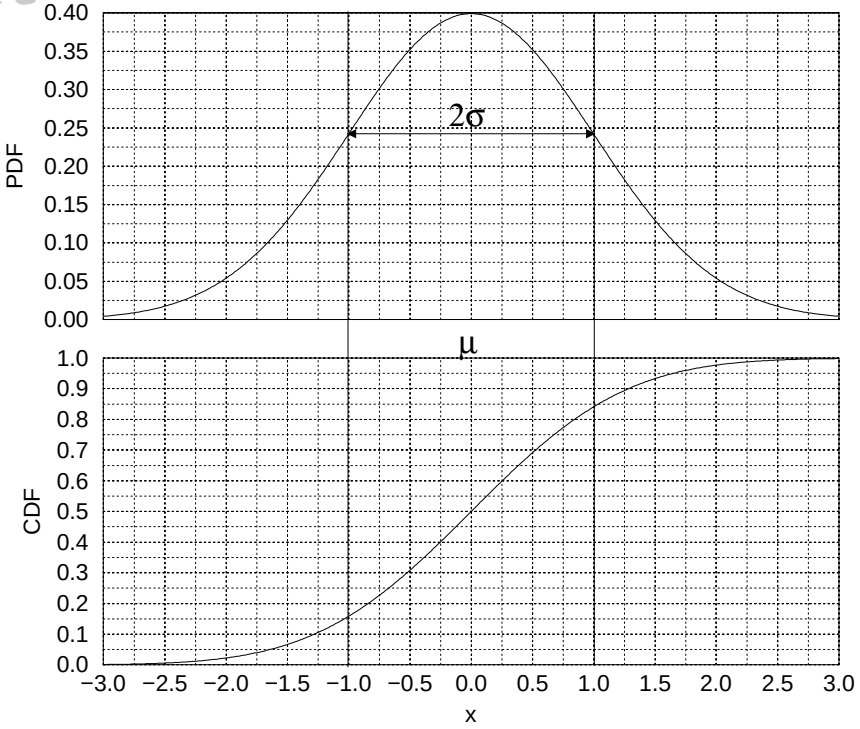


Figure 1.8: Normalized Gauss Distribution, and Cumulative Distribution Function

2. $f(x)$ is a "bell curve" with inflection points at $x = \mu \pm \sigma$.

3. $f(x)$ is a valid *probability distribution function* as:

$$\int_{-\infty}^{\infty} f(x) = 1 \tag{1.17}$$

4. The *probability* that $x_{min} < x < x_{max}$ is given by:

$$P(x_{min} < x < x_{max}) = \int_{x_{min}}^{x_{max}} f(x)dx \tag{1.18}$$

5. *Cumulative distribution functions* (cdf) of the normal distribution defined as:

$$\Phi(s) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{s} e^{-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2} dx \tag{1.19}$$

and is expressed in terms of the error function (erf).

6. The cdf of normalized normal distribution function is given by:

$$\Phi(s) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{s} e^{-\frac{x^2}{2}} dx \tag{1.20}$$

and is usually tabulated in books.

1.2.5.2 Assignment

1. Generate the data for a normal distribution with a mean of 100 and a standard deviation of 20 and plot the probability distribution function from the mean minus the standard deviation, to the mean plus the standard deviation.

2. Retrieve (through anonymous ftp to bechtel) the following two sets `ftp/pub/Structures/cven4837/set1.da` and `ftp/pub/Structures/cven4837/set2.dat`. Those sets contains results of concrete compressive strength f'_c for two different ready mix companies.

   (a) For each set determine the Mean, Standard Deviation, Coefficient of variation, Skewness, and Kurtosis.

   (b) Plot the probability distribution function $f(x)$ (based on a Normal Distribution), plotted from $\mu - 4.\sigma$ to $\mu + 4.\sigma$

   (c) Plot a histogram of the normalized strength (from the input file) with 20 bins.

   (d) Superimpose the two plots.

   Note This problem will be revisited later to: a) use MATLAB statistical functions, and b) Apply numerical integration for the evaluation of the probability of the stregth being below a certain value.

Chapter 2

WEEK II; MATLAB: Matrix Algebra

2.1 Syntax

2.1.1 Matrix Operations

2.1.1.1 Matrix Definition

1 Matrices can be entered in several different ways:

- Entered by an explicit list of elements, or built from blocks. For example,

  ```
  A=[1 4 6; 2 1 8; -2 7 -9];
  B = [A, ones(3,2); zeros(2,3), eye(2)]}
  will build a 5-by-5 matrix.
  ```

- Generated by built-in statements and functions, Table 2.1. For example, `ones(m,n)` produces an m-by-n matrix of ones; if A is a matrix, then `ones(A)` produces a matrix of ones of the same size as A.

  If `x` is a vector, `diag(x)` is the diagonal matrix with `x` down the diagonal; if `A` is a square matrix, then `diag(A)` is a vector consisting of the diagonal of `A`.

- Created in M-files,

- Loaded from external data files

2 The following session is self explanatory,

```
x=[1,2;6,7]
x =
     1     2
```

eye	identity matrix
zeros	matrix of zeros
ones	matrix of ones
diag	see below
triu	upper triangular part of a matrix
tril	lower triangular part of a matrix
rand	randomly generated matrix
hilb	Hilbert matrix
magic	magic square
toeplitz	see `help toeplitz`

Table 2.1: Built In Matrix Definition Functions

```
     6     7

x=[1 2 3
4 5 6
9 10 11]
x =
     1     2     3
     4     5     6
     9    10    11
```

3 "colon operation" is a much more effective way to deal with consecutive numbers than through loops (very slow). For instance

```
x=[3 -1 5]
x =
     3    -1     5

y=[x;x+1;2*x-1]
y =
     3    -1     5
     4     0     6
     5    -3     9

rot90(y)
ans =
     5     6     9
    -1     0    -3
     3     4     5

fliplr(y)
ans =
     5    -1     3
     6     0     4
     9    -3     5
```

```
flipud(y)
ans =
     5    -3     9
     4     0     6
     3    -1     5

y(:,3)
ans =
     5
     6
     9

y(2,:)
ans =
     4     0     6

x(1,:)
ans =
     3    -1     5

z=[y(1,:);y(3,:)]
z =
     3    -1     5
     5    -3     9

v=[0.2 1.4 2.6]
v =
    0.2000    1.4000    2.6000

diag(v)
ans =
    0.2000         0         0
         0    1.4000         0
         0         0    2.6000
```

2.1.1.2 Matrix Operations

Matrix operations are listed in Table 2.2.

The "matrix division" operations deserve special comment. If A is an invertible square matrix and b is a compatible column, vector, then $x = A\backslash b$ is the solution of $A * x = b$

```
>> A=[2 3;1 4];b=[8;9];AInv=inv(A);
>> x=AInv*b
x =
     1
     2

>> x=A\b
```

+	addition
−	subtraction
*	multiplication
^	power
'	transpose
\	left division
/	right division

Table 2.2: Matrix Operations

```
x =

     1
     2
```

2.1.1.3 Matrix functions

Matrix built in functions are listed in Table 2.3.

eig	eigenvalues and eigenvectors
chol	cholesky factorization
svd	singular value decomposition
inv	inverse
lu	LU factorization
qr	QR factorization
logm	matrix logarithm
expm	matrix exponential
sqrtm	matrix square root
poly	characteristic polynomial
det	determinant
size	size
norm	1-norm, 2-norm, F-norm, ∞-norm
cond	condition number in the 2-norm
rank	rank

Table 2.3: Matrix Functions

```
EDU>> x=[4 2 3; 1 6 5;-2 5 10]
x =
     4     2     3
     1     6     5
```

```
    -2     5    10

EDU>> y=inv(x)
y =
    0.2318   -0.0331   -0.0530
   -0.1325    0.3046   -0.1126
    0.1126   -0.1589    0.1457

EDU>> x*y
ans =
    1.0000         0    0.0000
    0.0000    1.0000    0.0000
    0.0000    0.0000    1.0000

EDU>> lu(x)
ans =
    4.0000    2.0000    3.0000
   -0.2500    6.0000   11.5000
    0.5000   -0.9167   -6.2917

EDU>> det(x)
ans =
   151

EDU>> lux=lu(x)
lux =
    4.0000    2.0000    3.0000
   -0.2500    6.0000   11.5000
    0.5000   -0.9167   -6.2917

EDU>> triu(lux)
ans =
    4.0000    2.0000    3.0000
         0    6.0000   11.5000
         0         0   -6.2917

EDU>> tril(lux)
ans =
    4.0000         0         0
   -0.2500    6.0000         0
    0.5000   -0.9167   -6.2917

EDU>> tril(lux,-1)
ans =
         0         0         0
   -0.2500         0         0
    0.5000   -0.9167         0

EDU>> diag(lux)
ans =
    4.0000
    6.0000
```

```
   -6.2917

EDU>> eig(x)
ans =
    4.4123
   12.9438
    2.6440

EDU>> x=[0.5 0.25;0.25 0.5]
x =
    0.5000    0.2500
    0.2500    0.5000

EDU>> [V,D]=eig(x)
V =
    0.7071    0.7071
   -0.7071    0.7071
D =
    0.2500         0
         0    0.7500

EDU>> x*V-V*D
ans =
     0     0
     0     0
```

2.1.2 Graphics Revisited

2.1.2.1 Polar Plots

```
theta=0:pi/60:2*pi;r=2*(1-cos(theta));polar(theta,r);axis square
print -deps2 polar.eps
```

resulting in Fig 2.1

2.1.2.2 3-D mesh plots.

7 Three dimensional mesh surface plots are drawn with the function `mesh`. The command `mesh(z)` creates a three-dimensional perspective plot of the elements of the matrix z. The mesh surface is defined by the z-coordinates of points above a rectangular grid in the x-y plane.

```
% Shape Functions for Quadrilateral Quadratic Elements
X=-1:1/20:1;
Y=X;
YT=Y';
XT=X';
N8=0.5*(1-YT.*YT)*(1-X);
meshc(X,Y,N8)
```

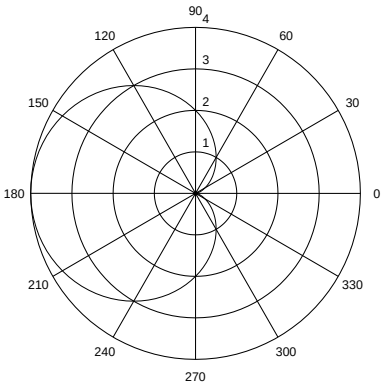


Figure 2.1: Polar Plot of a Cardioid

```
print -deps2 shap8-8.eps
c=contour(X,Y,N8);
clabel(c)
print -deps2 shap8-8-c.eps
```

and the plots are shown in Fig. 2.2

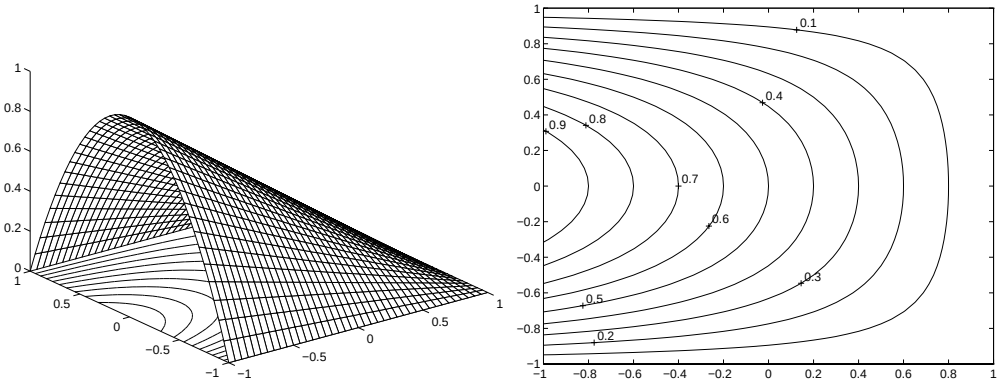


Figure 2.2: Sample of Surface and Contour Plots

2.1.2.3 Animation

8 Animation can be achieved in MATLAB (student's edition may not support it) through the `moviein` and `getframe` commands.

```
X=-1:1/5:1;
```

```
Y=X;
YT=Y';
XT=X';
N8=0.5*(1-YT.*YT)*(1-X);
k=0;
M=moviein(11); % set up for 11 frames
for fac=-5:1:5 % loop
k=k+1;
z=fac*N8;
surf(X,Y,z);
%axis off;
axis([-1 1 -1 1 -5 5]) % freeze the axis to user specified values
M(:,k)=getframe; % grab the frame into M
end
movie(M,10) % play back 10 times
```

will generate the animation of Fig. 2.3

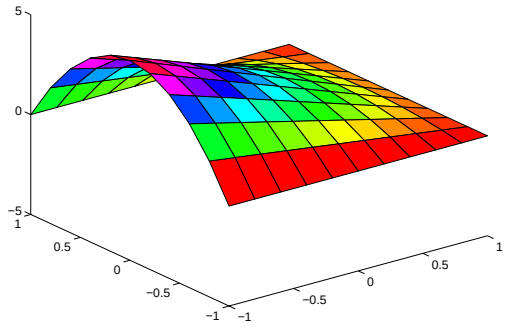


Figure 2.3: Picture Used for Animation

9 Note that alternatively you may generate the image within a loop, and insert a pause after each one.

2.1.3 Data Analysis

2.1.3.1 Statistical analysis

10 Statistical analysis predefined functions are shown in Table 2.4.

```
EDU>> x=rand(1,10) % generate a uniform distribution
x =
  Columns 1 through 7
```

max	maximum value
min	minimum value
mean	mean value
median	median value
std	standard deviation
sort	sorting
sum	sum of elements
prod	product of elements
cumsum	cumulative sum of elements
cumprod	cumulative product of elements
diff	approximate derivatives
hist	histograms
corrcoef	correlation coefficients
cov	covariance matrix
cplxpair	reorder into complex pairs

Table 2.4: Statistical Analysis Functions

```
    0.2190    0.0470    0.6789    0.6793    0.9347    0.3835    0.5194
  Columns 8 through 10
    0.8310    0.0346    0.0535

EDU>> rand('seed',0) %set seed
EDU>> rand('uniform') % specify a uniform random distribution
EDU>> rand(1,10)
ans =
  Columns 1 through 7
    0.2190    0.0470    0.6789    0.6793    0.9347    0.3835    0.5194
  Columns 8 through 10
    0.8310    0.0346    0.0535

EDU>> rand('seed',0) % reset the seed

EDU>> x=randn(1,100); %Generates random numbers with a normal distribution
EDU>> hist(x,20) % Histogram with 20 bins
EDU>> print -deps2 hist1.eps
EDU>> mean(x)
ans =
    0.0507

EDU>> max(x)
ans =
    2.9432

EDU>> min(x)
ans =
```

```
   -2.0186

EDU>> median(x)
ans =
    0.0239

EDU>> sum(x)
ans =
    5.0730

EDU>> std(x)
ans =
    0.9979

EDU>> sort(x)
ans =
  Columns 1 through 7
   -2.0186   -1.8769   -1.7651   -1.6984   -1.6853   -1.6237   -1.4921
.............
Columns 99 through 100
    2.3722    2.9432
```

where the file `hist1.eps` is shown in Fig. 2.4

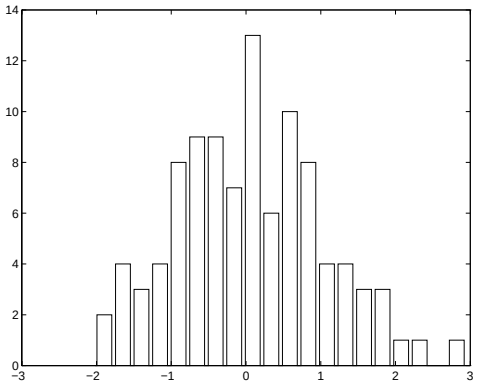


Figure 2.4: Histogram

2.1.3.2 Regression Analysis

The following example will generate data which approximately fit $y = 2x+10$, and then through a linear regression determine the parameters.

```
EDU>> x=[0:1:10];

EDU>> mnoise=0.5*rand(1,11)
mnoise =
  Columns 1 through 7
```

```
    0.0173    0.0267    0.2649    0.3356    0.0038    0.1917    0.0334
  Columns 8 through 11
    0.2087    0.3434    0.2945    0.4652

EDU>> bnoise=2*rand(1,11);

EDU>> y=(2+mnoise).*x+10+bnoise % noisy y=2x+10
y =
  Columns 1 through 7
   11.6923   13.0806   14.7136   18.3146   18.8474   22.3609   24.0212
  Columns 8 through 11
   26.9856   29.2720   30.7453   36.1243

EDU>> coef=polyfit(x,y,1)% get the coefficients
coef =
    2.3546   10.6048

EDU>> m=coef(1); b=coef(2);
yfit=m*x+b;plot(x,yfit,x,y,'o'),title('Linear Regression'),grid
EDU>> print -deps2 regres.eps
EDU>>
```

where the file `regres.eps` is shown in Fig. 2.5.

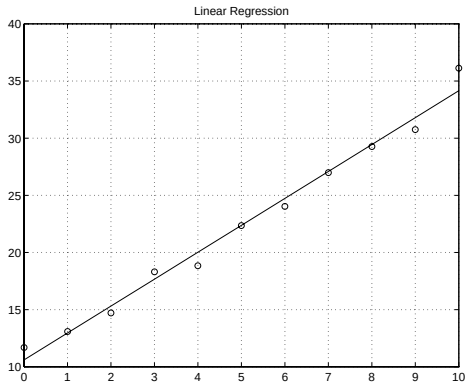


Figure 2.5: Regression Analysis

2.1.3.3 Signal Processing

11 MATLAB has a library of very powerful signal processing functions, Table 2.5.

2.1.4 Control Flow

12 MATLAB programming is very simple and supports the DO, WHILE, and IF constructs.

abs	complex magnitude
angle	phase angle
conv	convolution
corrcoef	correlation coefficients
cov	covariance
deconv	deconvolution
fft	radix-2 fast Fourier transform
fft2	two-dimensional FFT
ifft	inverse fast Fourier transform
ifft2	inverse 2-D FFT
fftshift	FFT rearrangement

Table 2.5: Signal Processing Functions

13 MATLAB provides flow control statements (such as DO, WHILE and IF) which operate like those in most computer languages.

14 Relations and logical operators in MATLAB are given in Table 2.6.

$<$	less than
$>$	greater than
$<=$	less than or equal
$>=$	greater than or equal
$==$	equal
$\sim=$	not equal.
&	and
|	or
$\sim$	not.

Table 2.6: Relations and Logical Operators

15 Note that "=" is used in an assignment statement while "==" is used in a relation.

```
EDU>> x=4*rand(1,10);
EDU>> for k=1:10
   if x(k)>=3
     disp('A')
   elseif x(k)<3 & x(k)>=2
     disp('B')
   else
      disp(' You failed')
   end
end
```

```
 You failed
B

A
 You failed
 You failed
A
B
A
B
```

another example

```
EDU>> max=10;x(1)=1;x(2)=1;k=3;
EDU>> while k<max
     x(k)=x(k-1)+x(k-2);
     k=k+1;
     end
EDU>> x
x =
     1     1     2     3     5     8    13    21    34
```

16 MATLAB has some predefined relational and logical functions, Table 2.7.

any	logical conditions
all	logical conditions
find	find array indices of logical values
isnan	detect NaNs
finite	detect infinities
isempty	detect empty matrices
isstr	detect string variables
strcmp	compare string variables

Table 2.7: Relational and Logical Functions

17 Control flow is described through Table 2.8

2.1.5 Function Files

18 Function files take one or more external arguments (enclosed in parenthesis) and return one or more output.

```
function  hyp=pyt(a,b)
% All commented lines following the function definition
```

if	conditionally execute statements
elseif	used with **if**
else	used with **if**
end	terminate **if, for, while**
for	repeat statements a number of times
while	do while
break	break out of **for** and **while** loops
return	return from functions
pause	pause until key pressed

Table 2.8: Control Flow

```
% will be displayed if the user types help pyt
if nargin < 2 disp('invalid number of arguments'; end
hyp=sqrt(a.^2+b.^2);
```

19 This file should be placed in a diskfile with filename `pyt.m` (corresponding to the function name). The first line declares the function name, input arguments, and output arguments; without this line the file would be a script file. Then a MATLAB statement `c=hyp(4,5)`, for example, will cause the numbers 4 and 5 to be passed to the variables `a` and `b` in the function file with the output result being passed out to the variable `c`. Since variables in a function file are local, their names are independent of those in the current MATLAB environment.

20 Note that use of `nargin` ("number of input arguments") permits one to set a default value of an omitted input variable.

21 A function may also have multiple output arguments. For example:

```
function  [mean, stdev] = stat(x)
% STAT  Mean and standard deviation
[m  n] = size(x);
if m == 1
  m = n;
end
mean = sum(x)/m;
stdev = sqrt(sum(x.^2)/m - mean.^2);
```

Once this is placed in a diskfile `stat.m`, a MATLAB command `[xm, xd] = stat(x)`, for example, will assign the mean and standard deviation of the entries in the vector `x` to `xm` and `xd`, respectively. Single assignments can also be made with a function having multiple output arguments. For example, `xm = stat(x)` (no brackets needed around `xm`) will assign the mean of `x` to `xm`.

2.2 Assignment

2.2.1 Practice

Start by repeating all the examples in this handout.

2.2.2 Polar Plot

Plot the following curve (Folium of Descartes)

$$r = \frac{3a \sin\theta \cos\theta}{\sin^3\theta + \cos^3\theta} \tag{2.1}$$

for $-\pi/6 \leq \theta \leq \pi/2$, use $a = 1$.

2.2.3 Animation

The transverse vibration of an undamped simply supported beam straight beam is given by

$$v = C_2 \sin\frac{n\pi x}{L} \tag{2.2}$$

at thee frequency $\omega_n = n^2\pi^2\sqrt{\frac{EI}{mL^4}}$ Write a function which will accept as argument the mode shape (n which is an integer), the number of frames N and will generate an animation of the beam vibration. Let C_2 vary from $-L/5$ to $L/5$, and take $L = 10$.

2.2.4 Strain Rosette

2.2.4.1 Theory

Experimentally, strains are measured by a *strain gage*. The most common type of strain gage is the bonded resistance strain gage shown in Fig. 2.6. These gages use a grid of fine wire or a

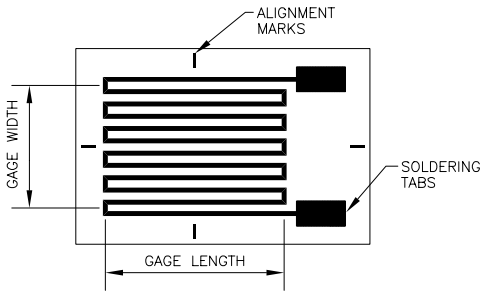


Figure 2.6: Bonded Strain Gage

metal foil grid encapsulated in a thin resin backing. The gage is glued to the carefully prepared test specimen by a thin layer of epoxy. The epoxy acts as the carrier matrix to transfer the

strain in the specimen to the strain gage. As the gage changes in length, the tiny wires either contract or elongate depending upon a tensile or compressive state of stress in the specimen. The cross sectional area will increase for compression and decrease in tension. Because the wire has an electrical resistance that is proportional to the inverse of the cross sectional area, $R\,\alpha\,\frac{1}{A}$, a measure of the change in resistance can be converted to arrive at the strain in the material.

Bonded resistance strain gages are produced in a variety of sizes, patterns, and resistance. One type of gage that allows for the complete state of strain at a point in a plane to be determined is a strain gage rosette. It contains three gages aligned radially from a common point at different angles from each other, Fig. 2.7. The strain transformation equations to

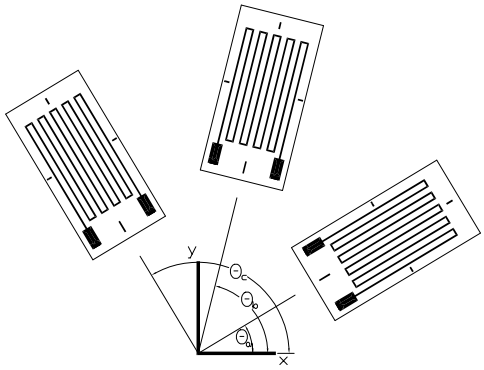


Figure 2.7: Strain Rosette

convert from the three strains at any angle to the strain at a point in a plane are

$$\varepsilon_a = \varepsilon_{xx}\cos^2\theta_a + \varepsilon_{yy}\sin^2\theta_a + \gamma_{xy}\sin\theta_a\cos\theta_a \quad (2.3\text{-a})$$
$$\varepsilon_b = \varepsilon_{xx}\cos^2\theta_b + \varepsilon_{yy}\sin^2\theta_b + \gamma_{xy}\sin\theta_b\cos\theta_b \quad (2.3\text{-b})$$
$$\varepsilon_c = \varepsilon_{xx}\cos^2\theta_c + \varepsilon_{yy}\sin^2\theta_c + \gamma_{xy}\sin\theta_c\cos\theta_c \quad (2.3\text{-c})$$

The angles are usually given by

$$\lfloor\ \theta_a \quad \theta_b \quad \theta_c\ \rfloor = \lfloor\ 0^o \quad 60^o \quad 120^o\ \rfloor \quad (2.4)$$

When the measured strains ϵ_a, ϵ_b, and ϵ_c, are measured at their corresponding angles from the reference axis and substituted into the above equations the state of strain at a point may be solved, namely, ϵ_{xx}, ϵ_{yy}, and γ_{xy}.

The stresses are in turn related to the strains through the elastic constants E and ν, Young's modulus and Poisson's ratio ($lt0.5$)respectively.

$$\begin{aligned}\sigma_{xx} &= \tfrac{E}{(1-2\nu)(1+\nu)}\left[(1-\nu)\varepsilon_{xx} + \nu(\varepsilon_{yy}+\varepsilon_{zz})\right]\\ \sigma_{yy} &= \tfrac{E}{(1-2\nu)(1+\nu)}\left[(1-\nu)\varepsilon_{yy} + \nu(\varepsilon_{zz}+\varepsilon_{xx})\right]\\ \tau_{xy} &= \tfrac{E}{2(1+\nu)}\gamma_{xy}\end{aligned} \quad (2.5)$$

Those would define the stress tensor

$$\begin{bmatrix} \sigma_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_{yy} & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_{zz} \end{bmatrix} \tag{2.6}$$

It can be shown that the solution for principal stresses, i.e. stresses acting on principal planes where there is no shear stress, yields

$$l(\sigma_{xx} - \sigma) + m\tau_{xy} + n\tau_{xz} = 0 \tag{2.7-a}$$
$$l\tau_{xy} + m(\sigma_{yy} - \sigma) + n\tau_{yz} = 0 \tag{2.7-b}$$
$$l\tau_{xz} + m\tau_{yz} + n(\sigma_{zz} - \sigma) = 0 \tag{2.7-c}$$

where l, m and n are direction cosines. Since those are linear and homogeneous equations in l, m and n a solution will exist only if

$$\begin{vmatrix} \sigma_{xx} - \sigma & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_{yy} - \sigma & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_{zz} - \sigma \end{vmatrix} = 0 \tag{2.8}$$

or

$$\sigma^3 - I_1\sigma^2 - I_2\sigma - I3 = 0 \tag{2.9}$$

where I_1, I_2 and I_3 are three invariants equal to

$$I_1 = \sigma_{xx} + \sigma_{yy} + \sigma_{zz} \tag{2.10-a}$$
$$I_2 = \tau_{xy}^2 + \tau_{xz}^2 + \tau_{yz}^2 - \sigma_{xx}\sigma_{yy} - \sigma_{xx}\sigma_{zz} - \sigma_{yy}\sigma_{zz} \tag{2.10-b}$$
$$I_3 = \begin{vmatrix} \sigma_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{xy} & \sigma_{yy} & \tau_{yz} \\ \tau_{xz} & \tau_{yz} & \sigma_{zz} \end{vmatrix} \tag{2.10-c}$$

The three roots ($\sigma_1,\ \sigma_2,\ \sigma_3$) are the three principal stresses.

The direction cosines of the three principal axes are obtained from Eq. 2.7-a-**??** by setting σ in turn equal to $\sigma_1,\ \sigma_2,\ \sigma_3$ and recalling that $l^2 + m^2 + n^2 = 1$.

2.2.4.2 Assignment

Write a function which accepts as input arguments:

1. ε_a ε_b and ε_c from a 60 degrees strain rosette.
2. Young's modulus and Poisson's ratio E, and ν.

Determines

1. Cartesian strains (Eq. 2.3-a-2.3-c)

2. Stresses (Eq. 2.5)
3. Stress invariants (Eq. 2.10-a-2.10-c)
4. Principal stresses (through the eigenvalues in Eq. 2.8)
5. Direction cosines (Eq. **??**; Tricky!).

Test your program for the values shown in Table 2.9. Note that μ equal one microstrain

	Mild Steel	Aluminum	High Strength Concrete
E	30,000 ksi	70 GPa	31 GPa
ν	0.30	0.33	0.20
ε_a	600 μ	2,000 μ	-400 μ
θ_a	0 deg	0 deg	15 deg
ε_b	500 μ	1,500 μ	-800 μ
θ_b	45 deg	120 deg	75 deg
ε_c	-200 μ	-1,300 μ	-1,200μ
θ_c	90 deg	60 deg	135 deg

Table 2.9: Strain Rosette Problems

(10^{-6}in/in)

2.2.5 Structural Design

2.2.5.1 Theory

The design of a steel truss entail the following steps:

1. Structural analysis in which the equations of equilibrium are applied at each node and written in terems of the unknown internal element forces and known externally applied load. This will result in a system of linear equations of the form

$$[\mathbf{B}]\{\mathbf{x}\} = \{\mathbf{P}\} \tag{2.11}$$

   where $\{\mathbf{P}\}$ is the vector of externally applied load, and $\{\mathbf{x}\}$ the vector of unknown element forces (tension if positive, compression if negative). Note that the unknown vector contains both element forces ass well as reactions.

2. Analysis is performed for different loads, and then multiple load cases are coonsidered. Each load case is typically a linear combination of the basic load investigated.
3. Results (internal forces) are tabulated for all load cases, maximum forces determined, and each element is separately designed.

