A Brief Introduction to Matlab

1 Mathematical operations

Try the following operations and commands on the command line in the command window of Matlab.

1.1 Defining an array and arranging in ascending and descending order

```
>> A = [2  4 -3 43 2 10 log(200)]

A =

    2.0000    4.0000   -3.0000   43.0000    2.0000   10.0000    5.2983

>> sort(A)

ans =

   -3.0000    2.0000    2.0000    4.0000    5.2983   10.0000   43.0000

>> sort(A,'descend')

ans =

   43.0000   10.0000    5.2983    4.0000    2.0000    2.0000   -3.0000
```

1.2 Defining a matrix and arranging in ascending and descending order

```
>> B = [2 3 1 4; -4 -2 0 -13; 66 3 0.2 1/3]

B =

    2.0000    3.0000    1.0000    4.0000
   -4.0000   -2.0000         0  -13.0000
   66.0000    3.0000    0.2000    0.3333

>> sort(B)

ans =

   -4.0000   -2.0000         0  -13.0000
    2.0000    3.0000    0.2000    0.3333
   66.0000    3.0000    1.0000    4.0000

>> sort(B,2)

ans =

    1.0000    2.0000    3.0000    4.0000
  -13.0000   -4.0000   -2.0000         0
    0.2000    0.3333    3.0000   66.0000
```

1.3 Powers of a matrix and its elements

```
>> C = [ 1 2 3; 4 2 3; 5 6 2]

C =

     1     2     3
     4     2     3
     5     6     2

>> C^2

ans =

    24    24    15
    27    30    24
    39    34    37

>> C.^2

ans =

     1     4     9
    16     4     9
    25    36     4

>> D = [ 3 3 2; 5 0 1; -2 3.5 6]

D =

    3.0000    3.0000    2.0000
    5.0000         0    1.0000
   -2.0000    3.5000    6.0000

>> C*D

ans =

    7.0000   13.5000   22.0000
   16.0000   22.5000   28.0000
   41.0000   22.0000   28.0000

>> C.*D

ans =

     3     6     6
    20     0     3
   -10    21    12
```

1.4 Finding maximum and minimum entries in an array (or matrix)

```
>> A = [ 4 2 -7 -0.2 33 12]

A =

    4.0000    2.0000   -7.0000   -0.2000   33.0000   12.0000

>> max(A)

ans =

    33

>> min(A)

ans =

    -7
```

Finding `argmax` and `argmin` in an array

```
>> find(A == max(A))

ans =

     5

>> find(A == min(A))

ans =

     3

>> D

D =

    3.0000    3.0000    2.0000
    5.0000         0    1.0000
   -2.0000    3.5000    6.0000

>> max(D)

ans =

    5.0000    3.5000    6.0000

>> max(D,[],2)
```

```
ans =

     3
     5
     6
```

Summing (and *cumulative* summing) rows and columns of a matrix

```
>> A = [ 3 1 -2 5; 0 2 11 -4; 9 0 7 6]

A =

     3     1    -2     5
     0     2    11    -4
     9     0     7     6

>> sum(A)

ans =

    12     3    16     7

>> sum(A,2)

ans =

     7
     9
    22

>> cumsum(A)

ans =

     3     1    -2     5
     3     3     9     1
    12     3    16     7

>> cumsum(A,2)

ans =

     3     4     2     7
     0     2    13     9
     9     9    16    22
```

Product of entries of a matrix

```
>> prod(A)
```

```
ans =

     0     0  -154  -120

>> prod(A,2)

ans =

   -30
     0
     0
```

1.5 Accessing certain sections of a matrix

```
>> A = [22 3 -4; 0 1 7; 2 -1 2]

A =

    22     3    -4
     0     1     7
     2    -1     2

>> A(:,2)

ans =

     3
     1
    -1

>> A(3,:)

ans =

     2    -1     2

>> [-2 -1 -3].*A(1,:)

ans =

   -44    -3    12
```

1.6 Some elementary mathematical functions

```
 >> exp(2)

ans =

    7.3891

>> sin(pi/4)
```

```
ans =

    0.7071

>> log(100)

ans =

    4.6052

>> log10(100)

ans =

     2

>> exp([1 2; 3 4])

ans =

    2.7183    7.3891
   20.0855   54.5982
```

