Introduction to Matlab - A Quick Tutorial [1][2]

Arvind Gopu [3]

Tuesday, Nov 25 2003

[1]Examples in this presentation are from the Matlab Help Section!
[2]Presentation to Bioinformatics Research Group
[3]Author Contact Info: http://peart.ucs.indiana.edu/contact.html

Contents

0.1 Introduction

MATLAB is a high-performance language for technical computing [1]. It integrates computation, visualization, and programming in an easy-to-use environment. Typical uses include:

- Mathematical processing
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics

... and much much more ...

0.1.1 Getting Matlab up and running

On any Unix-flavor machine (Linux, Solaris, etc), just typing matlab should start the Matlab program (if it's installed obviously!). For example I've shown below how I boot up Matlab on kitefin.cs.indiana.edu, a Sun Ultra 10 which sits in my old office in Lindley Hall (IU-CS).

```
ag@kitefin~/research/matlab_ppt\% matlab &
[1] 25439
ag@kitefin~/research/matlab_ppt%
```

The Matlab interface looks something like this, but for a few things which might look different in different OSs and such.

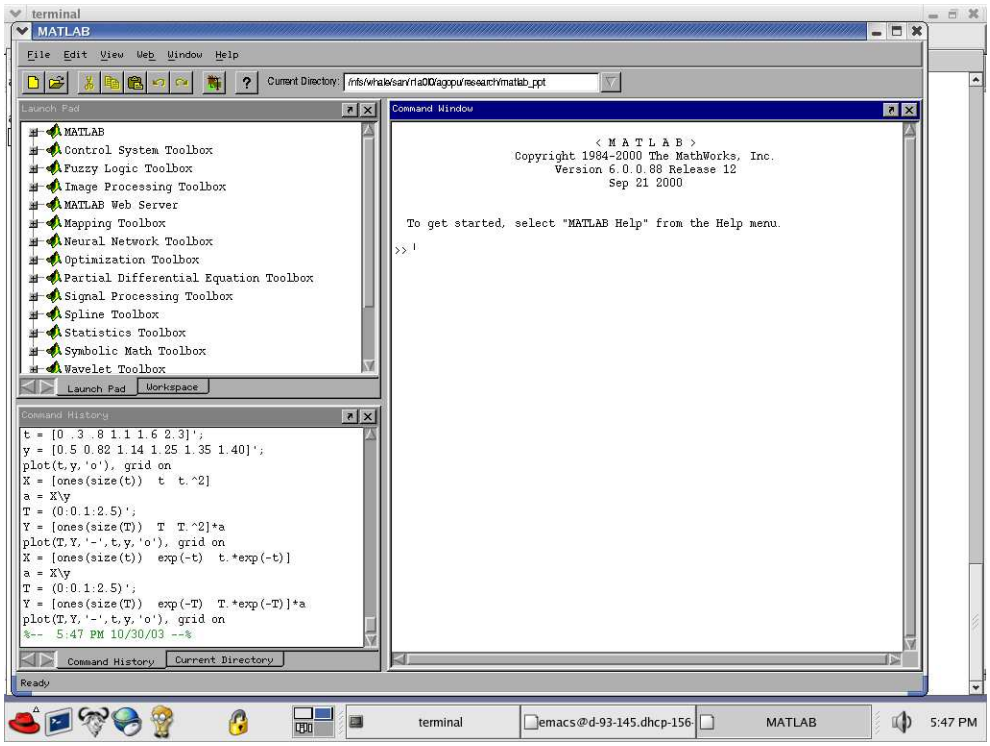

Figure 1: Matlab Interface (console)

On a Windows machine, you will need to find out where Matlab is installed. Doing a "Search for files" on Matlab*.* should give the information needed.

On the Windows XP boxes in I109, Matlab can be brought up using the Start menu shortcut

```
Start -> Programs -> Departmentally Sponsored -> Mathematics -> Matlab
```

(Or something like that, I can't seem to remember the exact shortcut!). I would expect it to be installed on the shared R drive.

Note that Matlab's licence is quite expensive (yeah it's NOT free). So don't expect it to be installed on all machines that strike your eye. For example it's not available on the Linux boxes in I105 here at Informatics. It's only available in the Windows machines in the general purpose lab I109, thanks to its association with UITS.

If you wanna bring up Matlab Help, do:

```
'Help' menu -> 'Matlab Help'
```

On any version/flavor you should see something like this:

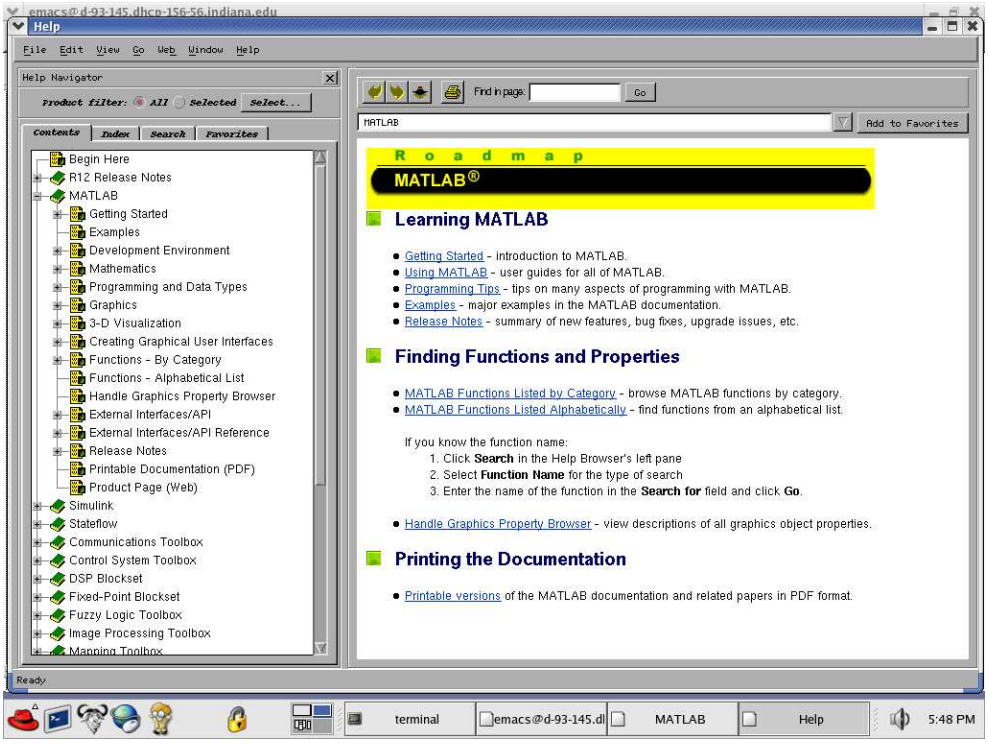

Figure 2: Matlab Help

0.1.2 Going beyond this tutorial

I have just tried to present some of the things Matlab can do for you in this presentation. If you are interested in exploring more funcionality that it provides (and there's ample amounts of that!), then I'd suggest you first look at the

Matlab documentation [1] before trying other resources like books and such. I find it to be an excellent source of information with nice descriptive examples. I usually use the Matlab available with the Sun systems on the CS network here at IU. But I am sure the Windows version of Matlab which is available at Informatics will suffice in a pinch.

0.2 Matrix Manipulation

How many times have you felt really irritated when you had to process matrices as 2-d arrays or the like. Worse to add two matrices, you needed a double for loop and even worse, to multiply two matrices you needed a triple for loop [2] like this:

```
  for (i=0; i < n; i++) {
    for (j=0; j < n; j++) {
      for (k=0; k < n; k++) {
C[i][j] += A[i][k] * B[k][j];
      }
    }
  }
```

Now would it not be nice to do something like use a simple '*' operator to multiply two matrices? That's exactly the kinda functionality Matlab provides for matrix manipulation. Computing inverse, transpose, eigen values of matrices is all easy to do using Matlab functions. The following two subsections explain some of these functions.

0.2.1 Simple Matrix Processing

The following code shows how you can:

- Define vectors and manipulate them
- Create matrices
- Perform Add, Subtract, Multiply operations on matrices

A discussion of this code follows...

```
% Hints: use ; at end of line to prevent debug'ish output from being printed

% Vector Product -- needs to be col * row or vice versa
u = [3; 1; 4]        % column vector   <-- note: vectors are defined here
v = [2 0 -1]         % Row vector
w = v'               % Transpose of 'v'

x = v*u             % Will return a scalar
```

```
X = u*v             % Will return a vector
% X = u*w;          % Will raise and error


% Matrices
A = pascal(3)
B = magic(3)
C = [
1 2 3 4
4 5 6 7
7 8 9 0
]

% Add A and B
X = A + B

% Subtract B from A
Y = A - B

% Multiply A and B (not equal to B * A). Must satisfy dim reqs
Z = A * B

% Multiply by scalar
s = 7;
Z = A*s

% To show how error messages are raised if dims dont match
C = fix(10*rand(3,2))
% X = A + C
```

Before going on to processing matrices, I thought I'll show you how Matlab handles vectors (1-D arrays!). You can define a column vector or row vector. In the former case you will need to use a ";" in between elements. In both cases you enclose the elements in "[....]". A trailing single-quote (" ' ") creates the transpose of a vector. This is especially useful when you want to multiply two vectors since you can only mutiply a dimensionally corresponding column vector to row vector or vice versa - whether you get back a scalar or a vector will depend on this order.

To define test matrices you can use predefined functions like pascal(int) or magic(int) to create some funky matrix – constructs a magic square kinda matrix so that each row/column adds up to a certain value [1]. The 'int' parameter is used to specify the dimension. Use two integers if you want non-square matrix.

In reality you will usually have a table of data points which you want to translate into a matrix. It's quite straight forward to do. It's just like defining

[1] Thanks to Scott Martin for pointing this out!

a few row vectors. Have each row's data comma/space separated and then a return char (newline) demarcates two rows.