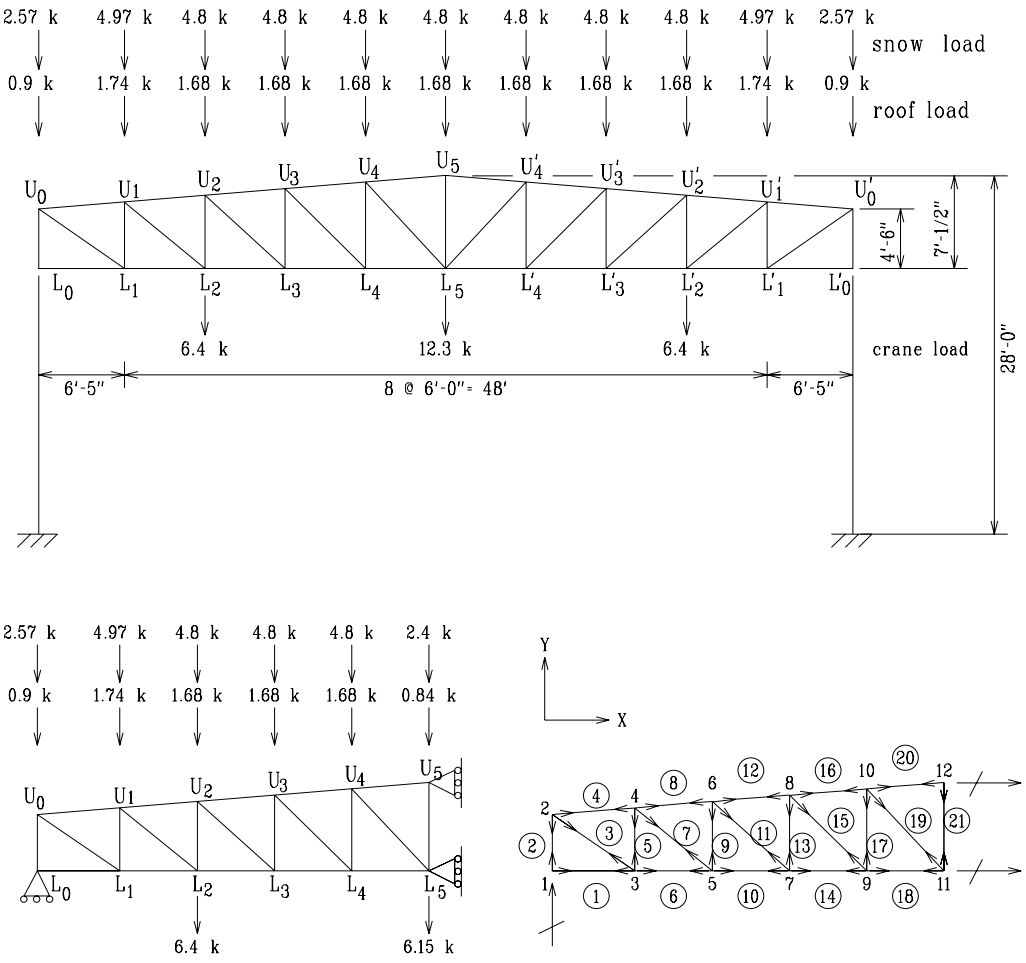


Figure 2.8: Steel Truss

2.2.5.2 Assignment

The truss shown in Fig. 2.8 is to be designed. The statics $[B]$ matrix can be assembled[1] The resulting statics matrix which can be copied from
`ftp/pub/Structures/cven4837/truss.m` is given by

```
b=[...
 1., 0. ,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., ...
0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0., 1. ,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., ...
0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 1. ,  0.,  0.;  ...
 0., 0. , .82, 1. ,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., ...
0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0., -1.,-.57, .08,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., ...
0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.;  ...
-1.,  0.,-.82,  0., 0. , 1. ,  0.,  0.,  0.,  0.,  0.,  0., ...
0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0.,  0., .57,  0., 1. , 0. ,  0.,  0.,  0.,  0.,  0.,  0., ...
0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0.,  0.,  0., -1., 0. ,  0.,-.77, 1. ,  0.,  0.,  0.,  0., ...
0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0.,  0.,  0.,-.08, -1.,  0., .64, .08,  0.,  0.,  0.,  0., ...
0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0.,  0.,  0.,  0.,  0., -1., .77,  0., 0. , 1. ,  0.,  0., ...
0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0.,  0.,  0.,  0.,  0.,  0.,-.64,  0., 1. , 0. ,  0.,  0., ...
0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0.,  0.,  0.,  0.,  0.,  0.,  0.,-1. , 0. ,  0.,.73 ,  1., ...
0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0.,  0.,  0.,  0.,  0.,  0.,  0.,-.08, -1.,  0.,-.68, .08, ...
0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -1.,-.73,  0., ...
0., 1. ,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., 0. ,.68 ,  0., ...
1., 0. ,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., -1., ...
0.,  0., .71,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,-.08, ...
-1.,  0.,-.71,.08 ,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.; ...
 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., ...
0., -1.,-.71,  0., 0. , 1. ,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., ...
0., 0. , .71,  0., 1. , 0. ,  0.,  0.,  0.,  0.,  0.,  0.;  ...
 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., ...
0.,  0.,  0., -1., 0. ,  0.,.68 , 1.0,  0.,  0.,  0.,  0.;  ...
 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., ...
0.,  0.,  0.,-.08, -1.,  0.,-.74, .08,  0.,  0.,  0.,  0.;  ...
 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., ...
0.,  0.,  0.,  0.,  0., -1.,-.68,  0., 0. ,  0., 1. ,  0.;  ...
 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., ...
0.,  0.,  0.,  0.,  0., 0. , .74,  0., 1. ,  0.,  0.,  0.;  ...
 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., ...
0.,  0.,  0.,  0.,  0.,  0.,  0.,-1. , 0. ,  0.,  0.,  1.;  ...
 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0., ...
0.,  0.,  0.,  0.,  0.,  0.,  0.,-.08, -1.,  0.,  0.,  0.];
r=-[0., 0., 0.,  -0.9, 0., 0., 0., -1.74, 0., 0., 0., -1.68,...
0., 0., 0., -1.68, 0., 0., 0., -1.68, 0., 0.,   0., -0.84];
```

[1]Note that in assembling the matrix, each column corresponds to an unknown internal element force, or an external reaction. Each row corresponds to an equation of equilibrium.

```
s=-[0., 0., 0., -2.57, 0., 0., 0., -4.97, 0., 0., 0., -4.8, ...
0., 0., 0., -4.8 , 0., 0., 0., -4.8 , 0., 0.,   0., -2.4];
l=-[0., 0., 0.,    0., 0., 0., 0.,    0., 0.,   -6.4,   0., ...
0., 0., 0.,    0., 0., 0., 0.,    0., 0., 0.,-6.15, 0., 0.];
```

The three load vectors (corresponding to roof, snow, and live loads) are:

```
r=-[0., 0., 0.,  -0.9, 0., 0., 0., -1.74, 0., 0., 0., -1.68,
0., 0., 0., -1.68, 0., 0., 0., -1.68, 0., 0.,   0., -0.84]

s=-[0., 0., 0., -2.57, 0., 0., 0., -4.97, 0., 0., 0., -4.8,
0., 0., 0., -4.8 , 0., 0., 0., -4.8 , 0., 0.,   0., -2.4]

l=-[0., 0., 0.,    0., 0., 0., 0.,    0., 0.,   -6.4,   0.,
0., 0., 0.,    0., 0., 0., 0.,    0., 0., 0.,-6.15, 0., 0.]
```

Write a MATLAB program which will

1. Read the statics matrix and the load vectors
2. Invert the statics matrix [**B**]
3. Solve for the roof, snow, and live load forces
4. Obtain the forces for each of the 3 following three load combinations

Load Case	Roof	Live	Snow
1	1.4	0	0
2	1.2	1.6	0.5
3	1.2	0.5	1.6

5. Determine the maximum load for each member
6. Select the cross sectional area of each member using $\sigma_{\text{allowable}} = 20$ kips/in^2 in compression and nal area of each member using $\sigma_{\text{allowable}} = 24$ kips/in^2 in tension.
7. Tabulate the results as shown in Table 2.10 plus an additional column showing the cross-sectional area in in^2.

Note that Note that the actual results are given by Table 2.10.

2.2.6 Dynamic Response of a Linear Oscillator

2.2.6.1 Theory

Newton's Second Law states that

$$\frac{dm\mathbf{v}}{dt} = \mathbf{f} \tag{2.12}$$

where m is the mass, v the velocity, t time, and $\mathbf{f}$ the force. for a constant mass (unlike a rocket) we have

$$\mathbf{f} = m\mathbf{a} \tag{2.13}$$

Member		Load			Load Cases			Design Load	
		Roof	Snow	Live	1	2	3	Compr.	Tens.
1	$L_0 - L_1$	0.	0.	0.	0.	0.	0.	0	0
2	$U_0 - L_0$	-8.5	-24.	-13.	-12.	-35.	-55.	-55.	0
3	$U_0 - L_1$	12.	34.	20.	17.	52.	79.	0	79.
4	$U_0 - U_1$	-9.8	-28.	-16.	-14.	-43.	-65.	-65.	0
5	$U_1 - L_1$	-6.8	-20.	-11.	-9.6	-30.	-45.	-45.	0
6	$L_1 - L_2$	9.8	28.	16.	14.	43.	65.	0.	65.
7	$U_1 - L_2$	7.3	21.	16.	10.	38.	50.	0.	50.
8	$U_1 - U_2$	-15.	-44.	-29.	-22.	-72.	-100.	-100.	0
9	$U_2 - L_2$	-4.6	-13.	-3.9	-6.5	-14.	-29.	-29.	0
10	$L_2 - L_3$	15.	44.	29.	22.	72.	100.	0	100.
11	$U_2 - L_3$	4.	11.	5.2	5.6	15.	26.	0	26.
12	$U_2 - U_3$	-18.	-52.	-32.	-26.	-83.	-120.	-120.	0
13	$U_3 - L_3$	-2.7	-7.8	-3.6	-3.8	-10.	-18.	-18.	0
14	$L_3 - L_4$	18.	52.	32.	26.	83.	120.	0	120.
15	$U_3 - L_4$	1.4	3.9	4.6	1.9	9.8	10.	0	10.
16	$U_3 - U_4$	-19.	-55.	-36.	-27.	-90.	-130.	-130.	0
17	$U_4 - L_4$	-0.97	-2.8	-3.3	-1.4	-6.9	-7.3	-7.3	0
18	$L_4 - L_5$	19.	55.	36.	27.	90.	130.	0	130.
19	$U_4 - L_5$	-0.89	-2.5	4.2	-1.2	5.1	-3.1	-3.1	5.1
20	$U_4 - U_5$	-19.	-53.	-38.	-26.	-93.	-130.	-130.	0
21	$U_5 - L_5$	0.66	1.9	3.1	0.92	6.	5.3	0	6.

Table 2.10: Result of Truss Design

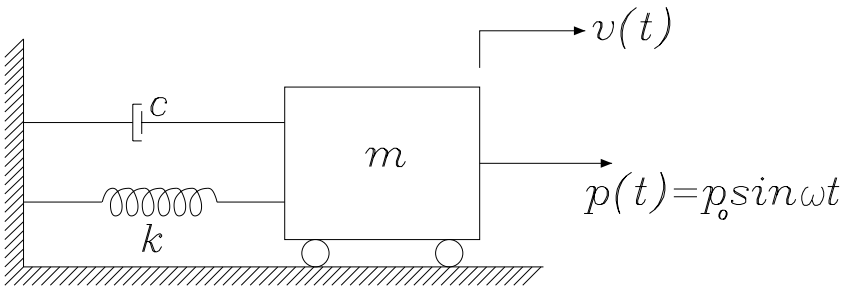


Figure 2.9: Single Degree of Freedom Oscillator

If we consider the linear oscillator shown in Fig. 2.9 setting up the sum of the horizontal forces we have

$$m\ddot{u} + c\dot{u} + ku = P\sin\omega t \tag{2.14}$$

where $\ddot{u}$ and $\dot{u}$ are the acceleration and velocity respectively, k the stiffness of the spring, $P\sin\omega t$ the driving force.

The solution to this problem is of the form

$$u = e^{\lambda t} \tag{2.15}$$

and substituting into Eq. 2.14 we obtain

$$(m\lambda^2 + c\lambda + k)e^{\lambda t} = P\sin\omega t \tag{2.16}$$

Since the exponential is never equal to zero we have

$$\lambda = \frac{-c \pm \sqrt{c^2 - 4mk}}{2m} \tag{2.17}$$

or

$$\lambda = -\frac{c}{2(km)^{1/2}}\left(\frac{k}{m}\right)^{1/2} \pm \left(\frac{k}{m}\right)^{1/2}\left(\frac{c^2}{4km} - 1\right)^{1/2} \tag{2.18}$$

If we define

$$\omega_n = \sqrt{\frac{k}{m}} \quad \text{Undamped natural frequency} \tag{2.19-a}$$

$$\xi = \frac{c}{2\sqrt{km}} = \frac{c}{2m\omega_n} \quad \text{Fraction of critical damping} \tag{2.19-b}$$

then Eq. 2.14 becomes

$$\ddot{u} + 2\xi\omega_n\dot{u} + \omega^2 u = \frac{P}{m}\sin\omega t \tag{2.20}$$

A solution of this equation is

$$u = C_1\sin\omega t - C_2\cos\omega t \tag{2.21}$$

where

$$C_1 = \frac{P}{k}\frac{1 - (\omega/\omega_n)^2}{\left[1 - (\omega/\omega_n)^2\right]^2 + \left[2\xi\omega/\omega_n\right]^2} \tag{2.22-a}$$

$$C_2 = \frac{P}{k}\frac{2\xi\omega/\omega_n}{\left[1 - (\omega/\omega_n)^2\right]^2 + \left[2\xi\omega/\omega_n\right]^2} \tag{2.22-b}$$

$$\phi = \tan^{-1}\frac{C_2}{C_1} \tag{2.22-c}$$

ϕ is the phase lag between the applied force, and the system response.

We can also express the displacement, velocities and accelerations as

$$u = \frac{P}{k} R_d \sin(\omega t - \phi) \tag{2.23-a}$$

$$\dot{u} = \frac{P}{\sqrt{km}} R_v \cos(\omega t - \phi) \tag{2.23-b}$$

$$\ddot{u} = -\frac{P}{m} R_a \sin(\omega t - \phi) \tag{2.23-c}$$

where

$$R_d = \frac{1}{\sqrt{\left[1 - (\omega/\omega_n)^2\right]^2 + \left[2\xi\omega/\omega_n\right]^2}} \tag{2.24-a}$$

$$R_v = \frac{\omega/\omega_n}{\sqrt{\left[1 - (\omega/\omega_n)^2\right]^2 + \left[2\xi\omega/\omega_n\right]^2}} \tag{2.24-b}$$

$$R_a = \frac{(\omega/\omega_n)^2}{\sqrt{\left[1 - (\omega/\omega_n)^2\right]^2 + \left[2\xi\omega/\omega_n\right]^2}} \tag{2.24-c}$$

$$\tag{2.24-d}$$

and correspond to the displacement, velocity and acceleration factors respectively.

2.2.6.2 Assignment

1. Generate surface plots showing
   (a) R_d in terms of $0 \leq \omega/\omega_n \leq 3$ and $0 \leq \xi \leq 1.2$
   (b) R_v in terms of $0 \leq \omega/\omega_n \leq 3$ and $0 \leq \xi \leq 5$
   (c) R_a in terms of $0 \leq \omega/\omega_n \leq 3$ and $0 \leq \xi \leq 5$
   (d) ϕ in terms of $0 \leq \omega/\omega_n \leq 3$ and $0 \leq \xi \leq 5$
2. A 2,000 lbm (or $\frac{2{,}000}{(32.2)(12)} = 5.176$ lb/in/sec^2), is supported by a 60" cantilever with a stiffness $k = 1{,}065 lb/in$; The natural frequency is $\omega_n = \sqrt{\frac{k}{m}} = \sqrt{1{,}0655.176} = 14.34$ rad/sec. An external force with an amplitude of 250 lb which oscillates at 3 cycles per second is applied. The excitation frequency is $\omega = 2\pi f = 2(3.1416)(3) = 18.85$ rad/sec. The system is damped to 2 percent of critical damping. Generate 4 plots (on the same display) showing the excitation force, displacement, velocity and accelerations for $0 \leq t \leq 5$ sec.

2.2.7 Nonlinear Equation

2.2.7.1 Theory

Considering a beam-column subjected to axial and shear forces as well as a moment, Fig. 2.10, taking the moment about i for the beam segment and assuming the angle $\frac{dv}{dx}$ between the axis

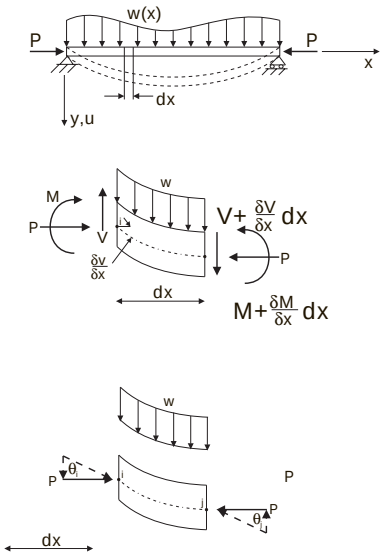


Figure 2.10: Simply Supported Beam Column; Differential Segment; Effect of Axial Force P

of the beam and the horizontal axis is small, leads to

$$M - \left(M + \frac{dM}{dx}dx\right) + w\frac{(dx)^2}{2} + \left(V + \frac{dV}{dx}\right)dx - P\left(\frac{dv}{dx}\right)dx = 0 \tag{2.25}$$

neglecting the terms in dx^2 which are small, and then differentiating each term with respect to x, we obtain

$$\frac{d^2M}{dx^2} - \frac{dV}{dx} - P\frac{d^2v}{dx^2} = 0 \tag{2.26}$$

However, considering equilibrium in the y direction gives

$$\frac{dV}{dx} = -w \tag{2.27}$$

From beam theory, neglecting axial and shear deformations, we have

$$M = -EI\frac{d^2v}{dx^2} \tag{2.28}$$

Substituting Eq. 2.27 and 2.28 into 2.26, and assuming a beam of uniform cross section, we obtain

$$\boxed{EI\frac{d^4v}{dx^4} - P\frac{d^2v}{dx^2} = w} \tag{2.29}$$

Introdcing $k^2 = \frac{P}{EI}$, the general solution of this fourth order differential equation to any set of boundary conditions is

$$v = C_1 \sin kx + C_2 \cos kx + C_3 x + C_4 \tag{2.30}$$

If we consider a column with one end fixed (at $x = 0$), and one end hinged (at $x = L$). The boundary conditions are

$$\begin{array}{rclrcl} v & = & 0, & v_{,xx} & = & 0 \quad \text{at} x = 0 \\ v & = & 0, & v_{,x} & = & 0 \quad \text{at} x = L \end{array} \tag{2.31}$$

These boundary conditions will yield $C_2 = C_4 = 0$, and

$$\sin kL - kL \cos kL = 0 \tag{2.32}$$

But since $\cos kL$ can not possibly be equal to zero, the preceding equation can be reduced to

$$\tan kL = kL \tag{2.33}$$

which is a transcendental algebraic equation and can only be solved numerically.

2.2.7.2 Assignment

Write a MATLAB function to solve for the first 5 roots of Eq. 2.33 by zooming into the intersection of the left hand side with the right hand side.

2.2.8 Newton Raphson Method

2.2.8.1 Theory

Given an equation $f(x) = 0$ with $f(x)$ expandable in a Taylor's series about an initial guess value x_0, we can write

$$f(x) = f(x_0) + f'(x)(x - x_0) + \cdots = 0 \tag{2.34}$$

neglecting high order terms. From this equation we can obtain

$$x = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{2.35}$$

or

$$\boxed{x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}} \tag{2.36}$$

2.2.8.2 Assignment

Write a MATLAB program which will

1. Ask the user (through the `input` function)
   (a) Equation $f(x)$ to be solved
   (b) First derivative of the equation $f'(x)$
   (c) Range of x for plotting (xmin and xmax).
   (d) Initial guess for the solution
   (e) Tolerance
2. Plot the equation for the specified range
3. Solve the equation within the specified tolerance
4. Display the root of the equation and the number of iterations.

Test your program by solving Eq. 2.33 and compare the number of flops (floating point operations) with the preceding solution.

Chapter 3

WEEK III; MATLAB: "Advanced" Topics

3.1 Background

3.1.1 Numerical Integration and Differentiation

1 The integration of $\int_a^b F(x)dx$ is essentially based on passing a polynomial $P(x)$ through given values of $F(x)$ and then use $\int_a^b P(x)dx$ as an approximation.

$$\boxed{\int_a^b F(x)dx \approx \int_a^b P(x)dx} \tag{3.1}$$

2 Using $P(x) = F(x)$ at n points, and recalling the properties of Lagrangian interpolation functions, we obtain

$$\begin{aligned} P(x) &= l_1(x)F(x_1) + l_2(x)F(x_2) + \cdots + l_n(x)F(x_n) && \text{(3.2-a)} \\ &= \sum_{i=1}^{n} l_i(x)F(x_i) && \text{(3.2-b)} \end{aligned}$$

3.1.1.1 Newton-Cotes Method

3 In Newton-Cotes integration, it is assumed that the sampling points are *equally spaced*, Fig. 3.6, thus we define

$$\int_a^b P(x)dx = \int_a^b \sum_i^n l_i(x)dxF(x_i) = \sum_i^n \int_a^b l_i(x)dxF(x_i) \tag{3.3}$$

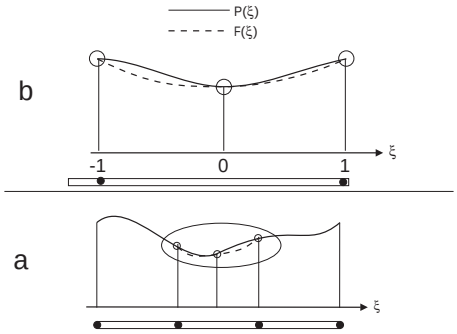


Figure 3.1: Newton-Cotes Numerical integration

or

$$\begin{array}{|r|rcl|}\hline \text{Approximation} & \int_a^b P(x)dx & = & \sum_{i=1}^n W_i^{(n)} F(x_i) \\ \text{Weights} & W_i^{(n)} & = & \int_a^b l_i(x)dx = (b-a)C_i^{(n)} \\ \hline \end{array} \tag{3.4}$$

where $C_i^{(n)}$ are the "weights" of the *Newton-Cotes quadrature* for numerical integration with n equally spaced sampling points.

4 newton-Cotes constants, and corresponding reminder are shown in Table 3.1, (Bathe 1982).

n	$C_0^{(n)}$	$C_1^{(n)}$	$C_2^{(n)}$	$C_3^{(n)}$	$C_4^{(n)}$	Error
2	$\frac{1}{2}$	$\frac{1}{2}$				$10^{-1}(b-a)^3F^{II}(x)$
3	$\frac{1}{6}$	$\frac{4}{6}$	$\frac{1}{6}$			$10^{-3}(b-a)^5F^{IV}(x)$
4	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$		$10^{-3}(b-a)^5F^{IV}(x)$
5	$\frac{7}{90}$	$\frac{32}{90}$	$\frac{12}{90}$	$\frac{32}{90}$	$\frac{7}{90}$	$10^{-6}(b-a)^7F^{VI}(x)$

Table 3.1: Weights for Newton-Cotes Quadrature Formulas

5 It can be shown that this method permits exact integration of polynomial of order $n-1$, and that if n is odd, then we can exactly integrate polynomials of order n. Hence we use in general odd values of n,

6 For $n = 2$ over $[-1, 1]$, we select equally spaced points at $x_1 = -1$ and $x_2 = 1$ to evaluate

$\int_{-1}^{1} P(x)dx$

$$\begin{aligned}
P(x) &= \textstyle\sum_{i=1}^{2} l_i(x)F(x_i) \\
l_1(x) &= \frac{x-x_2}{x_1-x_2} = \frac{1}{2}(1-x) \\
l_2(x) &= \frac{x-x_1}{x_2-x_1} = \frac{1}{2}(1+x) \\
W_1^{(2)} &= \int_{-1}^{1} l_1(x)dx = \frac{1}{2}\int_{-1}^{1}(1-x)dx = 1 \\
W_2^{(2)} &= \int_{-1}^{1} l_2(x)dx = \frac{1}{2}\int_{-1}^{1}(1+x)dx = 1 \\
\int_{-1}^{1} F(x)dx &\approx \int_{-1}^{1} P(x)dx = \textstyle\sum_{i=1}^{2} W_i^{(2)}F(x_i) = F(-1)+F(1)
\end{aligned}$$

which is the *trapezoidal rule*

7 For $n = 3$ over $[-1, 1]$, we select equally spaced points at $x_1 = -1$ $x_2 = 0$, and $x_3 = 1$, to evaluate $\int_{-1}^{1} P(x)dx$

$$\begin{aligned}
P(x) &= \textstyle\sum_{i=1}^{3} l_i(x)F(x_i) \\
l_1(x) &= \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} = \frac{1}{2}x(x-1) \\
l_2(x) &= \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} = -(1+x)(x-1) \\
l_3(x) &= \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} = \frac{1}{2}x(1+x) \\
W_1^{(3)} &= \int_{-1}^{1} l_1(x)dx = \frac{1}{2}\int_{-1}^{1} x(x-1)dx = \frac{1}{3} \\
W_2^{(3)} &= \int_{-1}^{1} l_2(x)dx = \int_{-1}^{1} -(1+x)(x-1)dx = \frac{4}{3} \\
W_3^{(3)} &= \int_{-1}^{1} l_3(x)dx = \frac{1}{2}\int_{-1}^{1} x(1+x)dx = \frac{1}{3} \\
\int_{-1}^{1} F(x)dx &\approx \int_{-1}^{1} P(x)dx = \textstyle\sum_{i=1}^{3} W_i^{(3)}F(x_i) = \frac{1}{3}[F(-1)+4F(0)+F(1)]
\end{aligned}$$

which is *Simpson's rule*

3.1.1.2 MATLAB Examples

8 MATLAB's functions for numerical (definite) integration and differentiation are shown in Table 3.2.

Numerical Integration	
trapz	trapezoidal numerical integration
quad	numerical function integration; Simpson's rule
quad8	Newton-Cotes 8 panel rule
diff	approximate derivatives

Table 3.2: Numerical Integration and Differentiation

9 For example, let us recall the expression for the center of gravity

$$\overline{y} = \frac{\int ydA}{A} \tag{3.5}$$

and moment of inertia

$$I_{xx} = \int_A y^2 dA \qquad I_{yy} = \int_A x^2 dA \tag{3.6-a}$$

$$I_{xx} = I_{xx}^{cg} + Ad_y^2 \qquad I_{yy} = I_{yy}^{cg} + Ad_x^2 \tag{3.6-b}$$

and moment of inertia for a given cross section

for a rectangular cross (width b, height h) section we would have

$$\overline{y} = \frac{\int y dA}{A} \tag{3.7-a}$$

$$= \frac{b\int_0^h y dy}{bh} \tag{3.7-b}$$

$$= \frac{h}{2} \tag{3.7-c}$$

Determining the moment of inertia I_{xx} with respect to the neutral axis (which passes through the centroid)

$$I_{xx} = \int_A y^2 dA \tag{3.8-a}$$

$$= 2b\int_0^{h/2} y^2 dy \tag{3.8-b}$$

$$= \frac{bh^3}{12} \tag{3.8-c}$$

We now seek to determine the centroid of the triangular cross section shown in Fig. 3.2 where $A(-2,0)$, $B(4,0)$ and $C(0,6)$ First we need to define the following functions (note that each one of them must be in a separate file).

```
function y=line1(x)
xx=[-2 0];yy=[0,6];coef=polyfit(xx,yy,1);
y=coef(1).*x+coef(2);

function y=line1x(x)
xx=[-2 0];yy=[0 6];coef=polyfit(xx,yy,1);
y=coef(1).*x.*x+coef(2).*x;

function x=line1y(y)
xx=[-2 0];yy=[0,6];coef=polyfit(yy,xx,1);
x=-(coef(1).*y.*y+coef(2).*y);

function y=line2(x)
```

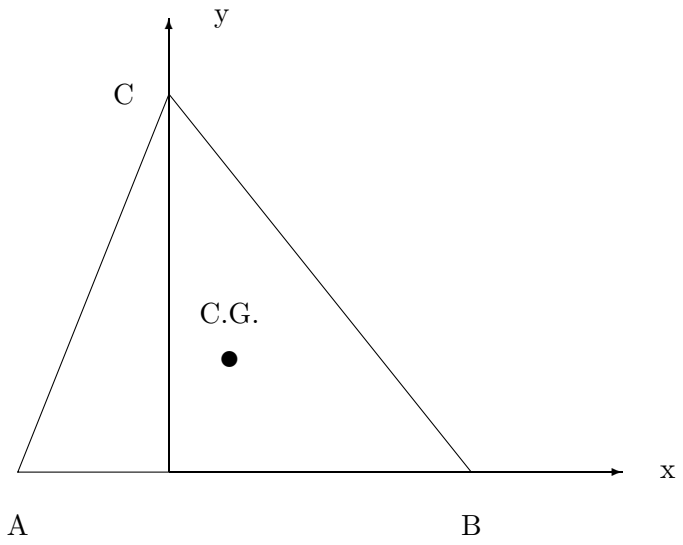


Figure 3.2: Center of Gravity and Moments of Inertia of A Triangular Cross-Section

```
xx=[0 4];yy=[6 0];coef=polyfit(xx,yy,1);
y=coef(1).*x+coef(2);

function y=line2x(x)
xx=[0 4];yy=[6 0];coef=polyfit(xx,yy,1);
y=coef(1).*x.*x+coef(2).*x;

function x=line2y(y)
xx=[0 4];yy=[6 0];coef=polyfit(yy,xx,1);
x=-(coef(1).*y.*y+coef(2).*y);
```

and now the following function will determine the areas and coordinates of the centroid

```
a1=quad('line1',-2,0)
cgx1=quad('line1x',-2,0)/a1
cgy1=quad('line1y',0,6)/a1

a2=quad('line2',0,4)
cgx2=quad('line2x',0,4)/a2
cgy2=-(quad('line2y',0,6)/a2)

a=a1+a2
cgx=(quad('line1x',-2,0)+quad('line2x',0,4))/a
```

```
cgy=(quad('line1y',0,6)-quad('line2y',0,6))/a
```

3.1.2 Nonlinear Equations and Optimization

Nonlinear Equations and Optimization	
fmin	minimum of a function of one variable
fmins	minimum of a multivariable function
fsolve	solution of a system of nonlinear equations (zeros of a multivariable function)
fzero	zero of a function of one variable

Table 3.3: Nonlinear Equations and Optimization

3.1.3 Ordinary Differential Equations

The solution of ordinary differential equations (ODE) can always be reduced to the study of sets of first order differential equations. For example

$$\frac{d^2y}{dx^2} + q(x)\frac{dy}{dx} = r(x) \tag{3.9}$$

can be rewritten as two sets of first order equations

$$\frac{dy}{dx} = z(x) \tag{3.10}$$

$$\frac{dz}{dx} = r(x) - q(x)z(x) \tag{3.11}$$

where z is a new variable. Hence, all problems in ordinary differential equations are reduced to the study of a set of n coupled first order differential equations for the function y_i, $i = 1, 2, \cdots, n$ with the general form

$$\frac{dy_i(x)}{dx} = f(x, y_1, \cdots, y_n) \qquad i = 1, \cdots, n \tag{3.12}$$

where the functions f_i are known derivatives of the $y's$ and the $x's$ are the independent variables. This is known as the *Cauchy form* and can be rewritten as

$$y_1'(x) = f_1(x, y) \tag{3.13}$$

$$y_n'(x) = f_n(x, y) \tag{3.14}$$

where n corresponds to the order of the ODE. In order to solve the differential equation, we also need to have its initial (or boundary) values.

The simplest method to solve for y_{n+1} is the **Euler method** through

$$y_{n+1} = y_n + hf(x_n, y_n) + O(h^2) \tag{3.15}$$

where $x_{n+1} \equiv x_n + h$, and in which a linear approximation is made by using the first two terms of the Taylor's series expansion. We observe that the formula is unsymmetrical (it advances the solution by h but uses the derivative only at the beginning of that interval).

One can improve on Euler's method by

Midpoint method where the derivative is taken at midpoint rather than at the beginning of the step, Fig. 3.3.

$$g_1 = f(x_n, y_n) \tag{3.16}$$

$$y_{n+\frac{1}{2}} = y_n + \frac{h}{2} g_1 \tag{3.17}$$

$$g_2 = f\left(x_n + \frac{h}{2}, y_{n+\frac{1}{2}}\right) \tag{3.18}$$

$$y_{n+1} = y_n + h g_2 \tag{3.19}$$

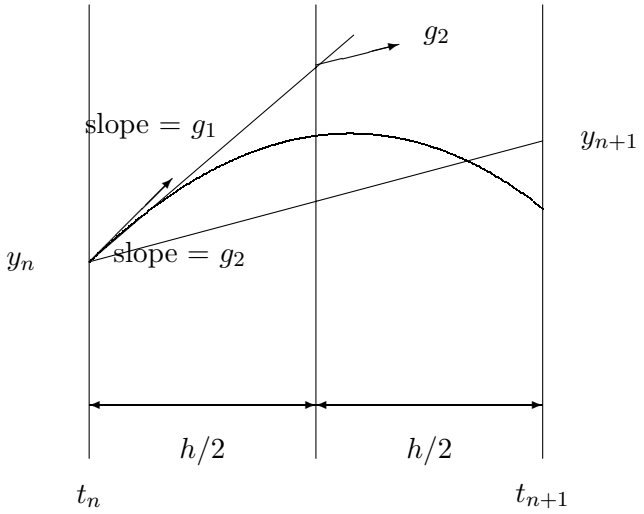


Figure 3.3: Runge's Midpoint Method

Trapezoid method where we consider the average values of the derivatives, Fig.3.4

$$g_1 = f(x_n, y_n) \tag{3.20}$$

$$g_2 = f(x_n + h, y_n + h g_1) \tag{3.21}$$

$$y_{n+1} = y_n + \frac{h}{2}(g_1 + g_2) + O(h^3) \tag{3.22}$$

hence, through the symmetrization the method is now second order[1]. Actually, we have just derived the equations for the *second-order Runge-Kutta* method. For example if

[1] A method is conventionally called n^{th} order if its error is $O(h^{n+1})$.