1.7 Random number generator

```
 >> rand

ans =

    0.3112

>> rand()

ans =

    0.5285

>> rand(3)

ans =

    0.1656    0.6541    0.4505
    0.6020    0.6892    0.0838
    0.2630    0.7482    0.2290

>> rand(1,2)

ans =

    0.9133    0.1524
```

```
>> a=2

a =

     2

>> b=5

b =

     5

>> r = a+(b-a).*rand(5,1)          % random numbers between a and b
                                       % from uniform distribution

r =

    4.4775
    3.6150
    4.9884
    2.2345
    3.3280

>> randi([-3 3],5,1)

ans =

    -3
     3
     1
     0
    -2

>> randperm(4)

ans =

     3     4     2     1

>> nchoosek(5,2)

ans =

    10
>> perms([1 2 3])

ans =

     3     2     1
     3     1     2
     2     3     1
```

```
     2     1     3
     1     3     2
     1     2     3
```

Normally distributed random numbers

```
 >> randn(5)

ans =

    1.1275   -1.7502   -0.5336   -0.0348    1.3514
    0.3502   -0.2857   -2.0026   -0.7982   -0.2248
   -0.2991   -0.8314    0.9642    1.0187   -0.5890
    0.0229   -0.9792    0.5201   -0.1332   -0.2938
   -0.2620   -1.1564   -0.0200   -0.7145   -0.8479
```

1.8 Calculus and functional operations

Writing matlab script files

It must be noted that the `command window` is not the only interface in Matlab to perform mathematical operations. In fact there is a more optimal and preferred editor where one can write Matlab script files and define functions of their choice. To familiarize the user with the matlab editor, the following sequence of operations are written as a script file using the editor. In order to execute the code, you simply have to hit the `run` icon on the editor at the top. The answers and results of the code appear on the `command window` as well as in the `workspace`.

The commented sections of the code appear to the right of the % symbol and are non-executable portions of the code. The comments should be self explanatory.

```
% function handles and function evaluations
f = @(y) exp(y)
f(1)

syms x          % define x as a symbolic variable
g = 5*exp(x) + sin(x) % symbolic function definition
dg = diff(g)    % symbolic differentiation
subs(dg,x,pi)   % replace x with pi in the expression given by dg


% function handles and function evaluations
A = [1 2; 3 4] % A is a matrix whose eigen system we want to find
fn = @eig
%two alternate ways of finding eigenvector and eigenvalues
[EV ev] = fn(A)
% OR
[V e] = feval(fn,A)
```

2 Graphical representation of data

2.1 Plotting functions

The following piece of program gives you an idea of plotting functions graphically, assigning appropriate labels, defining boundaries of graphs and using desired font size and markers. Additionally, you will also learn how to generate multiple plots within a single graphic frame using the matlab `subplot` functionality. Tweak the parameters and practise to get a better understanding of the different available options for pictorially depicting data in matlab.

```
x=[0:0.05:4*pi];
y=sin(x);
z=cos(x);
figure, subplot(3,1,1);
plot(x,y,'r*-',x,z,'bo-');
xlim([0 4*pi]);
ylim([-1.3 1.3]);
xlabel('x');
ylabel('fundamental oscillations');
legend('sin(x)','cos(x)');
title('periodic functions');

subplot(3,1,2);
xx=[0:0.1:4*pi];
t=tan(xx);
stem(xx,t,'mx')
xlim([0 4*pi]);
ylim([-10 10]);
xlabel('x');
ylabel('tan x');
title('periodic singular function');

%% drawing rectangular pulse %%
% This example generates a pulse train using the default rectangular
% pulse of unit width. The repetition frequency is 0.5 Hz,
% the signal length is 60 s, and the sample rate is 1 kHz.
% The gain factor is a sinusoid of frequency 0.05 Hz.
t = 0:1/1e3:60;
d = [0:2:60;sin(2*pi*0.05*(0:2:60))]';
x = @rectpuls;
y = pulstran(t,d,x);
subplot(3,1,3);
plot(t,y,'-bs',...
    'LineWidth',2,...
    'MarkerSize',10,...
    'MarkerEdgeColor','g',...
    'MarkerFaceColor',[0.5,0.5,0.5]);
hold on
xlabel('Time (s)')
ylabel('Waveform')
title('rectangular pulse train');
```