Following the definition of matrices A, B and C, I have shown simple operations like Add, Subtract and multiply using the '+', '-' and the '*' operators. It is obvious how easy it is to play around with matrices in Matlab. I have also shown one instance in which the dimensions don't match up for the specific operation in which case Matlab will raise an error (The last operation in which I've tried to add a 3 x 3matrix to a 3 x 2 matrix).

```
u =

     3
     1
     4

v =
     2     0    -1

w =

     2
     0
    -1

x =
     2

X =
     6     0    -3
     2     0    -1
     8     0    -4

A =

     1     1     1
     1     2     3
     1     3     6

B =

     8     1     6
     3     5     7
     4     9     2

C =
     1     2     3     4
```

```
     4     5     6     7
     7     8     9     0

X =
     9     2     7
     4     7    10
     5    12     8

Y =
    -7     0    -5
    -2    -3    -4
    -3    -6     4

Z =
    15    15    15
    26    38    26
    41    70    39

Z =
     7     7     7
     7    14    21
     7    21    42

C =
     4     6
     8     8
     8     6
```

0.2.2 Inverses, Powers and such

The following code shows how you can:

- Create an identity matrix
- Compute determinant, inverse of a matrix
- Compute powers of a matrix

A discussion of this code follows...

```
% Hints: use ; at end of line to prevent debug'ish output from being printed

% Matrices
A = pascal(3)
B = magic(3)

% Identity Matrices
```

```
eye(3,4);           % Returns a 3 x 4 identity matrix

% Determinant -- WARNING: Round off error
d = det(A)

% Inverse of Matrix
X = inv(A)
d = det(A)         % A is symmetric and has integer values -- same det


X = A^2        % Computes matrix squared
X = A.^2       % Squares each element

Y = B^(-3)

sqrt(A)        % Element-based sqrt

sqrtm(A)       % Matrix-based sqrt
```

We start with the usual test-matrix definitions. Then I have shown how you can create an identity matrix using the eye(int, int) function. This is pretty handy when you want to do scalar-vector or scalar-matrix conversions.

Then I have shown the use of det(matrix) function to compute the determinant and the use of inv(matrix) function to compute its inverse. Now that's cool, I think because usually to compute matrix inverses you end up doing a lot of processing.

Following that I have shown the use of the '$\hat{}$' operator to compute powers of a matrix. Note that '$\hat{2}$' (for example) is used to compute the squared-matrix whereas '.$\hat{2}$' is used to calculate the sqaure of each individual element. Negative powers can also be computed as shown.

Another useful function is the squareroot-computing function. Again it has two variants depending on whether we want to handle individual matrix elements or the matrix as a whole.

```
A =
     1     1     1
     1     2     3
     1     3     6

B =
     8     1     6
     3     5     7
     4     9     2

d =
```

```
     1

X =
     3    -3     1
    -3     5    -2
     1    -2     1

d =
     1

X =
     3     6    10
     6    14    25
    10    25    46

X =
     1     1     1
     1     4     9
     1     9    36

Y =
    0.0053   -0.0068    0.0018
   -0.0034    0.0001    0.0036
   -0.0016    0.0070   -0.0051

ans =
    1.0000    1.0000    1.0000
    1.0000    1.4142    1.7321
    1.0000    1.7321    2.4495

ans =

    0.8775    0.4387    0.1937
    0.4387    1.0099    0.8874
    0.1937    0.8874    2.2749
```

0.3 Data Analysis

In this section I will show you how you can:

- Do statistical analysis (for example compute mean, standard deviation and stuff like that)
- Fit curves
- Do a moving average filter

- Perform Fourier Transform

Quite obviously this is only part of the stuff you can do in terms of analysing data points. There are ways to work with partial differential equations, sparse matrices, etc. But we will not go into such detail. There's always the documentation [1] (or other reference books) you can fall back on!

0.3.1 Simple Statistical Functions - Mean, S.D., Covariance, Difference

One of the first things you need to do while working with large number of data points is to define the data else where i.e., in a different file. This not only helps in having cleaner and more readable code but also modularizes the whole process! Here we show how you can store tabular data to be used as a matrix (first code listing) and also how you can load that kinda data (First line in the second code listing).

A brief discussion of what's being done follows the code listing...

```
% File: data_for_stat.dat
1 1 1
1 2 1
1 2 2
1 2 5
2 1 4
```

```
% Hint: use ; at end of line to prevent debug'ish output from being printed

load data_for_stat.dat

% Number of rows and columns
[n, p] = size(data_for_stat);
t = 1:n;                % Where you might use such information


% Really simple stat functions

% Max/Min -- Returns a row of max/min elements
X = max(data_for_stat)
X = min(data_for_stat)

% Max/Min -- One value from entire dataset
%   data_for_stat(:) rearranges the m x n  matrix into a mn x 1 column vector.
X = max(data_for_stat(:))
```

```
% Mean and Standard Deviation
mu = mean(data_for_stat)
sigma = std(data_for_stat)

% Further???
% For example if you wanted (x-mu) for each element, say to compute a normal distr
unity = ones(n,1)          %% <---- Note Use of ones( , )
x = data_for_stat - unity * mu


% Correlation and Covariance

covar = cov(data_for_stat(:,1))
corr  = corrcoef(data_for_stat)
```

As mentioned earlier we load some data from a file called "data_for_stat.dat". Note that once a .dat file is loaded, we drop the extension (.dat) in future references and use "data_for_stat" just like we would do in the case of a variable.

The size() function returns the number of rows and columns present in the data. (Consider the data points to form rows and columns of a matrix).

Then we have shown some basic functions like Min(), Max() and variants in their usage. Just using the data file name (yeah without the extension) returns a row of min/max values. If we need the min/max value in the entire data set then we need to use data_for_stat(:) meaning all rows and columns. In general (int:int) corresponds to rows:columns. If either integer is left blank then it means "all" in the corresponding dimension.

After that we have shown how the functions mean(), std(), covar() and corrcoef() can be used to compute mean, standard deviation, covariance and the correlation coefficient respectively.

A couple of points to be noted:

- The use of data_for_stat(int:int) is again shown in a couple of places
- A function ones(rows, columns) has been used – it generates a matrix with ones in it. Umm just in case you are thinking, how is this different from an identity matrix? :) Think again!
- I have also shown the use of multiplying a scalar to a vector to generate a new vector (which is weighted according to the mean in this case)

```
X =
     2     2     5

X =
     1     1     1
```

```
X =
     5

mu =
    1.2000    1.6000    2.6000

sigma =
    0.4472    0.5477    1.8166

unity =
     1
     1
     1
     1
     1

x =
   -0.2000   -0.6000   -1.6000
   -0.2000    0.4000   -1.6000
   -0.2000    0.4000   -0.6000
   -0.2000    0.4000    2.4000
    0.8000   -0.6000    1.4000

covar =
    0.2000

corr =

    1.0000   -0.6124    0.4308
   -0.6124    1.0000    0.0503
    0.4308    0.0503    1.0000
```

0.3.2 More Statistical Functions - Difference

In this section I've shown the use of various difference functions in Matlab. A discussion follows the source code.

```
% Hint: use ; at end of line to prevent debug'ish output from being printed

load data_for_stat.dat

% Finite Differences
% Simple Difference
A = [9 -2 3 0 1 5 4];
diff(A)                  % Produces difference between successive elements
```

```
% Gradient
v = -2:0.2:2
[x,y] = meshgrid(v)
z = x .* exp(-x.^2 - y.^2)
[px,py] = gradient(z,.2,.2)
%contour(v,v,z), hold on, quiver(v,v,px,py), hold off

% Discrete Laplacian -- See discussion for more information
[x,y] = meshgrid(-4:4,-3:3)
U = x.*x+y.*y
V = 4*del2(U)
```

First up I've shown the use of the *diff()* function to do a forward difference computation. If X is a vector, then *diff(X)* returns a vector, one element shorter than X, of differences between adjacent elements:

$$Diff(X) = [X(2) - X(1)X(3) - X(2)...X(n) - X(n-1)]$$

Some typical uses of the *diff()* function to check for conditions like monotonicity, etc.

Tests	Description
diff(x)==0	Test for repeated elements
all(diff(x)¿0)	Test for monotonicity
all(diff(diff(x))==0)	Test for equally spaced vector elements

The function *gradient()* gives the numerical partial derivatives a matrix which for *F(x,y)* (2 variables) is equal to:

$$\nabla F = \frac{\partial F}{\partial x} i + \frac{\partial F}{\partial y} j$$

We can decide how many dimensions need to be taken into account when computing this. For example: *FX = gradient(F)* where F is a vector: Returns the one-dimensional numerical gradient of F. FX corresponds to $\frac{\partial F}{\partial x}$, the differences in the x (column) direction whereas...

[FX,FY] = gradient(F) where F is a matrix: Returns the x and y components of the two-dimensional numerical gradient. FX corresponds to $\frac{\partial F}{\partial x}$, the differences in the x (column) direction. FY corresponds to $\frac{\partial F}{\partial Y}$, the differences in the y (row) direction. The spacing between points in each direction is assumed to be one. And so on so forth... For more information refer [1] and [3]

The function *del2()* computes the discrete laplacian for a given data set. If the matrix U is regarded as a function *u(x,y)* evaluated at the point on a square grid, then *4 * del2(U)* is a finite difference approximation of Laplace's differential operator applied to u, that is:

$$l = \frac{\nabla^2 u}{4}$$

$$l = \frac{1}{4}\left(\frac{d^2u}{dx^2} + \frac{d^2u}{dy^2}\right)$$

where in the interior:

$$l_{ij} = \frac{1}{4}\left(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}\right) - u_i$$

For functions of more variables than 'x' and 'y', and more information about the Laplacian operator in different coordinate systems refer [1] and [4].