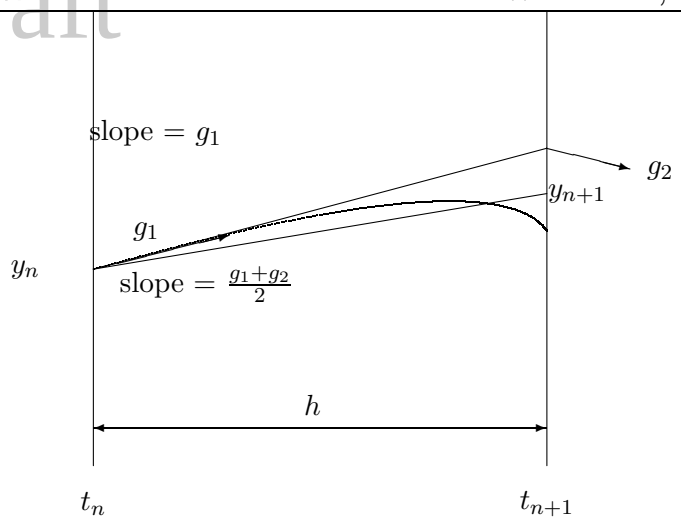


Figure 3.4: Runge's Trapezoid Method

$x_n = 2$, $y_n = 1$, $h = 0.1$ and the differential equation is $\frac{dy}{dx} = x^2 + y(x)^2$, then

$$g_1 = x_n^2 + y_n^2 = 5 \tag{3.23}$$
$$g_2 = (t_n + h)^2 + (y_n + hg_1)^2 = 2.1^2 + 1.5^2 = 6.66 \tag{3.24}$$
$$y_{n+1} = y_n + h\left(\frac{g_1 + g_2}{2}\right) = 1 + 0.1(5.83) = 1.583 \tag{3.25}$$

Fourth-Order Runge-Kutta method is given by, Fig. 3.5.

$$g_1 = f(x_n, y_n) \tag{3.26}$$
$$g_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}g_1) \tag{3.27}$$
$$g_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}g_2) \tag{3.28}$$
$$g_4 = f(x_n + h, y_n + g_3) \tag{3.29}$$
$$y_{n+1} = y_n + h\left(\frac{g_1}{6} + \frac{g_2}{3} + \frac{g_3}{3} + \frac{g_4}{6}\right) + O(h^5) \tag{3.30}$$

and it will require four evaluations of the right hand side per step h

Fourth-Order Runge-Kutta method is given by

$$g_1 = f(x_n, y_n) \tag{3.31}$$
$$g_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}g_1) \tag{3.32}$$
$$g_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}g_2) \tag{3.33}$$
$$g_4 = f(x_n + h, y_n + hg_3) \tag{3.34}$$
$$y_{n+1} = y_n + h\left(\frac{g_1}{6} + \frac{g_2}{3} + \frac{g_3}{3} + \frac{g_4}{6}\right) + O(h^5) \tag{3.35}$$

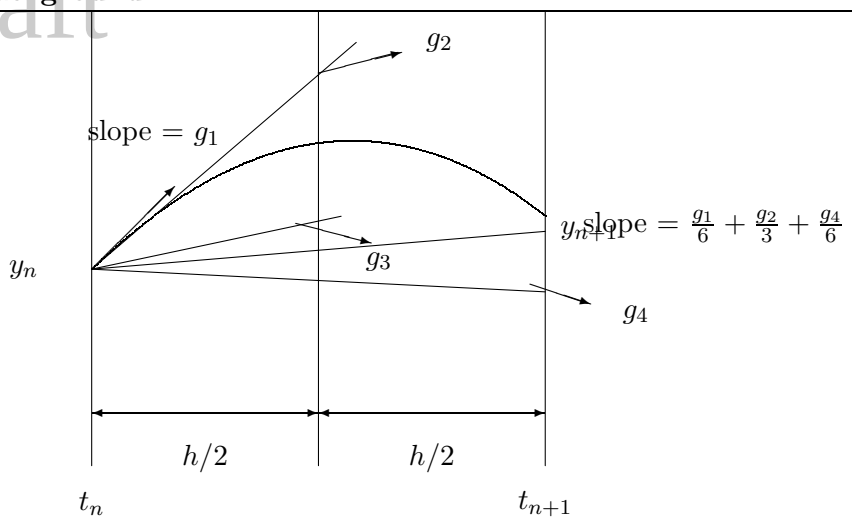


Figure 3.5: 4th Order Runge-Kutta Method

and it will require four evaluations of the right hand side per step h.

Note, in many applications x corresponds to the time t, and thus Equation 3.12 can be rewritten as:

$$\frac{dx_i(t)}{dt} = f(t, x_1, \cdots, x_n) \qquad i = 1, \cdots, n \tag{3.36}$$

10 As an illustrative example, let us consider the following ODE $\frac{dy}{dx} = 1 + y(x)^2$ with $y(0) = 0$, we seek $y(\pi/4)$, and stepsize $h = \pi/4 \simeq 0.78540$

$$\begin{array}{rclcrcl} y_0 & = & 0 & & g_1 & = & 1 \\ y_0 + \frac{h}{2}g_1 & = & 0.39270 & & g_2 & = & 1.1542 \\ y_0 + \frac{h}{2}g_2 & = & 0.45326 & & g_3 & = & 1.2054 \\ y_0 + \frac{h}{2}g_3 & = & 0.94676 & & g_2 & = & 1.8963 \\ y_1 & = & 0.99687 & & & & \end{array} \tag{3.37}$$

11 In this other example, let us consider the following second order ordinary differential equation

$$\frac{d^2y}{dt^2} = \frac{y}{e^t + 1} \qquad y(0) = 1 \qquad \frac{dy}{dt}(0) = 0 \tag{3.38}$$

we use $\Delta t = 0.1$and solve for the first step only. But first we transform it into two coupled equations

$$z = \frac{dy}{dt} \tag{3.39-a}$$

$$\frac{dz}{dt} = \frac{y}{e^t + 1} \qquad z(0) = 1 \tag{3.39-b}$$

$$\frac{dy}{dt} = z \qquad y(0) = 1 \tag{3.39-c}$$

For the first step $y_0 = 1$ and $z_0 = 0$ and we seek y_1 and z_1. We treat the dependent variables and their derivatives as vectors and each component of each vector must be calculated before proceding. Hence,

$$g^1(y,z,t) = \frac{y}{e^t+1}; \quad g^2(y,z,t) = z; \tag{3.40}$$

are components of (g^1 and g^2) the derivative vector $\vec{g}$. Now we can start by evaluating $\vec{g}$ at (y_0, z_0, t_0).

$$\begin{array}{rcl} g_1^1(y_0,z_0,t_0) & = & \frac{y}{e^0+1} = 0.5 \\ g_1^2(y_0,z_0,t_0) & = & 0 \end{array} \tag{3.41}$$

Solve for the locations of the intermediary point

$$\begin{array}{rcl} z_{1/2} & = & z_0 + \frac{\Delta t}{2} g_1^1(y_0,z_0,t_0) = 0 + \frac{0.1}{2}(0.5) = 0.025 \\ y_{1/2} & = & y_0 + \frac{\Delta t}{2} g_1^2(y_0,z_0,t_0) = 1 + \frac{0.1}{2}(0) = 1 \end{array} \tag{3.42}$$

Derivatives

$$\begin{array}{rcl} g_2^1(y_{1/2},z_{1/2},t_{1/2}) & = & \frac{1}{e^{0.05}+1} = 0.487503 \\ g_2^2(y_{1/2},z_{1/2},t_{1/2}) & = & z_{1/2} = 0.025 \end{array} \tag{3.43}$$

compute again the dependent variables

$$\begin{array}{rcl} z_{1/2} & = & z_0 + \frac{\Delta t}{2} g_2^1(y_{1/2},z_{1/2},t_{1/2}) = 0 + \frac{0.1}{2}(0.487503) = 0.024375 \\ y_{1/2} & = & y_0 + \frac{\Delta t}{2} g_2^2(y_{1/2},z_{1/2},t_{1/2}) = 1 + \frac{0.1}{2}((0.025) = 1.001250 \end{array} \tag{3.44}$$

Derivatives

$$\begin{array}{rcl} g_3^1(y_{1/2},z_{1/2},t_{1/2}) & = & \frac{1.001250}{e^{0.05}+1} = 0.488112 \\ g_3^2(y_{1/2},z_{1/2},t_{1/2}) & = & z_{1/2} = 0.024375 \end{array} \tag{3.45}$$

Dependent variables

$$\begin{array}{rcl} z_1 & = & z_0 + \Delta t g_3^1(y_{1/2},z_{1/2},t_{1/2}) = 0 + (0.1)(0.488112) = 0.048811 \\ y_1 & = & y_0 + \Delta t g_3^2(y_{1/2},z_{1/2},t_{1/2}) = 1 + (0.1)(0.024375) = 1.002438 \end{array} \tag{3.46}$$

Derivatives

$$\begin{array}{rcl} g_4^1(y_1,z_1,t_1) & = & \frac{1.002438}{e^{0.1}+1} = 0.476179 \\ g_4^2(y_1,z_1,t_1) & = & z_1 = 0.048811 \end{array} \tag{3.47}$$

First step is now completed, solve for z_1 and y_1

$$z_1 = z_0 + \Delta t \left[\frac{1}{6} g_1^1(y_0,z_0,t_0) + \frac{1}{3} g_2^1(y_{1/2},z_{1/2},t_{1/2}) + \frac{1}{3} g_3^1(y_{1/2},z_{1/2},t_{1/2}) + \frac{1}{6} g_4^1(y_1,z_1,t_1)\right] \tag{3.48-a}$$

$$= 0 + 0.1\left[\frac{1}{6}(0.5) + \frac{1}{3}(0.487503) + \frac{1}{3}(0.488112) + \frac{1}{6}(0.476179)\right] \tag{3.48-b}$$

$$= \boxed{0.048790} \tag{3.48-c}$$

$$y_1 = y_0 + \Delta t \left[\frac{1}{6} g_1^2(y_0,z_0,t_0) + \frac{1}{3} g_2^2(y_{1/2},z_{1/2},t_{1/2}) + \frac{1}{3} g_3^2(y_{1/2},z_{1/2},t_{1/2}) + \frac{1}{6} g_4^2(y_1,z_1,t_1)\right] \tag{3.48-d}$$

$$= 0 + 0.1\left[\frac{1}{6}(0) + \frac{1}{3}(0.0.025) + \frac{1}{3}(0.0.024375) + \frac{1}{6}(0.048811)\right] \tag{3.48-e}$$

$$= \boxed{1.002459} \tag{3.48-f}$$

12 There are two MATLAB functions for the solution of a system of ordinary differential equations, Table 3.4.

Differential Equations	
ode23	2nd/3rd order Runge-Kutta method
ode45	4th/5th order Runge-Kutta-Fehlberg method

Table 3.4: Differential Equations

13 As An illustrative example, let us reconsider the mass-spring-damper system previously analyzed in Sect. 2.2.6.1, the governing differential equation was given by Eq. 2.14

$$m\ddot{x} + c\dot{x} + kx = P\sin\omega t \tag{3.49}$$

we now seek to rearrange this equation in Cauchy form suitable for the Runge-Kutta MATLAB solution.

$$\left.\begin{array}{rcl} \ddot{x} &=& -\frac{c}{m}\dot{x} - \frac{k}{m}x + P\sin\omega t \\ y_1 &=& \dot{x} \\ y_2 &=& x \end{array}\right\} \quad \boxed{\begin{array}{rcl} y_1' &=& -\frac{c}{m}y_1 - \frac{k}{m}y_2 + P\sin\omega t \\ y_2' &=& y_1 \end{array}} \tag{3.50}$$

the initial boundary conditions are $x = 0$ and $\dot{x} = 0$ or

$$y_1(0) = 0 \tag{3.51}$$

$$y_2(0) = 0 \tag{3.52}$$

To analyze this problem with MATLAB we first need to define an M-file which we call `msd.m`

```
function yp=msd(t,y)
% This function defines the differential equations, written in Cauchy
% form, governing the response of a damped mass-spring system subjected
% to a harmonic excitation.
% Assign constants
p= 2250.;      %  lbf
m= 2000.;     % lbm
c=250;        %
k= 1065;     % lb/in
omega= 120.; % radiands
% Initialize the yd matrix
[rows, cols]=size(y); yp=zeros(rows,cols);
% define the derivatives
yp(1)=-c/m*y(1)-k/m*y(2)+p/m*sin(omega*t);
yp(2)=y(1);
```

Now, to see how the system oscillates, we define the initial boundary values

```
t0=0.;y0=[0.;0.];
tf=100;
```

and then the simulation is run through

```
t0=0.;y0=[0.;0.];
tf=30;
[t,y]=ode23('msd',t0,tf,y0);
plot(t,y(:,2)),grid % Note that x corresponds to y(2)
xlabel('Time, t [sec]')
ylabel('Displacement, x [mm]')
pause
plot(y(:,1),y(:,2))
xlabel('Velocity')
ylabel('Displacement')
```

Which will generate Figs. **??**

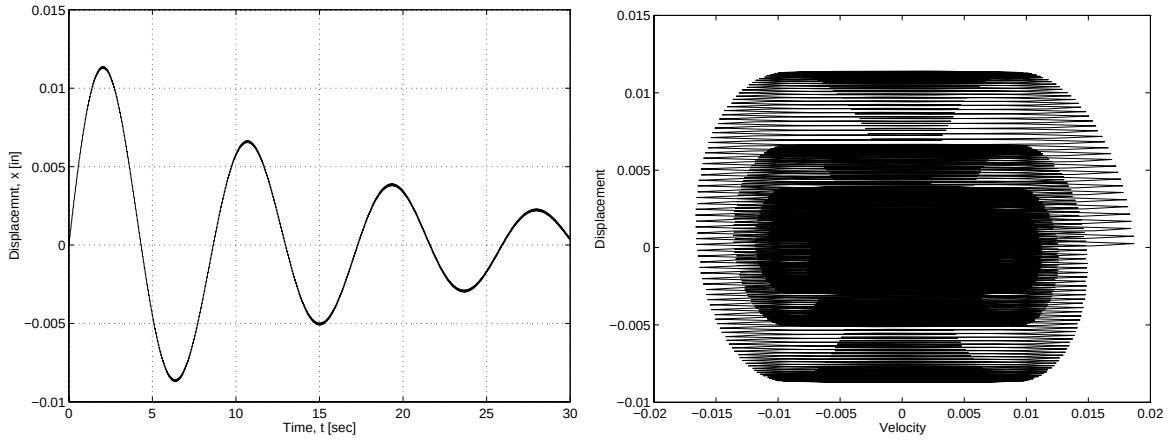


Figure 3.6: Runge-Kutta Solution for the Mass-Spring-Damper System

3.2 Assignment

3.2.1 Practice

Start by repeating all the examples in this handout.

3.2.2 Probability

For the data set 1 of the first homework (`ftp/pub/Matlab/set1.dat`), determine the probability that the concrete strength is between 3,950 psi and 4,050 psi using the following two approaches:

1. From the computed mean and standard deviation, use Eq. 1.18

$$P(x_{min} < x < x_{max}) = \int_{x_{min}}^{x_{max}} f(x)dx \tag{3.53}$$

   where

$$\phi(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left[\frac{x-\mu}{\sigma}\right]^2} \tag{3.54}$$

2. From the raw data.

3. Compare the two probabilities.

3.2.3 Moment of Inertias

Determine the Moment of inertias I_{xx} and I_{yy} for the triangular section of Fig. 3.2.

3.2.4 Reinforced Concrete Ultimate Stress Distribution

In reinforced concrete (r/c) beams, the strain distribution along the depth is assumed to be linearly varying, positive and negative at the lower and upper extreme fibers respectively. At failure, the maximum compressive strain is 0.003. The stress-strain curve of (normal strength) concrete is nonlinear, hence the exact stress distribution is also nonlinear.

In order to determine the ultimate load carrying capacity of a r/c beam, all we need is the resultant and location of the resultant force. Through extensive experimental studies, it was determined that a rectangular stress block, Fig. 3.7, would have the same resultant force F

$$F = \underbrace{0.85 f'_c}_{\sigma} \underbrace{ab}_{A} \tag{3.55}$$

acting at a distance $a/2$ from the top where

$$a = \beta_1 c \tag{3.56-a}$$
$$\beta_1 = 0.85 \tag{3.56-b}$$

In order to assess the accuracy of this equation, the following stress-strain curve was experimentally obtained

$$\sigma = \frac{2\frac{f'_c}{\varepsilon_{max}}}{1 + \left(\frac{\varepsilon}{\varepsilon_{max}}\right)^2}\varepsilon \tag{3.57}$$

Using this equation,

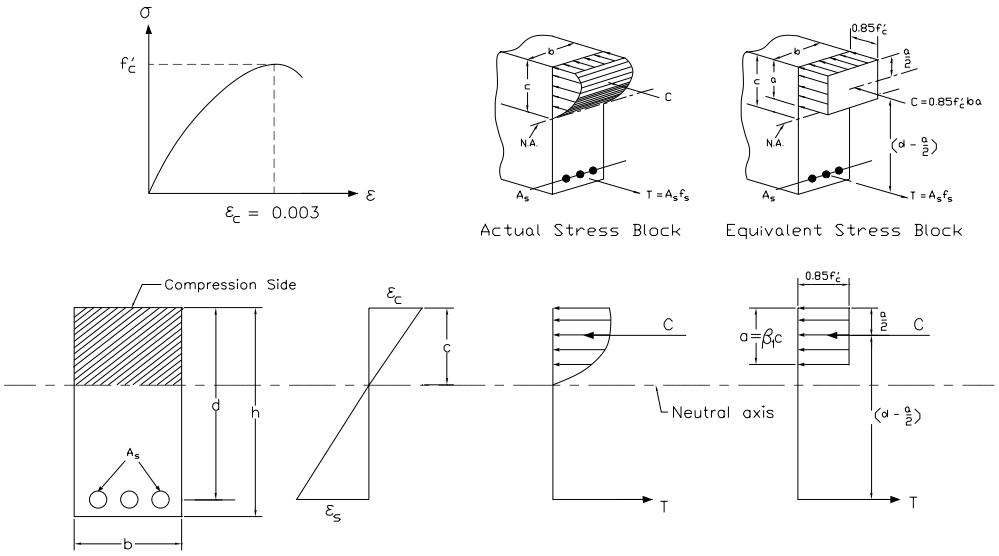


Figure 3.7: Equivalent and Exact Stress Distribution in Reinforced Concrete Beams

1. Determine the resultant force and its location
2. Compare with the actual simplified approximation (where the resultant force is at $a/2$ fro the top and is equal to $0.85 f'_c ab$).

Use: $f'_c = 3,000$ lbs/in^2, $\varepsilon_{max} = 0.003$, $c = 4.in$.

3.2.5 Mixture Problem, [2]

A 120 gallon tank contains 90 pounds of salt dissolved in 90 gallons of water. Brine containing 2 pounds of salt per gallon is flowing into the tank at a rate of 4 gallons per minute. The mixture flows out of the tank at the rate of 3 gallons per minute. The differential equation that specifies the amount of slat $x(t)$ in pounds in the tank at time t is

$$x' = 8 - \frac{3}{90 - t}.x \tag{3.58}$$

The tank is full after 30 minutes.

1. Determine and plot the amount of salt in the tank from time=0 until the tank is full.
2. Determine the amount of time required for the tank to contain 150 pounds of salt.
3. The analytical solution to the differential equation is

$$x(t) = 2(90 + t) - \frac{90^4}{(90 + t)^3} \tag{3.59}$$

compare your numerical results with the exact ones.

3.2.6 Dog-Tracking, [1]

A number of interesting curves are trajectories of a point D moving at constant speed while always pointing toward another point M that moves along a known path. Such a method of tracking is known as *dog-tracking* because it resembles the path followed by a dog chasing his master. Similar curves are those followed by a vessel chasing another one, and by some ground-to-air or air-to-air missiles.

Determine and plot the trajectories of a dog chasing his master for each of the following two cases:

1. The master moves on a straight line and
   (a) Initial dog position (x_0, y_0)=(0,0)
   (b) Dog speed v_d=10m/s
   (c) Master position at time t, $x_m = v_m t$, and y_m=100m
   (d) Master speed v_m=5m/s
   (e) Initial and final times $t_0 = 0$ and t_f=10 s.
2. The dog is at the center of a circular pond of radius r while his master walks on the bank at constant velocity v_m. The dog swims at a constant speed v_d always toward his master and
   (a) Initial dog position (x_0, y_0)=(0,0)
   (b) Dog speed v_d=2.5m/s
   (c) Master speed v_m=2m/s
   (d) Radius r = 15m

Hint: $\vec{v}_d = v_d \frac{\vec{DM}}{|\vec{DM}|}$, this will result into two separate equations (one for the x component and one for the y component).

3.2.7 Ballistic Model, [1]

A simple ballistic model assumes that only two forces act on a projectile, namely gravity and drag. According to this model, the equation of motion are

$$mv'_x = -W\frac{v_x}{v} \tag{3.60-a}$$

$$mv'_y = -W\frac{v_y}{v} - mg \tag{3.60-b}$$

where v_x and v_y are components of the velocity and v its magnitude. The drag W is given by

$$W = c_w \frac{\rho}{2} v^2 \frac{\pi}{4} d^2 \tag{3.61}$$

where c_w is the drag coefficient and is nearly constant as long as the flight is at subsonic speed, ρ is the air density, and d the diameter of the projectile.

Plot the trajectory, using the following parameters: g=9.81 ms^{-2}, m=10 kg, c_w=0.2, ρ=1.225kgm^{-3}, d=0.005m. The initial conditions are x_0=0m, y_0=0m, v_0=250 m/s and the gun elevation is $\pi/6$, Use `ode23` with a tolerance of 5.10^{-4}.

Chapter 4

Week IV: MATHEMATICA

4.1 Background

4.1.1 Introduction

4.1.1.1 What is *Mathematica* ?

1 *Mathematica* is a program for *symbolic mathematics*.

2 With *Mathematica*, you can perform algebraic operations of various complexities. Hence, rather than numerically solving an Engineering problem, you can solve it algebraically.

3 Because of its power, *Mathematica* will enable you to easily address problem of such complexities that it would have been impractical to solve otherwise.

4 Finally, *Mathematica* has a very powerful set of graphics capabilities.

4.1.1.2 Availability

5 Mathematica is available in the Bechtel Laboratory.

6 A relatively inexpensive Student Edition can be purchased from the Buffalo Chip, and the University used to have a site license for it.

4.1.1.3 Front End and Kernel

7 *Mathematica* consists of two parts: the Kernel which is the computation engine, and the Front End which is the user interface. The Kernel is identical on all computers, however the Kernel may vary.

8 There are two Front Ends supported in the Bechtel Lab:

Command-Line Interface: where the current input line is displayed on the last line of the screen (it can be edited by using the arrow, insert and delete keys). This Front End does not support graphics, and is invoked by the `math` command. It is a convenient (and only available one) to use if you connect to the server via a remote terminal which does not support graphics (such as a home pc). Files storing *Mathematica* commands can be created by any ASCII editor and are usually given the file extension `.m`

Notebook Front End: which can contain a mixture of text, graphics, and *Mathematica* definitions, Fig. 4.1.

In this environment there is no need to use an external editor since the Notebook Front End plays also that role. Notebook files usually have a `.ma` or `.mb` for ASCII (to facilitate transfer to other computers) or binary files.

9 In both cases *Mathematica* is used interactively

4.1.1.4 Accessing Mathematica

10 In the Bechtel Lab, Mathematica is accessible through by clicking the left mouse button and then selecting Mathematica. Note that the program is licensed to run only on the server (bechtel).

11 If you want to run the program differently, you need to
`xhost +` on your console
`rlogin bechtel` To connect you to the server
`setenv DISPLAY yourmachine:0.0` To have the graphics displayed on your workstation
Type **math** for the Command Line Front End. To exit you simply type `Exit`.

12 Alternatively, you may simply type **mathematica** for the Notebook Front End.

4.1.1.5 Help

13 For help, simply type `?`. For a specific function, you can type `?Sqrt` or `?P*`.

14 Note that after each command, you should hit the following two keys *simultaneously* <*Shift*> <*return*>

15 The Notebook Front End also provides extensive help, Fig. 4.2.

4.1.1.6 Notebook Front End

4.1.1.6.1 Pointers 16 As the mouse moves around the Notebook, the shape of the pointer changes:

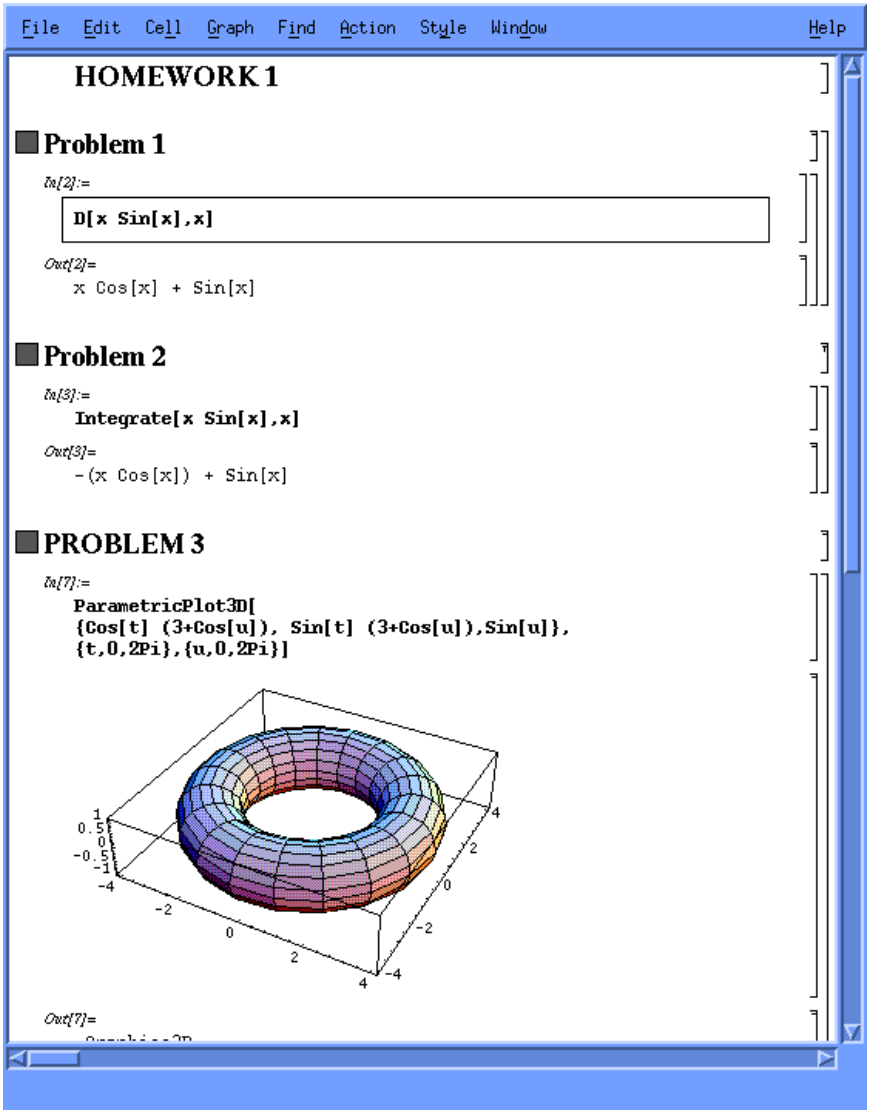


Figure 4.1: Notebook Front End for Mathematica

Figure 4.2: Notebook Front End Help for Mathematica

Vertical I-bar appears when pointing to text (or input) that can be edited. If you click on the mouse, the pointer will then blink.

Horizontal I-bar appears when pointing to the space between two cells, or before the first and after the last one. If you click the mouse, a horizontal line is drawn. Anything typed after will be part of a new cell.

Cell-bracket Pointer appears when pointing to a cell bracket. If you click the mouse to select a group a cell (or group of cells), then you can copy the cell, paste something else into it, or re-execute the cell. If you double click, it will open or close the cell.

Formatted-Cell Pointer appears when pointing to cells that are formatted (such as *Mathematica* output or graphics which can not be changed).

Graphics Pointer appears when pointing inside the bounding box of a selected graphics. You can drag the graphic by depressing the mouse button and then moving the mouse.

Sizing Pointers appears when pointing to handles of graphic's bounding box. Click and drag the mouse to rescale the graphic in the direction shown by the pointer.

Watch Pointer *Mathematica* is busy can not be interrupted!.

4.1.1.6.2 Cell Brackets [17] A cell is the basic unit of organization in a Notebook. It can contain text, *Mathematica* input or output, or other cells. The style of the bracket in the right margin gives an indication of the cell attributes, and its size indicates its extent.

Unformatted Cell contains ordinary text that can be edited such as *Mathematica* input or text.

Inactive Cell can not be executed (such as *Mathematica* output, and they are usually formatted.

Initialization Cell can be automatically evaluated when the Notebook is opened

Locked Cell means that its content can not be altered unless it is first unlocked.

Closed Group implies that only the first cell in the group is visible, and all others are not.

4.1.1.7 Brackets, Parentheses, and Braces

Brackets are used to specify arguments of functions.

Parenthesis are used for grouping. Without them, multiplication and division have a higher precedence than addition and subtraction.

Braces are used to specify lists, vectors, and matrices.

Double Brackets are used for indexing.

Comments are enclosed by `(* Comment *)`.

4.1.2 Examples

4.1.2.1 Basic Arithmetic Operation

18 You can perform arithmetic operations with *Mathematica* as you would perform them with a pocket calculator:

1. Simple

```
In[1]:= 6 + 5
Out[1]= 11
```

2. or slightly more complex operations:

```
In[2]:=6 ((3+4)^2-(5/6*7)+8 (9+10))
Out[2]=1171
```

   The arguments of all *Mathematica* functions are enclosed in square brackets, and the names of built-in functions begin with capital letters.

3. and now try to do this on your calculator:

```
In[1]:= 3^99
Out[1]= 1717925069106704436788203765885404242340358406667
```

   which is the **exact** value of 3^{99}.

4.1.2.2 Approximate Numerical Values

19 You can use the *Mathematica* function N to get approximate numerical results.

```
= N[3^99]
                   47
Out[3]= 1.71793 10
In[4]:= 3^99//N
                   47
Out[4]= 1.71793 10
```

4.1.2.3 Complex Numbers

20 *Mathematica* can also handle complex numbers:

```
In[2]:= (2+5 I) (2-3 I)
Out[2]= 19 + 4 I
```

4.1.2.4 Derivatives

21 Here is an example of a derivative.

```
In[1]:= D[x^2 Sin[a x],x]

           2
Out[1]= a x  Cos[a x] + 2 x Sin[a x]
```

4.1.2.5 Integration

```
In[1]:= Integrate[(x+y)^5,x]

         6             4  2       3  3      2  4
        x     5     5 x  y    10 x  y    5 x  y       5
Out[1]= -- + x  y + ------- + -------- + ------- + x y
        6              2         3          2

In[2]:=  Integrate[Sin[x],{x,a,b}]

Out[2]= Cos[a] - Cos[b]

In[3]:= Integrate[Sin[x]+Sin[y],{x,0,Pi/2},{y,0,x}]

        Pi
Out[3]= --
        2
 In[4]:= NIntegrate[Sin[x]^2-Sin[2 x],{x, 0,Pi}]

 Out[4]= 1.5708
```

4.1.2.6 Algebraic Formulae

22 *Mathematica* can expand or reduce mathematical expressions

```
In[5]:= Expand[(x+y)^5]

         5      4         3  2       2  3        4    5
Out[5]= x  + 5 x  y + 10 x  y, + 10 x  y  + 5 x y  + y
```

4.1.2.7 Solving equations

23 *Mathematica* can determine the root of equations:

```
In[15]:= Solve[x^2+3x-a==0,x]

                 -3 + Sqrt[9 + 4 a]          -3 - Sqrt[9 + 4 a]
Out[15]= {{x -> ------------------}, {x -> ------------------}}
                        2                           2
```

4.1.2.8 Matrices

```
In[28]:= m=Table[(i),{i,9},{j,1}]
Out[28]= {{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}}

In[26]:= m=Table[(i+j),{i,0,8},{j,1}]
Out[26]= {{1}, {2}, {3}, {4}, {5}, {6}, {7}, {8}, {9}}

In[29]:=  m=Table[(i),{i,9},{j,2}]
Out[29]= {{1, 1}, {2, 2}, {3, 3}, {4, 4}, {5, 5}, {6, 6}, {7, 7}, {8, 8},
 >    {9, 9}}
```

24 Matrices are presented in *Mathematica* as a list of lists. *Mathematica* can add, multiply, determine the inverse, determinant, eigenvalues and eigenvectors of matrices:

```
In[37]:= m=Table[(j+i),{i,0,1},{j,2}]
Out[37]= {{1, 2}, {2, 3}}

In[41]:=Inverse[m]              or      In[40]:= Inverse[%37]
Out[40]= {{-3, 2}, {2, -1}}

In[41]:= Det[%]
Out[41]= -1

In[43]:= %37.%40
Out[43]= {{1, 0}, {0, 1}}
```

```
In[44]:= a={{12,7,9},{4,3,1},{11,6,3}}

Out[44]= {{12, 7, 9}, {4, 3, 1}, {11, 6, 3}}

In[45]:= Inverse[a]

            3      33   5     1   63    6      9     5      2
Out[45]= {{-(--), -(--), --}, {--, --, -(--)}, {--, -(--), -(--)}}
            52     52   13    52  52    13     52    52     13

In[46]:= %2.%1

Out[46]= {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}}

In[47]:= Eigenvalues[a]//N

Out[47]= {20.4216, -3.21391, 0.79228}

In[48]:= Det[a]

Out[48]= -52
```

4.1.2.9 Graphics and Three-Dimensional Plots

```
In[13]:= Plot[Sin[x],{x,0,2 Pi}]
Out[13]= -Graphics-


In[10]:= Plot3D[Exp[x 2 y],{x,-2,2},{y,-2,2}]
Out[10]= -SurfaceGraphics-
```

4.1.2.10 Interfacing with *Mathematica*

4.1.2.10.1 Input [25] Rather than typing all the input commands, you can store them in an ASCII file, the extension `.m` is recommended. To execute the file, simply type:

```
In[1]:= << <filename>>
```

where `<filename>` is the file name. If you had an `.m` extension, you do not need to specify it.