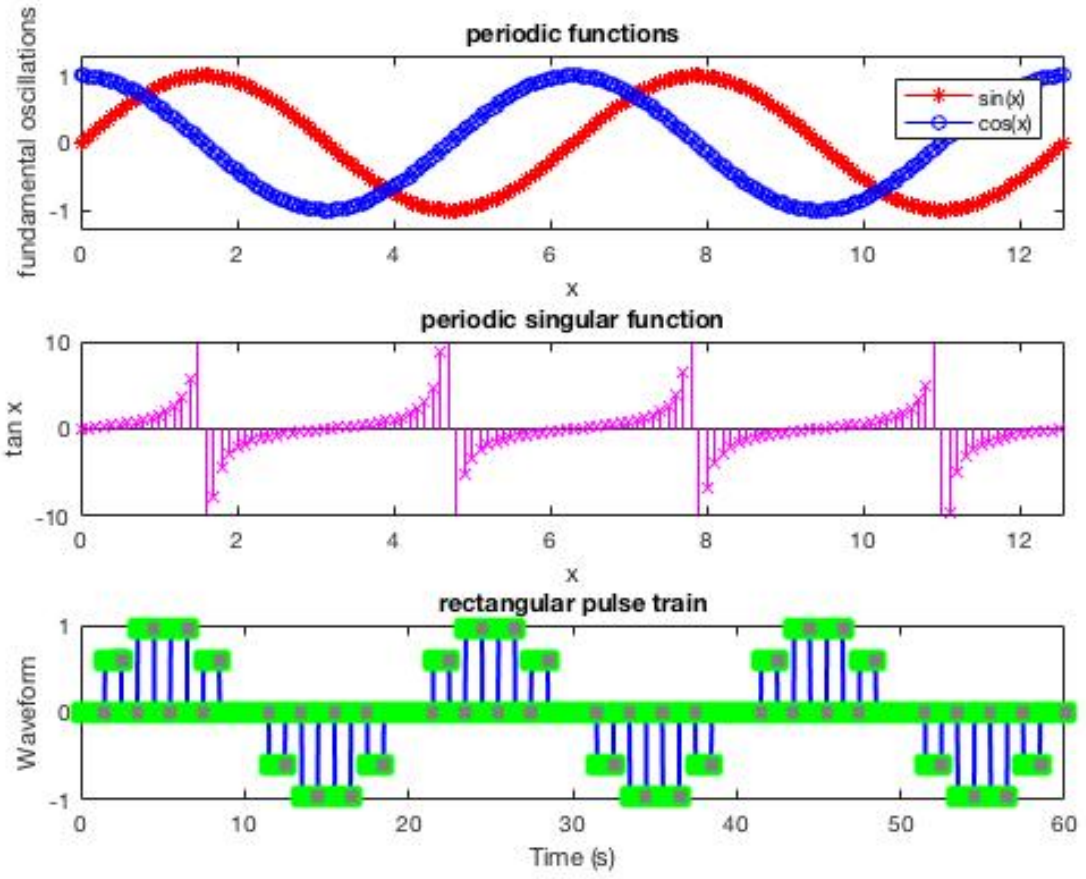

Figure 1: How to make subplots in matlab with labels and legends?

2.2 Plotting data on a bar graph

In a matlab script file, store data as a matrix as shown below. Then call a user defined function `mydataPlot` which takes the input matrix as its argument and draws a joint-histogram of the data over time (years). The data must be normalized on a scale of 0-2 and depicted as a bar graph.

The script file may contain the following lines of code.

```
 %%%%%%%%%%%%%%%% making histograms %%%%%%%%%%
% Ip stores input data in 3 columns:
% (Year, Rainfall, Temperature) in appropriate units
Ip=[2009 1000 39; 2010 997 34;2011 1152 41; 2012 855 31; ...
    2013 1013 40; 2014 878 30; 2015 1243 43];
mydataPlot(Ip);
```

Writing user defined functions

`mydataPlot` is a user defined function which needs to be programmed by writing a short matlab function and saving it as `mydataPlot.m`. The function may be written as follows.

```
function [X] = mydataPlot(Ip)

A = Ip(:,1);
```

```
B = Ip(:,2);
C = Ip(:,3);
Bnew=(B(:)-min(B))/(max(B)-min(B));
Cnew=(C(:)-min(C))/(max(C)-min(C));
newData = [Bnew+1 Cnew+1]; % setting it on a normalized scale 0-2
figure, bar(A,newData(:,1:2));
xlabel('Year','fontsize',18);
ylabel('Normalized data on a scale of 0-2','fontsize',18);
legend('Amount of rainfall','temperature','Location','northwest');
title('Visualizing trend & correlation qualitatively using joint-histogram',...
    'fontsize',14);

end % denoting end of function
```