This is the output you can expect to see when you run the above shown .m file.

```
>> more_complex_stat

ans =

   -11     5    -3     1     4    -1

v =

  Columns 1 through 7

   -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000

  Columns 8 through 14

   -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000

  Columns 15 through 21

    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000

x =

  Columns 1 through 7

   -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
   -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
   -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
   -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
   -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
   -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
   -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
   -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
   -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
   -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
   -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
```

```
  -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
  -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
  -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
  -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
  -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
  -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
  -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
  -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
  -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000
  -2.0000   -1.8000   -1.6000   -1.4000   -1.2000   -1.0000   -0.8000

 Columns 8 through 14

  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000
  -0.6000   -0.4000   -0.2000         0    0.2000    0.4000    0.6000

 Columns 15 through 21

   0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
   0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
   0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
   0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
   0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
   0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
   0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
   0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
   0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
```

```
    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
    0.8000    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000

y =

  Columns 1 through 7

   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000
   -1.8000   -1.8000   -1.8000   -1.8000   -1.8000   -1.8000   -1.8000
   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000
   -1.4000   -1.4000   -1.4000   -1.4000   -1.4000   -1.4000   -1.4000
   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000
   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000
   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000
   -0.6000   -0.6000   -0.6000   -0.6000   -0.6000   -0.6000   -0.6000
   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000
   -0.2000   -0.2000   -0.2000   -0.2000   -0.2000   -0.2000   -0.2000
         0         0         0         0         0         0         0
    0.2000    0.2000    0.2000    0.2000    0.2000    0.2000    0.2000
    0.4000    0.4000    0.4000    0.4000    0.4000    0.4000    0.4000
    0.6000    0.6000    0.6000    0.6000    0.6000    0.6000    0.6000
    0.8000    0.8000    0.8000    0.8000    0.8000    0.8000    0.8000
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
    1.2000    1.2000    1.2000    1.2000    1.2000    1.2000    1.2000
    1.4000    1.4000    1.4000    1.4000    1.4000    1.4000    1.4000
    1.6000    1.6000    1.6000    1.6000    1.6000    1.6000    1.6000
    1.8000    1.8000    1.8000    1.8000    1.8000    1.8000    1.8000
    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000

  Columns 8 through 14

   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000
   -1.8000   -1.8000   -1.8000   -1.8000   -1.8000   -1.8000   -1.8000
   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000
   -1.4000   -1.4000   -1.4000   -1.4000   -1.4000   -1.4000   -1.4000
   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000
```

```
   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000
   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000
   -0.6000   -0.6000   -0.6000   -0.6000   -0.6000   -0.6000   -0.6000
   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000
   -0.2000   -0.2000   -0.2000   -0.2000   -0.2000   -0.2000   -0.2000
         0         0         0         0         0         0         0
    0.2000    0.2000    0.2000    0.2000    0.2000    0.2000    0.2000
    0.4000    0.4000    0.4000    0.4000    0.4000    0.4000    0.4000
    0.6000    0.6000    0.6000    0.6000    0.6000    0.6000    0.6000
    0.8000    0.8000    0.8000    0.8000    0.8000    0.8000    0.8000
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
    1.2000    1.2000    1.2000    1.2000    1.2000    1.2000    1.2000
    1.4000    1.4000    1.4000    1.4000    1.4000    1.4000    1.4000
    1.6000    1.6000    1.6000    1.6000    1.6000    1.6000    1.6000
    1.8000    1.8000    1.8000    1.8000    1.8000    1.8000    1.8000
    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000

  Columns 15 through 21

   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000   -2.0000
   -1.8000   -1.8000   -1.8000   -1.8000   -1.8000   -1.8000   -1.8000
   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000   -1.6000
   -1.4000   -1.4000   -1.4000   -1.4000   -1.4000   -1.4000   -1.4000
   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000   -1.2000
   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000
   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000   -0.8000
   -0.6000   -0.6000   -0.6000   -0.6000   -0.6000   -0.6000   -0.6000
   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000   -0.4000
   -0.2000   -0.2000   -0.2000   -0.2000   -0.2000   -0.2000   -0.2000
         0         0         0         0         0         0         0
    0.2000    0.2000    0.2000    0.2000    0.2000    0.2000    0.2000
    0.4000    0.4000    0.4000    0.4000    0.4000    0.4000    0.4000
    0.6000    0.6000    0.6000    0.6000    0.6000    0.6000    0.6000
    0.8000    0.8000    0.8000    0.8000    0.8000    0.8000    0.8000
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
    1.2000    1.2000    1.2000    1.2000    1.2000    1.2000    1.2000
    1.4000    1.4000    1.4000    1.4000    1.4000    1.4000    1.4000
    1.6000    1.6000    1.6000    1.6000    1.6000    1.6000    1.6000
    1.8000    1.8000    1.8000    1.8000    1.8000    1.8000    1.8000
    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000    2.0000

z =

  Columns 1 through 7

   -0.0007   -0.0013   -0.0023   -0.0036   -0.0052   -0.0067   -0.0077
```

```
  -0.0014   -0.0028   -0.0048   -0.0077   -0.0111   -0.0144   -0.0165
  -0.0028   -0.0054   -0.0096   -0.0152   -0.0220   -0.0284   -0.0326
  -0.0052   -0.0099   -0.0174   -0.0278   -0.0400   -0.0518   -0.0594
  -0.0087   -0.0167   -0.0293   -0.0467   -0.0674   -0.0872   -0.0999
  -0.0135   -0.0259   -0.0455   -0.0725   -0.1046   -0.1353   -0.1552
  -0.0193   -0.0372   -0.0652   -0.1040   -0.1499   -0.1940   -0.2224
  -0.0256   -0.0492   -0.0863   -0.1376   -0.1984   -0.2567   -0.2943
  -0.0312   -0.0601   -0.1054   -0.1680   -0.2423   -0.3135   -0.3595
  -0.0352   -0.0677   -0.1188   -0.1895   -0.2732   -0.3535   -0.4053
  -0.0366   -0.0705   -0.1237   -0.1972   -0.2843   -0.3679   -0.4218
  -0.0352   -0.0677   -0.1188   -0.1895   -0.2732   -0.3535   -0.4053
  -0.0312   -0.0601   -0.1054   -0.1680   -0.2423   -0.3135   -0.3595
  -0.0256   -0.0492   -0.0863   -0.1376   -0.1984   -0.2567   -0.2943
  -0.0193   -0.0372   -0.0652   -0.1040   -0.1499   -0.1940   -0.2224
  -0.0135   -0.0259   -0.0455   -0.0725   -0.1046   -0.1353   -0.1552
  -0.0087   -0.0167   -0.0293   -0.0467   -0.0674   -0.0872   -0.0999
  -0.0052   -0.0099   -0.0174   -0.0278   -0.0400   -0.0518   -0.0594
  -0.0028   -0.0054   -0.0096   -0.0152   -0.0220   -0.0284   -0.0326
  -0.0014   -0.0028   -0.0048   -0.0077   -0.0111   -0.0144   -0.0165
  -0.0007   -0.0013   -0.0023   -0.0036   -0.0052   -0.0067   -0.0077

Columns 8 through 14

  -0.0077   -0.0062   -0.0035         0    0.0035    0.0062    0.0077
  -0.0164   -0.0133   -0.0075         0    0.0075    0.0133    0.0164
  -0.0324   -0.0263   -0.0149         0    0.0149    0.0263    0.0324
  -0.0590   -0.0480   -0.0271         0    0.0271    0.0480    0.0590
  -0.0992   -0.0808   -0.0455         0    0.0455    0.0808    0.0992
  -0.1540   -0.1254   -0.0707         0    0.0707    0.1254    0.1540
  -0.2207   -0.1797   -0.1013         0    0.1013    0.1797    0.2207
  -0.2921   -0.2378   -0.1341         0    0.1341    0.2378    0.2921
  -0.3567   -0.2905   -0.1637         0    0.1637    0.2905    0.3567
  -0.4022   -0.3275   -0.1846         0    0.1846    0.3275    0.4022
  -0.4186   -0.3409   -0.1922         0    0.1922    0.3409    0.4186
  -0.4022   -0.3275   -0.1846         0    0.1846    0.3275    0.4022
  -0.3567   -0.2905   -0.1637         0    0.1637    0.2905    0.3567
  -0.2921   -0.2378   -0.1341         0    0.1341    0.2378    0.2921
  -0.2207   -0.1797   -0.1013         0    0.1013    0.1797    0.2207
  -0.1540   -0.1254   -0.0707         0    0.0707    0.1254    0.1540
  -0.0992   -0.0808   -0.0455         0    0.0455    0.0808    0.0992
  -0.0590   -0.0480   -0.0271         0    0.0271    0.0480    0.0590
  -0.0324   -0.0263   -0.0149         0    0.0149    0.0263    0.0324
  -0.0164   -0.0133   -0.0075         0    0.0075    0.0133    0.0164
  -0.0077   -0.0062   -0.0035         0    0.0035    0.0062    0.0077

Columns 15 through 21
```

```
    0.0077    0.0067    0.0052    0.0036    0.0023    0.0013    0.0007
    0.0165    0.0144    0.0111    0.0077    0.0048    0.0028    0.0014
    0.0326    0.0284    0.0220    0.0152    0.0096    0.0054    0.0028
    0.0594    0.0518    0.0400    0.0278    0.0174    0.0099    0.0052
    0.0999    0.0872    0.0674    0.0467    0.0293    0.0167    0.0087
    0.1552    0.1353    0.1046    0.0725    0.0455    0.0259    0.0135
    0.2224    0.1940    0.1499    0.1040    0.0652    0.0372    0.0193
    0.2943    0.2567    0.1984    0.1376    0.0863    0.0492    0.0256
    0.3595    0.3135    0.2423    0.1680    0.1054    0.0601    0.0312
    0.4053    0.3535    0.2732    0.1895    0.1188    0.0677    0.0352
    0.4218    0.3679    0.2843    0.1972    0.1237    0.0705    0.0366
    0.4053    0.3535    0.2732    0.1895    0.1188    0.0677    0.0352
    0.3595    0.3135    0.2423    0.1680    0.1054    0.0601    0.0312
    0.2943    0.2567    0.1984    0.1376    0.0863    0.0492    0.0256
    0.2224    0.1940    0.1499    0.1040    0.0652    0.0372    0.0193
    0.1552    0.1353    0.1046    0.0725    0.0455    0.0259    0.0135