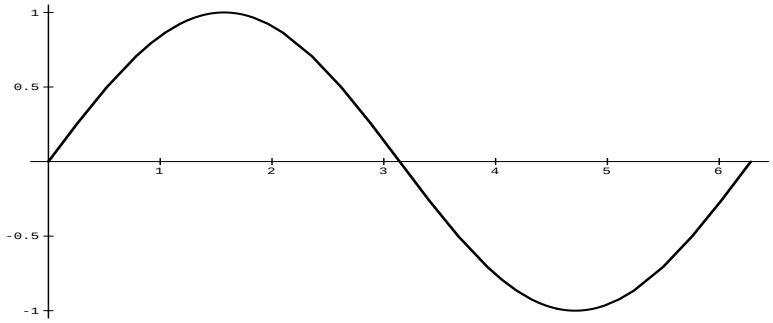


Figure 4.3: Two-dimensional Plot generated from *Mathematica*

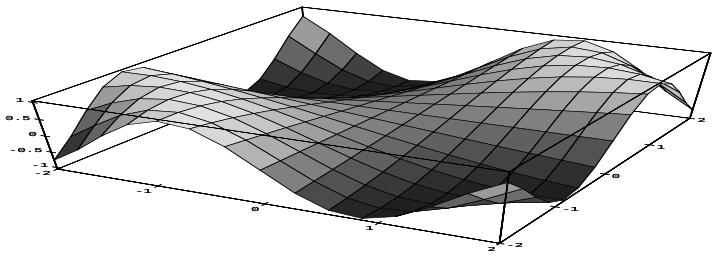


Figure 4.4: Three-dimensional Plot generated from *Mathematica*

4.1.2.10.2 Output

[26] *Mathematica* can be interfaced

```
In[10]:= Integrate[x/(1+xsinx),x]

                2
               x
Out[10]= -------------
         2 (1 + xsinx)
```

with:

Mathematica **Form** `In[14]:= InputForm[%]`

```
Out[14]//InputForm= x^2/(2*(1 + xsinx))
```

Fortran `In[12]:= FortranForm[%]`

```
Out[12]//FortranForm= x**2/(2*(1 + xsinx))
```

C `In[11]:= CForm[%]`

```
Out[11]//CForm= Power(x,2)/(2*(1 + xsinx))
```

TEX `In[13]:= TeXForm[%]`

```
Out[13]//TeXForm= {{{x^2}}\over {2 \left( 1 + {\it xsinx} \right) }}
```

4.1.2.11 Packages

[27] Most function in *Mathematica* are written in C. However, some functions are written in *Mathematica* itself. Such functions are defined in files called *packages* which will allow you to:

1. Define a function or set of functions that are often used
2. Hide the implementation from the user
3. Save the functions and reload them when needed.

4.1.2.12 Graphics hardcopy

[28] The function PSPrint will generate a postscript file which can be later sent to the laser printer.

`Type: PSPrint[%n]`

29 If you want to save your graphics use the *Mathematica* command:

```
Display["file name",%n]
```

Where n is the number of your graphics output. Then in shelltool window use the psfix command to create a PostScript file.

4.1.2.13 Input file

30 You can store all your operations into an ASCII file through a text editor, and then "load" it into *Mathematica*

```
See the {\it Mathematica} manuals for further info on input files.
```

4.1.3 Some *Mathematica* Commands

4.1.3.1 Basic Operations

31 Following are the basic operations supported by *Mathematica* Note that multiplication can be specified by either using an asterisk or by leaving a blank space between arguments.

x^y	power
-x	minus
x/y	devide
x y z or x*y*z	multiply
x+y+z	add

4.1.3.2 Mathematical Functions

32 Following is a list of the most commonly used functions, a complete list is presented at the end of the chapter.

Sqrt[x]	square root
Exp[x]	exponential
Log[x]	natural logarithm
Log[b,x]	logarithm to base b
Sin[x], Cos[x]	trigonometric functions (with arguments in radians)
Tan[x]	
ArcSin[x]	inverse trigonometric functions
ArcCos[x],ArcTan[x]	
n!	factorial (product of integers 1,2,...n)
Abs[x]	absolute value
Round[x]	closest integer to x
Mod[n, m]	n modulo m (remainder on division of n by m)
Random[ ]	pseudorandom number between 0 and 1
Max[x,y,..],	maximum,minimum of x,y,...
Min[x,y,..]	
Grad[f]	Gradient in the given system
Div[f]	Divergence
Curl[f]	Curl

33 Note that all Mathematica built-in functions start with an upper case letter, and the arguments are enclosed in square brackets.

4.1.3.3 Some *Mathematica* Constants

34 Mathematica also has some physical constants hardwired into the system.

Pi	3,14159
E	2.71828
Degree	3.14159/180:degrees to radians conversion factor
I	$\sqrt{-1}$
Infinity	∞

4.1.3.4 Complex Numbers

x + Iy	The complex number x + i y
Re[z]	Real part of z
Im[z]	Imaginary part of z
Conjugate[z]	Complex conjugate of z
Abs[z]	Absolute value of z
Arg[z]	The argument phi in the exponential form of a complex number

4.1.3.5 Recall of Previous Expressions

%	Last result
%%	Next to last result
% n	Result on output line *Out[n]*

4.1.3.6 Assignment of Variables

x = value	Assign value to x
x = y = value	Assign value to x and y
x = . or Clear[x]	Remove assignment of x

4.1.3.7 Brackets

(term)	Parentheses for grouping
f[x]	Square brackets for functions
{a, b, c}	Braces for lists
v[[i]]	Double brackets for indexing

4.1.3.8 Help

?name	Display information on name
??name	Display more information on name
?Aaaa*	Display information on all commands whose names begin with C*

4.1.3.9 Interrupting Mathematica

Continue	Continue calculation
Show	Show what *MATHEMATICA* is doing
Inspect	Inspect current state
Abort	Abort this particular calculation
Exit	Exit *MATHEMATICA*

4.1.3.10 Transformation of Algebraic Expressions

Expand[expr]	Expand an expression into sum of terms
Factor[expr]	Write expr as a minimal product of factors
Simplify[expr]	Simplify expression
Coefficient[expr, form]	Coefficient of form in expr
Numerator[expr]	Numerator
Denominator[expr]	Denominator
expr // Short	Show one line outline of output
Short[expr, n]	Show n-line outline of result
Eliminate[{lhs1 == rhs1,lhs2 == rhs2, ...}, {x, ...}]	Eliminate x, ... from the set of simultaneous equations
Reduce[{lhs1 == rhs1, lhs2 == rhs2, ...}, {x, y, ...}]	Give a set of simplified equations, including all possible solutions

4.1.3.11 Differentiation

D[f, x]	Partial derivative of f with respect to x
D[f, x1, x2, ...]	Multiple derivative of f with respect to x1, x2, ...
D[f, {x, n}]	The nth derivative of f with respect to x
Dt[f]	The total derivative of f
Dt[f, x]	The total derivative of f with respect to x
Limit[f, x-¿xo]	The limit of f as x goes to xo

4.1.3.12 Integration

Integrate[f, x]	Indefinite integral of f with respect to x
Integrate[f, {x, xmin, xmax}]	Definite integral of f wrt to x from xmin to xmax
Integrate[f, {x, xmin, xmax}, {y, ymin, ymax}]	Multiple integral of f with respect to y and x

4.1.3.13 Summation & Products

Sum[f, {i, imin, imax, di}]	Summation with i incremented by di
Sum[f, {i, imin, imax}, {j, jmin, jmax}]	Nested summation of f with respect to j then i
Product[f, {i, imin, imax}]	The product of f from imin to imax
{imax}	Iterate imax times without incrementation of a variable
{i, imax}	Increment i from 1 to imax by steps of 1
{i, imin, imax}	Increment i form imin to imax by steps of 1
{i, imin, imax, di}	Increment i from imin to imax by steps of di

4.1.3.14 Equations

4.1.3.14.1 Preliminaries

x = y	Assigns the value of y to x
x == y	Tests for x = y
x != y	Unequal
x > y	Greater than
x ≥ y	Greater than or equal to
x < y	Less than
x <= y	Less than or equal to
x == y == z	All equal
x != y != z	All unequal
!p	not
p && q && ...	and
p || q || ...	or
If[p, then, else]	Execute *then* if p is true, *else* otherwise

4.1.3.14.2 Solution

Solve[lhs == rhs, x]	Solve equation in terms of x
x /. expr	Use expr to get values of x
expr1 /. expr2	Use expr2 to get values for expr1
Solve[{lhs1 == rhs1, lhs2 == rhs2, ...}, {x, y, ...}]	Solve simultaneous set of equations for x,y,...

4.1.3.14.3 Differential Equations

DSolve[eqns, y[x], x]	Solve differential equation for y[x], taking x as the independent variable
DSolve[eqns, y, x]	Give sol'n for y in pure functional form

4.1.3.14.4 Numerical Values

N[expr]	Numerical value of expr
NIntegrate[f. {x, xmin, xmax}]	Numerical approximation of integral
NSolve[lhs == rhs, x]	Numerical approximation of the solution to the equation
NDSolve[eqns, y, {x, xmin, xmax}]	Solve numerically for y with independent variable x

4.1.3.15 Functions

f[x_]	Define the function f
?f	Show definition of f
Clear[f]	Clear definition of f

4.1.3.16 Vectors & Matrices

{a, b, c}	Vector (a,b,c)
{{a,b},{c,d}}	Matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$
Array[a, n]	Build a length-n vector of the form {a[1],...
Range[n]	Create a list {1, 2, 3, ..., n}
Length[list]	Give number of elements in list
IdentityMatrix[n]	Generate n x n identity matrix
MatrixForm[list]	Display list in matrix form
a . b	Matrix product
c m	Multiply a matrix by a scalar c
Inverse[m]	Matrix inverse
MatrixPower[m, n]	nth power of a matrix
Det[m]	Determinant
Transpose[m]	Transpose
Eigenvalues[m]	Eigenvalues
Eigenvectors[m]	Eigenvectors
Eigenvalues[N[m]]	and Numerical eigenvalues
Eigenvectors[N[m]]	Numerical eigenvectors

4.1.3.17 Graphics

4.1.3.17.1 Preliminaries

Plot[f, {x, xmin, xmax}]	Plot f in terms of x from xmin to xmax
Plot[Evaluate[f], {x, xmin, xmax}	First evaluate f then plot it

4.1.3.17.2 Plot Options

AspectRatio	Height-to-width ratio for plot
Axes	Whether to include axes (True or False)
AxesLabel	Labels to be put on axes {xlabel, ylabel} or None
AxesOrigin	Point at which axes cross
DefaultFont	The default font for text in the plot
DisplayFunction	How to display graphics
Frame	Whether to draw frame around plot
FrameLabel	Labels to be put around the frame
FrameTicks	What tick marks to draw if frame is included
GridLines	What grid lines to include
PlotLabel	An expression to be printed as a label for the plot
PlotRange	Plot range of coordintes to include in the plot
Ticks	What tick marks to draw if there is an axes
Automatic	Use internal algorithms
None	Do not include this
All	Include everything
True	Do this
False	Do not do this
PlotPoints	Minimum number of points at which to sample the function
MaxBend	Maximum kink angle between successive segments of a curve
PlotDivision	Maximum factor by which to subdivide in sampling the function
Show[plot]	Redraw a plot
Show[plot, option-¿value]	Redraw plot with changed options
Show[plot1, plot2, ...]	Combine several plots

4.1.3.17.3 3 Dimensional Plots

Plot3D[f, {x, xmin, xmax}, {y, ymin, ymax}]	Make 3D plot of f as a function of x and y

4.1.3.17.4 Parametric Plots [35] First you must load the ParametricPlot3D package:

```
In[1]:=<<Graphics'ParametricPlot3D'
```

ParametricPlot[{fx, fy}, {t, tmin, tmax}]	Make a parametric plot
ParametricPlot[{fx, fy}, {gx, gy}, {t, tmin, tmax}]	Parametric plot of several curves simultaneously

4.1.3.17.5 File Manipulation

¡¡name	Load an ASCII file containing *Mathematica* commands
expr >> name	Output expr to an ASCII file
expr >>> name	Append expr to an ASCII file
!!name	Display contents of an ASCII file
Save[”name”, f, g, ...]	Save definitions for variables or functions in a file

4.1.3.17.6 Generating C, Fortran & TEXFiles

CForm[expr]	Write expr as C code
FortranForm[expr]	Write expr as Fortran code
TexForm[expr]	Write expr in TEXform

4.1.4 Programming in Mathematica

36 *Programming* in Mathematica is the process of writing functions, or packages, that can subsequently be executed by simply typing in the name of the package. Writing programs with Mathematica allows you to develop specialized functions in your field of interest that can then be run from the *Mathematica* environment.

4.1.4.1 Building a Package

37 The goal of writing a package is to make the new function behave as much like a Mathematica function as possible. This includes being able to type `?FunctionName` for documentation on how to use the new package `FunctionName`. Also the output from the package should not be dependant on any previous calculations you have done during your Mathematica session.

38 Let's look at a few examples:

```
(*This function returns the first n powers of x*)
PowerSum[x_,n_] :=
    Block[{i},
        Sum[x^i,{i,1,n}]
    ]
```

39 This would then be saved in a file with an extension *.m*, such as *Example.m*

1. PowerSum[ ] is the name of the function were are creating.
2. All the variables local to PowerSum[ ] are declared in a **Block**[ ] statement, this isolates them from any values they might hold globally.
3. For this example, you must make sure that you don't pass in variables that are used locally inside the function.

For example:

```
In[1]:= <<Example.m
In[2]:= PowerSum[x,5]
Out[2]= x + x^2 + x^3 + x^4 + x^5
```

but if we were to call PowerSum with the following:

```
In[3]:= PowerSum[i,5]
Out[3]= 3413
```

notice that we get a number for our answer and not an expression. This is because the variable i, which we passed in, was captured by the variable (also called i) in the range of the summation. The following example shows how to avoid such a problem.

```
PowerSum::usage = "PowerSum[x,n] returns the sum of first n powers of x."
Begin["Private`"]
PowerSum[x_,n_] :=
    Block[{i},
        Sum[x^i,{i,1,n}]
    ]
End[]
```

40 Note we have done two things:

1. The `usage` statement defines a help message to be printed out if the user types `?PowerSum`.
2. The local variable i is now created in the *context* `Private`` which is not searched when you type in a variable name later on.

41 For example, with the revised `Example.m` you could type in the following:

```
In[1]:= <<Example.m
In[2]:= PowerSum[i,5]
Out[2]= i + i^2 + i^3 + i^4 + i^5
```

we now get the expression and not a numerical value.

4.1.4.2 A Complete Example

42 Here is an example of a package that will plot a random walk of length n. It will start at the origin and then randomly choose a direction to follow for a line of length 1.

```
BeginPackage["RandomWalk`"]
RandomWalk::usage = "RandomWalk[n] plots a random walk of length n."
Begin["`Private`"]
RandomWalk[n_Integer] :=
    Block[{loc = {0.0,0.0}, dir, points = Table[0,{n+1}], range = N[{0, 2Pi}]},
        points[[1]]=loc;
        Do[
            dir = Random[Real,range];
            loc += {Cos[dir],Sin[dir]};
            points[[i]] = loc,
```

```
          {i,2,n+1}];
        Show[ Graphics[{Point[{0,0}], Line[points]}],
            Frame->True, AspectRatio->Automatic]
    ]
End[]
EndPackage[]
```

The package is started by `BeginPackage["RandomWalk‘"]` and ended by `EndPackage[]`. Once a package is complete these statements need to be inserted, but during development of the package it is suggested that they be left out. If they are included in the development stage they may interfere with the debugging process.

Next is the `usage` statement followed by a short description of the package. This is the message that will be displayed when the user requests help for this command.

The `Begin["‘Private‘"]` and `End[]` statements are to keep values of variables used outside the package from interfering with the variables used locally by the package.

The `RandomWalk[n_Integer] :=` statement assigns the name `RandomWalk` and the parameter `n` (defined as an integer) to the function.

All the variable declarations and operations are done within the `Block[........]` statement.

The first part of `Block[....]` declares some variables:

1. `loc = {0.0,0.0}` defines `loc` as a collection of two values, these will be used as x and y coordinates later on.
2. `points` is declared as a `Table`, see Mathematica reference for more details if you are not familiar with this command. Basically, `points` will be a list going from 0 to n+1.
3. `range` is exactly that, the `N` is to give an approximate value of Pi.

The first point of `points` is then assigned the origin. The semicolon at the end of the line is to separate statements.

The next construct is a `Do[...]`, this enable the user to iterate. The format is `Do[`*statement , interator*`]`. This Do loop has more than one statement, so they are separated by semicolons. The iterator is `{i,2,n+1}`.

The following occur within the Do loop:

1. dir is chosen randomly from the range indicated in `range` and returned as a real.
2. loc is then assigned the cosine and sine of dir
3. `points` is assigned the value of `loc`.

The next statements; `Show[...]`, and `Graphics[..]`, are straight forward Mathematica commands. For more information on these commands see the Mathematica reference text.

4.1.5 List of All Mathematica Functions

43 Table 4.1 contains all *Mathematica* functions, with the most commonly used ones in boldfaced.

4.2 Assignment

4.2.1 Practice

Start by repeating all the examples in this handout.

4.2.2 Problems

Write a *Mathematica* Notebook to execute the following:

1. Solve $x^2 + 2x + 1 = 0$
2. Integrate $\int_0^1 \int_0^{\sqrt{x}} ye^{x^2} dydx$
3. Solve the following differential equation $y'(x) = y(x)$
4. Plot the function $x \sin x$ for x in the range $[0, 6\pi]$, and
   (a) Change the `AspectRatio` so that the width of the plot is twice its height.
   (b) Give the plot the label ``` ``x Sin[x]'' ```.
   (c) Label the x axis "x", and the y axis "y".
5. Plotsin$(\pi \sin x + y)$ for $-3 < x < 3$ and $-3 < y < 3$ using 33 grid points, no display of the axes.
6. Generate a contour plot of the previous equation.
7. Write a function which takes a pair x, y and returns $\sqrt{x^2 + y^2}$.

4.3 References

1. Wolfram, S. "*Mathematica: A System for Doing Mathematics by Computer* second edition, Addison-Wesley, 1991.
2. Maeder, R., *Programming in Mathematica*, Addison Wesley, 1991
3. Blachman, N., *Mathematica a Practical Approach*, Prentice Hall, 1991.
4. Gray, T., and Glynn J., *The Beginner's Guide to Mathematica*, Addison-Wesley, 1992.

Above	CoefficientList	EllipticK	Headers	†**LightSources**	OneIdentity	Re	StringMatchQ
†**Ab**	†**Collect**	EllipticLog	HermiteH	†**Lighting**	OpenAppend	Read	StringSkeleton
Accumulate	ColonForm	Encode	†**HiddenSurf***	Limit	OpenRead	ReadList	SubValue
Accuracy	ColumnForm	End	Hide	Line	OpenTemporary	ReadProtected	Subscript
AccuracyGoal	Complement	EndAdd	Hold	LineBreak	OpenWrite	Real	Subscripted
†**AddTo**	†**Complex**	EndOfFile	HoldAll	LinearProgra*	Operate	Recall	Subtract
AiryAi	ComplexInfini*	EndPackage	HoldFirst	LinearSolve	Optional	Rectangle	†**SubtractFr***
AlgebraicRules	Compose	EndProcess	HoldForm	List	Options	†**Reduce**	†**Sum**
AlgebraicRule*	ComposeSeries	EngineeringFo*	HoldRest	ListContourPl*	Or	Release	Superscript
Alias	CompoundExp*	Environment	HorizontalForm	ListDensityPl*	Order	Remove	SurfaceGraphics
All	Condition	Epilog	Hypergeometri*	ListPlot	OrderedQ	Removed	Switch
†**AmbientLig***	Conjugate	Equal	Hypergeometri*	ListPlot3D	Orderless	RenderAll	Symbol
And	Constant	EquatedTo	Hypergeometri*	Listable	Out	R	Tab
Apart	Constants	Erf	Hypergeometri*	Literal	Outer	RepeatedNull	Table
Append	ConstrainedMax	EulerE	Hypergeometri*	Locked	OutputForm	RepeatedString	TableForm
AppendTo	ConstrainedMin	EulerGamma	Hypergeometri*	†**Log**	Overflow	Replace	TagSet
Apply	Construct	EulerPhi	Hypergeometri*	LogIntegral	OwnValue	ReplaceAll	TagSetDelayed
ArcCos	Context	EvenQ	I	LogicalExpand	†**PSPrint**	ReplaceRepeat*	TagUnset
ArcCosh	ContextToFile*	†**Exit**	Identity	MachineID	PageHeight	ResetMedium	Take
ArcCot	Continuation	†**Exp**	IdentityMatrix	MachineName	PageWidth	Residue	Tan
ArcCoth	Continue	ExpIntegralE	If	MainSolve	†**Parametri***	Rest	T
ArcCsc	†**ContourGr***	ExpIntegralEi	Im	MakeRules	Part	Resultant	†**TeXForm**
ArcCsch	ContourLevels	†**Expand**	Implies	Map	Partition	Resultant2	TensorRank
ArcSec	†**ContourPlot**	ExpandAll	In	MapAll	PartitionsP	Return	Terms
ArcSech	ContourSpacing	ExpandDenom*	Increment	MapAt	PartitionsQ	Reverse	Text
ArcSin	Cos	ExpandNumer*	Indent	MatchBox	Pattern	Right	TextFont
ArcSinh	CosIntegral	Exponent	Indeterminate	MatchQ	PatternTest	†**Roots**	†**TextForm**
ArcTan	Cosh	ExponentStep	Inequality	MatrixExp	Permutations	RotateLeft	TextLeading
ArcTanh	Cot	Expression	†**Infinity**	†**MatrixForm**	†**Pi**	RotateRight	TextRendering
Arg	Coth	ExtendedGCD	Infix	MatrixPower	Plain	Round	Thickness
ArithmeticGeo*	Count	ExtraFactors	Information	MatrixQ	†**Plot**	RowReduce	Thread
Array	Csc	ExtraTerms	Inner	Max	†**Plot3D**	Rule	Through
AspectRatio	Csch	FaceForm	Input	MaxBend	Plot3Matrix	RuleDelayed	Throw
AtomQ	Cubics	Factor	InputForm	MaxIterations	PlotColor	RuleForm	Ticks
Attributes	Cyclotomic	FactorComplete	InputString	MaxMemoryUsed	PlotDivision	RuleTable	TimeConstrained
Automatic	†**D**	FactorInteger	Insert	MaxRecursion	PlotJoined	Run	Times
Auxiliary	DSolve	FactorList	Integer	Medium	PlotLabel	RunThrough	TimesBy
Axes	DampingFactor	FactorSquareF*	IntegerQ	MemberQ	PlotPoints	SameQ	Timing
AxesEdge	Dashing	FactorSquareF*	†**Integrate**	MemoryConstra*	†**PlotRange**	Save	ToASCII
AxesLabel	†**Debug**	FactorTerms	InterpolatingPo	MemoryInUse	PlotStyle	Scaled	ToExpression
AxesStyle	†**Decompose**	FactorTermsLi*	Interrupt	Mesh	†**Plus**	Scan	ToRules
Background	Decrement	Factorial	Intersection	MeshRange	Pochhammer	†**ScientificFo***	ToString
BaseForm	Default	Factorial2	†**Inverse**	MeshStyle	Point	Sec	†**Together**
Begin	DefaultColor	Factors	InverseFourier	Message	PointSize	Sech	Tolerance
BeginPackage	DefaultValue	Fail	InverseFuncti*	MessageName	Points	Second	TooBig
Below	Definition	False	InverseFuncti*	Messages	PolyGamma	SeedRandom	Top
BernoulliB	Degree	FindMinimum	InverseJacobi*	Method	PolyLog	Select	TotalHeight
BesselI	Delimiters	†**FindRoot**	InverseJacobi*	Min	Polygon	Sequence	TotalWidth
BesselJ	Denominator	First	InverseJacobi*	MinRecursion	PolynomialG*	SequenceForm	Transpose
BesselK	DensityGraphi*	Fit	InverseJacobi*	Minors	PolynomialQ	SequenceLimit	TreeForm
BesselY	DensityPlot	FixedPoint	InverseJacobi*	Minus	PolynomialQu*	Series	†**TrigExpand**
Beta	Depth	Flat	InverseJacobi*	Mod	PolynomialQu*	SeriesCoeffic*	True
BetaRegulariz*	†**Derivative**	Flatten	InverseJacobi*	Mode	PolynomialRe*	SeriesData	TrueQ
Binomial	Det	Floor	InverseJacobi*	†**Modular**	Position	Set	UnAlias
Blank	DiagonalMatrix	†**Font**	InverseJacobi*	Modulus	Positive	SetAccuracy	Underflow
BlankForm	DigitBlock	FontForm	InverseJacobi*	MoebiusMu	†**PostScript**	SetAttributes	Unequal
BlankNullSequ*	Digits	For	InverseSeries	Multinomial	Postfix	SetDelayed	Union
BlankSequence	Dimensions	Format	JacobiAmplitu*	Multiplicity	Power	SetOptions	Unique
Block	DirectedInfin*	FormatType	JacobiCD	N	PowerMod	SetPrecision	Unprotect
Bottom	Disk	†**FortranForm**	JacobiCN	NBernoulliB	PreDecrement	Shading	UnsameQ
BoxRatios	Dispatch	Fourier	JacobiCS	†**NIntegrate**	PreIncrement	Share	Unset
BoxStyle	Display	Framed	JacobiDC	NProduct	PrecedenceFo*	Sh	UpSet
Boxed	DisplayFuncti*	FreeQ	JacobiDN	†**NRoots**	Precision	Show	UpSetDelayed
Break	Distribute	FromASCII	JacobiDS	NSum	Prefix	Sign	UpValue
Byte	†**Divide**	FullDefinition	JacobiNC	NameQ	Prepend	Signature	Update
ByteCount	DivideBy	FullForm	JacobiND	Names	PrependTo	†**Simplify**	Using
C	DivisorSigma	Function	JacobiNS	Needs	Prime	†**Sin**	ValueForm
CForm	Divisors	GCD	JacobiP	Negative	PrimeQ	SinIntegral	ValueList
CallProcess	Do	Gamma	JacobiSC	Nest	Print	SingularValues	ValueQ
Cancel	†**Dot**	GammaRegul*	JacobiSD	NestList	PrintForm	SingularityDe*	Variables
Cases	DoubleExpone*	GaussianQu*	JacobiSN	NonAssociati*	PrintValue	Sinh	Vector
Catalan	DoublyInfinite	†**Gegenbaue***	JacobiSymbol	NonCommutati*	ProbablePrimeQ	Skeleton	VerticalForm
Catch	DownValue	General	Jacobian	NonConstants	Product	Slot	ViewPoint
Ceiling	Drop	Generic	Join	NonNegative	Prolog	SlotSequence	WeierstrassP
CellArray	DropFrom	Get	LCM	None	PromptForm	†**Solve**	WeierstrassP*
Center	Dt	GoldenRatio	Label	Normal	Protect	SolveAlways	Which
Character	Dump	Goto	LaguerreL	Not	Protected	Sort	While
Characters	†**E**	Gradient	LaserPrint	Null	PseudoInverse	Space	WorkingPrecis*
ChebyshevT	EdgeForm	Graphics	Last	NullSpace	Put	SpaceForm	Write
ChebyshevU	Edit	Graphics3D	LatticeReduce	NumValue	PutAppend	SphericalHarm*	WriteString
Check	EditDef	GraphicsFont	LeafCount	Number	Quartics	Splice	WynnDegree
Chop	EditIn	GraphicsLeading	Left	NumberForm	Quit	†**Sqrt**	Xor
Circle	Eigensystem	GrayLevel	LegendreP	NumberPoint	Quotient	StartProcess	ZeroTest
Clear	†**Eigenvalues**	Greater	LegendreQ	NumberQ	RGBColor	StirlingS1	Zeta
ClearAll	Eigenvectors	GreaterEqual	LegendreType	NumberSepar*	Random	StirlingS2	
ClearAttribut*	ElapsedTime	GroebnerBasis	Length	Numerator	Range	String	
ClipFill	†**Eliminate**	Hash	LerchPhi	OddQ	Rational	StringBreak	
Close	EllipticE	HashTable	Less	Off	Rationalize	StringForm	
CodeAddress	EllipticExp	Head	LessEqual	On	Raw	StringJoin	
Coefficient	EllipticF	HeadCompose	Level	RawMedium	StringLength		

Table 4.1: *Mathematica* Functions

Chapter 5

SAMPLES of MATLAB PROGRAMS

This chapter contains the description and listing of a number of simple (and not so simple) MATLAB codes.

The source codes can be freely copied from
`/pub/Structures/cven4837`

5.1 arches

5.1.1 Description

SUBDIRECTORY: arches

SYNOPSIS: axial, shear, and normal force plots for an arch (Structures)

DESCRIPTION: arches.m plots the Bending Moment, Shear Force, and Axial Force for a three pin semi circular arch under uniform load.