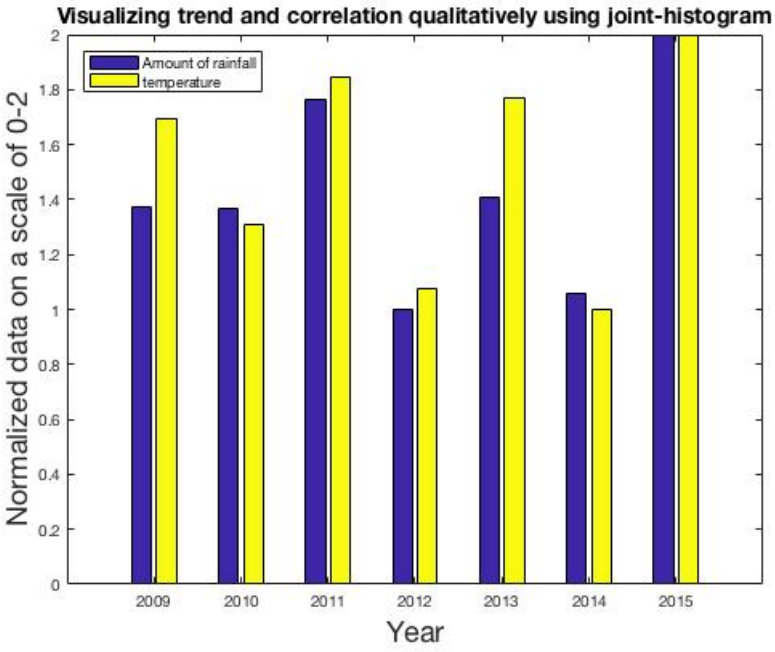

Figure 2: How to make bar graphs in matlab with labels and legends?

3 Loops and conditional statements

Matlab provides in-built *loop* functionalities using the following syntax.

```
for
    .........
    .........
end
```

or

```
while
    .........
    .........
end
```

3.1 `for` loop to generate a matrix with labels in lexicographical order

```
% use for loops to construct a matrix with entries
% that store the serial location (lexicographical order)
% also compute the sum of all entries of the same matrix

imax=4; jmax=5;
running_sum=0;
for i=1:imax
    for j=1:jmax
        A(i,j) = (j + (i-1)*jmax);
        running_sum = running_sum + A(i,j);
    end
end
```

3.2 Conditional statement using `if-else` for comparing results

```
if running_sum == sum(sum(A))
    disp('my calculation is correct: Hurrah!');
else
    disp('I made an error in calculation');
end
```

3.3 `for` loops for printing prime numbers less than or equal to N

Exercise: Following is a pseudocode for printing all the prime numbers less than or equal to N where $N = 30$ for example. Convert the pseudocode to matlab executable code and display your answer.

Pseudocode for printing prime numbers:

```
INPUT: N.

for all i from 2 to N
   reset prime number flag to default ON
   for all j from 2 to i/2
      if remainder of i ÷ j is not equal to 0
         continue the current for loop
      else
         turn OFF prime number flag
         break from the current for loop
      end if-else condition
   end for loop for j
   if prime number flag is ON
      store prime number i in a dynamic array
   end if condition
end for loop for i

OUTPUT: display array containing the prime numbers.
```

While implementing the above algorithm you will learn to use the following matlab commands: `mod`, `continue` and `break`. Use >>`doc` ⟨`function name`⟩ or >>`help` ⟨`function name`⟩ on the command line to learn how to use them. Also, the comparative clause *'is not equal to'* is written in matlab as $\sim=$.

4 Data structures: loading, organizing, accessing and writing data

There are many different data structures available in matlab. Here we will discuss only some of the important ones that may be relevant to us.

4.1 Data as graphs of matrices

A *graph* is a data structure that consists of the following two components.

1. A finite set of *vertices* also known as *nodes*.
2. A finite set of ordered pairs of the form (u, v) known as *edges*. The pair is ordered because (u, v) is not same as (v, u) in case of a directed graph (di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

Graphs are used to represent many real-life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex (or node). Each node is a structure and contains information like id, name, gender and location of a person.