    0.0999    0.0872    0.0674    0.0467    0.0293    0.0167    0.0087
    0.0594    0.0518    0.0400    0.0278    0.0174    0.0099    0.0052
    0.0326    0.0284    0.0220    0.0152    0.0096    0.0054    0.0028
    0.0165    0.0144    0.0111    0.0077    0.0048    0.0028    0.0014
    0.0077    0.0067    0.0052    0.0036    0.0023    0.0013    0.0007

px =

  Columns 1 through 7

   -0.0031   -0.0040   -0.0058   -0.0074   -0.0078   -0.0063   -0.0023
   -0.0066   -0.0085   -0.0124   -0.0157   -0.0167   -0.0135   -0.0050
   -0.0131   -0.0168   -0.0245   -0.0310   -0.0330   -0.0266   -0.0098
   -0.0238   -0.0307   -0.0446   -0.0566   -0.0601   -0.0484   -0.0179
   -0.0401   -0.0516   -0.0751   -0.0951   -0.1011   -0.0815   -0.0300
   -0.0623   -0.0801   -0.1165   -0.1477   -0.1570   -0.1265   -0.0467
   -0.0893   -0.1148   -0.1670   -0.2117   -0.2250   -0.1813   -0.0669
   -0.1181   -0.1518   -0.2210   -0.2802   -0.2977   -0.2399   -0.0885
   -0.1443   -0.1855   -0.2699   -0.3422   -0.3636   -0.2930   -0.1081
   -0.1627   -0.2091   -0.3043   -0.3858   -0.4100   -0.3303   -0.1218
   -0.1693   -0.2176   -0.3168   -0.4016   -0.4267   -0.3438   -0.1268
   -0.1627   -0.2091   -0.3043   -0.3858   -0.4100   -0.3303   -0.1218
   -0.1443   -0.1855   -0.2699   -0.3422   -0.3636   -0.2930   -0.1081
   -0.1181   -0.1518   -0.2210   -0.2802   -0.2977   -0.2399   -0.0885
   -0.0893   -0.1148   -0.1670   -0.2117   -0.2250   -0.1813   -0.0669
   -0.0623   -0.0801   -0.1165   -0.1477   -0.1570   -0.1265   -0.0467
   -0.0401   -0.0516   -0.0751   -0.0951   -0.1011   -0.0815   -0.0300
   -0.0238   -0.0307   -0.0446   -0.0566   -0.0601   -0.0484   -0.0179
   -0.0131   -0.0168   -0.0245   -0.0310   -0.0330   -0.0266   -0.0098
```

```
  -0.0066   -0.0085   -0.0124   -0.0157   -0.0167   -0.0135   -0.0050
  -0.0031   -0.0040   -0.0058   -0.0074   -0.0078   -0.0063   -0.0023

Columns 8 through 14

   0.0037    0.0104    0.0156    0.0176    0.0156    0.0104    0.0037
   0.0079    0.0222    0.0334    0.0376    0.0334    0.0222    0.0079
   0.0156    0.0438    0.0659    0.0743    0.0659    0.0438    0.0156
   0.0285    0.0797    0.1200    0.1353    0.1200    0.0797    0.0285
   0.0480    0.1341    0.2019    0.2276    0.2019    0.1341    0.0480
   0.0745    0.2083    0.3135    0.3535    0.3135    0.2083    0.0745
   0.1067    0.2985    0.4493    0.5066    0.4493    0.2985    0.1067
   0.1412    0.3950    0.5945    0.6703    0.5945    0.3950    0.1412
   0.1725    0.4824    0.7261    0.8187    0.7261    0.4824    0.1725
   0.1945    0.5439    0.8187    0.9231    0.8187    0.5439    0.1945
   0.2024    0.5661    0.8521    0.9608    0.8521    0.5661    0.2024
   0.1945    0.5439    0.8187    0.9231    0.8187    0.5439    0.1945
   0.1725    0.4824    0.7261    0.8187    0.7261    0.4824    0.1725
   0.1412    0.3950    0.5945    0.6703    0.5945    0.3950    0.1412
   0.1067    0.2985    0.4493    0.5066    0.4493    0.2985    0.1067
   0.0745    0.2083    0.3135    0.3535    0.3135    0.2083    0.0745
   0.0480    0.1341    0.2019    0.2276    0.2019    0.1341    0.0480
   0.0285    0.0797    0.1200    0.1353    0.1200    0.0797    0.0285
   0.0156    0.0438    0.0659    0.0743    0.0659    0.0438    0.0156
   0.0079    0.0222    0.0334    0.0376    0.0334    0.0222    0.0079
   0.0037    0.0104    0.0156    0.0176    0.0156    0.0104    0.0037

Columns 15 through 21

  -0.0023   -0.0063   -0.0078   -0.0074   -0.0058   -0.0040   -0.0031
  -0.0050   -0.0135   -0.0167   -0.0157   -0.0124   -0.0085   -0.0066
  -0.0098   -0.0266   -0.0330   -0.0310   -0.0245   -0.0168   -0.0131
  -0.0179   -0.0484   -0.0601   -0.0566   -0.0446   -0.0307   -0.0238
  -0.0300   -0.0815   -0.1011   -0.0951   -0.0751   -0.0516   -0.0401
  -0.0467   -0.1265   -0.1570   -0.1477   -0.1165   -0.0801   -0.0623
  -0.0669   -0.1813   -0.2250   -0.2117   -0.1670   -0.1148   -0.0893
  -0.0885   -0.2399   -0.2977   -0.2802   -0.2210   -0.1518   -0.1181
  -0.1081   -0.2930   -0.3636   -0.3422   -0.2699   -0.1855   -0.1443
  -0.1218   -0.3303   -0.4100   -0.3858   -0.3043   -0.2091   -0.1627
  -0.1268   -0.3438   -0.4267   -0.4016   -0.3168   -0.2176   -0.1693
  -0.1218   -0.3303   -0.4100   -0.3858   -0.3043   -0.2091   -0.1627
  -0.1081   -0.2930   -0.3636   -0.3422   -0.2699   -0.1855   -0.1443
  -0.0885   -0.2399   -0.2977   -0.2802   -0.2210   -0.1518   -0.1181
  -0.0669   -0.1813   -0.2250   -0.2117   -0.1670   -0.1148   -0.0893
  -0.0467   -0.1265   -0.1570   -0.1477   -0.1165   -0.0801   -0.0623
  -0.0300   -0.0815   -0.1011   -0.0951   -0.0751   -0.0516   -0.0401
```

```
   -0.0179   -0.0484   -0.0601   -0.0566   -0.0446   -0.0307   -0.0238
   -0.0098   -0.0266   -0.0330   -0.0310   -0.0245   -0.0168   -0.0131
   -0.0050   -0.0135   -0.0167   -0.0157   -0.0124   -0.0085   -0.0066
   -0.0023   -0.0063   -0.0078   -0.0074   -0.0058   -0.0040   -0.0031

py =

  Columns 1 through 7

   -0.0038   -0.0073   -0.0129   -0.0206   -0.0296   -0.0383   -0.0440
   -0.0054   -0.0104   -0.0182   -0.0291   -0.0419   -0.0543   -0.0622
   -0.0093   -0.0179   -0.0314   -0.0501   -0.0723   -0.0935   -0.1072
   -0.0146   -0.0281   -0.0494   -0.0787   -0.1135   -0.1468   -0.1683
   -0.0208   -0.0400   -0.0702   -0.1119   -0.1614   -0.2088   -0.2394
   -0.0266   -0.0512   -0.0898   -0.1432   -0.2064   -0.2670   -0.3062
   -0.0302   -0.0581   -0.1020   -0.1626   -0.2344   -0.3033   -0.3478
   -0.0297   -0.0573   -0.1005   -0.1602   -0.2309   -0.2988   -0.3426
   -0.0241   -0.0464   -0.0814   -0.1297   -0.1870   -0.2420   -0.2775
   -0.0135   -0.0261   -0.0457   -0.0729   -0.1051   -0.1360   -0.1559
         0         0         0         0         0         0         0
    0.0135    0.0261    0.0457    0.0729    0.1051    0.1360    0.1559
    0.0241    0.0464    0.0814    0.1297    0.1870    0.2420    0.2775
    0.0297    0.0573    0.1005    0.1602    0.2309    0.2988    0.3426
    0.0302    0.0581    0.1020    0.1626    0.2344    0.3033    0.3478
    0.0266    0.0512    0.0898    0.1432    0.2064    0.2670    0.3062
    0.0208    0.0400    0.0702    0.1119    0.1614    0.2088    0.2394
    0.0146    0.0281    0.0494    0.0787    0.1135    0.1468    0.1683
    0.0093    0.0179    0.0314    0.0501    0.0723    0.0935    0.1072
    0.0054    0.0104    0.0182    0.0291    0.0419    0.0543    0.0622
    0.0038    0.0073    0.0129    0.0206    0.0296    0.0383    0.0440

  Columns 8 through 14

   -0.0436   -0.0355   -0.0200         0    0.0200    0.0355    0.0436
   -0.0617   -0.0503   -0.0283         0    0.0283    0.0503    0.0617
   -0.1064   -0.0867   -0.0489         0    0.0489    0.0867    0.1064
   -0.1670   -0.1360   -0.0767         0    0.0767    0.1360    0.1670
   -0.2376   -0.1935   -0.1091         0    0.1091    0.1935    0.2376
   -0.3039   -0.2474   -0.1395         0    0.1395    0.2474    0.3039
   -0.3451   -0.2810   -0.1584         0    0.1584    0.2810    0.3451
   -0.3400   -0.2768   -0.1561         0    0.1561    0.2768    0.3400
   -0.2754   -0.2242   -0.1264         0    0.1264    0.2242    0.2754
   -0.1547   -0.1260   -0.0710         0    0.0710    0.1260    0.1547
         0         0         0         0         0         0         0
    0.1547    0.1260    0.0710         0   -0.0710   -0.1260   -0.1547
    0.2754    0.2242    0.1264         0   -0.1264   -0.2242   -0.2754
```

```
    0.3400    0.2768    0.1561         0   -0.1561   -0.2768   -0.3400
    0.3451    0.2810    0.1584         0   -0.1584   -0.2810   -0.3451
    0.3039    0.2474    0.1395         0   -0.1395   -0.2474   -0.3039
    0.2376    0.1935    0.1091         0   -0.1091   -0.1935   -0.2376
    0.1670    0.1360    0.0767         0   -0.0767   -0.1360   -0.1670
    0.1064    0.0867    0.0489         0   -0.0489   -0.0867   -0.1064
    0.0617    0.0503    0.0283         0   -0.0283   -0.0503   -0.0617
    0.0436    0.0355    0.0200         0   -0.0200   -0.0355   -0.0436

  Columns 15 through 21

    0.0440    0.0383    0.0296    0.0206    0.0129    0.0073    0.0038
    0.0622    0.0543    0.0419    0.0291    0.0182    0.0104    0.0054
    0.1072    0.0935    0.0723    0.0501    0.0314    0.0179    0.0093
    0.1683    0.1468    0.1135    0.0787    0.0494    0.0281    0.0146
    0.2394    0.2088    0.1614    0.1119    0.0702    0.0400    0.0208
    0.3062    0.2670    0.2064    0.1432    0.0898    0.0512    0.0266
    0.3478    0.3033    0.2344    0.1626    0.1020    0.0581    0.0302
    0.3426    0.2988    0.2309    0.1602    0.1005    0.0573    0.0297
    0.2775    0.2420    0.1870    0.1297    0.0814    0.0464    0.0241
    0.1559    0.1360    0.1051    0.0729    0.0457    0.0261    0.0135
         0         0         0         0         0         0         0
   -0.1559   -0.1360   -0.1051   -0.0729   -0.0457   -0.0261   -0.0135
   -0.2775   -0.2420   -0.1870   -0.1297   -0.0814   -0.0464   -0.0241
   -0.3426   -0.2988   -0.2309   -0.1602   -0.1005   -0.0573   -0.0297
   -0.3478   -0.3033   -0.2344   -0.1626   -0.1020   -0.0581   -0.0302
   -0.3062   -0.2670   -0.2064   -0.1432   -0.0898   -0.0512   -0.0266
   -0.2394   -0.2088   -0.1614   -0.1119   -0.0702   -0.0400   -0.0208
   -0.1683   -0.1468   -0.1135   -0.0787   -0.0494   -0.0281   -0.0146
   -0.1072   -0.0935   -0.0723   -0.0501   -0.0314   -0.0179   -0.0093
   -0.0622   -0.0543   -0.0419   -0.0291   -0.0182   -0.0104   -0.0054
   -0.0440   -0.0383   -0.0296   -0.0206   -0.0129   -0.0073   -0.0038

x =

    -4    -3    -2    -1     0     1     2     3     4
    -4    -3    -2    -1     0     1     2     3     4
    -4    -3    -2    -1     0     1     2     3     4
    -4    -3    -2    -1     0     1     2     3     4
    -4    -3    -2    -1     0     1     2     3     4
    -4    -3    -2    -1     0     1     2     3     4
    -4    -3    -2    -1     0     1     2     3     4

y =

    -3    -3    -3    -3    -3    -3    -3    -3    -3
```

```
    -2    -2    -2    -2    -2    -2    -2    -2    -2
    -1    -1    -1    -1    -1    -1    -1    -1    -1
     0     0     0     0     0     0     0     0     0
     1     1     1     1     1     1     1     1     1
     2     2     2     2     2     2     2     2     2
     3     3     3     3     3     3     3     3     3

U =

    25    18    13    10     9    10    13    18    25
    20    13     8     5     4     5     8    13    20
    17    10     5     2     1     2     5    10    17
    16     9     4     1     0     1     4     9    16
    17    10     5     2     1     2     5    10    17
    20    13     8     5     4     5     8    13    20
    25    18    13    10     9    10    13    18    25

V =

     4     4     4     4     4     4     4     4     4
     4     4     4     4     4     4     4     4     4
     4     4     4     4     4     4     4     4     4
     4     4     4     4     4     4     4     4     4
     4     4     4     4     4     4     4     4     4
     4     4     4     4     4     4     4     4     4
     4     4     4     4     4     4     4     4     4
```