CONTENTS;

`arches.m` main program

Written by: Brian Rose

Determine the reactions of the three-hinged arch shown in Fig. 5.1 **Solution:**

Four unknowns, three equations of equilibrium, one equation of condition ⇒ statically determinate.

$$(+\circlearrowright)\ \Sigma M_z^C = 0; \quad (R_{Ay})(140) + (80)(3.75) - (30)(80) - (20)(40) + R_{Ax}(26.25) = 0$$

$$\Rightarrow 140R_{Ay} + 26.25R_{Ax} = 2.900$$

$$(+\rightarrow)\ \Sigma F_x = 0; \qquad 80 - R_{Ax} - R_{Cx} = 0$$

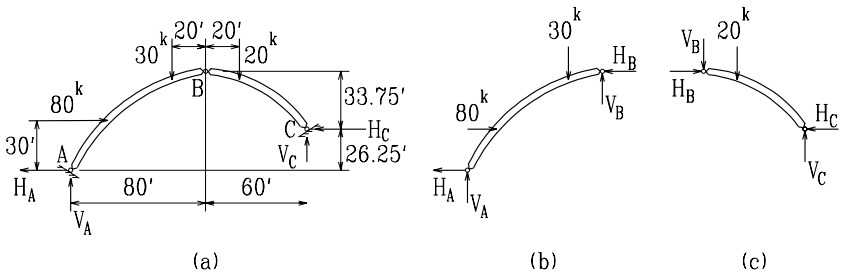


Figure 5.1:

$$
\begin{aligned}
(+\uparrow)\,\Sigma F_y = 0; \qquad & R_{Ay} + R_{Cy} - 30 - 20 = 0 \\
(+\circlearrowleft)\,\Sigma M_z^B = 0; \qquad & (R_{ax})(60) - (80)(30) - (30)(20) + (R_{Ay})(80) = 0 \\
& \Rightarrow 80R_{Ay} + 60R_{Ax} = 3,000 \qquad (5.1\text{-a})
\end{aligned}
$$

Solving those four equations simultaneously we have:

$$
\begin{bmatrix} 140 & 26.25 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 80 & 60 & 0 & 0 \end{bmatrix}
\begin{Bmatrix} R_{Ay} \\ R_{Ax} \\ R_{Cy} \\ R_{Cx} \end{Bmatrix}
=
\begin{Bmatrix} 2,900 \\ 80 \\ 50 \\ 3,000 \end{Bmatrix}
\Rightarrow
\begin{Bmatrix} R_{Ay} \\ R_{Ax} \\ R_{Cy} \\ R_{Cx} \end{Bmatrix}
=
\begin{Bmatrix} 15.1 \text{ k} \\ 29.8 \text{ k} \\ 34.9 \text{ k} \\ 50.2 \text{ k} \end{Bmatrix}
\qquad (5.2)
$$

We can check our results by considering the summation with respect to b from the right:

$$
(+\circlearrowleft)\,\Sigma M_z^B = 0; -(20)(20) - (50.2)(33.75) + (34.9)(60) = 0\surd \qquad (5.3)
$$

Determine the reactions of the three hinged statically determined semi-circular arch under its own dead weight w (per unit arc length s, where $ds = rd\theta$). 5.2

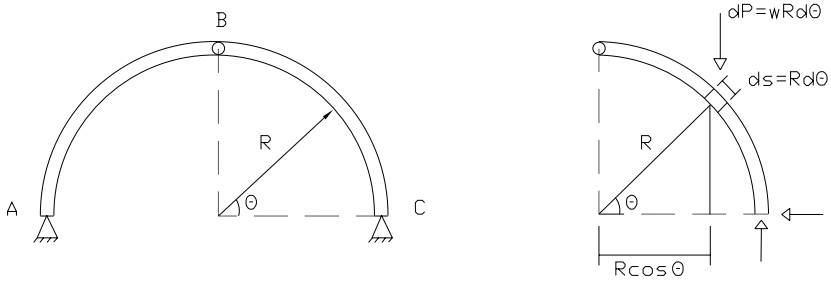


Figure 5.2:

The reactions can be determined by **integrating** the load over the entire structure

$$(+\circlearrowright)\,\Sigma M_A = 0; \qquad -(V_C)(2R) + \int_{\theta=0}^{\theta=\pi} \underbrace{wRd\theta}_{dP}\ \underbrace{R(1+\cos\theta)}_{\text{moment arm}} = 0$$

$$\Rightarrow V_C = \tfrac{wR}{2}\int_{\theta=0}^{\theta=\pi}(1+\cos\theta)d\theta$$

$$= \tfrac{wR}{2}[\theta - \sin\theta]\ |_{\theta=0}^{\theta=\pi} = \tfrac{wR}{2}[(\pi - \sin\pi) - (0 - \sin 0)]$$

$$= \tfrac{\pi}{2}wR \tag{5.4-a}$$

Next we determine the horizontal reaction

$$(+\circlearrowleft)\,\Sigma M_B = 0; \quad -(H_C)(R) + (V_c)(R) - \int_{\theta=0}^{\theta=\frac{\pi}{2}} \underbrace{wRd\theta}_{dP}\ \underbrace{R\cos\theta}_{\text{moment arm}} = 0$$

$$\Rightarrow H_C = \tfrac{\pi}{2}wR - \tfrac{wR}{2}\int_{\theta=0}^{\theta=\frac{\pi}{2}}\cos\theta d\theta$$

$$= \tfrac{\pi}{2}wR - wR[\sin\theta]\ |_{\theta=0}^{\theta=\frac{\pi}{2}} = \tfrac{\pi}{2}wR - wR(\tfrac{\pi}{2} - 0)$$

$$= (\tfrac{\pi}{2} - 1)\,wR \tag{5.5-a}$$

By symmetry the reactions at A are equal to those at C

Draw the shear and moment diagram for the three hinged statically determined semi-circular arch under its own dead weight w, Fig. 5.3. **Solution:**

Reactions: Those were determined earlier, Example **??**. For the sake of clarity, we repeat their derivation:

1. Starting with C_Y

$$(+\circlearrowright)\,\Sigma M_A = 0; \qquad -(C_Y)(2R) + \int_{\theta=0}^{\theta=\pi} \underbrace{wRd\theta}_{dP}\ \underbrace{R(1+\cos\theta)}_{\text{moment arm}} = 0$$

$$\Rightarrow C_Y = \tfrac{wR}{2}\int_{\theta=0}^{\theta=\pi}(1+\cos\theta)d\theta$$

$$= \tfrac{wR}{2}[\theta - \sin\theta]\ |_{\theta=0}^{\theta=\pi} = \tfrac{wR}{2}[(\pi - \sin\pi) - (0 - \sin 0)]$$

$$= \tfrac{\pi}{2}wR \tag{5.6-a}$$

2. Next we determine the horizontal reaction

$$(+\circlearrowleft)\,\Sigma M_B = 0; \quad -(C_x)(R) + (C_y)(R) - \int_{\theta=0}^{\theta=\frac{\pi}{2}} \underbrace{wRd\theta}_{dP}\ \underbrace{R\cos\theta}_{\text{moment arm}} = 0$$

$$\Rightarrow C_x = \tfrac{\pi}{2}wR - \tfrac{wR}{2}\int_{\theta=0}^{\theta=\frac{\pi}{2}}\cos\theta d\theta$$

$$= \tfrac{\pi}{2}wR - wR(\sin\theta]\ |_{\theta=0}^{\theta=\frac{\pi}{2}} = \tfrac{\pi}{2}wR - wR(\tfrac{\pi}{2} - 0)$$

$$= (\tfrac{\pi}{2} - 1)\,wR \tag{5.7-a}$$

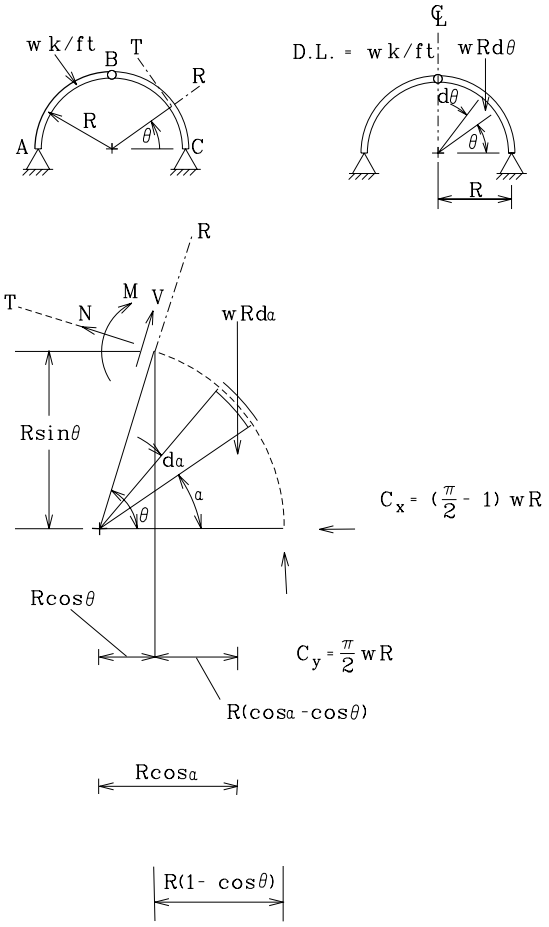


Figure 5.3:

3. By symmetry the reactions at A are equal to those at C

Shear Forces: Considering the free body diagram of the arch, and summing the forces in the radial direction ($\Sigma F_R = 0$):

$$\underbrace{-(\frac{\pi}{2}-1)wR\cos\theta}_{C_x} + \underbrace{\frac{\pi}{2}wR\sin\theta}_{C_y} - \int_{\alpha=0}^{\theta} wRd\alpha\sin\theta + V = 0 \tag{5.8}$$

$$\Rightarrow \quad V = wR\left[(\frac{\pi}{2}-1)\cos\theta + (\theta-\frac{\pi}{2})\sin\theta\right] \tag{5.9}$$

Axial Forces: Similarly, if we consider the summation of forces in the axial direction ($\Sigma F_N = 0$):

$$(\tfrac{\pi}{2}-1)wR\sin\theta + \tfrac{\pi}{2}wR\cos\theta - \int_{\alpha=0}^{\theta} wRd\alpha\cos\theta + N = 0 \tag{5.10}$$

$$N = wR\left[(\theta-\tfrac{\pi}{2})\cos\theta - (\tfrac{\pi}{2}-1)\sin\theta\right] \tag{5.11}$$

Moment: Now we can consider the third equation of equilibrium ($\Sigma M_\theta = 0$):

$$(\tfrac{\pi}{2}-1)wR\cdot R\sin\theta - \tfrac{\pi}{2}wR^2(1-\cos\theta) + \int_{\alpha=0}^{\theta} wRd\alpha\cdot R(\cos\alpha-\cos\theta) + M = 0 \tag{5.12}$$

$$M = wR^2\left[\tfrac{\pi}{2}(1-\sin\theta) + (\theta-\tfrac{\pi}{2})\cos\theta\right] \tag{5.13}$$

We seek to determine the vertical deflection of the crown of the three hinged statically determined semi-circular arch, Fig. 5.4 under its own dead weight w, Fig. **??**.

1. We first seek to determine the analytical expression of the moment diagram. From statics, it can be shown that the vertical and horizontal reactions are $R_v = \frac{\pi}{2}wR$ and $R_h = (\frac{\pi}{2}-1)wR$.

2. Next considering the free body diagram of the arch, and summing the forces in the radial direction ($\Sigma F_R = 0$):

$$-(\tfrac{\pi}{2}-1)wR\cos\theta + \tfrac{\pi}{2}wR\sin\theta - \int_{\alpha=0}^{\theta} wRd\alpha\sin\theta + V = 0 \tag{5.14-a}$$

$$V = wR\left[(\tfrac{\pi}{2}-1)\cos\theta + (\theta-\tfrac{\pi}{2})\sin\theta\right] \tag{5.14-b}$$

3. Similarly, if we consider the summation of forces in the axial direction ($\Sigma F_T = 0$):

$$(\tfrac{\pi}{2}-1)wR\sin\theta + \tfrac{\pi}{2}wR\cos\theta - \int_{\alpha=0}^{\theta} wRd\alpha\cos\theta + N = 0 \tag{5.15-a}$$

$$N = wR\left[(\theta-\tfrac{\pi}{2})\cos\theta - (\tfrac{\pi}{2}-1)\sin\theta\right] \tag{5.15-b}$$

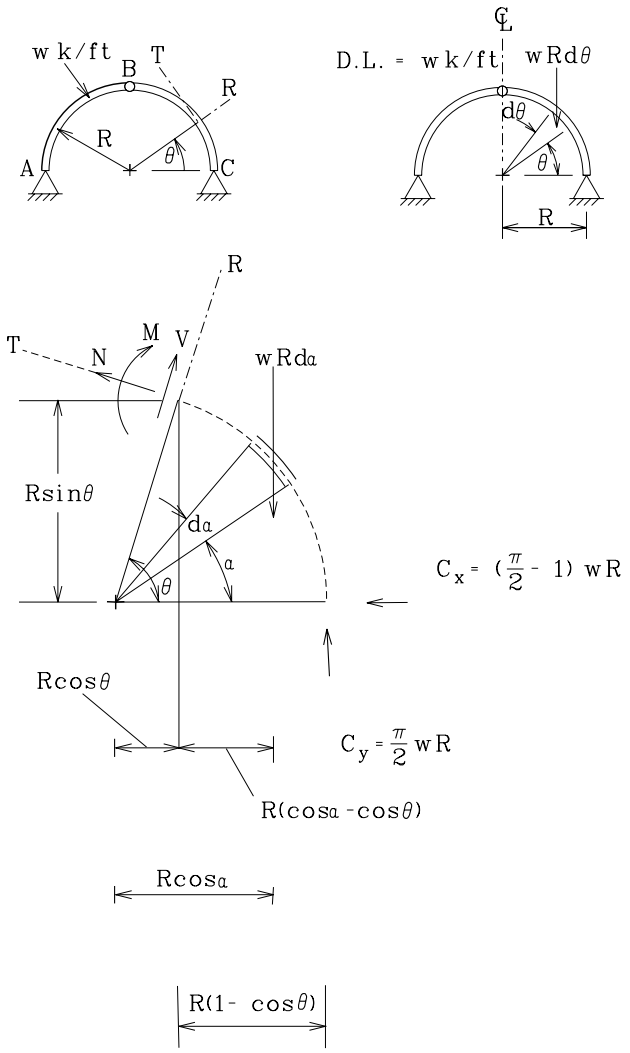


Figure 5.4:

4. Now we can consider the third equation of equilibrium ($\Sigma M_\theta = 0$):

$$\begin{aligned}&(\tfrac{\pi}{2}-1)wR\cdot R\sin\theta - \tfrac{\pi}{2}wR^2(1-\cos\theta) + \\ &\int_{\alpha=0}^{\theta} wRd\alpha\cdot R(\cos\alpha-\cos\theta) + M = 0 \qquad (5.16\text{-a})\\ &M = wR^2\left[\tfrac{\pi}{2}(1-\sin\theta) + (\theta-\tfrac{\pi}{2})\cos\theta\right] \qquad (5.16\text{-b})\end{aligned}$$

5. The real curvature ϕ is obtained by dividing the moment by EI

$$\phi = \frac{M}{EI}\frac{wR^2}{EI}\left[\frac{\pi}{2}(1-\sin\theta) + (\theta-\frac{\pi}{2})\cos\theta\right] \qquad (5.17)$$

6. The virtual force $\delta\overline{P}$ will be a unit vertical point in the direction of the desired deflection, causing a virtual internal moment

$$\delta\overline{M} = \frac{R}{2}\left[1-\cos\theta-\sin\theta\right] \qquad 0\le\theta\le\frac{\pi}{2} \qquad (5.18)$$

7. Hence, application of the virtual work equation yields:

$$\begin{aligned}\underbrace{1}_{\delta\overline{P}}\cdot\Delta &= 2\int_{\theta=0}^{\frac{\pi}{2}} \underbrace{\frac{wR^2}{EI}\left[\frac{\pi}{2}(1-\sin\theta)+(\theta-\frac{\pi}{2})\cos\theta\right]}_{\phi}\cdot\underbrace{\frac{R}{2}\cdot[1-\cos\theta-\sin\theta]}_{\delta\overline{M}}\underbrace{Rd\theta}_{dx}\\ &= \frac{wR^4}{16EI}\left[7\pi^2-18\pi-12\right]\\ &= \boxed{.0337\tfrac{wR^4}{EI}} \qquad (5.19\text{-a})\end{aligned}$$

5.1.2 Listing

5.1.2.1 `arches.m`

Equations used for calculation of shear force, axial force, and bending moment are as follows:

$$\begin{aligned}V &= \omega R[(\frac{\pi}{2}-1)\cos\theta + (\theta-\frac{\pi}{2})\sin\theta]\\ N &= \omega R[(\theta-\frac{\pi}{2})\cos\theta - (\frac{\pi}{2}-1)\sin\theta]\\ M &= \omega R^2[\frac{\pi}{2}(1-\sin\theta) + (\theta-\frac{\pi}{2})\cos\theta]\end{aligned}$$

arches.m file starts here:

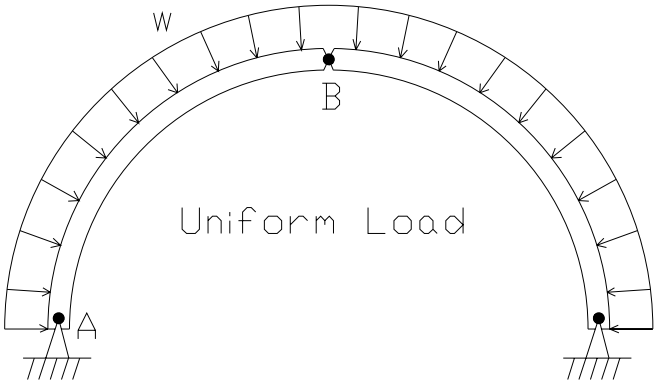


Figure 5.5: Simply Supported Arch with Uniform Load

```
%Arches is a script that plots the Bending Moment, Shear Force,
%and Axial Force for a three pin semi circular arch under uniform
%load

%clear the command screen
clc

%prompt the user for some information
R=input('Enter the radius of the arch: ');
w=input('Enter the load on the of the arch: ');

%create a vector of theta from 0 to 90 degrees
theta=(0:pi/100:pi/2);

%calculate V, N, M for a half the arch
V=w*R*((pi/2-1)*cos(theta)+(theta-pi/2).*sin(theta));
N=w*R*((theta-pi/2).*cos(theta)-(pi/2-1)*sin(theta));
M=w*R^2*(pi/2*(1-sin(theta))+(theta-pi/2).*cos(theta));

%since the structure is symmetric, get V,N,M values
%for theta from 90 to 180 degrees by flipping V,N,M
%notice the negative sign for Shear
%also note that if we just flip V,N,M we'll
%repeat V(90 degrees),N(90 degrees), M(90 degrees)
%so we'll pull those values out
V=[V fliplr(-V(1:length(V)-1))];
N=[N fliplr(N(1:length(N)-1))];
M=[M fliplr(M(1:length(M)-1))];

%redefine theta so that it goes from 0 to 180 degrees
theta=(0:pi/100:pi);
```

```
%Plot the shear diagram
%The axislimit=...  commands are there so that I can
%tell Matlab to do a polar plot with a radial axis starting from
%-V to +V, instead of 0 to +V.
%The tickmarks=... command makes it so that the tick labels are nice round numbers
%Also note that we are not using the command polar, but instead we're using
%polarhg. Polarhg is a more flexible version of polar, but it doesn't come
%standard with Matlab
subplot(221)
axislimits=max(abs(V))*1.2;
axislimits=ceil(axislimits/10.^floor(log10(axislimits)))*10.^floor(log10(axislimits))*[-1 1];
tickmarks=(axislimits(1):2*10^floor(log10(axislimits(2))):axislimits(2));
polarhg([theta;theta],[V;0*V],'rlim',axislimits,'rtick',tickmarks);
title('Shear Force')

%plot the Axial force diagram
subplot(222)
axislimits=max(abs(N));
axislimits=ceil(axislimits/10.^floor(log10(axislimits)))*10.^floor(log10(axislimits))*[-1,1];
tickmarks=(axislimits(1):2*10^floor(log10(axislimits(2))):axislimits(2));
polarhg([theta;theta],[N;0*N],'rlim',axislimits,'rtick',tickmarks);
title('Axial Force')

%plot the Moment diagram
subplot(223)
axislimits=max(abs(M));
axislimits=ceil(axislimits/10.^floor(log10(axislimits)))*10.^floor(log10(axislimits))*[-1,1];
tickmarks=(axislimits(1):2*10^floor(log10(axislimits(2))):axislimits(2));
polarhg([theta;theta],[-M;0*M],'rlim',axislimits,'rtick',tickmarks);
title('Bending Moment')
```

5.2 beam1

5.2.1 Description

SUBDIRECTORY : beam1

SYNOPSIS: shear and bending moment diagram for a simple beam (Structures)

DESCRIPTION: BMdemo.m gets the bending moment and deflection for a simply supported beam subjected to either a point load or a distributed load.

CONTENTS:

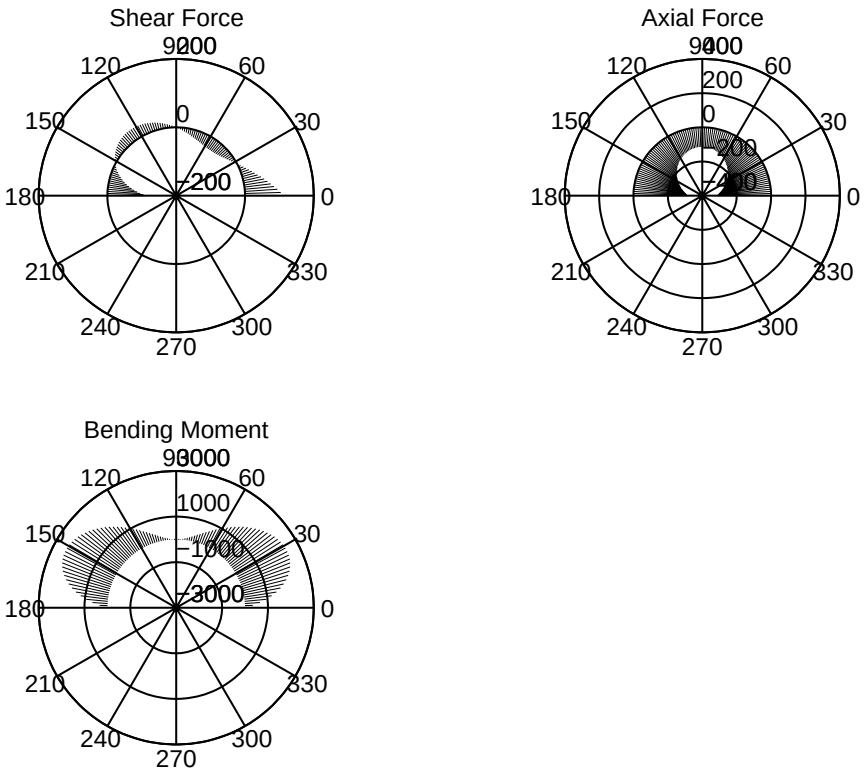


Figure 5.6: Sample output for *arches.m* using r = 100 and ω= 2

`BMdemo.m`	the main program
`M2deflection.m`	converts bending moments to deflections by integration
`P2M.m`	converts point loads to moment using a formula
`P2V.m`	converts point loads to shear using a formula
`w2M.m`	converts uniform loads to moment using a formula
`w2V.m`	converts uniform loads to shear using a formula

Written by: Brian Rose

5.2.2 Listing

5.2.2.1 `BMdemo.m`

Equations used for calculation of shear, bending moment, and deflection diagrams:

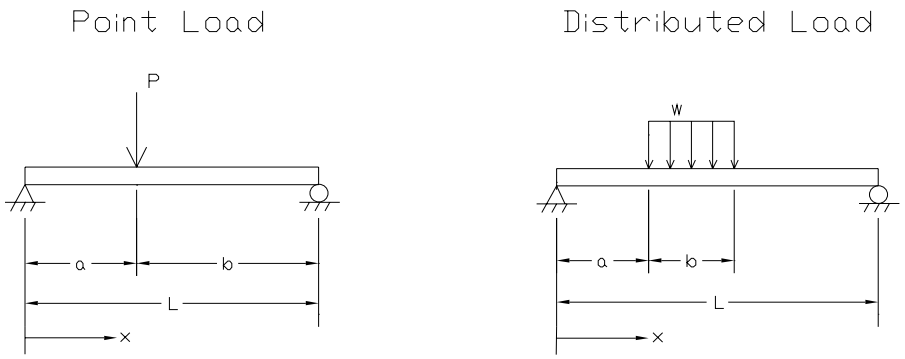


Figure 5.7: Simply Supported Beams with Various Loadings

- Pointload

$$R_1 = \frac{Pb}{L}$$

$$R_2 = \frac{Pa}{L}$$

$$M = R_1 x \qquad \text{for } (x < a)$$

$$= R_2(\text{L - x}) \qquad \text{for } (x \geq a)$$

$$V = R_1 \qquad \text{for } (x < a)$$

$$= -R_2, \qquad \text{for } (x \geq a)$$

$$v'' = \frac{M(i)}{EI} = \frac{v(i-1) - 2v(i) + (i+1)}{dx^2}$$

- Continuous Load

$$R1 = \frac{\omega b}{2L}(2a + b)$$

$$R2 = \frac{\omega b}{2L}(2c + b)$$

$$M = R_1 x \qquad \text{for } (x < a)$$

$$= R_1 x - \frac{\omega}{2}(x - a) \qquad \text{for } (a \le x \le a + b)$$

$$= R_2(\text{L - x}) \qquad \text{for } (x \ge a + b)$$

$$V = R_1 \qquad \text{for } (x < a)$$

$$= \frac{(x - a)(-R_2 - R_1)}{b} + R_1 \qquad \text{for } (a \le x \le a + b)$$

$$= -R2 \qquad \text{for } (x \ge a + b)$$

$$v'' = \frac{M(i)}{EI} = \frac{v(i-1) - 2v(i) + (i+1)}{dx^2}$$

BMdemo.m file starts here:

```
%BMdemo
%this script gets the bending moment and
%deflection for a simply supported beam subjected to either
%a point load or a distributed load

%prompt the user for the load type
choice=input('Analyze for point load (1) or a uniform load (2) ? ');

%clear the screen
clc

%do the point load case
if choice==1

  %prompt the user for some parameters
  P=input('Enter P: ');
  a=input('Enter a: ');
  L=input('Enter L: ');
  E=input('Enter E: ');
  I=input('Enter I: ');
  number_points=input('Enter the number of discretization points: ');

  %calculate the mesh size
```

```
  dx=L/(number_points-1);

  %get the BM
  M=P2M(P,a,L,dx);

  %get the shear
  V=P2V(P,a,L,dx);

  %get the deflection
  v=M2deflection(M,E,I,dx);

%do the distributed load case
elseif choice==2

  %prompt the user for some parameters
  w=input('Enter w: ');
  a=input('Enter a: ');
  b=input('Enter b: ');
  L=input('Enter L: ');
  E=input('Enter E: ');
  I=input('Enter I: ');
  number_points=input('Enter the number of discretization points: ');

  %calculate the mesh size
  dx=L/(number_points-1);

  %get the deflection
  M=w2M(w,a,b,L,dx);

  %get the Shear
  V=w2V(w,a,b,L,dx);

  %get the deflection
  v=M2deflection(M,E,I,dx);
end

%plot the BM and deflection

%make a vector for location along beam
x=(0:dx:L);

%plot the BM
subplot(311)
plot(x,M)
ylabel('Bending Moment')
grid

%plot the Shear
subplot (312)
plot(x,V)
ylabel('Shear')
grid
```

```
%plot the deflection
subplot(313)
plot(x,v)
ylabel('Displacement')
grid
```

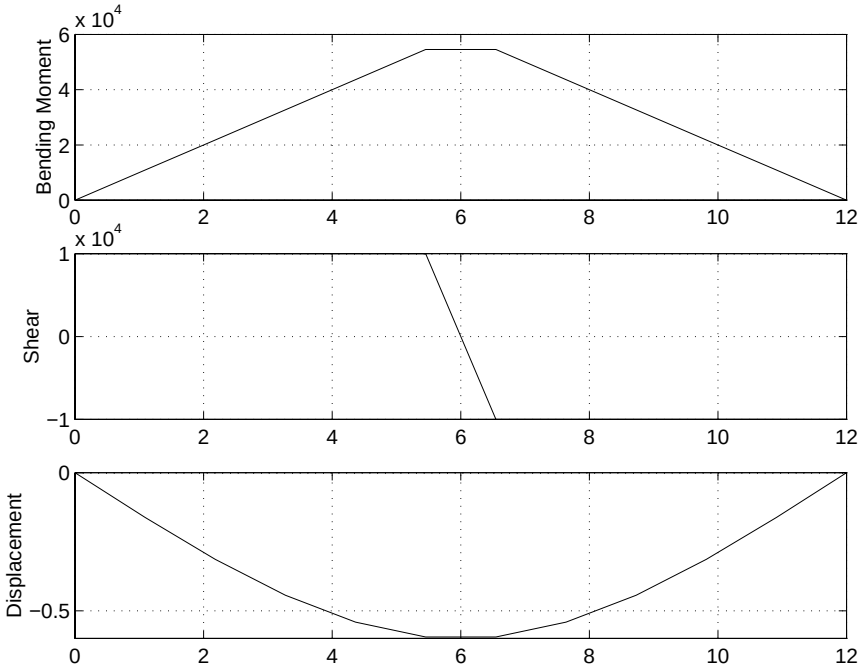


Figure 5.8: Sample output of *BMdemo.m* with a pointload using P = 20 kN, a = 6 m, L = 12 m, E = 200 GPa, I = 6e6 mm^4, and 12 discretization points.

5.2.2.2 M2deflection.m

```
function v=M2deflection(M,E,I,dx)
%M2DEFLECTION(M,E,I,dx) returns the deflection of a simply
%supported beam, given the descritized moment, M;
%E, Young's modulus; I, moment of inertia; and dx,
%the mesh size.
%
%This algorithm integrates the bending moment twice
%using the central difference method.
%The second derivative is defined as
%
```

```
%      v(i-1) - 2v(i) + (i+1)
%v''=  ----------------------
%               dx^2
%
%
%Using the central difference method we can w%The central difference method will take on the form Av=b.rite
%
%            v(i-1) - 2v(i) + (i+1)
%M(i)/EI  =  ----------------------
%                   dx^2
%
%If you write the above equation for every mesh point
%you get the following matrix equation.
%
%        ----    --    --         --    --
%        |  1 -2  1  |    | v(-1) |        | M(0) |
%        |     1 -2  1    |    | v(0)  |        | M(1) |
%        |        1 -2  1  |    | v(1)  |        | M(2) |
%        |              .  |    | v(2)  |        |  .   |
%1/dx^2  |                 .  |    |  .    | = 1/EI |  .   |
%        |                    .    |    |  .    |        |  .   |
%        |                   1 -2  1    |    |  .    |        | M(n) |
%        |     1  |    | v(n)  |        |  0   |
%        |                        1       |    | v(n+1)|        |  0   |
%        ----    --     --        --    --
%
%        |_________________________________|   |_______|        |______|
%                       A                          v              b
%
%v(-1) and v(n+1) are known as the phantom points,and
%they do not represent any real displacements. The last two
%lines embody the boundary conditions: v(0)=0 and v(n)=0. By
%solving for v in the matrix equation Av=b, you will obtain
%the displacements, including the phantom points.

%get the number of discretization points
m=length(M);

%make a diagonals for the matrix
firstdiagonal=ones(m+2,1)/dx^2;
seconddiagonal=-2*ones(m+1,1)/dx^2;
thirddiagonal=ones(m,1)/dx^2;

%make a tridiagonal matrix of dimensions (m+2,m+2)
A=diag(firstdiagonal,0)+diag(seconddiagonal,1)+diag(thirddiagonal,2);

%replace the last row with a bunch of zeros
A(m+1:m+2,:)=zeros(2,m+2);

%now put ones in the appropriate places
A(m+1,2)=1;
A(m+2,m+1)=1;
```

```
%make the vector b
b=M/(E*I);
b(m+1)=0;
b(m+2)=0;

%make sure that b is a column vector
if size(b,1)==1
  b=b';
end

%solve the system of equations
v=(A\b);

%discard the phantom points
v=v(2:m+1);
```

5.2.2.3 P2M.m

```
function M=P2M(P,a,L,dx)
%FUNCTION M=P2M(P,A,L,dx) returns the bending moment diagram for a
%point load on a simply supported beam
%
%              |P
%              |
%             \|/
%    --------------------------
%   /\                        /\
%    |----A----|
%    |------------L------------|

%              |-------B-------|
%    |---> x

%get b
b=L-a;

%get the reactions
R1=P*b/L;
R2=P*a/L;

%make a vector x
x=(0:dx:L);

%get the moment
M=R1*(x<a).*x + R2*(x>=a).*(L-x);

%the above statement is a shorthand way of saying

%for i=1:length(x)
%  if x(i) < a
```

```
%    M(i)=R1*x(i);
%  elseif x(i) >= a
%    M(i)=R2*(L-x(i));
%  end
%end

%Matlab is very fast at evaluating matrix operations
%like the shorthand command for M. Matlab is not so fast
%at doing if statements and for loops.
```

5.2.2.4 P2V.m

```
function V=P2V(P,a,L,dx)
%FUNCTION V=P2V(P,A,L,dx) returns the Shear diagram for a
%point load on a simply supported beam
%
%              |P
%              |
%             \|/
%    --------------------------
%   /\                        /\
%    |----A----|
%    |------------L------------|

%              |-------B-------|
%    |---> x

%get b
b=L-a;

%get the reactions
R1=P*b/L;
R2=P*a/L;

%make a vector x
x=(0:dx:L);

%get the shear
V=(x<a)*R1 - (x>=a)*R2;

%the above statement is a shorthand way of saying

%for i=1:length(x)
%  if x(i) < a
%    V(i)=R1;
%  elseif x(i) >= a
%    V(i)=-R2;
%  end
%end

%Matlab is very fast at evaluating matrix operations
```

```
%like the shorthand command for M. Matlab is not so fast
%at doing if statements and for loops.
```

5.2.2.5 w2M.m

```
function M=w2M(w,a,b,L,dx)
%FUNCTION M=w2M(W,a,b,L,dx) returns the bending moment diagram for a
%distributed load on a simply supported beam
%
%
%          -------
%          |||||||  w
%    --------------------------
%   /\                         /\
%    |--A--|--B--|
%    |------------L------------|

%    |---> x
%                |--------C----|


%get c
c=L-a-b;

%get the reactions
R1=w*b/2/L*(2*c+b);
R2=w*b/2/L*(2*a+b);

%make a vector x
x=(0:dx:L);

%get the bending moment
M=(x<a).*(x*R1) + ((x>=a)&(x<(a+b))).*(R1*x-w/2*(x-a).^2) + (x>=(a+b)).*(R2*(L-x));

%the above statement is a shorthand way of saying

%for i=1:length(x)
%  if x(i) < a
%    M(i)=(x(i)*R1);
%  elseif (x(i) >= a) &| (x(i) < (a+b))
%    M(i)=(R1*x(i)-w/2*(x(i)-a).^2);
%  elseif (x(i) >= (a+b))
%    M(i)=(R2*(L-x(i)));
%  end
%end

%Matlab is very fast at evaluating matrix operations
%like the shorthand command for M. Matlab is not so fast
%at doing if statements and for loops.
```

5.2.2.6 w2V.m

```
function V=w2V(w,a,b,L,dx)
%FUNCTION V=w2V(W,a,b,L,dx) returns the shear diagram for a
%distributed load on a simply supported beam
%
%
%          _______
%          |||||||  w
%    ---------------------------
%   /\                          /\
%    |--A--|--B--|
%    |------------L------------|

%    |---> x
%                |--------C----|


%get c
c=L-a-b;

%get the reactions
R1=w*b/2/L*(2*c+b);
R2=w*b/2/L*(2*a+b);

%make a vector x
x=(0:dx:L);

%get the shear
V=(x<a)*R1 +((x>=a)&(x<(a+b))).*((x-a)*(-R2-R1)/b+R1) - (x>=(a+b))*R2;

%the above statement is a shorthand way of saying

%for i=1:length(x)
%  if x(i) < a
%    V(i)=R1
%  elseif (x(i) >= a) & (x(i) < (a+b))
%    V(i)=(x-a)*(-R2-R1)/b+R1
%  elseif (x(i) >= (a+b))
%    V(i)=-R2
%  end
%end

%Matlab is very fast at evaluating matrix operations
%like the shorthand command for M. Matlab is not so fast
%at doing if statements and for loops.
```

5.3 beam2

5.3.1 Description

SUBDIRECTORY: beam2

SYNOPSIS: principle stress plots in a simple beam under midspan point load (Structures)

DESCRIPTION: beam_under_load.m plots the principle stresses in a prismatic rectangular beam under uniform loads, with simple supports, assuming plane stress conditions.