4.1.1 Adjacency Matrix

A graph may be represented as an *adjacency matrix*. An *adjacency matrix* is a 2D array (or matrix) of size $V \times V$ where V is the number of vertices in a graph. Let the 2D array be $adj(i, j)$, a slot $adj(i, j) = 1$ indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If $adj(i, j) = w$, then there is an edge from vertex i to vertex j with weight w.

Drawing a graph from adjacency matrix:

```
% Define a matrix adj.
adj = [0 1 1 0 ; 1 0 0 1 ; 1 0 0 1 ; 0 1 1 0];

% Draw a picture showing the connected nodes.
cla % clear current axis
subplot(1,2,1);
gplot(adj,[0 1;1 1;0 0;1 0],'.-');
% gplot(A,xy) plots the graph (as in 'graph theory') specified by A and xy
text([-0.2, 1.2 -0.2, 1.2],[1.2, 1.2, -.2, -.2],('1234')', ...
   'HorizontalAlignment','center')
axis([-1 2 -1 2],'off')
title('undirected graph','fontsize',14);

% Draw a picture showing the adjacency matrix.
subplot(1,2,2);
xtemp = repmat(1:4,1,4); ytemp = reshape(repmat(1:4,4,1),16,1)';
text(xtemp-.5,ytemp-.5,char('0'+adj(:)),'HorizontalAlignment','center');
```

```
line([.25 0 0 .25 NaN 3.75 4 4 3.75],[0 0 4 4 NaN 0 0 4 4])
axis off tight
title('adjacency matrix','fontsize',14);
```

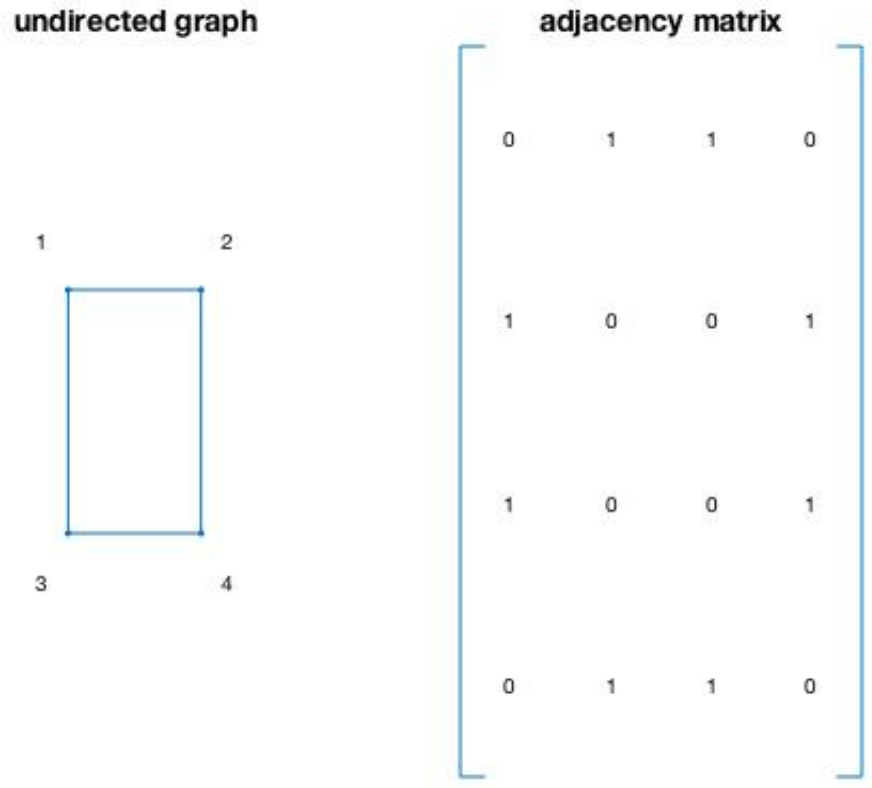

Figure 3: A *graph* and its corresponding *adjacency matrix*.