0.3.3 Curve Fitting (Regression)

I will show two examples of fitting a curve for a given set of data points - Polynomian Regression and Linear in the Parameter Regression. A discussion follows the source code snippet.

```
% Hints: use ; at end of line to prevent debug'ish output from being printed

% Initial plot of just the data points
t = [0 .3 .8 1.1 1.6 2.3]';
y = [0.5 0.82 1.14 1.25 1.35 1.40]';
plot(t,y,'o'), grid on

X = [ones(size(t))  t  t.^2]
a = X\y

% Fit curve of the data points
T = (0:0.1:2.5)';
```

```
Y = [ones(size(T))  T  T.^2]*a
plot(T,Y,'-',t,y,'o'), grid on

%% Increase degree of polynomial to increase accuracy of fit OR use linear in parameter

X = [ones(size(t))  exp(-t)  t.*exp(-t)]
a = X\y
T = (0:0.1:2.5)';
Y = [ones(size(T))  exp(-T)  T.*exp(-T)]*a
plot(T,Y,'-',t,y,'o'), grid on
```

In the polynomian regression code snippet, we've tried to fit an equation of the form:

$$y = a_0 + a_1 t + a_2 t^2$$

a_0, a_1 and a_2 are found using the backslash operator. It is evident that we do not get that good a fit, curve misses most of the data points by a considerable margin. The accuracy can be increased by using a higher order polynomial. Or the next method could be used...

In the second example shown, instead of a polynomial function, I've used a function that is linear-in-the-parameters - an exponential function of the form:

$$y = a_0 + a_1 \exp -t + a_2 t \exp -t$$

Again a_0, a_1 and a_2 are found using the backslash operator. But we get a much better fit with this method. Refer the Matlab documentation [1] for more information on curve fitting, they have some nice examples!

This is the output you can expect to see when you run the above shown .m file.

```
X =

    1.0000         0         0
    1.0000    0.3000    0.0900
    1.0000    0.8000    0.6400
    1.0000    1.1000    1.2100
    1.0000    1.6000    2.5600
    1.0000    2.3000    5.2900

a =

    0.5318
    0.9191
   -0.2387

Y =
```

```
    0.5318
    0.6213
    0.7060
    0.7860
    0.8612
    0.9316
    0.9973
    1.0582
    1.1143
    1.1656
    1.2121
    1.2539
    1.2909
    1.3231
    1.3506
    1.3733
    1.3912
    1.4043
    1.4127
    1.4163
    1.4151
    1.4091
    1.3984
    1.3829
    1.3626
    1.3375

X =

    1.0000    1.0000         0
    1.0000    0.7408    0.2222
    1.0000    0.4493    0.3595
    1.0000    0.3329    0.3662
    1.0000    0.2019    0.3230
    1.0000    0.1003    0.2306

a =

    1.3974
   -0.8988
    0.4097

Y =
    0.4986
    0.6212
```

```
    0.7286
    0.8226
    0.9047
    0.9764
    1.0390
    1.0934
    1.1408
    1.1818
    1.2174
    1.2482
    1.2747
    1.2976
    1.3172
    1.3339
    1.3482
    1.3604
    1.3707
    1.3793
    1.3866
    1.3926
    1.3976
    1.4017
    1.4050
    1.4077
```

0.3.4 Difference Equations & Filters

Matlab allows us to implement filters often on data points in vectors form. In this example I've shown how we could implement a moving average filter for a sample dataset (indicating traffic conditions at various times at 3 locations). The data itself is represented in the following format: each row of data corresponds to a time with the 3 columns indicating traffic at the 3 locations.

```
% File: count.dat
    11    11     9
     7    13    11
     ...
    10     9     7


% Hints: use ; at end of line to prevent debug'ish output from being printed

load count.dat
a = 1
b = [1/4 1/4 1/4 1/4]        % vector for 4 hour moving average
```

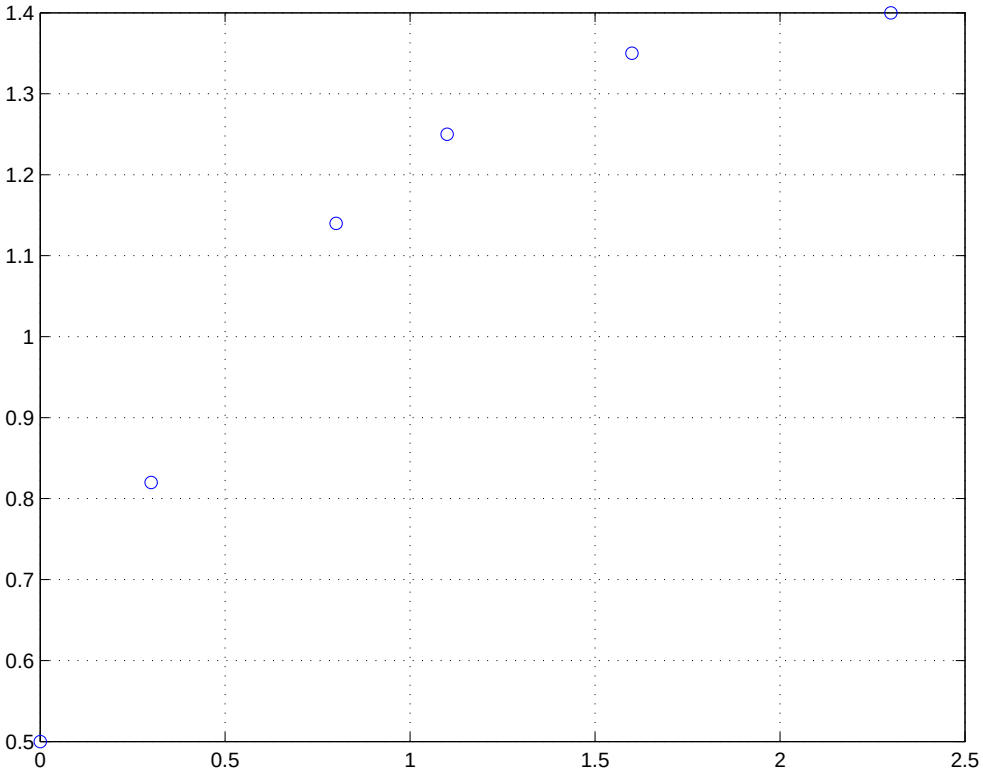

Figure 3: Plot of data points

```
% -- essentially a fourth of the current and past 3 inputs

x = count(:,1)
y = filter(b,a,x)            % Does the averaging

t = 1:length(x)
plot(t,x,'-.',t,y,'-'), grid on
legend('Original Data','Smoothed Data',2)
```

The filter command can be thought of as an implementation of the difference equation the output y(n) of which is a linear combination of current and previous inputs, x(n) x(n-1) ..., and previous outputs, y(n-1) y(n-2) ...

$$a(1)y(n) = (b(1)x(n)+b(2)x(n-1)+...+b(nb)x(n-nb+1))-(a(2)y(n-1)-...-a(na)y(n-na+1))$$

The filter structure shown above is the general tapped delay-line filter described by the difference equation below, where n is the index of the current sample, na is the order of the polynomial described by vector a and nb is the