CONTENTS:

`beam_under_load.m` main program

Written by: Brian Rose

5.3.2 Listing

5.3.2.1 `beam_under_load.m`

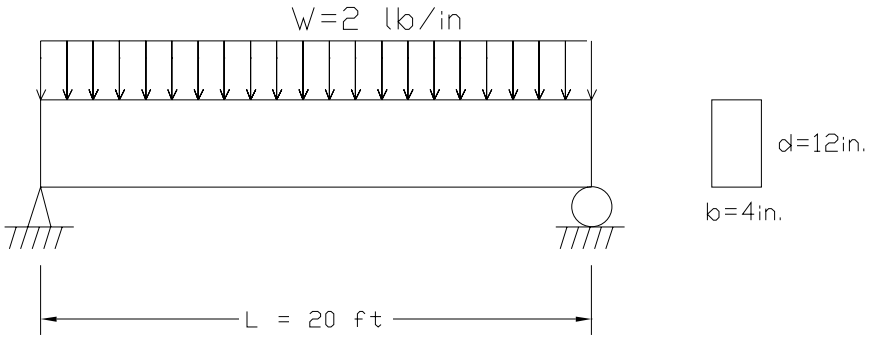


Figure 5.9: Beam Under Continuous Load

Major formulas used:

$$
\begin{aligned}
M &= \frac{\omega}{2}x(L-x) \\
V &= \omega 9\frac{L}{2}-x) \\
Q &= b(\frac{d}{2}-y)(\frac{y}{2}+\frac{d}{4})
\end{aligned}
$$

$$\begin{aligned} \sigma_{xx} &= -\frac{My}{I} \\ \tau &= \frac{VQ}{Ib} \\ \sigma_{1,2} &= \frac{\sigma_{xx}}{2} \pm \sqrt{\frac{\sigma_{xx}^2}{4} + \tau^2} \end{aligned}$$

beam_under_load.m file starts here:

```
%script beam_under_load will plot the principle stresses in a prismatic
%rectangular beam under uniform loads, with simple supports. We will
%assume plane stress conditions.
%beam is depth d, width b, length L and has uniform load of w

%define dimensions
b=4;
d=12;
L=20*12;

%define the uniform load
w=2;

%get some cross sectional properties
A=b*d;
I=b*d^3/12;

%set up x and y coordinate system
numberx=100;
numbery=40;
x=0:L/numberx:L;
y=-d/2:d/numbery:d/2;

%get a mesh of x and y's
[x,y]=meshgrid(x,y);

%calculate moment and shear
M=w/2*x.*(L-x);
V=w*L/2-w*x;

%calculate the first moment of intertia
Q=b*(d/2-y).*(y/2+d/4);

%calculate bending stress and shear stress for each point in the beam
sb=-M.*y/I;
sv=V.*Q/I/b;

%calculate the principle and maximum shear stresses
tmax=sqrt(sb.^2/4+sv.^2);
p1=sb/2+tmax;
p2=sb/2-tmax;
```

```
%set the colorscale for the plots
%for more details type "help color" and "help colormap"
colormap(jet)

%plot results
%the shading interp takes away ugly gridlines
subplot(311)
pcolor(x,y,p1)
shading interp
title('Principle Stress 1')
colorbar

subplot(312)
pcolor(x,y,p2)
shading interp
title('Principle Stress 2')
colorbar

subplot(313)
pcolor(x,y,tmax)
shading interp
title('Maximum Shear Stress')
colorbar

%for those of you who appreciate psychedelic stuff try this
%spinmap(20)
```

5.4 boussinesq

5.4.1 Description

SUBDIRECTORY: boussinesq

SYNOPSIS: stress plot for semi-infinite domain under a point load (Geotech)

DESCRIPTION: boussinesq.m plots the stress distribution of a point load on a semi-infinite half space

CONTENTS:

`boussinesq.m` main program

Written by: Brian Rose

5.4.2 Listing

5.4.2.1 `boussinesq.m`

Main formula:

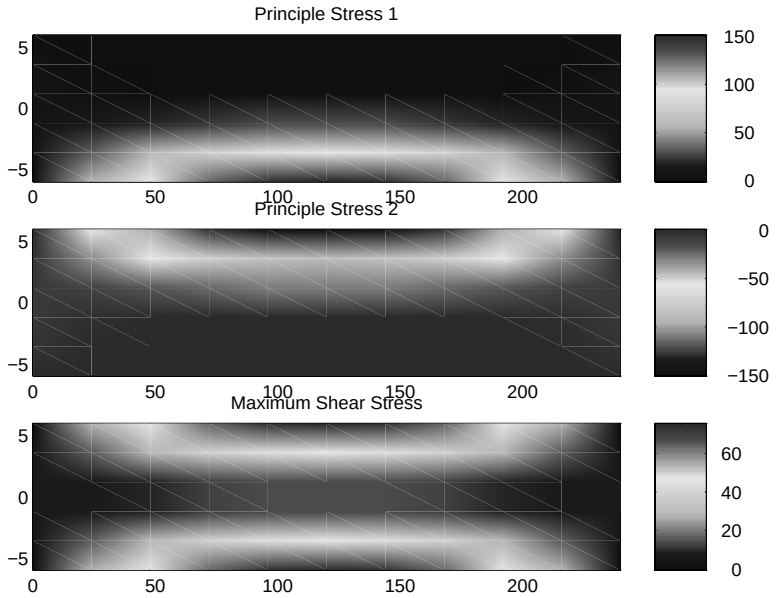


Figure 5.10: Sample output of principle stresses for *beam_under_load.m* with $\omega = 2, b = 4, d = 12,$ and $L = 20$.

$$\sigma_y = \frac{3P(\sin\theta)^3}{2\pi R^2}$$

boussinesq.m file starts here:

```
%script boussinesq plots the stress distribution
%of a point load on a semi-infinite half space
%
%
%         |  P
%         |
%         |
%        \|/
%------------------------------
%
%  --->x
%  |
%  |
%  |/ y

%set up a radial grid
%define the r's and theta's for our grid as vectors
r=0.01:.002:0.1;
theta=0:pi/20:pi;
P=100;
```

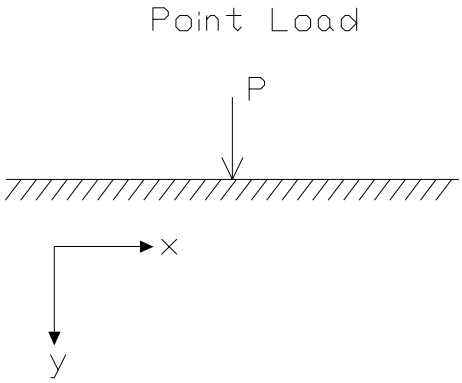


Figure 5.11: Point Load on Semi-Infinite Domain

```
%Now define a R and THETA for each grid point
%this means that R and THETA are both matrices
[R,THETA]=meshgrid(r,theta);

%get the stress in the y direction
stressy=3*P/(2*pi)*(sin(THETA)).^3./R.^2;

%take R and THETA and convert them to cartesian coordinates
[x,y] = pol2cart(THETA,R);

%plot the results (note the -y is so that
%it plots upside down
pcolor(x,-y,stressy)


%note that since the solution blows up at R=0 and theta=0,
%we have left out the solution for that point.
```

5.5 dodgecity

5.5.1 Description

SUBDIRECTORY: dodgecity

SYNOPSIS: statistical analysis of wind power in Dodge City (Energy Management)

DESCRIPTION: wind.m makes a histogram of windspeed and does some simple statistics on it. Accompanies Jan Kreider's handout "Wind Power Assessment"

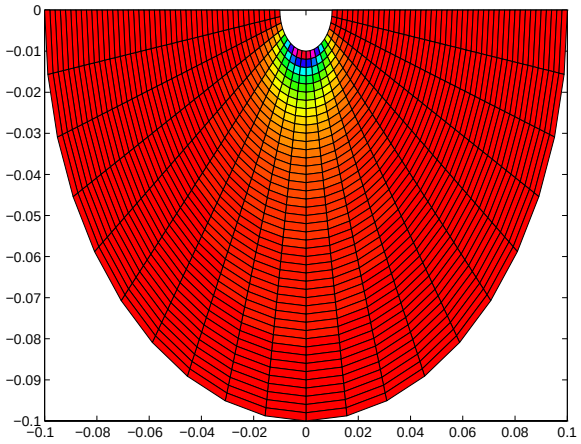


Figure 5.12: Stress plot for point Load on Semi-Infinite Domain

CONTENTS:

`apr`	data file
`aug`	data file
`dec`	data file
`feb`	data file
`jan`	data file
`jul`	data file
`jun`	data file
`mar`	data file
`may`	data file
`nov`	data file
`oct`	data file
`sep`	data file
`wind.m`	main program

Written by: Brian Rose

5.5.2 Listing

5.5.2.1 `wind.m`

Equation used:

$$P = \frac{1}{2}\rho v^3$$

wind.m file starts here:

```
%Script wind.m
%Makes a histogram of windspeed and does some simple statistics‘
%on it. Accompanies Jan Kreider’s handout Wind Power Assessment

%load data for all 12 months
%use the -ascii extension to tell Matlab that we’re reading ascii
%files. Matlab will take the data in the file jan and stuff it into
%a matrix called jan. Same goes for all the other months

load jan -ascii
load feb -ascii
load mar -ascii
load apr -ascii
load may -ascii
load jun -ascii
load jul -ascii
load aug -ascii
load sep -ascii
load oct -ascii
load nov -ascii
load dec -ascii


%Concatentate them into one big matrix called data
data=[jan;feb;mar;apr;may;jun;jul;aug;sep;oct;nov;dec];

%now pull out the wind speed from the 7th column of data
V=data(:,7);

%specify the bins
%the bin goes from 0 to the maximum windspeed rounded up to the
%next whole number. Try typing ’help ceil’ in the matlab window
bin=(0:ceil(max(V)));

%sort the velocity into bins
hours=hist(V,bin);

%make a histogram
%on your own try doing bar(bin,hours) and bar(midpoint,hours).
%the results will be slightly different.
midpoint=bin+0.5;
bar(midpoint,hours)
title(’Dodge City’)
xlabel(’Wind Velocity (m/s)’)
ylabel(’Hours’)

%Compute power.
%Note here that the we use .^3 instead of ^3
%because the former is an array operation (means operate
%element by element), whereas the latter is a matrix operation.
rho=1.124;
Power=rho/2*(V.^3);
```

```
%do some statistics on power
mean_power=mean(Power)
max_power=max(Power)
std_power=std(Power)
median_power=median(Power)
```

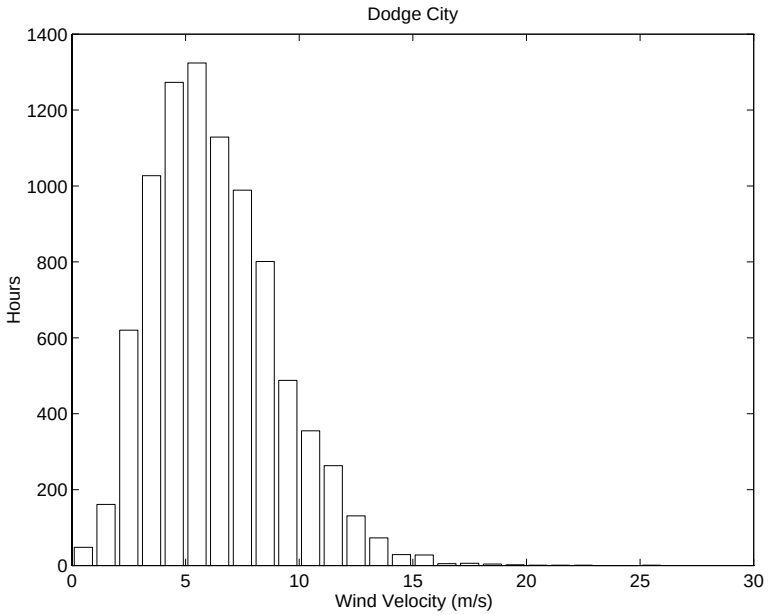


Figure 5.13: Sample histogram of windspeed

```
Simple Statistics calculated from wind data:

mean_power =  194.0820

max_power =  8.7812e+03

std_power =  311.3429

median_power =  70.2500
```

5.6 effectiveL

5.6.1 Description

SUBDIRECTORY: effectiveL

SYNOPSIS: effective length of a sidesway inhibited column of a building (Structures)

DESCRIPTION: effectiveL.m returns the effective length factor, K given the G values for the top and bottom of a sidesway inhibited column.

CONTENTS:

`effectiveL.m` main function. Note, not a scipt.

Recall that the Euler buckling load was derived for a pinned column. In many cases, a column will have different boundary conditions. It can be shown that in all cases, the buckling load would be given by

$$\boxed{P_{cr} = \frac{\pi^2 EI}{(KL)^2}} \tag{5.20}$$

where K is called **effective length factor**, and KL is the **effective length**. and

$$\boxed{\sigma_{cr} = \frac{\pi^2 E}{\left(\frac{KL}{r}\right)^2}} \tag{5.21}$$

The ratio $\frac{KL}{r}$ is termed the **slenderness ratio**. The effective length, can only be determined by numerical or approximate methods, and is the distance between two adjacent (real or virtual) inflection points, Fig. 5.14, 5.15

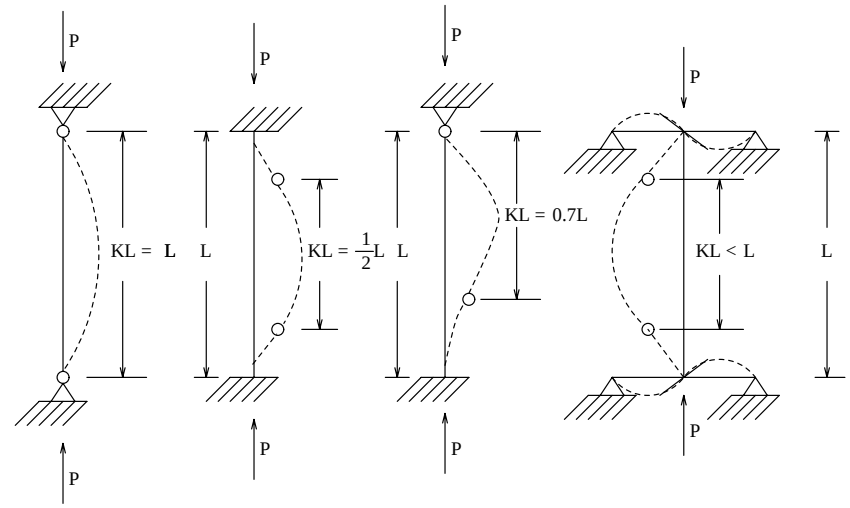


Figure 5.14: Column Effective Lengths

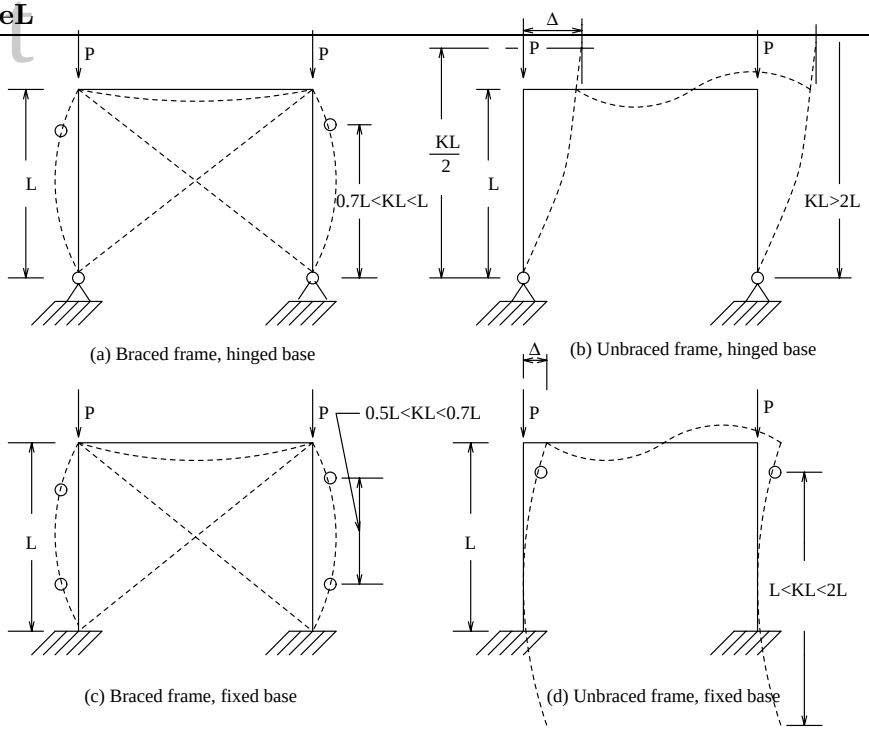


Figure 5.15: Frame Effective Lengths

The most widely used charts for the effective length determination are those produced by the Structural Stability Research Council. The alignment chart, for an individual column, Fig. 5.16 is shown in Fig. 5.17. It should be noted that this chart assumes that all members are still in the **elastic** range.

The use of the alignment chart involces computing G at each end of the column using the following formula

$$\boxed{G_a = \frac{\sum \frac{I_c}{L_c}}{\sum \frac{I_g}{L_g}}} \tag{5.22}$$

where G_a is the stiffness at end a of the column, I_c, I_g are the moment of inertias of the columns and girders respectively. The summation must include only those members which are **rigidly** connected to that joint and lying in the plane for which buckling is being considered.

Hence, once G_a and G_b are determined, those values are connected by a straight line in the appropriate chart, and k is the point of intersection of that line with the midle axis.

Alternatively :-)

$$\boxed{\frac{G_A G_B}{4}\left(\frac{\pi}{K}\right)^2 + \left(\frac{G_A + G_B}{2}\right)\left(1 - \frac{\pi/K}{\tan \pi/K}\right) + \frac{2\tan \pi/2K}{\pi/K} = 1} \tag{5.23}$$

(Ref. McGuire P. 467).

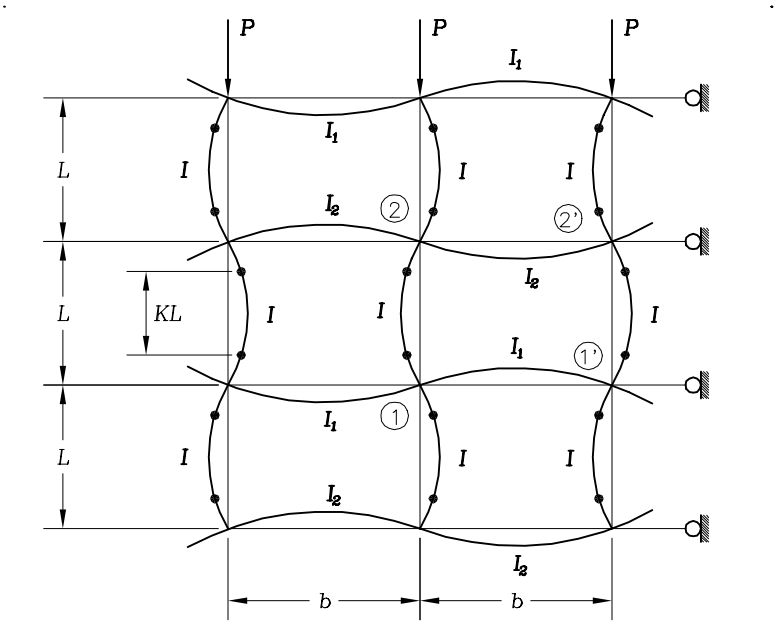


Figure 5.16: Column Effective Length in a Frame

5.6.2 Listing

5.6.2.1 effectiveL.m

```
function K=effectiveL(Ga,Gb)
%K=effectiveL (Gtop, Gbot) returns the effective length
%factor, K given the G values for the top and bottom of a
%sidesway inhibited column.
%
%K can be found by solving the following equation numerically
%
% _     _   _    _ 2     _       _    _             _
%| Ga Gb | |  pi  |     |  Ga+Gb  |  |        pi/K    |      2 tan(pi/2K)
%| ----- | | ---- |   + |-------- |  | 1 -  --------- |  +  -------------  = 1
%|   4   | |  K   |     |    2    |  |      tan(pi/K) |          pi/K
% -     -   -    -       -       -    -              -
%
%This function uses a root finding scheme, namely the bisection
%method to find the root of the following equation.
%
%     _     _   _    _ 2     _       _    _             _
%    | Ga Gb | |  pi  |     |  Ga+Gb  |  |        pi/K    |      2 tan(pi/2K)
%R = | ----- | | ---- |   + |-------- |  | 1 -  --------- |  +  -------------  - 1
%    |   4   | |  K   |     |    2    |  |      tan(pi/K) |          pi/K
%     -     -   -    -       -       -    -              -
%
```

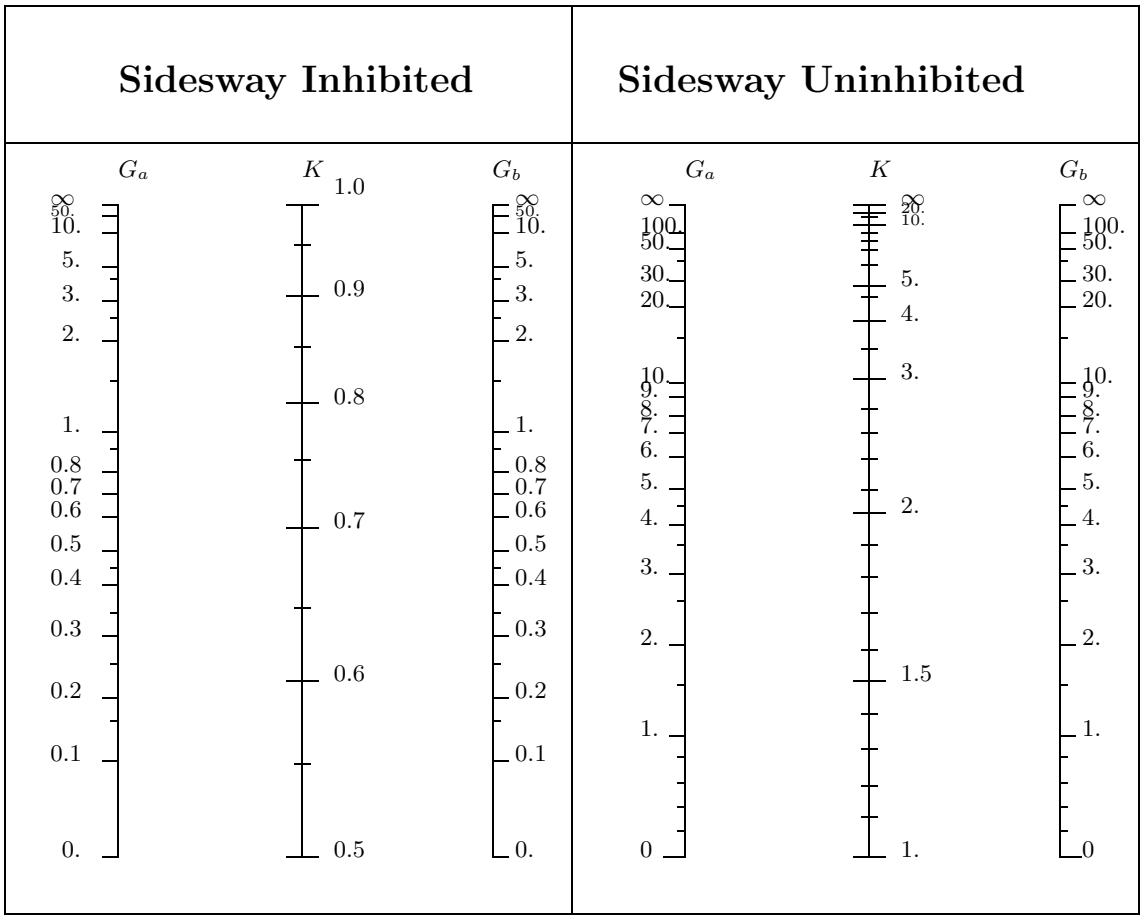


Figure 5.17: Standard Alignment Chart (AISC)

```
%     note: R is the residual and we want to find K such that R=0;
%
%For this method, we make two guesses of K, Klower and Kupper, that
%bound the correct K. Then, we estimate K as the average of the
%two initial guesses. Next we figure out if our if the estimated
%K and Kupper or the estimated K and Klower bound the solution.
%Then with the two new bounds, we start over again.
%
%More ambitious people can use the Matlab function fzero


%define upper and lower bounding estimates of K
Kupper=.98;
Klower=.52;

%define the tolerance for rootfinding
tolerance=1e-3;

%get the residual for the upper and lower bounding estimates
Rupper=Ga*Gb/4*(pi/Kupper)^2 + (Ga+Gb)/2*(1-pi/Kupper/tan(pi/Kupper))+2*tan(pi/2/Kupper)/(pi/Kupper)-1;
Rlower=Ga*Gb/4*(pi/Klower)^2 + (Ga+Gb)/2*(1-pi/Klower/tan(pi/Klower))+2*tan(pi/2/Klower)/(pi/Klower)-1;

%set R to something big so that we can enter the loop
R=100*tolerance;

%loop until we find the root
while abs(R)>tolerance

  %let our estimate of K be halfway between Kupper and Klower
  K=(Kupper+Klower)/2;

  %calculate the residual corresponding to K
  R=Ga*Gb/4*(pi/K)^2 + (Ga+Gb)/2*(1-pi/K/tan(pi/K))+2*tan(pi/2/K)/(pi/K)-1;

  %if K and Kupper bound the root let Klower=K
  if R>0
    Klower=K;
    Rlower=R;

  %if K and Klower bound the root let Kupper=K
  else
    Kupper=K;
    Rupper=R;
  end
end
```

5.7 eigenvalues

5.7.1 Description

SUBDIRECTORY: eigenvalues

SYNOPSIS: Principle stresses and directions of a 3-Dimensional stress element

DESCRIPTION: eigenvalues.m prompts the user for stress components of a 3-dimensional stress element and then calculates the principal stresses and directions.

CONTENTS:

`eigenvalues.m` main program

Written by: Brian Rose

5.7.2 Listing

5.7.2.1 `eigenvalues.m`

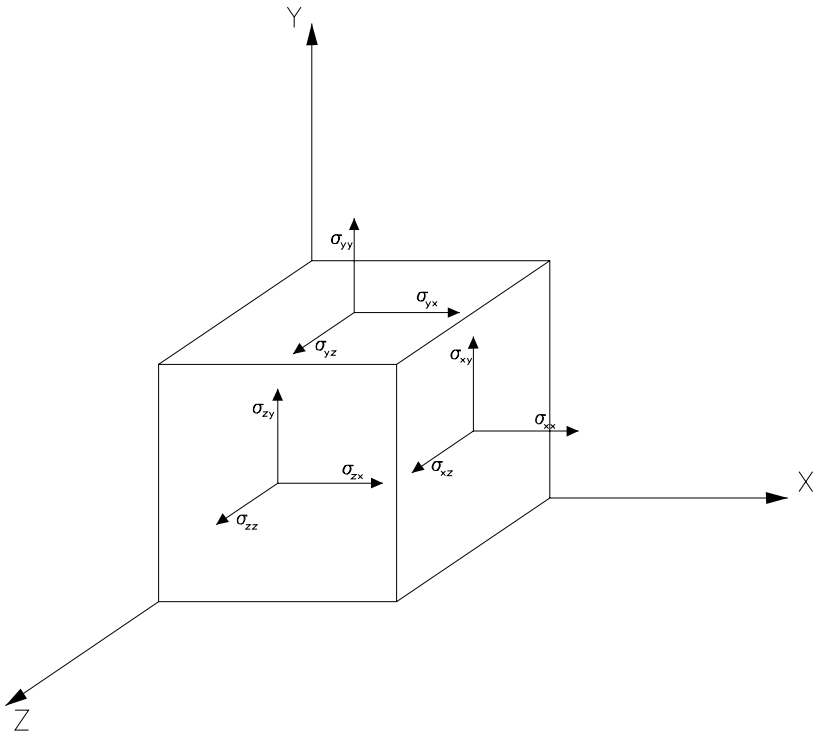


Figure 5.18: 3-Dimensional Stress Element

```
%The script "eigenanalysis" prompts the user for stress components
%in 3-D. Then is calculates the principle stresses and the
%principle directions
```

```
%clear the screen
clc

%prompt user for some numbers
%note that I can define the elements of the matrix
%without even defining the size of the matrix before hand
Stress(1,1)=input('Enter normal stress on x face: ');
Stress(2,2)=input('Enter normal stress on y face: ');
Stress(3,3)=input('Enter normal stress on z face: ');
Stress(1,2)=input('Enter shear stress xy:         ');
Stress(1,3)=input('Enter shear stress xz:         ');
Stress(2,3)=input('Enter shear stress yz:         ');

%Make the stress matrix symmetric
%First pull out the upper triangular part of the matrix (without the diagonal).
%Second transpose the part we pulled out and add it to the Stress matrix
%for more details on the command triu, try typing "help triu"
Stress
temp=triu(Stress,1)
Stress=Stress+temp'

%Now do the eigenanalysis
[eigenvectors,principle_stress]=eig(Stress);

%Display the results as well as the stress matrix
%clc
disp('The stress tensor is')
disp(Stress)
disp(' ')
disp(' ')
for i=1:3
  disp('Principle stress number')
  disp(i)
  disp(principle_stress(i,i))
  disp('Its unit normal is:')
  disp(eigenvectors(:,1)')
  disp(' ')
  disp(' ')
end
```

Sample run:

```
>> eigenvalues

Enter normal stress on x face: 3

Enter normal stress on y face: 4

Enter normal stress on z face: 1

Enter shear stress xy:         2

Enter shear stress xz:         5
```

```
Enter shear stress yz:         8

Stress =

     3     2     5
     0     4     8
     0     0     1


temp =

     0     2     5
     0     0     8
     0     0     0


Stress =

     3     2     5
     2     4     8
     5     8     1

The stress tensor is
     3     2     5
     2     4     8
     5     8     1



Principle stress number
     1

    1.4822

Its unit normal is:
   -0.8460    0.5314    0.0442



Principle stress number
     2

   -6.4509

Its unit normal is:
   -0.8460    0.5314    0.0442



Principle stress number
     3
```

```
   12.9688

Its unit normal is:
   -0.8460    0.5314    0.0442
```

5.8 infiltration

5.8.1 Desctiption

SUBDIRECTORY: infiltration

SYNOPSIS: infiltration of water in soil where ponding has occured (Hydrology)

DESCRIPTION: infiltration.m calculates the the depth of infiltration of water in a soil medium after ponding has occured. The problem is taken from "Applied Hydrology" by V.Chow, D. Maidment, and L. Mays page 120, example 4.4.2.

CONTENTS:

`infiltration.m` main program

Written by: Brian Rose

5.8.2 Listing

5.8.2.1 infiltration.m

Equation used:

$$F = F_p + \psi\Delta\theta \ln[\frac{\psi\Delta\theta + F}{\psi\Delta\theta + F_p}] + k(t - t_p)$$

infiltration.m file starts here:

```
%script infiltration will calculate the the depth of infiltration
%of water in a soil medium after ponding has occured. The problem is
%taken from "Applied Hydrology" by V.Chow, D. Maidment, and L. Mays
%page 120, example 4.4.2.
%
%The equations is
%
%                           psiDtheta + F
% F - Fp - (psiDtheta)*ln ---------------- = K (t-tp).
%                           psiDtheta + Fp
%
%
%We will solve this equation for F by rewriting the above as
%
```

```
%                          psiDtheta + F
% F = Fp + (psiDtheta)*ln ---------------- + K (t-tp).
%                          psiDtheta + Fp
%
%First we will guess an F. Then we will plug this F into the right hand
%side of the equation. This will give us a new F. We will then
%plug this new F into the right hand side and so on.

%define the contants
Fp=0.85
psiDtheta=5.68
K=0.65
tp=.17
t=1

%make an initial guess of F
F=.2;

%iterate
for i=1:1000
  F=Fp+psiDtheta*log((psiDtheta+F)/(psiDtheta+Fp))+K*(t-tp);
end

%display F
F

Sample run:

>> infiltration

Fp =  0.8500

psiDtheta =  5.6800

K =  0.6500

tp =  0.1700

t =  1

F =  3.0176
```

5.9 montecarlo

5.9.1 Description

SUBDIRECTORY: montecarlo

SYNOPSIS: montecarlo simulation of a simple beam under midspan point load (Structures and Statistics)

DESCRIPTION: montecarlo.m does a monte carlo simulation for a simply supported beam under a point load at the midspan. The load, P and the yield stress, fy are normally distributed.