Drawing the adjacency matrix of a given graph:

For fun, we will first use the matlab in-built function `bucky` to first draw the graph of a *geodesic dome*.[1]

```
%% using bucky and find the adjacency matrix of a given graph
[B,V] = bucky;
G = graph(B);
figure,
p = plot(G);
```

In order to investigate the graph `G` in more detail, type `>> G.Edges` on the command line. Now, let us say we were given the graph `G` (and not the matrix `B`), and we had to find the corresponding adjacency matrix `A`, we would do as follows:

```
A = adjacency(G);
H = graph(A(1:10,1:10)); % graph only a section of the adjacency matrix
figure,
h = plot(H);
```

[1] `https://en.wikipedia.org/wiki/Geodesic_dome`

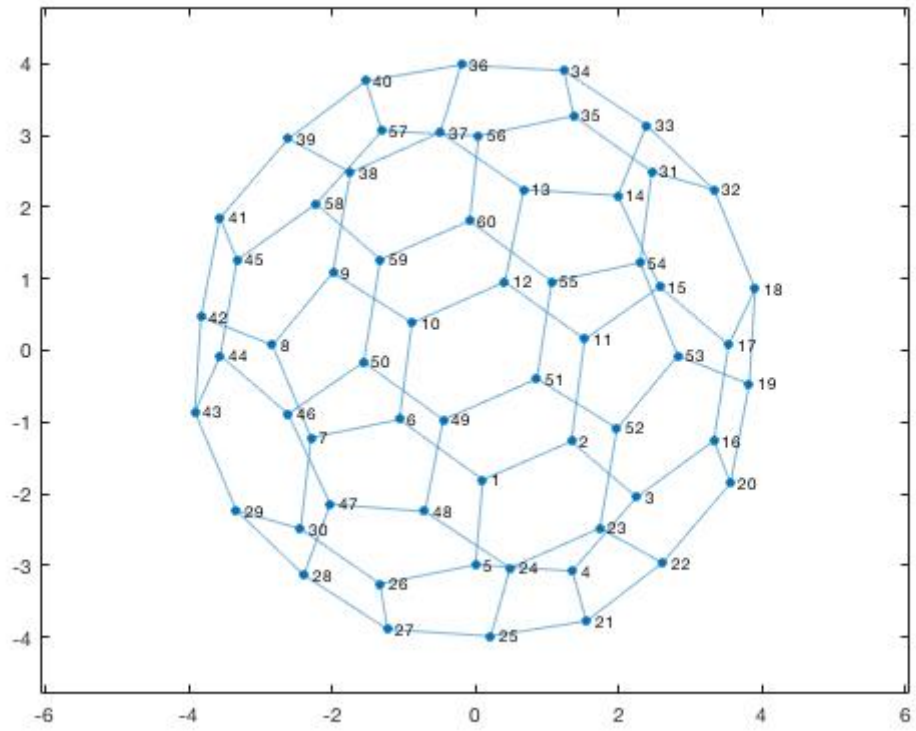

Figure 4: Graph of the geodesic dome using the matlab function `bucky`.

The new plot graphs the selected section of the adjacency matrix `A` specified by the first 10×10 entries of `A`. The new adjacency matrix `A` can be seen from the command line as `>> A` and the new truncated graph is shown below. The information pertaining to the new truncated graph can be found thusly.

```
>> H.Edges

ans =

  112 table

    EndNodes     Weight
    ________     ______

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

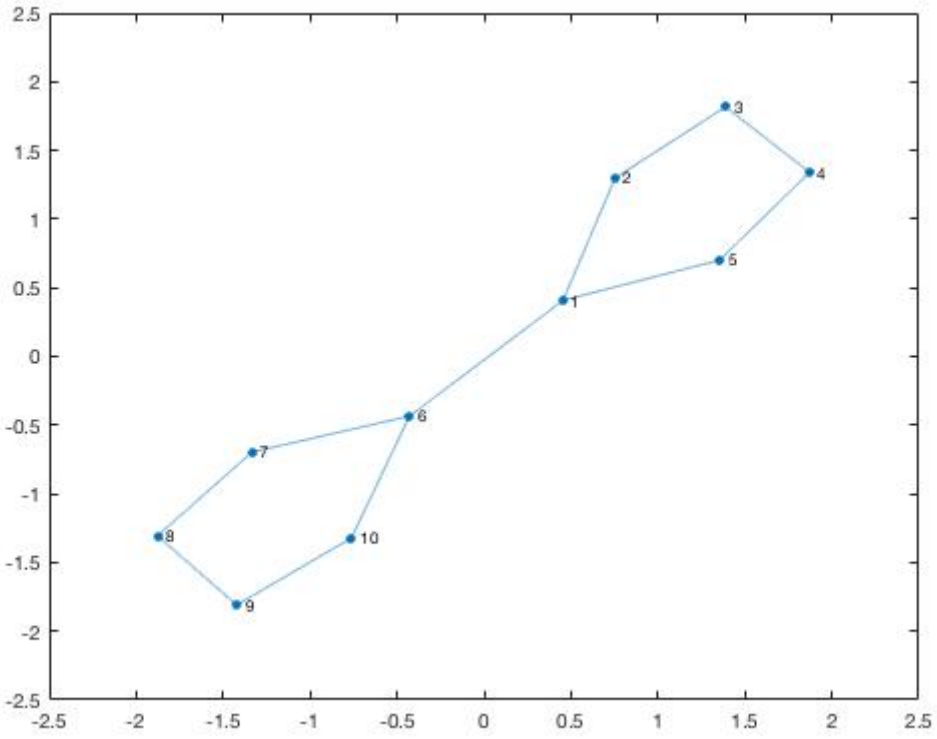

Figure 5: Graph of the truncated geodesic dome.

You can also use similar commands to generate the adjacency matrix of the graph in the previous example (the box graph) and compare it with the matrix `adj` you defined to begin with.