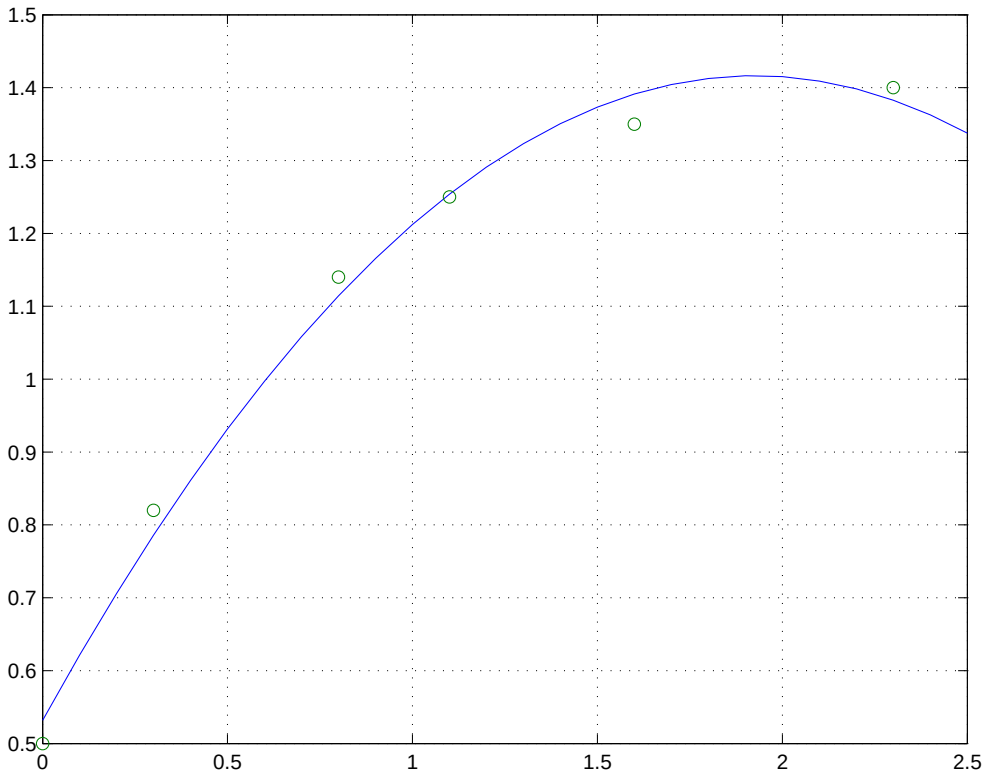

Figure 4: Case 1: Polynomial Regression (Quite Inaccurate)

order of the polynomial described by vector b.In our example, vectors 'a' and 'b' correspond to values '1' and [1/4, 1/4, 1/4, 1/4]. So we get:

$$y(n) = \frac{1}{4}\left(x(n) + x(n-1) + x(n-2)\right) + x(n-3))$$

The corresponding bit of code in Matlab becomes: filter(b, a, x). A plot shows how the filter smooths out the data over a 4-hour period.

```
a =
     1

b =
    0.2500    0.2500    0.2500    0.2500

x =
    11
     7
    14
```

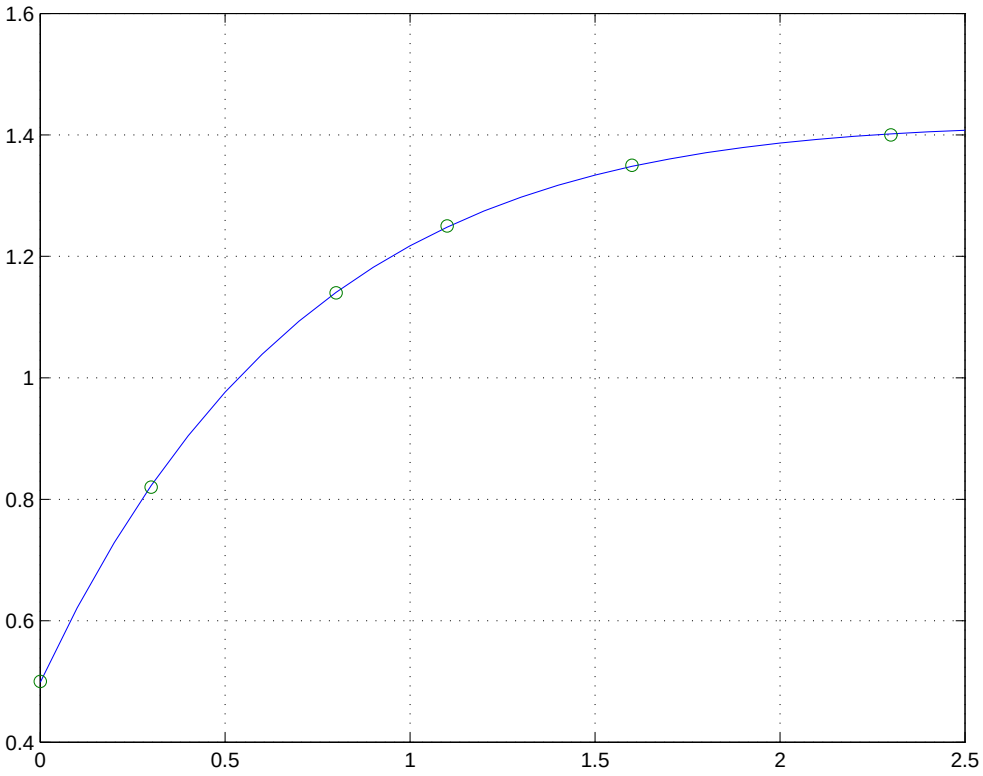

Figure 5: Case 2: Linear in Parameters Regression (More Accurate)

11
43
38
61
75
38
28
12
18
18
17
19
32
42
57
44

```
   114
    35
    11
    13
    10

y =
    2.7500
    4.5000
    8.0000
   10.7500
   18.7500
   26.5000
   38.2500
   54.2500
   53.0000
   50.5000
   38.2500
   24.0000
   19.0000
   16.2500
   18.0000
   21.5000
   27.5000
   37.5000
   43.7500
   64.2500
   62.5000
   51.0000
   43.2500
   17.2500

t =
  Columns 1 through 12
     1     2     3     4     5     6     7     8     9    10    11    12
  Columns 13 through 24
    13    14    15    16    17    18    19    20    21    22    23    24
```

0.3.5 Fourier Transforms

Fourier transforms are used in numerous scientific applications. Let me give you a bit of background on what it is all about.

For any one-dimensional function (usually in time) f(x), the fourier transform is defined as:

$$F(u) = \int_{-\infty}^{\infty} f(x) \exp(-t2\pi ux) dx$$

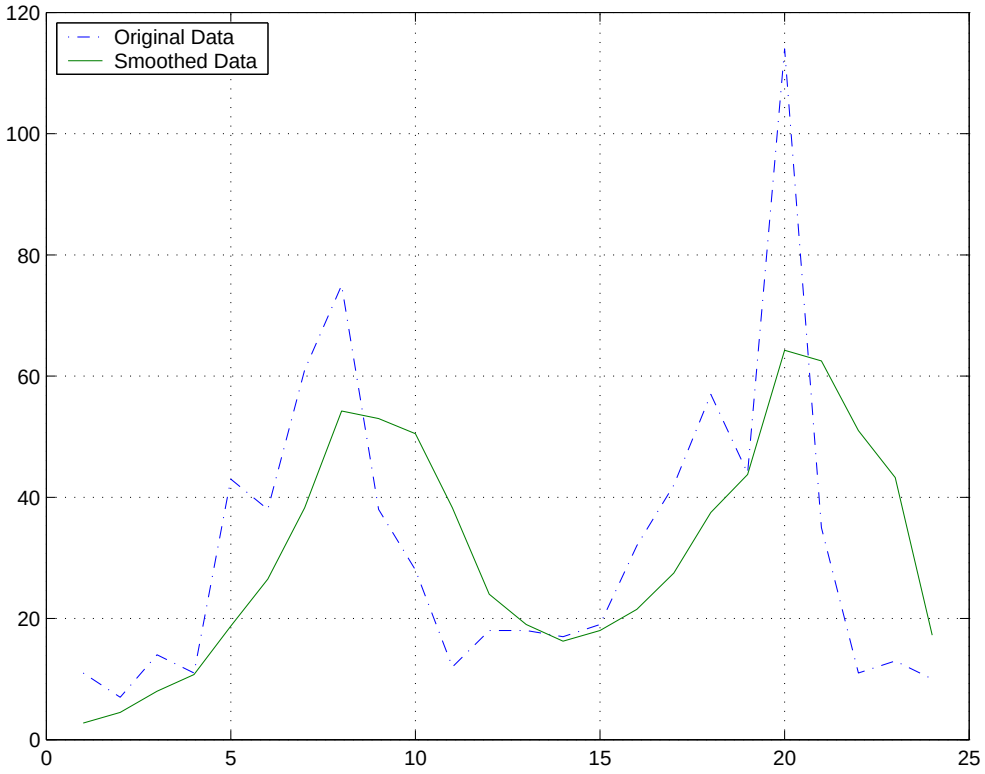

Figure 6: Effect of 4-Hour Moving Average Filter

And the inverse fourier transform (IFT) is defined as:

$$f(x) = \int_{-\infty}^{\infty} F(u) \exp(t2\pi ux) du$$

Note that the above shown transforms are based on a continuous function f(x). Hence they are not computationally feasible. Fast Fourier transform (FFT) is just a modified discretized version of the Fourier transform algorithm (DFT) which reduces the number of computations needed for N points from $2N^2$ to $2N \lg N$.

DFT for a sequence 'x' of length 'N' goes like this ('X' is a vector of length 'N' again)...

$$X(k) = \sum_{n=1}^{N} x(n) \exp(-j2\pi(k-1)\left(\frac{n-1}{N}\right)), 1 \leq k \leq N$$

And the inverse discrete fourier transform (IDFT) is:

$$x(n) = \frac{1}{N}\sum_{k=1}^{N} X(k)\exp\left(j2\pi(k-1)\left(\frac{n-1}{N}\right)\right), 1 \leq n \leq N$$

Fourier transforms have a whole bunch of useful properties, which is the reason they are so widely used in the scientific community. For such information refer [5] and [6] as well as other references you might find!