CONTENTS:

`montecarlo.m` main program

Written by: Brian Rose

5.9.2 Listing

5.9.2.1 `montecarlo.m`

```
%script montecarlo does a monte carlo simulation for a simply
%supported beam under a point load at the midspan
%the load, P and the yield stress, fy are normally distributed

%define a sample size
samplesize=5000;

%define the properties of the beam
L=480;
S=150;

%define the mean and standard deviation for
%yield stress and load
meanfy=36
meanP=40
stdfy=1
stdP=1

%get a normally distributed set of stress values and
%a normally distributed set of load values
%note that randn gives me a standard normal
%distribution so I have to convert to a normal
%distribution
fy=meanfy+stdfy^2*randn(samplesize,1);
P=meanP+stdP^2*randn(samplesize,1);

%calculate the max stress in a simple span beam loaded in the center
M=P*L/4;
f=M/S;

%define X=capacity/demand
X=fy./f;

%plot f and fy
subplot(211)
bin=(30:.1:42);
plot(bin,hist(f,bin)/samplesize,bin,hist(fy,bin)/samplesize)
legend('computed stress','yield stress');
```

```
xlabel ('stress (ksi)')
ylabel('PDF')
grid

%plot X
subplot(212)
bin=(-0.5:0.1:2);
plot(bin,hist(log(X),bin)/samplesize);
legend('computed stress','yield stress');
xlabel ('ln of C/D (ksi)')
ylabel('PDF of ln(C/D)')
legend off
grid


%get the safety index
%note that log denotes the natural log
beta=mean(log(X))/std(log(X))

%get the probability of failure
failureP=sum(f>fy)/length(f)
```

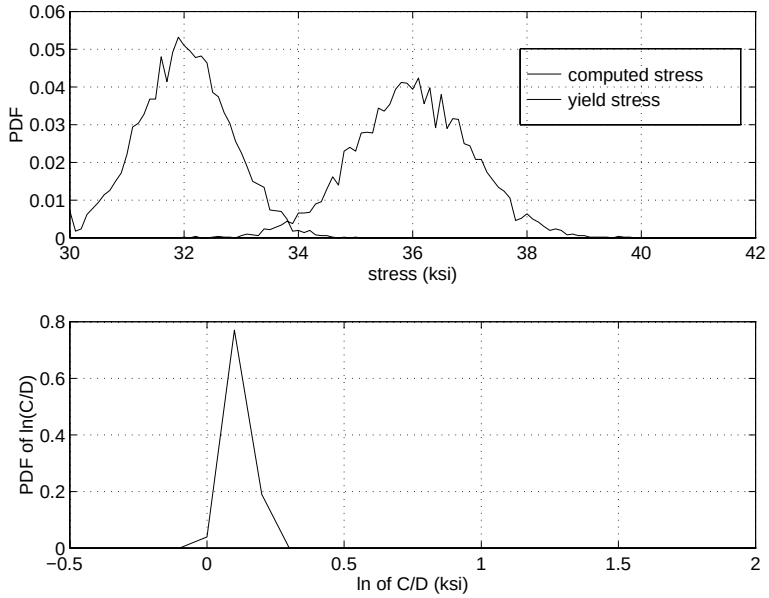


Figure 5.19: Sample output for a monte carlo simulation for a simply supported beam under a point load at the midspan.

5.10 3dplot

5.10.1 Description

SUBDIRECTORY : 3Dplots

SYNOPSIS: takes elevation readings and plots them in 3D (Surveying)

DESCRIPTION: pix.m will plot the elevation data for some mountains. The elevations are contained in the file mtns.mat. Matrices x and y contains x and y coordinates of each elevation reading. Matrix z contains the elevations. The first part of the code sets up buttons that allows the user to change the perspective of the plots: the first slider changes the azimuth and the second slider changes the elevation. The second part of the code cylces through each of type of plot.

CONTENTS:

`cont.m`	creates the a fictitius elevations readings for a mountain
`mtns.mat`	contains the elevation reading of mountains
`pix.m`	the main program
`smooth.m`	takes the elevation readings and smooths them out

Written by: Brian Rose

5.10.2 Listing

5.10.2.1 `cont.m`

```
[x,y]=meshdom(0:.1:2*pi,0:.1:2*pi);
a=rand(1,5);
b=rand(1,5);
a=3.5.^(a+.1);
b=3.5.^(b+.1);
z=zeros(size(x));

for i=1:size(a,2)
  z=z+log(abs(sin(a(i)*x)))+log(abs(cos(b(i)*y)));
end

z=z+rand(size(z));
mesh(x,y,smooth(z,2))
figure(1)
```

5.10.2.2 `pix.m`

```
%script pix.m
%will plot the elevation data for some mountains.
%The elevations are contained in the file mtns.mat.
%Matrices x and y contains x and y coordinates of
%each elevation reading. Matrix z contains the elevations.
%The first part of the code sets up buttons that allows
```

```
%the user to change the perspective of the plots: the first
%slider changes the azimuth and the second slider changes
%the elevation. The second part of the code cylces through
%each of type of plot.

%clear screens, etc
clear
figure(1)
clg

%load the elevation data
load mtns

%set up the buttons for the 3D plots
hAZ=uicontrol('style','slider','position',[.7 .95 .3 .05],...
'units','normalized','min',0,'max',360,...
'callback','[az,el]=view; az=get(gco,''val''); view(az,el);');

hEL=uicontrol('style','slider','position',[.7 .89 .3 .05],...
'units','normalized','min',0,'max',180,...
'callback','[az,el]=view; el=get(gco,''val''); view(az,el);');

hdef=uicontrol('style','pushbutton','position',[.88 .83 .12 .05],...
'units','normalized','String','Default',...
'callback','view(-37.5, 30)');

%do the 3D plots
mesh(x,y,z)
title('mesh(x,y,z)')
xlabel('press any key to continue')
pause

meshz(x,y,z)
title('meshz(x,y,z)')
xlabel('press any key to continue')
pause

contour3(z)
title('contour3(z)')
xlabel('press any key to continue')
pause

meshc(x,y,z)
title('meshc(x,y,z)')
xlabel('press any key to continue')
pause

surf (x,y,z)
title('surf (x,y,z)')
pause

surfc(x,y,z)
```

```
title('surfc(x,y,z)')
xlabel('press any key to continue')
pause

surfl(x,y,z)
title('surfl(x,y,z)')
xlabel('press any key to continue')
pause

waterfall (x,y,z)
title('waterfall (x,y,z)')
xlabel('press any key to continue')

%clear the figure window
clg

%do the 2D plots
contour(x,y,z)
title('contour(x,y,z)')
xlabel('press any key to continue')
pause

pcolor  (x,y,z)
title('pcolor  (x,y,z)')
colorbar
xlabel('press any key to continue')
pause
```

5.10.2.3 smooth.m

```
function newz=smooth(z,weight)

[m,n]=size(z);
if ~exist('weight')
  weight=5;
end

newz=z;

for i=2:m-1
  for j=2:n-1
    newz(i,j)=(z(i-1,j-1)+  z(i-1,j+1)+  z(i+1,j-1)+  z(i+1,j+1)+ weight*z(i,j))/(4+weight);
  end
end
```

5.11 zec

5.11.1 Description

SUBDIRECTORY: zec

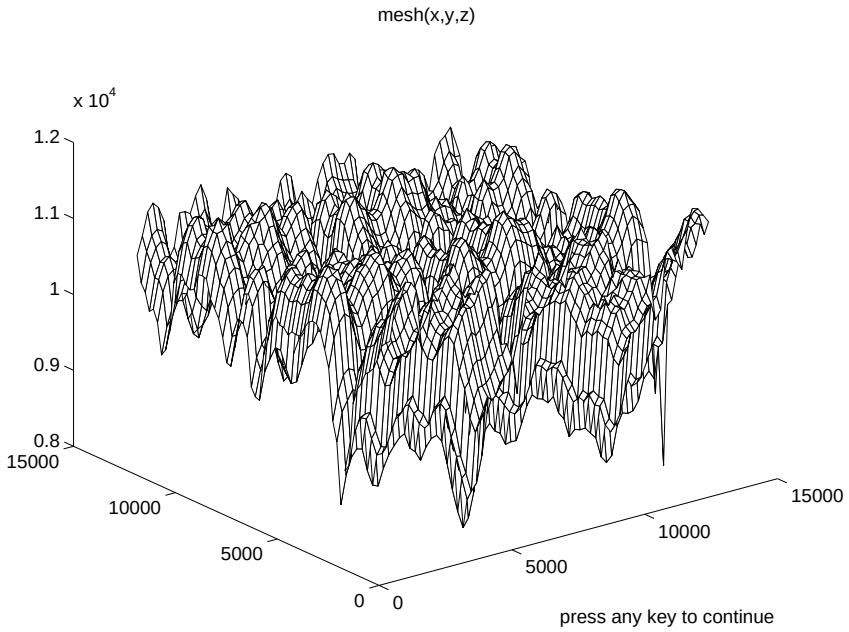


Figure 5.20: Sample plot from pix.m

SYNOPSIS: statistical analysis for heating energy of a building (Energy Management)

DESCRIPTION: zec.m does a linear regression to find a relationship between

CONTENTS:

`linregress.m`	function that performs linear regression
`zec.m`	main program
`zecdata`	contains heat and temperature data

Written by: Brian Rose

5.11.2 Listing

5.11.2.1 `linregress.m`

```
function [p,Rsquared,sigmabeta1]=linregress(x,y)
%[p,Rsquared,sigmabeta1]=linregress(x,y) performs a linear regression on x and y.
%p is a vector where y=p(1)+p(2)*x.
%
%[p,Rsquared,sigmabeta1]=linregress(x,y) will return R^2 as well as the standard
%deviation in p(2)


%make x and y into row vectors
```

```
if size(x,1)<size(x,2)
  x=x';
end
if size(y,1)<size(y,2)
  y=y';
end

%print an error message is x and y are not the same lengths
if size(x)~=size(y)
  disp('x and y need to be the same size')
  return
end

n=length(x)

%linear regress. A contains a column of ones then the x values
A=[ones(length(x),1) ,x];

%find p
p=inv(A'*A)*A'*y;

%get r, which is the residual
r=y-(p(1)+p(2)*x);


%get SSE, SST, and Rsquared
SSE=sum(r.^2);
SST=sum((y-mean(y)).^2);
sigmasquared=SSE/(length(y)-2);
sigmasquaredbeta1=sigmasquared/sum((x-mean(x)).^2);
sigmabeta1=sqrt(sigmasquaredbeta1);
Rsquared=1-SSE/SST;
```

5.11.2.2 zec.m

Emperical linear relationship between heating energy and outdoor temperature:

$$\text{HeatEnergy} = k_1 + k_2 * \text{Temperature}$$

k_1 and k_2 may be found using regression features of Matlab. Regression improves as the correlation coefficient(R-squared) value is found closer to '1.0'.

zec.m file starts here:

```
%script zec does a linear regression to find a relationship between
%the heat energy and the outside temperature of a building. This script
%uses data from the Zachry Engineering Center at Texas A&M

%load the data
```

```
load -ascii zecdata

%extract only the columns we need
energy=zecdata(:,11);
temperature=zecdata(:,5);

%do a linear regression. You can use the built in function polyfit
%but that will not give Rsquared or delta. Note that delta is
%the standard deviation of the slope.
[p,Rsquared,delta]=linregress(temperature,energy);
k1=p(1)
k2=p(2)
Rsquared
delta

%plot the results
%plot the raw data
plot(temperature,energy,'ro')
hold on

%plot the equation obtained from the linear regression
x=[min(temperature) max(temperature)];
y=k1+k2*x;
plot(x,y,'g-')
hold off
grid
legend('raw data','linear regression')
ylabel('Heat energy (MMBtu/hr)')
xlabel('Temperature (F)')
title('Heating vs. Temperature Plot')
```

Sample run of it zec.m:

```
>> zec

n =  744

k1 =  7.6999

k2 =  -0.0764

Rsquared =  0.6639

delta =  0.0020
```

5.12 Stress/Strain Programs

5.12.1 Description

SUBDIRECTORY : Stress/Strain Programs

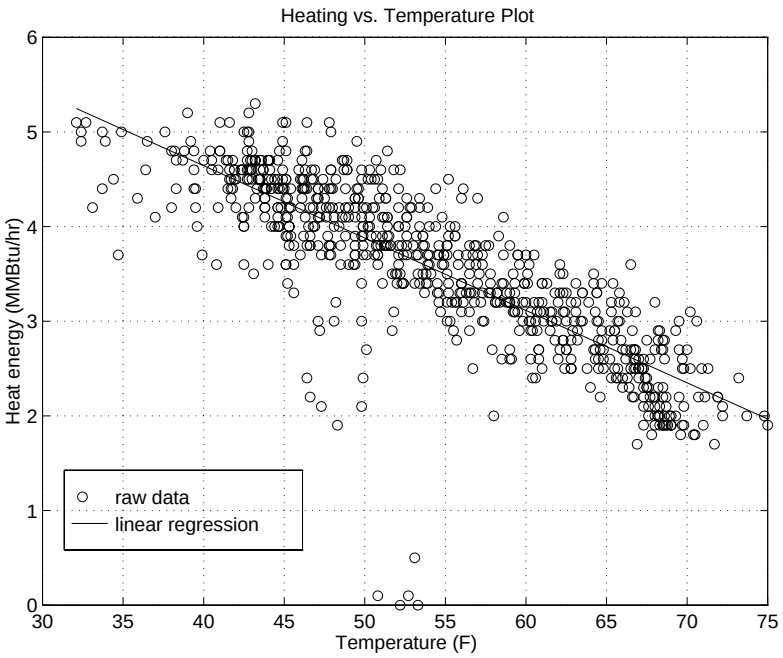


Figure 5.21: Sample output for zec.m

SYNOPSIS: stress and strain analysis for 2-D and 3-D elements

DESCRIPTION: Two main startup commands are used for different analysis:

1. *stressanalysis.m* is a program which provides a menu driven interface to execute 2-D and 3-D stress/strain calculations.
2. *stressdecomp.m* creates a menu for 3-Dimensional analysis of stresses including options to Input Stresses, find Normal and Shear stress components, Principal Invariants, plot the Principal Mohr's Circle with principal stresses and absolute max shear stresses, find Mean and Deviatoric stresses, find deviatoric principal invariants, and plot the Deviatoric principal Mohr's circles.

CONTENTS:
```
Program startup commands.
   stressanalysis.m      - Start stress/strain analysis program.
   stressdecomp.m        - Start stress decomposition program.

Print commands.
   elemprint.m           - Save figure to file and ask to print.
   circprint.m           - Save Mohr's Circles and data to files.

2-Dimensional Stress/Strain Analysis.
```

```
   stress2dmenu.m        - Plane stress/plane strain menu options.
   planestress2d.m       - Plane stress menu.
   planestrain2d.m       - Plane strain menu.
   definitions.m         - Definitions of plane stress and plane strain.
   principalangle.m      - Principal stresses/strains and directions.
   maxshearangle.m       - Maximum shear stress/strain and directions.
   plotelement.m         - Plot of unit stress element.
   plotprincipal.m       - Plot of stress/strain principal values.
   plotmaxshear.m        - Plot of stress/strain maximum shear values.
   strainplotelement.m   - Plot of unit strain element.
   subplots2d.m          - Plot original, principal, & max shear elements.
   transform.m           - Transformation equations.
   rotelement2d.m        - Rotate element by an angle theta.
   plotcircle.m          - Plot 2-D Mohr's Circle.
   mohrcircle.m          - 2-D Mohr's Circle with rotated angle.

3-Dimensional Stress/Strain Analysis.
   stress3dmenu.m        - Stress/Strain menu options.
   stress3d.m            - Stress menu.
   strain3d.m            - Strain menu.
   plotelement3d.m       - Plot of unit stress/strain element.
   plotcircle3d.m        - Plot 3-D Mohr's Circle.

Generalized 2-D and 3-D & Misc. functions.
   hlp.m                 - General help to use menus in stress programs.
   arrow.m               - Draw a line with an arrowhead.
   parammenu.m           - Menu driven way of entering variables.
   principal.m           - Find Principal stresses/strains.
   straintostress.m      - Convert from strain to stress.
   stresstostrain.m      - Convert from stress to strain.

3-D Stress Decomposition.
   mkmatrix.m            - Build a 3-D stress matrix.
   normcomp.m            - Normal stress component.
   shrcomp.m             - Shear stress component.
   invariant.m           - Principal invariants.
   mndevcomp.m           - Mean and Deviatoric stresses.
   devinvariant.m        - Deviatoric principal invariants.
```

Written by: Melissa Holland

5.12.2 Sample Output:

How to access program:

At Matlab prompt type:
p=path
path(p,'/bechtel/users1/graduate/hollanmm/matlab/')
stressanalysis or *stressdecomp*

Typing *stressanalysis* will cause the following menu to appear:

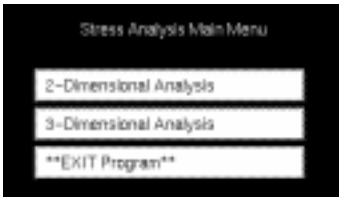


Figure 5.22: First menu encountered after starting *stressanalysis*

Next, choosing *2-Dimensional Analysis* from this menu brings up the next menu.

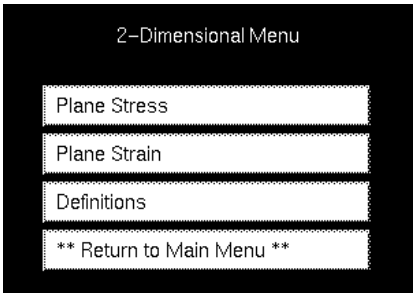


Figure 5.23: Menu encountered after clicking on *2-Dimensional Analysis* button

Finally, choosing *Plane Stress* button brings up the next menu(see next figure).

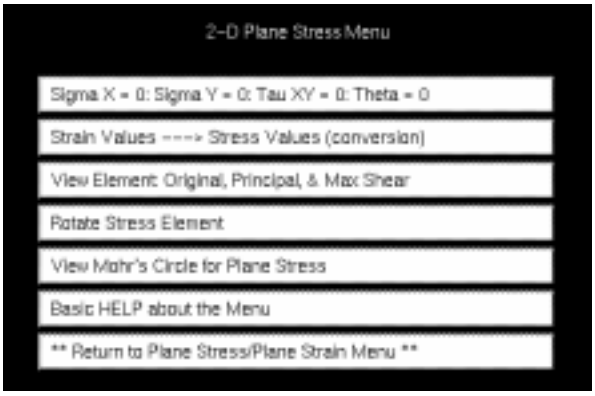


Figure 5.24: Menu encountered after clicking on *Plane Stress* button.

These menus allow easy movement through the program. Clicking on other buttons in the previous menus will bring up other menus or ask for different information at the matlab prompt.

Following are some examples which used *stressanalysis* program menus and input prompts to solve them.

5.12.2.1 Example 1:

Given: A state of plane strain exists at a point in a member, with the nonzero strain components $\epsilon_{xx} = -0.200$, $\epsilon_{yy} = 0.040$, $\epsilon_{xy} = -0.900$.

Objective:

1. Determine the principle strains in the (x,y) plane.
2. Determine the maximum shear strain in the (x,y) plane.
3. Show schematically the deformed shape of a rectangular element with the original undeformed element.
4. Plot Mohr's circle of strain.

Results:

```
                2-D ELEMENT DATA

Original Values:

        XX    = -0.2
        YY    = 0.04
        XY    = -0.9

Principal Values:

  (a)   S1    = 0.827965
        S2    = -0.987965

Maximum Shear Values:

        Avg         = -0.08
  (b)   Max shear   = 1.81593

                2-D MOHR'S CIRCLE DATA

Inputs:

        P1    = (-0.2,-0.9)
        P2    = (0.04,0.9)

        C     = (-0.08,0)

        Tmax  = (-0.08,0.907965)
```

```
        Tmin  = (-0.08,-0.907965)

Principal Values:

        S1    = (0.827965,0)
        S2    = (-0.987965,0)

        Principal Angle = 131.20 degrees
```

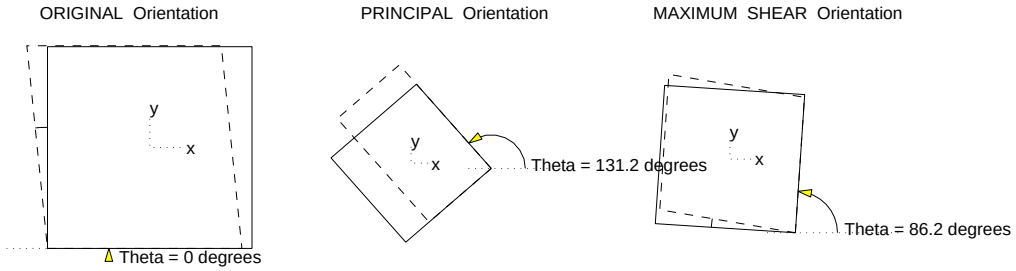


Figure 5.25: Example 1(c): Original, Principal, Maximum Shear Stresses, and Deformed shapes.

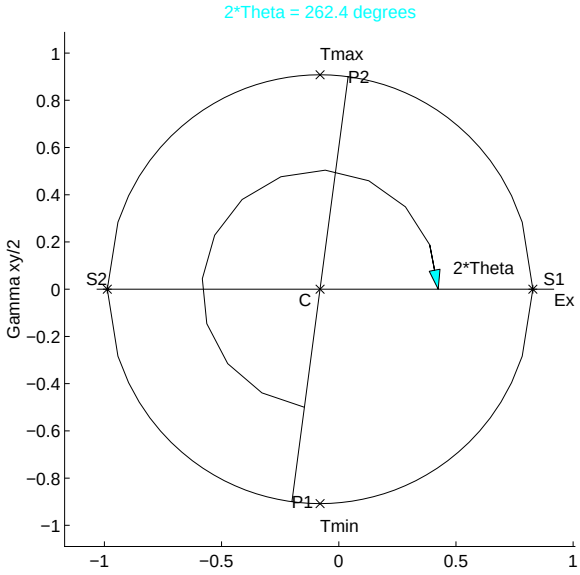


Figure 5.26: Example 1(d): Mohr's circle of 2-D stress.

5.12.2.2 Example 2:

Given: A rectangular strain rosette is cemented to a free surface of a structural member made of alluminum alloy(7075 T6/ $\nu = 0.33$ and E = 72.0 GPa). Under load, the strain readings are $\epsilon_a = 0.00250$, $\epsilon_b = 0.00140$, and $\epsilon_c = -0.00125$ from which it may be found that $\epsilon_{xx} = 0.00250$, $\epsilon_{yy} = -0.00125$, and $\epsilon_{xy} = 0.000775$. **Note:** *a free surface means $\sigma_{zz} = \sigma_{zx} = \sigma_{zy} = 0$, therefore, we will treat as a 2-Dimensional stress case. However, eventhough the stress acts as if 2-Dimensional, strain is 3-Dimensional in this case.*

Objective:

1. Determine principal stresses.
2. Determine the orientation of the volume element on which the principal stresses in the plane of the rosette act.
3. Determine maximum shear stress.
4. Determine the orientation of the volume element on which τ_{max} acts.
5. Plot Mohr's circle of stress.

Results:

```
                2-D ELEMENT DATA

Original Values:

       XX    = 1.68668e+08
       YY    = -3.43396e+07
       XY    = 4.19549e+07

Principal Values:

  (a)  S1    = 1.76997e+08
       S2    = -4.26685e+07

Maximum Shear Values:

       Avg        = 6.71642e+07
  (c)  Max shear  = 1.09833e+08

                2-D MOHR'S CIRCLE DATA

Inputs:

       P1    = (1.68668e+08,4.19549e+07)
```

```
        P2    = (-3.43396e+07,-4.19549e+07)

        C     = (6.71642e+07,0)

        Tmax  = (6.71642e+07,1.09833e+08)
        Tmin  = (6.71642e+07,-1.09833e+08)

Principal Values:

        S1    = (1.76997e+08,0)
        S2    = (-4.26685e+07,0)

        Principal Angle = 11.23 degrees
```

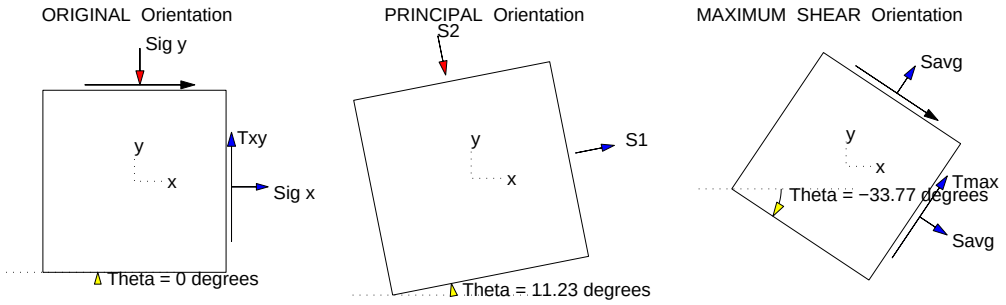


Figure 5.27: Example 2(b,d): Original, Principal and Maximum Shear Stresses.

5.12.2.3 Example 3:

Given: Let the state of stress at a point be given by $\sigma_{xx} = -120$ MPa, $\sigma_{yy} = 140$ MPa, $\sigma_{zz} = 66$ MPa, $\sigma_{xy} = 45$ MPa, $\sigma_{yz} = -65$ MPa, and $\sigma_{zx} = 25$ MPa.

Objective:

1. Determine the principal stresses.
2. Determine the maximum shear stress.
3. Plot Mohr's circle.

Results:

```
                    3-D MOHR'S CIRCLE DATA

Inputs:
```

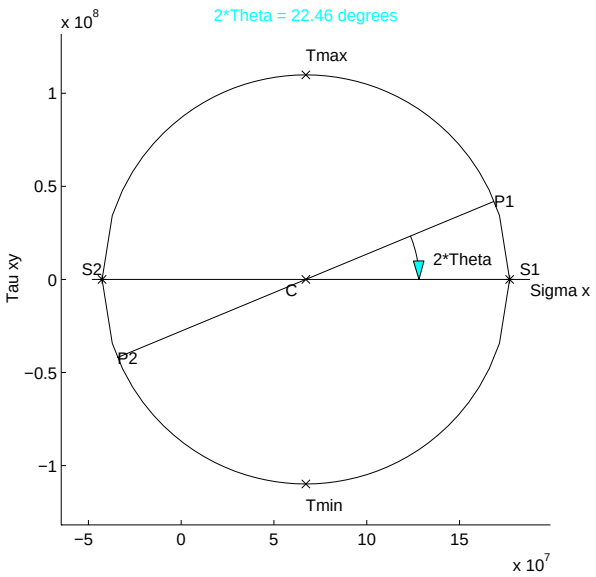


Figure 5.28: Example 2(e): Mohr's circle of 2-D stress.

```
        X       = -120
        Y       = 140
        Z       = 66
        XY      = 45
        XZ      = 25
        YZ      = -65

Principal Values:

  (a)   S1      = (180.207,0)
        S2      = (40.0573,0)
        S3      = (-134.264,0)

Maximum Shear Values:

        Tmax1   = (-47.1035,87.1608)
        Tmax2   = (22.9714,157.236)
        Tmax3   = (110.132,70.0748)

(b)  Tmax = 157.236
```

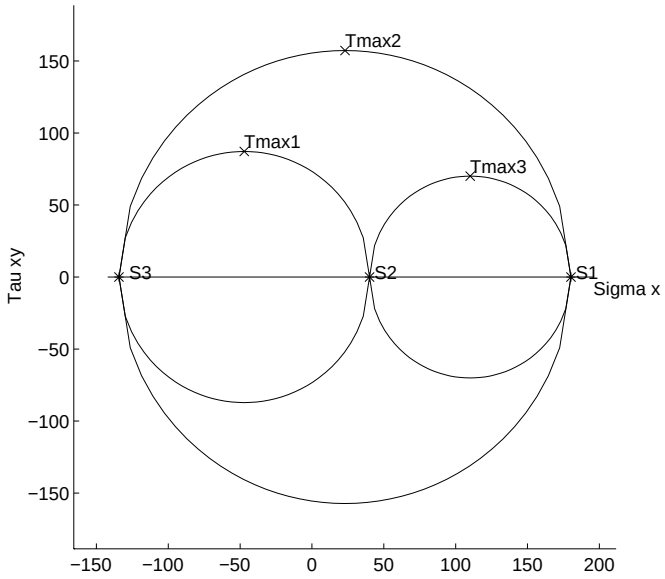


Figure 5.29: Example 3(c): Mohr's circle of 3-D stress.

Appendix A

INTRODUCTION TO UNIX

UNIX is a multi-user operating system that was invented by AT&T in 1969 and has grown to be the operating system of choice for universities and research organizations. This section will introduce the novice to some basic commands and also provide more advanced commands or tricks of the trade for those who are interested.

A.1 logging In

You may login to a Unix system in a number of way; **rlogin, telnet** from one computer (including pc) into the (Unix) computer.

In the Bechtel Lab., simply press return and the login prompt should appear. At the prompt, type in the `username` which was given to you, and then press **return**. Then the passwprd prompt will appear and you must *carefully* enter the password given to you. Note that as you type the password the characters will not appear on the screen. Note that Unix is case sensitive.

If this is the first time you are loggin in, then it may bee safe to change your password.

A.2 Changing Your Password

Usage: **passwd**

After you typed **passwd**, the system will prompt you for your *old* password (for security reason), and then will prompt twice (to make sure that there was no "typo error") for the new one.

The system will check after the first entry that the specified password can not be easily guessed by intruders (no dictionary name or permutations on your name allowed), that it has at least six characters, and that it contains at least one upper case or numeral or special character.

A.3 Logging Out

In general logging out is from a specific device.

If you are remotely connected to a computer, then simply type **exit**.

To logg out from the Bechtel Lab., then click the right button of the mouse on the screen and select `quit`. Once you are out, type contrl D. If you have followed properly the procedure, then a new login prompt should appear.

Note: If you are using a workstation, never turn it off when you have finished.

A.4 On-Line Help

A.4.1 man

Usage: **man** *command_name*
Man is the UNIX on-line help system. If you have questions about the appropriate syntax of a command, by typing in **man** *command*, you will get a complete description of the command and associated parameters. After the screen fills with text, at the bottom of the screen will be:

```
--More--(nn%)
```

This indicates how much of the file has been displayed. To continue to the next page hit the spacebar, to continue through line by line, press return. Unfortunately, because of the way the **man** system is set up you can not page back to previously displayed text, you can only page forward. If you are not to the end of the **man** file and wish to quit, **Ctrl C** will exit you from the help system. The following example first prints the manual page for ls to the screen, and the second line prints the manual page for **ls** to the printer.

```
> man ls
> man ls | enscript -2r -G
```

If you are not sure about a Unix command for which you are seeking help, then type

```
man -k copy
```

this will search the keyword copy in all the man files, and give you a list of the commands which man files contain the keyword `copy`.

A.4.2 Apropos

Usage: **apropos** *keyword*
apropos when used with a keyword, displays the man page name, section number, and a short description for each man page whose NAME line contains *keyword*. Words which are part of other words are considered; for example, when looking for 'compile', **apropos** will find all instances of 'compiler' as well. and **apropos** is not case sensitive.

A.5 UNIX File System

A.5.1 Directory Structure

UNIX uses a hierarchal directory structure to organize files. In this way user files can be kept separate from system files and other programs. The following figure summarizes the directory structure of the Bechtel lab.

Your user account is located under `/bechtel/users1/undergraduate` if you are an under graduate and `/bechtel/users1/graduate` if you are a graduate student.

Many local programs can be found in the /usr/local directory. Here is the hierarchy for local programs.

A.5.2 Your User Account

When you log into your user account you log into your own directory. The name of this directory will typically be you last name. This directory is called your home directory. Your working directory is whatever directory you are currently in, this will change many times during the average loggin session.

A.5.3 Absolute Pathnames

The *absolute pathname* tells you the path you must travel to get from the root directory to where you want to go. For example if Smith had a subdirectory called `temp`, the absolute pathname would be `/bechtel/users1/undergraduate/smith/temp`.

A.5.4 Relative Pathnames

A *relative pathname* points to a particular subdirectory, relative to your current directory. The shell assumes you are using relative pathnames unless you explicitly use an absolute pathname. If the user Smith were in his home directory the path to temp would be `temp`.

A.6 Shell

In Unix a **shell** is a command-line interpreter. You can choose the shell you want to use. The most popular ones being `/bin/csh` and `/bin/tcsh` or better known as C shell and turbo C shell.

A.7 File Management

A.7.1 Changing Directories

Usage: **cd** or **cd** *directory*

This command is used to change your working directory. If the directory you wish to change to is the next level down you would type **cd** *directoryname*. If the directory you wish to change to is several levels down you would type **cd** *pathname*. The *pathname* includes:

`/bechtel/users1/undergraduate/username/...`

A useful shortcut is to type:

`~username/...`

Where *username* is your user account name. When you type **cd** without a directory name, you will be placed in your home directory. Also typing **cd** (space) .. will move you up one directory, not to be confused with the MS DOS command **cd**.., you must remember to put a space between the **cd** and the ..