```
Gnew=graph(adj)
Anew = adjacency(Gnew);
Hnew = graph(Anew);
figure,
hnew = plot(Hnew);
```

Compare `Anew` and `adj` to check for consistency.

```
>> Anew

Anew =

   (2,1)         1
   (3,1)         1
   (1,2)         1
   (4,2)         1
   (1,3)         1
   (4,3)         1
   (2,4)         1
   (3,4)         1

>> adj
```

```
adj =

     0     1     1     0
     1     0     0     1
     1     0     0     1
     0     1     1     0
```

4.2 Data as tables

Often we will encounter situations where we may have to import data from an excel file (.csv file). This data in the excel file will be assumed to be stored in tabular (columnar) format under field headings. An example at hand is the file named

```
EnergyConsumptionMP_1996-2018.csv
```

where the data is stored in two columns under the field headings:

```
Year
```

and

```
EnergySupply_MU_
```

This data may be imported in matlab local workspace as follows:

```
%% reading data from .csv file
T = readtable('EnergyConsumptionMP_1996-2018.csv', 'ReadVariableNames', ...
    true, 'Format', '%f %f');
xdata = T.Year;
ydata = T.EnergySupply_MU_;
>> xdata

xdata =

        1996
        1998
        2000
        2002
        2004
        2006
        2008
        2010
        2012
        2014
        2016
        2018

>> ydata

ydata =

       27094
       28599
       30624
```

```
32232
33435
36073
35503
35563
38799
42945
47858
51976
```

4.3 Local data structures in matlab

4.3.1 Structure arrays

Below is a code to show a single variable (named student) which contains two fields which are "sub-components" of what it means to be a student. Each field has its own name and its own type.

```
students(1).name  = 'jim';
students(1).age   = 21;

students(2).name  = 'Jane';
students(2).age   = 33;

students(3).name  = 'Joe';
students(3).age   = 25;

students(4).name  = 'Janet';
students(4).age   = 24;
```

A querry about the first student yields the following information.

```
>> students(1)

ans =

  struct with fields:

    name: 'jim'
     age: 21
```

4.3.2 Cell arrays

Cell arrays contain data in cells that you access by numeric indexing. Common applications of cell arrays include storing separate pieces of text and storing heterogeneous data from spreadsheets. For example, store temperature data for three cities over time in a cell array.

```
temperature(1,:) = {'2009-12-31', [45, 49, 0]};
temperature(2,:) = {'2010-04-03', [54, 68, 21]};
temperature(3,:) = {'2010-06-20', [72, 85, 53]};
temperature(4,:) = {'2010-09-15', [63, 81, 56]};
temperature(5,:) = {'2010-12-09', [38, 54, 18]};

>> temperature
```

```
temperature = 5x2 cell array
    {'2009-12-31'}    {1x3 double}
    {'2010-04-03'}    {1x3 double}
    {'2010-06-20'}    {1x3 double}
    {'2010-09-15'}    {1x3 double}
    {'2010-12-09'}    {1x3 double}
```

Plot the temperatures for each city by date.