Alright now with the theory of Fourier transforms out of the way, let me show how you can do DFT in Matlab. As usual a discussion follows the source code.

```
% Hint: use ; at end of line to prevent debug'ish output from being printed

x = [4 3 7 -9 1 0 0 0]'
y = fft(x)               % Compute Discrete FFT
ifft(y)                  % Inverse

pause;                   % Pause before proceeding to next example

t = 0:1/100:10-1/100;
x = sin(2*pi*15*t) + sin(2*pi*40*t);

y = fft(x);
m = abs(y);              % get magnitude and phase angle (next line)
p = unwrap(angle(y));    % Unwrap - removes multiples of Pi

f = (0:length(y)-1)'*100/length(y);
subplot(2,1,1), plot(f,m),
ylabel('Abs. Magnitude'), grid on
subplot(2,1,2), plot(f,p*180/pi)
ylabel('Phase [Degrees]'), grid on
xlabel('Frequency [Hertz]')
```

I've shown two examples in the above code. The first one defines a simple vector 'x' and performs FFT on it. I've also shown the computation of inverse transform using the ifft() function.

The second function defines a time span on the variable 't'. Then 'x' is defined as sinuisoidal function of 't'. Then the fft() function is used.

Also note I've used a few other functions: abs(), angle() and unwrap(). abs() essentially gives the magnitude of the result while angle() gives the phase angle. unwrap() removes any redundant multiples of π in the phase angle.

Following these computations is a plotting of the data. I will come back to it, if possible, after looking into plotting capabilities in Matlab.

```
x =
     4
     3
     7
    -9
     1
     0
     0
     0

y =
   6.0000
  11.4853 - 2.7574i
  -2.0000 -12.0000i
  -5.4853 +11.2426i
  18.0000
  -5.4853 -11.2426i
  -2.0000 +12.0000i
  11.4853 + 2.7574i

ans =
    4.0000
    3.0000
    7.0000
   -9.0000
    1.0000
   -0.0000
         0
    0.0000
```

0.4 Plotting Capabilities

In this section I will show some examples of how Matlab can be used to plot data. (These are plots I did for another class project report, so please don't get psyched by the weird numbers! It was all script-generated obviously!). I have three examples in the following subsections which show different types of simple plots you can make with Matlab. Of course there is a lot more which you can do than this, but this should get you started. If you are interested GNUPlot is another useful tool that can be used to make nice graphs of datasets [7].

0.4.1 Simple Multicolumn Plot

This example shows how a bunch of columns against a common (x-axis) column can be plotted in case there is a matrix of data to be analyzed. This is probably

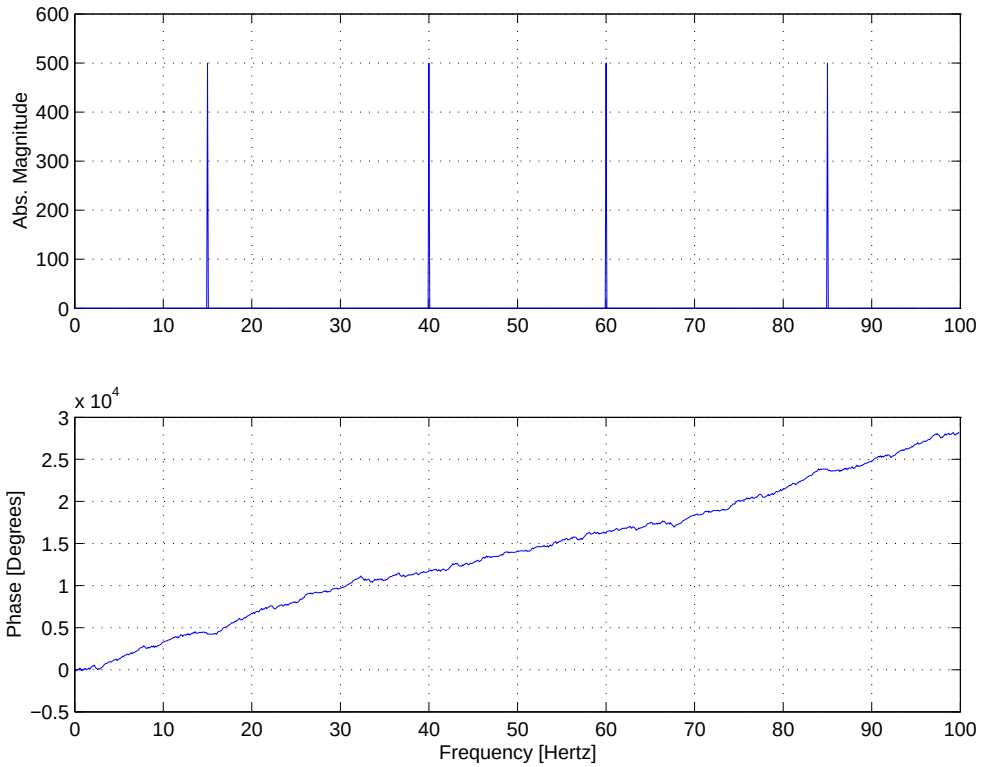

Figure 7: Plot of Magnitude and Phase after doing a FFT

the simplest form of graph that can be plotted on Matlab or any other plotting tool like GNUplot [7]. A brief discussion of the script follows the code snippet.

```
% log2mul, LogMaxN, maxM, SetPrec, IntgrPrec, GotPrec, 4 times, TotalTime %

A=[
18,1522,172,1000,760,722,0.268435,5.100274,0.268435,0.000000,6.174016
18,1884,216,1250,933,895,0.536871,5.905580,0.805306,0.000000,7.784628
18,2268,264,1500,1117,1077,1.073742,6.979322,1.610613,0.000000,10.200547
18,2651,312,1750,1298,1261,1.073742,8.321499,2.684355,0.000000,12.884902
18,3016,360,2000,1475,1435,1.879048,9.126805,4.294967,0.000000,16.106127
18,3399,408,2250,1650,1618,2.415919,10.200547,6.979322,0.000000,20.401094
18,3761,456,2500,1822,1791,2.952790,11.274289,10.200547,0.000000,25.501368
18,4145,508,2750,2005,1974,3.758096,12.616467,15.032386,0.000000,32.480690
18,4528,554,3000,2166,2157,5.100274,13.421773,20.401094,0.000000,39.996883
18,4893,600,3250,2325,2325,5.637145,14.227079,27.648851,0.000000,48.855251
18,5276,646,3500,2483,2482,6.710886,15.569257,37.849400,0.000000,61.471718
18,5639,692,3750,2639,2638,8.321499,16.642998,46.976204,0.000000,73.551315
```

```
18,6022,738,4000,2793,2793,9.932112,17.448305,59.324234,0.000000,88.852135
];

maxtimeX = 600:10:2462;
maxtimeY=60;
bestX=2462;
bestY=0:1:60;

plot(A(:,6), A(:,7), 'm-+', A(:,6), A(:,8), 'g-+', A(:,6), A(:,9), 'b-+', A(:,6), A(:,11), 'r-+',

legend('Precision Achieved (vs) Time for Log n computation', 'Precision Achieved (vs) Time for Ha
title('Precision (vs) Various Times', 'FontWeight', 'bold');
xlabel('Precision Acheived');
ylabel('Time (in seconds) [Max permissible = 60]');
axis([700 2900 0 100]);
```