A.7.2 Changing File Protection

Usage: **chmod**

Since UNIX is intended for multiple users, protection rights for read, write and execute are assigned to each file. There are three groups to which protection may be separately assigned: **u**user (yourself), **g**roup (a group of users working together and who need to share files), and **o**ther (every one who has access to the same computer as you).

There are three types of protection: **r**ead, **w**rite, and **x** execute. Note that directories as well as executable files have their protection defined by **x**.

To add a permission, or to take it away, use + or −.

Examples:

- **chmod w+r myfile** gives the world read access to `myfile`
- **chmod g+x mydirectory** gives group read access to `mydierectory`

Alternatively, you may use `chmod` through the following command:

Usage : chmod mode filename

Modes : Modes can be absolute or symbolic. An absolute mode is an octal number constructed from one or more of the following modes. Following is the list of basic mode values:

400 Read by owner.
200 Write by owner.
100 Execute (search in directory) by owner.
040 Read by group.
020 Write by group.
010 Execute (search) by group.
004 Read by others.
002 Write by others.
001 Execute (search) by others.

For example, typing:

```
> chmod 666 junk.tex
```

would set the permission codes on the file "junk.tex", such that it could be read and written by owner, group, and world.

The symbolic mode has the form: [who] [operator] [permission], where **who** is a combination of :

- u User's permission
- g Group permission
- o Others

Operator is one of:

- \+ To add permission
- \- To remove permission
- = To assign the permission explicitly

Permission is any combination of:

- r Read.
- w Write.
- x Execute.
- X Give execute permission if the file is a directory or if there is execute permission for one of the other user classes.
- s Set owner or group ID. This is only useful with u or g. Also, the set group ID bit of a directory may only be modified with '+' or '-'.
- t Set the sticky bit to save program text between processes.

For example:

```
> chmod ugo=rw junk.tex
```

will set the permission such that the user, group, and others can read from and write to the file "junk.tex".

A.7.3 Copying Files

Usage: **cp** *file1 file2*
This command copies file1 to file2. If file2 already exists, it is replaced with file1. If file1 is in you current working directory but file2 is not, you must include the pathname for file2. The following example copies file1 to file2, which is in a different directory.

```
> cp file1 ~username/directory/file2
```

If you have many files you wished to copy, and they all have similar names i.e. file1, file2, file3, file4, ..., file10, you can use what's called a wildcard to copy all the files at once. This example puts all the files named file1 through file10 into the subdirectory called *directory*.

```
> cp file[1-10] ~username/directory
```

If the files are not named in such a way that permitts you to use the wildcard option, you can list out the names of the files you wish to copy. Here the file *sample* and *exmpl* are put into the subdirectory *temp*. If a destination for the copy is omitted the files will be put in your current working directory.

```
> cp ~username/direc/sample ~username/direc/exmpl ~username/temp
```

If you wish to copy the contents of one directory to another directory this can be done as follows:

```
> cp ~username/directory1 ~username/directory2
```

A.7.4 Directory Listing

Usage: **ls**
Lists all files in your current directory. For example if you would like a listing of all files with the extension .f in your current directory you could type:

```
> ls *.f
```

Another type of directory listing that will give you the permissions on the file, the owner, the file size, the last date modified, and file name is **ls** used with the **-l** option, this is the same as the **dir** command. For more information on the * see **wildcards**.

If you type **ls -F**, then you may get additional information regarding the nature of the files. Try it and guess what those are.

Trt also **ls -l**

A.7.5 Make Directory

Usage: **mkdir** *directory*
Makes a directory of the name specified. You may only create directories in your user account. To create subdirectories, change your current directory to the directory in which you wish to create a subdirectory, then type **mkdir** *directoryname*. Directories are useful for organizing your files. For example you might have a directory for each of the different classes you are using the computer system for, or you may want to create separate directories for your different term projects.

A.7.6 Moving Files

Usage: **mv** *oldname newname*
Mv is similar to the dos rename command. This command will move the file *oldname* to the file *newname*. It can also be used to move files from one directory to another. The syntax of **mv** is the same as that of the copy command. Be careful when using this command to move

files between directories, since it does not make a copy of the exsisting file but physically moves it from one directory to another, so if something goes wrong in the process your file is as good as deleted.

A.7.7 Removing Files

Usage: **rm** *file*
rm removes (deletes) a file from the system. It is a good idea to regularly go through all your files and eliminate all obsolete and unnecessary files to free up disk space. Wildcards can be used to delete files, for instance if you want to delete all your output files and they end with the extention .out you would type:

```
beethoven% rm *.out
```

Be careful when using the wildcard option, files that are removed are destroyed and can not be retrieved. It is a good idea to do an **ls** of the files you are going to delete. Use **rm** * with great care, otherwise you could end up deleting everything in your current directory.

A.7.8 Linking Files

Usage: **ln** *filename otherfile*
When you use the **cp** command, you create a copy of a file. Any new modifications made to either file does not affect the other. But, say you are working on a project with a partner, and you have a copy of the project in your directory, and your partner has a copy in his/her directory. You both want to be kept up to date on all changes that have been made to the project files. Well, instead of mailing new copies back and forth everytime someone makes a change, in UNIX you can *link* files. A feature of UNIX is that you can refer to a single file under different names in different directories. Thus, when you use the **ln** command you are not duplicating the file, just referencing the same file with different names. For example either name *filename* or *otherfile* can be used to call up the file, they are both valid names for the same file.

A.7.9 Removing Directories

Usage: **rmdir** *directory*
Rmdir removes (deletes) directories. But first the directory must be empty of all files. If you are certain you want to delete the directory first delete all files in the directory using **rm**, then delete the directory using the **rmdir** command.

```
> rm ~username/directory/temp/*
> rmdir ~username/directory/temp
```

A.7.10 Wildcards

Another function the shell provides is wildcard expansion. UNIX shells provide two special characters, * and ? as wildcards. The * will match any number of arbitrary characters, including none. The ? will match any single character. The shell looks for files that match the pattern typed and replaces the pattern with a list of those files that matched. It sends this list to the program. This process is called expanding the wildcards.

As an example, say you want to list all the files beginning with "a". You could type

```
ls a*
```

into the shell. All files of any length starting with "a" would be listed, including the file "a" if it existed. It would also match the 'file a.b since "." is not a special character in Unix as it is in DOS, for instance. Or, if you wanted to list all files of length two that start with "a" you would type

```
ls a?
```

You should be especially careful with wildcards when using rm until you are familiar with how they match. Use **ls** to see what files match before using **rm**.

A.8 Printing Files

For printing ASCII, or post-script files.

A.8.1 Enscript

Usage: **enscript** -2r -G*file*
When you are printing large source files on the laser printer, this command will rotate the output and place it in two columns. Also see **Print**.

A.8.2 Checking the Print Que

Usage: **lpq**
When you type the command to print a file and hit return the system prompt almost immediately returns. This does not mean that your file has been printed, all it means is that it has been spooled to a print que. If you would like to find out how busy the printer is, or how long the print que is, type **lpq**.

A.8.3 Laser Printer

Usage: **lpr** *file*
The command **lpr** will print a file on the laser printer. If you are printing source code or data files *please* see **Enscript** or **Print**.

If a file size exceeds 1 Mb, you should use **lpr -s fn**.

If you want to select the printer you should use **lpr -P**`printer`, where

lpr -Php1 Hp printer in the Bechtel Lab, single sided.

lpr -Php2 Hp printer in the Bechtel Lab, double sided.

lpr -Pivy1 Hp printer in the Leung Lab, single sided.

lpr -Pivy2 Hp printer in the Leung Lab, double sided.

lpr -Pnacps Hp printer in NAC room, on the second floor.

If you do not use the **lpr** command without the **-P**`printer` option, then the file is sent to the default printer. To determine what is the default printer, type **echo $PRINTER**.

A.8.4 Killing a Print Job

Usage: **lprm** *job_number* If you have started a job printing and then for some reason you wish to kill it, for example you have accidently printed source code using **lpr** instead of **print**. You can kill the print print job as follows: first check the print que to get the job number then use the **lprm** command.

```
> lpq
Rank    Owner      Job    Files                 Total Size
active  gosselin   350    sample.ps             9450 bytes
> lprm 350
```

A.8.5 More

Usage: **more** *file*
More will display a text file to the screen in pages. To advance a page, type the space bar. Unfortunately, because of the way the **man** system is set up you can not page back to previously displayed text, you can only page forward.At the botom of each page will appear:

```
--More--(nn%)
```

this indicates how much of the file has been displayed. If you are not to the end of the text file and wish to quit, **Ctrl C** will put you back at the system prompt.

A.8.6 Print

Usage: **print** *filename*
When printing out source code or data use the print command, like **enscript**, it rotates the output and forms two columns on a page, but it is a little easier to use than **enscript**. Most importantly, like **enscript**, the **print** command saves paper!

A.9 Compressing and Archiving Files

Often users will be asked to limit their disk usage to ten megabytes, and will be notified when their usage exceeds this amount. If you find yourself nearing this limit you can **compress** files that you are no longer using, or won't be using for a while. A text file can be compressed by up to 90%, executables, slightly less, this is a considerable space saving tool. Also, you can archive files directly to tape, freeing up diskspace, you can also use the archive tool to do backups of your files. Users will be allowed to use the cartrigde and 8mm tape drives on Bechtel, with permission from the system administrator, or they can do backups directly to a diskette in the floppy drive of each machine.

A.9.1 Zipping Files

Usage: **gzip** *filename*
The **gzip** command will generate a file named *filename* but will tack on the extension `.gz`. For example:

```
> gzip filename
```

the file *filename.gz* will be generated.

```
> gzip file1.f
```

the file *file1.f.gz* will be generated. To unzip files use the command **gunzip** in the same manner you use **gzip**. If we were to uncompress the FORTRAN source code we compressed in the previous example:

```
> gunzip file1.f.gz
```

this will restore *file1.f* to it's normal size and strip off the *.gz* extension.

A.9.2 Compressing Files

Usage: **compress** *filename*

Note that this utility is not as efficient as `gzip` but is supported on all Unix machines.

The **compress** command will generate a file named *filename* but will tack on the extension `.Z`. For example:

```
> compress filename
```

the file *filename.Z* will be generated.

```
> compress file1.f
```

the file *file1.f.Z* will be generated. To uncompress files use the command **uncompress** in the same manner you use **compress**. If we were to uncompress the FORTRAN source code we compressed in the previous example:

```
> uncompress file1.f.Z
```

this will restore *file1.f* to it's normal size and strip off the *.Z* extension.

A.9.3 Archiving Files

Usage: **tar** *-options newfilename oldfilename*
The **tar** command lets you archive files, or entire directories to tape, or diskette. To archive a file do the following:

```
> tar -cvf newfile oldfile
```

this will archive `oldfile` and call it `newfile.tar`. If you want to archive directly to tape you would replace `newfile` with the device name of the drive. Archiving a file to the floppy drive would be done as follows:

```
> tar -cvf /dev/rfd0 oldfile
```

this will put `oldfile.tar` in its current size directly to floppy disk. To archive something to 8mm or cartridge tape you must be rlogged in to Bechtel, since he owns these drives. The device names of these are:

8mm: /dev/rst0 - must be video quality tape.

Cartridge: /dev/rst1 - must be a 600 type tape to write, 300-type tapes are read-only.

If you wish to archive a directory in a compressed version. You need to archive it to a scratch directory first, then compress it, then do a disk-dump of that directory to tape.

```
> tar -cvf scratchdirectory olddirectory
> compress scratchdirectory.tar
> dd if=scratchdirectory.tar.Z of=devicename
```

To dump something from tape to disk:

```
> dd if=devicename of=destination
```

You can also list the contents of a tarfile

```
>tar -tfv filename.tar
```

If your archive is on a tape replace the filename with the device name. To extract archived files, it is done as follows:

```
> tar -xvf devicename
```

For more information on **tar** see the man page on **tar**.

A.9.4 Scratch Directories

If your disk requirement has exceeded the quota assigned to you, first zipp as many files as possible, then if you are still short of disk space yo can access one of the following scratch directories: `/bechtel/scratch1` and `/bechtel/scratch2`. For example

```
> cd /bechtel/scratch1
> mkdir clinton
> cd clinton
> mv ~clinton/whitewater/* .
> chmod 600 *
> cd ~
```

A.10 Spooling Files

A.10.1 Color Laser Printer

Color post-script files can be printed on the color laser printer on the NAC printer on the second floor of the Engineering building by typing

```
printcolor fn
```

where `fn` is your filename

A.11 Pipe and Filters

You may wish the output from one command, be the input for another. Connecting commands together in this fashion is known as a *pipe*. A pipe is designated by a vertical bar (|) on the command line between two commands. When a pipe is set up between two commands, the output from the command on the left becomes the input for the command on the right. Pipes can also be set up between programs.

When a program takes the output from another program as its input and does some sort of operation on it and writes out a new output file, this is refered to as a *filter*. A filter can be thought of as some sort of post-processor. For example, if we wished to take a listing of a directory and sort it by file size would could do this as follows:

```
> ls -l | sort +4n
```

The *pipe* consists of the use of both commands **ls** and **sort**. The *filter* is **sort**, since it is rearranging the output of **ls**. For more information on **ls** and **sort** please see Utilities.

A.12 Multi-Tasking

Suppose you have a large job to run, for example, you might be running a statistical package of some sort that will probably take a long time. If you were using an MS-DOS machine, like

a pc, you would start the job running and come back later when it finshed. This is called a single process system. UNIX, on the other hand, is a multi-process system, you can run jobs in the "background" while still doing things in the "foreground". You may even log off, and the background process will continue to run to completetion.
Note: it is unethical the submitt batch jobs on a different workstation than the one you are logged in on, this practice will not be tolerated. Under **no** circumstances are you to submitt a batch job to *Bechtel*.

A.12.1 Job Control

To start a job running in the background, or a batch job, use the & after a command. After you hit **return** the system will show you the *process ID number*, **PID**. The prompt will then return immediately. The **PID** is useful for keeping track of how long the job has been running, and you will also need it if you wish to kill the job. You dont't need to remember your process ID, however, since there is a simple UNIX command to check on what processes you have running, (see Checking on a Process). You can also bring the program back as the foreground job by using the **fg** function. Here is a sample session:

```
> stat_package &
[1] 29890
>
> fg (become interactive with a background process)
stat_package
```

A.12.2 Checking on a Process

If you wish to check the status of a process simply type in the command **ps**, this will give you a list of processes currently running on your machine, the elapsed time, and the command that started them. Even if you haven't started any batch jobs, when you type **ps** you will still get a list of processes, this is due to the windows environment.

A.12.3 Killing Processes

Sometimes a process gets out of control and refuses to quit on its own. To stop these processes you must use the kill command. To kill a process you will need its process number. This can be obained by using the command `ps` as shown:

```
> ps -gux
ER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
tomf     10561  0.0  0.0   96    0 co IW   Jan 17  0:00 -csh (csh)
tomf     24280  0.0  0.0   96    0 p0 IW   08:35   0:00 -bin/csh (csh)
tomf     24279  0.0  0.0  104    0 p1 IW   08:35   0:00 -bin/csh (csh)
tomf     24271  0.0  1.7  144  536 co S    08:35   0:01 olwm -3
```

```
tomf     24659  0.0  1.6  184  480 p3 R    09:54   0:00 /bin/ps -gux
tomf     24570  0.0  0.0 1488    0 p2 IW   09:22   0:02 xrn -calvin
tomf     24441  0.0  0.0  120    0 p2 IW   09:03   0:00 -sh (csh)
```

The process number is the second column. For example, if you wanted to kill the *xrn* process you would type:

```
> kill 24570
```

If that did not kill the process, then you will have to use:

```
> kill -9 24570
```

Note that you can kill a process from another machine, by *rlogging* on from another machine. This is useful if you have a program that is writing garbage to the screen, or has hung the screen, making it useless.

A.12.4 Prioritizing Processes

Usage: **nice** *-number command arguments*
If you start a process running in the background and do not prioritize it, you will be competing with that process for cpu time. By using a command called **nice**, you can proiritize the process, that is, as you are working at the console, the machine will devote more cpu time to what you are running in the foreground, but when the console is idle, the machine will devote more time to the background process. So, as you sit talking to a friend, or looking something up, the background process will get more of the cpu than it does when you are working.

You can set the **nice** value to anything between 0 and 19. The default value, if no number is given with the **nice** command, is 10. Process's set with 0 will run fast, process's set with a **nice** number of 19 will only run when nothing else wants to. If you do not use the **nice** command, a process's **nice** number will be 0. A user can only prioritize those process he has started, and can only increase the **nice** value i.e decrease a process's priority. In order to change the priorty of a process you must use the **renice** command. See the man page on **renice** for instructions on how to do this.

```
> nice -19 ps -l
```

this will set the nice number of a process to 19, then list all the process running on the machine (see **ps** for more information) and the **-l** option displays the **nice** numbers for the processes.

Note: if someone has started a process on your machine, notify the system administrator immediatley.

A.13 Utilities

A.13.1 du

Usage: **du** *options filename*
The **du** command displays your disk usage . It gives the number of kilobytes contained in all files and, recursively, subdirectories within the specified directory. If the filename is omitted, the current directory is used. The **-s** option will display the grand total for each of the specified filenames. The **-a** option will generate an entry for each file in the listed directory, and recursively go through the subdirectories, of the directory. So doing **du** in your home directory will list all the subdirectories and the total space they are using.

A.13.2 Find

Usage: **find** [pathname-list] [-options] expression
Used to find files by name, or by other characteristics. **Find** recursively descends the directory hierarchy for each pathname under the pathname-list, seeking files that match a given file name or other logical expression defined by any of the aviailable operators. For example, the following command will search all directories below the current directory for the files having the name "misc.f". The file path will be printed each time the file is found.

```
> find . -name misc.f -print
./Isds/Isds2/misc.f
```

The **find** command has several operators that can be used for defining search parameters. For instance, files can be searched for by, name of file, type of file, date accessed, date modified, files newer than arguement file, etc. For more information on the find command, type "man find" from a shell tool prompt.

A.13.3 Script

Usage: **script** *filename*
The command **script** will create a typescript file of a loggin session. Everything that is printed on your screen will be logged in this file. The file is written to `filename`, or it can be appended to an exsisting file by using the **-a** option as follows:

```
> script -a oldfile
```

If a file name is not specified the session will be logged in a file called `typescript`. The **print** command can be used to print the file out. The script file is ended when the forked shell exits. The way **script** has been written, an internal buffer of 8K or 16K needs to be filled before the information is dumped to a file. If you exit before this requirement is met, the information in the buffer will be lost.

A.13.4 Sort

Usage: **sort** *file*
Sort will alphabetically sort a file, by default, and print it to the screen. There are other options that will control what the file is sorted on:

-f Sort regardless of upper or lower case
-n Sort by arithmatic value, ignoring blanks and tabs
-r Reverse the order of the sort
+x Skip to the xth field and begin sort

A.13.5 Spell Checker

Usage: **spell** *file*
Spell is used to spell check a document. Any words that are not found in the dictionary are printed to the screen. Please consult the **man** pages on **spell** for additional information.

A.13.6 XtoPS

Usage: **XtoPS** *-options filename*
This command allows you to do a screen dump. It will take a picture of the entire screen, or a section defined by you, and store it in a PostScript format. You may use this file using **pageview**, or print it out.

```
> XtoPS -windows filename
```

will do an entire screen dump. If you want to print out only a specific window, you would type the following:

```
> XtoPS -windows windowID filename
```

To get the windowID, run **xwininfo** and then click on the window of interest. Another way to encapsulate a particular window is to use the **-screen** option.

```
> XtoPS -frame -screen filename
```

the **-frame** includes the frame around the window. After you hit return the mouse pointer will turn into cross hairs, move the mouse to the window you want, and click the left mouse button. The computer will beep twice, then it will return the system prompt when done creating the file.

A.13.7 Whereis

Usage: **whereis** [-options] filename ...
Whereis locates source, binary and manual page files for a command. This command searches for programs in a list of standard places. The available options are:

-b Search only for binaries.
-m Search only for manual sections.
-s Search only for sources.
-u Search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus 'whereis -m -u *' asks for those files in the current directory which have no documentation.

Other options are also available for changing or limiting directories in which to search. For more information type "man whereis" from a shell tool prompt.

A.13.8 Who

Usage: **who**
This function will show the login names of all users on the computer you are currently logged in on.

A.14 Shells and "dotfiles"

A shell is a program that serves as an interface between the user and the operating system. The shell is a command line interface as opposed to a graphical interface like the windows. That is, the user types a command line into the shell for execution. Although much work can be done solely through the graphical interface, some things can be done more efficiently with the command lines, and some things absolutely require the use of the command line.

A.14.1 *.cshrc* & *.login*

When a user first logs in, a shell called csh (C SHell) is started. This shell first reads the contents of the file *.cshrc* (C SHell Run Commands) in the users home directory and performs the commands in that file. This file is very similar to the *autoexec.bat* file in MS-DOS. After that is done, the shell reads the contents of the file *.login* in the home directory and performs the commands in that file. After the shell is done with the commands in the *.login* file, it is ready to read commands from the user. This is when the user generally starts openwindows.

Other shells are started for every cmdtool window that is started under openwindows. Each of these shells also reads the *.cshrc* file and performs the commands in it. These other shells do not read the *.login* file, only the first (login) shell does that. You can put commands you want every shell to perform in the *.cshrc* file. Such commands might include setting the prompt to something other than the default.

One of the abilities of the shell is to alias a command to another. For instance, say you normally run the command verylongprogramname and you get tired of typing it. You could alias it to something else like myprog. Then, you would just have to type myprog to start verlongprogramname. The format of the alias command is:

```
alias substitute_name 'original_program_name'
```

So for verlongprogramname, the alias would be

```
alias myprog 'verlongprogramname'
```

Another ability of the shell tool is to set a *path* to specific commands. If there is a command that you use, for example invoking a compiler, then you would want to set the path for that command using the **PATH**= command in your *.cshrc* file. This command is very similar to the PATH command found in an *autoexec.bat* file, in that it tells the shell what order to search through directories when looking for a command. For example:

```
set path=(/bechtel/tools/etc..)
```

The original program could include flags. For instance, many people like to get a long directory listing with ls -al. They alias this to something like *dir* with the command:

```
alias dir 'ls -al'
```

Since probably you want every shell to see this alias, you could put the command in the *.cshrc* file. From then on, you will be able to use dir in every shell you start, including cmdtools.

A.14.2 *.logout*

The last file shells look at is *.logout*. When you quit the login shell, it reads the contents of *.logout* and performs the commands in it. Since the loggin shell was the first shell, quiting it means logging off the system. Many people will display a message or clear the screen. Perhaps you'd like to display the contents of a file in which you've placed important reminders. In that case, you see these reminders every time you log out.

A.14.3 *.rhosts*

Another file you may be interested in is .rhosts. This file lists machines and logins which are trusted to use your account. For instance, in the lab, you can put each machine name into the *.rhosts* file. If you do this, you will not have to give a password when you rlogin to another machine in the Bechtel Lab. You must have this file set up if you want to use *rcp* to copy between machines. See the section on RCP for more information. Since all the suns in the bechtel lab share files, a *.rhosts* file on one machine applies to all machines. So, if you put spot in *.rhosts*, for example, you could log in from spot to any machine in the lab without giving a password or use *rcp* (remote copy) from spot to any machine.

Another use is to allow another person to use your account. Be warned however, that if you do this, that person effectively becomes you while they are logged in. They can do anything you can do including deleting files, *rlogin* to other machines where you have a *.rhosts* set up and not give a password, send mail that appears to have come from you, etc. The advantage of this method to giving out your password is that that person cannot change your password. In order to change a password, you need the old password. Since they don't need the password to use your account, don't give it to them. They won't be able to change it then. They also

won't be able to give it out to others. Still, since this forgoes security for the sake of convience, make sure you trust the other person explicitly.

The format of the *.rhosts* file is as follows: *machine.name login*. The machine name needs to include the domain, *.colorado.edu* for all university machines. For other machines, it would be something different. The login only needs to be given if it is different from the login that the file is under. For instance, in the lab, part of the *.rhosts* might look like this:

```
grieg.colorado.edu
beethoven.colorado.edu
spot.colorado.edu
spot.colorado.edu buddy
```

Assuming your login is "mylogin", this file allows the login "mylogin" to login to any machines in the lab from grieg, beethoven, and spot without giving a password. Generally, you would list all the machines in the bechtel lab. There is a program called *rhosts.setup* which will create a *.rhosts* file for you and put each of the lab machines in it. If you want to add other machines like spot or tramp, you will have to edit *.rhosts* by hand.

The example above also allows "buddy" to log into any of the machines in the lab using your account from spot. buddy would log in with the command `rlogin -l mylogin bechtel`. The *-l* means that "buddy" wants to be "mylogin" on bechtel instead of "buddy". This format is also useful if you have different login names on different machines.

A.14.4 *.signature*

This file is used when you post news (see the section on news for information on how to get started with news). When you post an article on news, this file is appended to your post like a signature. Generally people will put their real name and e-mail address into the *.signature* file so others reading news will know who they really are and how to reach them. Some people al from spotso include a humorous or thoughful saying that they identify with. People reading news will often remember a saying better than a name. However, don't make the *.signature* file too long. Generally, four lines or less is always acceptable. As you increase the size of the file, you increase the risk of people sending you nasty e-mail saying they don't like waiting for your *.signature* to be sent over their slow modem. A page long *.signature*, no matter how cute it may be, will definitely bring about such mail.

A.14.5 *.mailrc*

This file will contain various setups for your mail. One usefull application is the definition of aliases to shorten the length of e-mail addresses. For example, by having the following inside your `.mailrc` file:

```
alias bicanic bicanic@cvvaxa.swan.ac.uk
```

you simply have to address your e-mail to `bicanic` rather than to: `bicanic@cvvaxa.swan.ac.uk`

A.15 Further Reference

You can find more information on any of the topics discussed in this section in the following:

The UNIX Programming Environment by Brian W. Kernighan and Rob Pike. Prentice Hall Software Series.

Learning the UNIX Operating System by Grace Todino and John Strang. O'Reilly and Associates, Inc.

And any of the numerous reference guides and user's manuals available in the lab.

A.16 Remote Loggin; Telnet

Telnet is your key to accessing applications on Internet host computers located around the world. In a TCP/IP environment telnet is used to login to any computer on the network. Your first connection to a UNIX computer is made using telnet unless you login via a directly connected serial terminal. So what is telnet? It is terminal emulation program with a limited set of commands for you to remember. You might be surprised to learn that telnet has no way to transfer files from machine to machine. This is very different from serial telecommunications programs that you may be familiar with. Let's look at these commands.¡p¿

The telnet client must be told the hostname or the IP address of the host to which you want to connect. The machine on which the telnet client is started is the "local" machine. The host to which the client connects is known as the "remote" machine.

Once connected, you must provide a valid username and password. These may be specifically for you if you have an account on the remote machine. Otherwise, the username and password may be known to several users (or everyone); there may be no password required. In this case, the account is probably set up for a specific purpose. A program is usually executed automatically (such as gopher or lynx), or the user is given a menu of choices when he or she logs in.

Having successfully logged in, your local machine is functioning as a terminal of the remote.

When logging in, make sure you are emulating the same type of terminal that the remote thinks you are emulating. Some common terminal types are: DEC vt100 family (vt100, vt102, vt220), ansi, and IBM 3270. The terminal type is set in the terminal emulation software (the telnet software) on the local machine. The remote may or may not prompt you to enter the terminal type after you login. Sometimes the remote gets a signal from the telnet client letting it know what type of terminal is in use. Some sites expect that only certain terminal types will be connecting to them and only function properly for that type.

A.16.1 Telnet Commands

open establishes a connection to the specified host.

close closes an open connection and leaves you in telnet

quit closes any open telnet sessions and exits the program

set echo toggles screen echo on and off between full and half duplex

CTRL-] this key sequence puts you in telnet's command mode

There are two methods for using telnet at the Unix prompt. First, you can just start the program, then issue the open command to get you connected to the desired host. Second, you can specify the desired host on the command line when you start telnet. This has the same effect as starting telnet then using the open command. It is a form of a short cut in that you are typing only one command. You are free to use telnet either way. You should also be aware that telnet uses a standard UNIX port to answer connections.

A.17 File Transfer Protocol (ftp)

Since telnet does not have the ability to transfer files, the program ftp becomes very important. Ftp will allow you to transfer files between two computers systems. Additionally, one of the vast resources on the Internet is public domain software. In order for you to get to this software you will use a special instance of ftp known as anonymous ftp. This allows you to transfer files between two computers without having an account on the remote computer system.

A.17.1 Definitions

FTP: The Internet standard file transfer protocol. An ftp program is a user interface to this protocol which allows the transfer to and from a remote network host.

FTP Client: The local program which allows connection to an FTP server for the purpose of sending or receiving files.

FTP Server: An Internet host which allows clients to connect and transfer files to or from the hosts.

Anonymous FTP: Use of ftp to transfer files to or from a server which allows anyone to connect. The FTP server allows a user login ID of "anonymous" or "guest" and accepts any password input. The accepted courtesy on the Internet is for the use to provide his or her Internet email address.

ASCII File: Text files are files that you can display on your screen and/or pull into a text editor. For example, AUTOEXEC.BAT and CONFIG.SYS are text files in DOS. Files you would usually transfer in ASCII mode include: "readme" files, source code, uuencoded, binhex, PostScript and HTML files. Files transferred in ASCII mode are "translated" so that they appear to be the same on the local machine as the remote. Text is stored differently on different types of computers. PC text is represented differently than Mac text. Neither PC nor Mac text is represented the same as Unix text. Those files are usually ASCII files: Text (.txt, tex, .asc); Source code (.c, .f, .for); e-mail messages; uuencoded files; PostScript (.ps); Hypertext (.htm, .html).

Binary File: Binary files are not text files. This includes many file types such as: executables, images, sound, movies, spreadsheets, databases, word processing, and compressed files. Files transferred in binary mode are copied exactly, bit-for-bit.

Those files are usually binary: Spreadsheets (.xls, ...); Databases (.dbf, .dbt); Word Processing (.doc, wp); Compressed files (.Z, .zip); Images (.gif, .tif, .tiff, .jpg, .jpeg, bmp, pcx); Sounds (.au); Movies (.mpg, .mpeg, .mov).

Download: An FTP client receiving a file from an FTP server using the "get" command is downloading the file.

Upload: An FTP client sending a file to an FTP server using the "put" command is uploading the file.

A.17.2 Connecting to an FTP Server

To establish a connection with a remote host (FTP server) from the command line, simply type "ftp <host name>". Once the connection is made, the remote host will require the user enter a valid username and password. To establish an anonymous ftp session, the username is anonymous and the password is your full internet email address.

A.17.3 Basic Commands

The words with the "less-than" (<) and "greater-than" (>) signs are not literal, but indicate a real, situtationally-dependent word should be substituted. The information contained within the brackets ("[" and "]") are not required.

cd <directory>: Change current directory to the indicated directory. Similar to DOS, there are several unique inputs for <directory>. They are "/" (go to the top most directory) and ".." (go to the next higher directory).

dir [<**directory**> :] List extended information for the contents of the indicated directory. If no directory is given, the extended information of the current directory is listed.

ls [<**directory**> :] List the contents of the indicated directory. If no directory is given, the current directory is listed.

type <transfer type>: Set the transfer type to the indicated type. The transfer type should be either ASCII or binary as previously defined.

get <remote file> [<**local file**> :] Requests the FTP server send the file indicated and save it on the local client with the name provided. If no <local file> is provided, the file is saved using the <remote file> name.

put <local file> [<**remote file**> :] Sends the indicated file to be saved on the FTP server with the name provided. If no <remote file> is provided, the file is saved using the <local file> name.

mget <file pattern>: Related to the "get" command, this retrieves multiple files matching the indicated pattern. Using the "*" as a wildcard character, the command "mget *.txt" would retrieve all files with the extension ".txt" in the current remote directory.

mput <file pattern>: Related to the "put" command, this sends multiple files matching the indicated pattern. Using the "*" as a wildcard character, the command "mput *.txt" would put all files with the extension ".txt" in the current local directory to the FTP server.

delete <remote file>: Deletes the indicated file from the FTP server. Most FTP clients allow use of "del" for the entire command "delete"

hash: This is a "toggle" command. When on, it causes a "#" to be printed to the screen for each block (1024, 2048, or 4096 bytes) of the file transferred. It is used to indicated continuing activity during transfer of large files. Some client software require the commands "hash on" and "hash off" to be used, although most simply toggle between the on and off modes.

bye: Close this ftp session. "quit" also works on most FTP clients.

Bibliography

[1] A. Biran and M. Breiner. *MATLAB for Engineers*. Addisoon-Wesley, 1995.

[2] D.M. Etter. *Engineering Problem Solving with MATLAB*. Prentice-Hall, 1993.