```
allTemps = cell2mat(temperature(:,2));
dates = datetime(temperature(:,1));

plot(dates,allTemps)
title('Temperature Trends for Different Locations')
xlabel('Date')
ylabel('Degrees (Fahrenheit)')
```

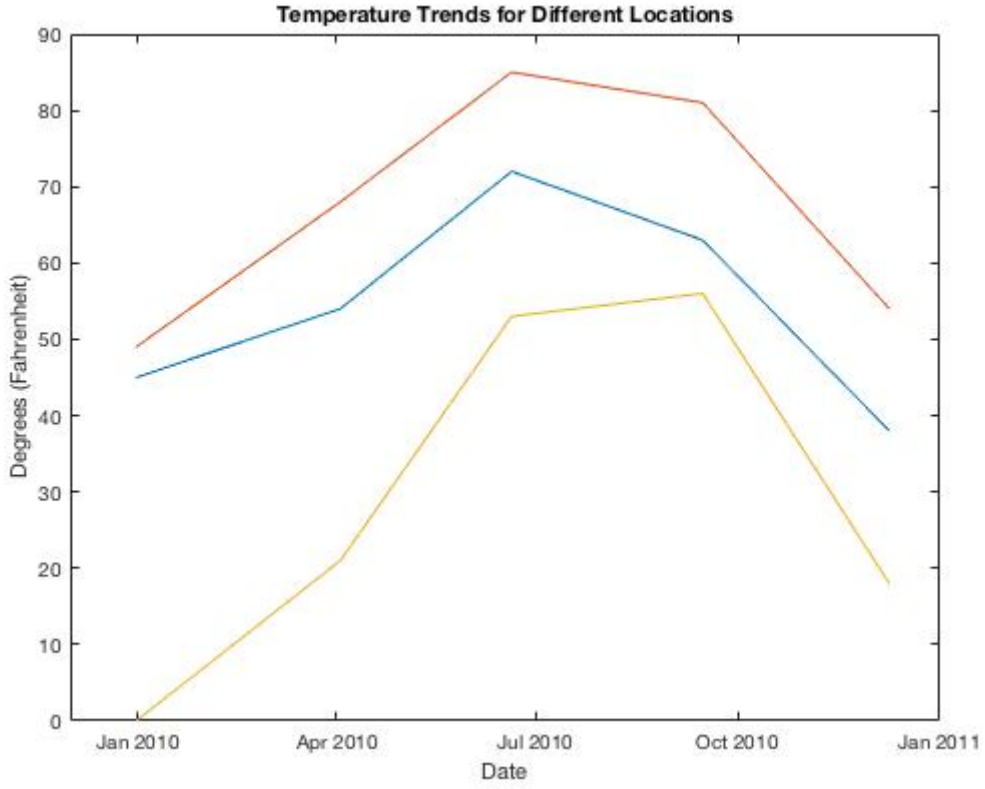

Figure 6: Graph of the truncated geodesic dome.

4.4 Writing to a binary file and reading from a binary file

Write a nine-element vector to a sample file, `nine.bin`.

```
fileID = fopen('nine.bin','w');
fwrite(fileID,[1:9]);
fclose(fileID);
```

Now, read the contents of the same file and store it in a local variable `A`.

```
fileID = fopen('nine.bin');
A = fread(fileID)
```

4.5 Reading a graphic image

Read a sample image.

```
A = imread('ngc6543a.jpg');
```

`imread` returns a 650-by-600-by-3 array, `A`. Now, display the image.

```
>> image(A)
```

Use
`>> doc imread`
to learn more about the function. Likewise, you may use the matlab command `imwrite` to write data to an image file.

4.6 Recording and playing a movie in matlab

For recording a set of graphic frames and constructing a movie, you may use `getframe` as shown below immediately after the graphic frame is generated each time in a loop.

```
while n <= 100
    .......
    .......
    plot(x,y,'r--');
    F(n) = getframe;
    n=n+1;
end
```

In order to make a movie and play it twice, use the following.

```
 >> figure, movie(F,2);
```

5 Final remarks

Finally, extensively use `>> doc doc` and `>> help` ⟨`function name`⟩ to explore different functionalities available in matlab. Also visit the mathworks online help center at
`https://in.mathworks.com/help/`
in order to learn how to use matlab more effectively.

All the best. Have fun learning!

Amrik Sen
Autumn, 2019.

□