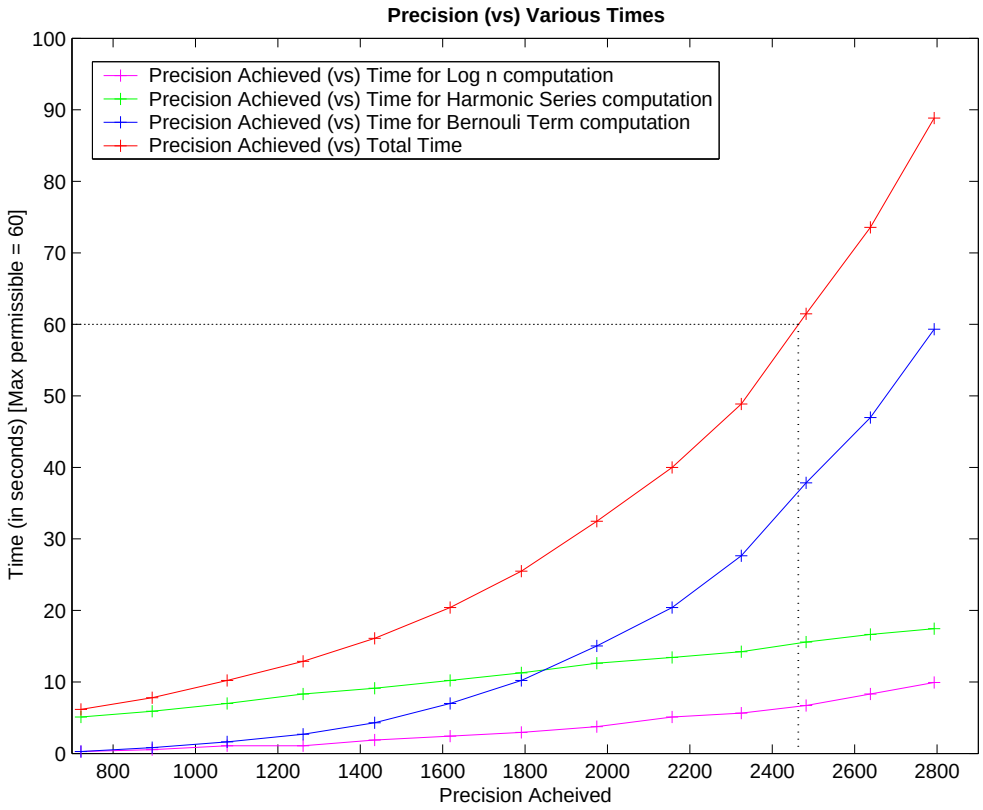

Figure 8: Simple Multicolumn Plot

0.4.2 Stacked Bar Graph

This example shows how a stacked bar chart can be constructed using Matlab. A brief discussion of the script follows the code snippet.

```
A=[
10,5413,1136,3420,1351,1350,7.516193,0.000000,179.851761,0.000000,188.441696
11,5413,1032,3420,1581,1581,7.247757,0.268435,136.902084,0.000000,145.223587
12,5413,946,3420,1770,1769,7.516193,0.268435,109.790100,0.000000,118.648468
13,5415,874,3420,1928,1928,7.247757,0.268435,86.704651,0.000000,95.563019
14,5415,812,3420,2062,2062,7.247757,0.805306,70.866959,0.000000,80.262199
15,5415,758,3420,2176,2175,8.589934,1.610613,58.787365,0.000000,70.598526
16,5417,710,3420,2272,2272,7.247757,2.684355,48.855251,0.000000,60.397976
17,5417,668,3420,2356,2356,7.247757,6.979322,41.607494,0.000000,57.445187
18,5417,632,3420,2435,2434,7.247757,16.106127,35.433479,0.000000,60.666412
19,5419,598,3420,2498,2498,6.979322,34.091305,30.601643,0.000000,73.819748
20,5419,568,3420,2557,2556,7.784628,70.866959,26.306675,0.000000,107.105743
21,5419,542,3420,2614,2581,7.516193,142.807663,23.353884,0.000000,174.483047
22,5419,508,3420,2617,2581,7.516193,279.709747,19.595789,0.000000,307.895477
];

maxtimeX = 9:0.05:23;
maxtimeY=60;
bestX=2462;
bestY=0:1:60;

dY=[A(:,7) A(:,8) A(:,9)];
h = bar(A(:,1), dY, 0.5, 'stack');
hold on
plot (maxtimeX, maxtimeY , 'r--');
hold off

legend(h, 'lg N vs Time for Log n computation', 'lg N vs Time for Harmonic Series compu
title('Lg N (vs) Various Times', 'FontWeight', 'bold');
xlabel('Lg N [N = pow (2, x-axis)]');
ylabel('Time (in seconds) [Max permissible = 60]');
axis([9 23 0 320]);
```

0.4.3 Dual Axes Example

This example shows how Matlab can be used to plot a common column (x-axis) not only against two different columns (in the y-axis) but also when they are at different scales i.e. the two columns to plot against the common x-axis are in different ranges altogether. As usual, a brief discussion of the script follows the code snippet.

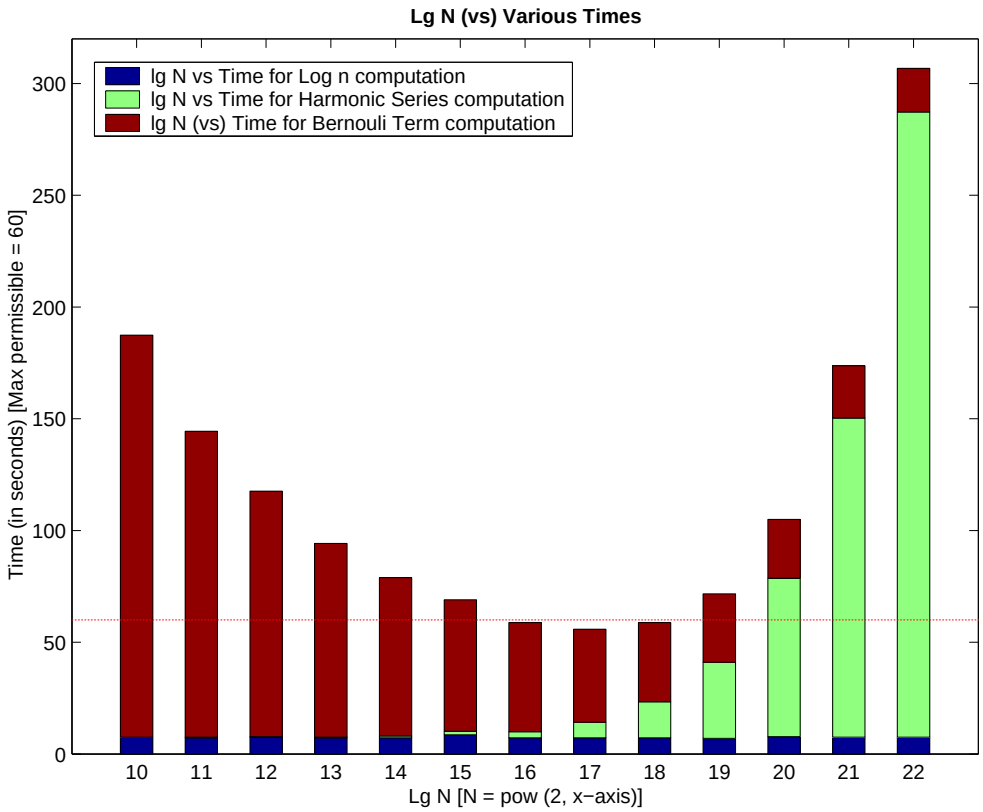

Figure 9: Stacked Bar Graph

```
% log2mul, LogMaxN, maxM, SetPrec, IntgrPrec, GotPrec, 4 times, TotalTime %
A=[
18,2203,256,3460,1087,1048,0.536871,7.247757,1.610613,0.000000,9.663676
18,2410,280,3460,1178,1146,0.805306,7.516193,1.879048,0.000000,10.737418
...
18,6567,640,3460,2462,2462,12.616467,19.058918,44.560287,0.000000,81.067505
];

X1=[
1.0
1.1
...
3.0
];

X2=[
```

```
A(1,4)*1.0
A(1,4)*1.1
...
A(1,4)*3.0
]

bestX = 2.37;
bestY = 1000:10:3000;

[haxes, hline1, hline2] =  plotyy(X1(:,1), A(:,6), X1(:,1), A(:,11), 'plot', 'plot');

axes(haxes(1));
ylabel('Precision achieved');
hold on
plot (bestX, bestY, 'r')
hold off

axes(haxes(2));
ylabel('Computation Time (in seconds)');
xlabel('Precision Multiplier [Working Precision = precision * precision multiplier]');
title('Working Precision (vs) Precision achieved/Computation Time', 'FontWeight', 'Bold
```

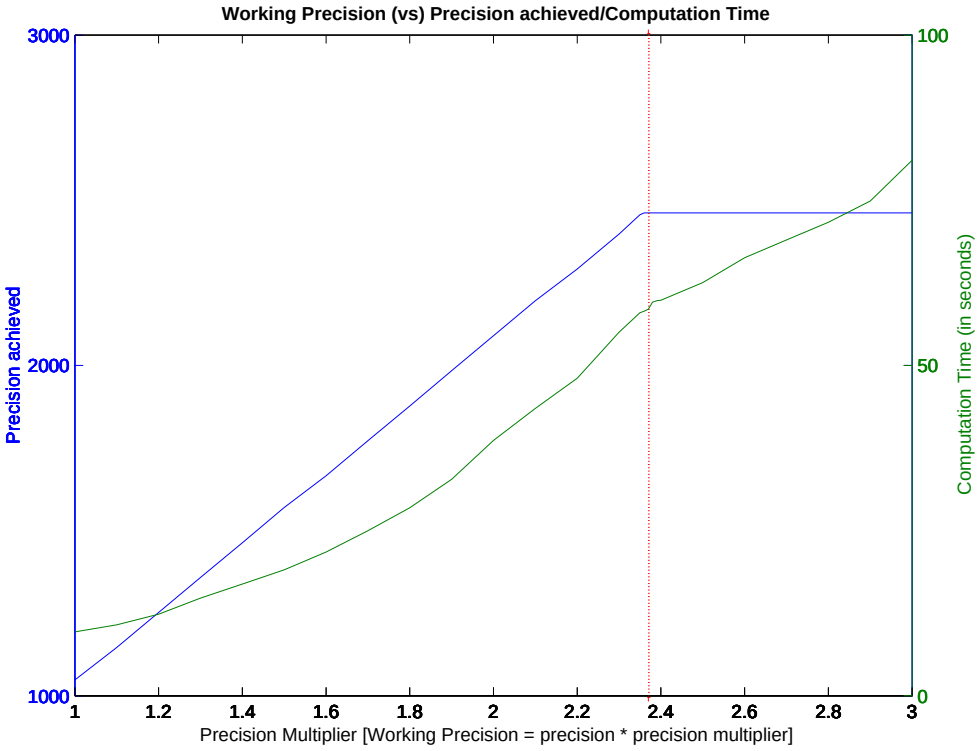

Figure 10: Dual Axes Example

Bibliography

[1] MATLAB Documentation *'Matlab Help' section under the 'Help' menu on the Matlab main window.*

[2] Matrix Multiplication - Basic Stuff! *http://www.csun.edu/ steve/COMP_535/MatrixMultiply.html*

[3] Partial Derivatives *http://mathworld.wolfram.com/PartialDerivative.html.*

[4] Laplacian Operator *http://mathworld.wolfram.com/Laplacian.html.*

[5] A Crash course in Fourier Transforms *http://www.astro.psu.edu/users/mce/A451_2/A451/downloads/notes2.pdf.*
Fourier Transforms and their properties *http://aurora.phys.utk.edu/ mahan/papers/fourier/section3_1.html#SECTION00010000000000000000.*
Fourier Transforms, DFTs and FFTs *http://www.me.psu.edu/me82/Learning/FFT/FFT.html.*

[6] Fourier Transforms from Mathworld *http://mathworld.wolfram.com/FourierTransform.html.*

[7] Introduction to GNUPlot *http://www.cs.uni.edu/Help/gnuplot/.*

Change Log

Should I call this change log or erratta?! What do you think?

Oct 30 2003 Section 0.2.1 "Transpose of 'w"' =¿ "Transpose of 'v' "

Have each row's data comma/space separated and then a return char (newline) demarcates between different rows. =¿ Have each row's data comma/space separated and then a return char (newline) demarcates two rows.

Section 0.2.2 Section Heading changed from Inverses, Powers, Eigenvalues and such. I decided not to include Eigen value stuff in this presentation!

Section 0.3.2 "Following that..." =¿ "First up..."

The .m source code and Sample Output updated to show 'gradient()' results.

Section 0.3.3 Figure [5] was showing the wrong plot - fixed.

The .m source code and Sample Output updated to show a bit more detail

One more plot showing just data points added. (Figure [3])

Section 0.1.1 Added in this sub section; Just realized this might be necessary to get started! Also has a couple of images in it (Figures [1] and [2])

Section 0.2.1 Added in a little bit of explanation about the magic(int) function.

Section ?? Fixed broken reference link.