

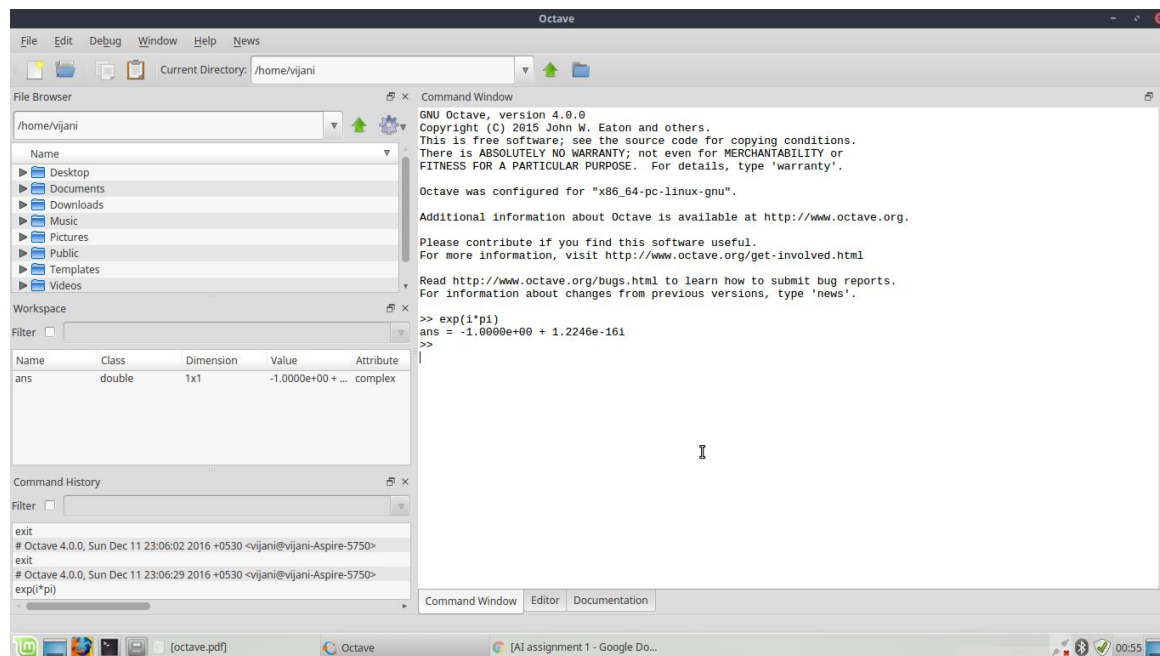
Author : Vijani Piyawardana, BSc (Honours) in Computer Science, UCD, Ireland.

[Download Octave and install on your machines.](#)

[Octave Introduction + Some Simple Calculations](#)

GNU Octave is a high-level language, primarily intended for numerical computations. It is typically used for such problems as solving linear and nonlinear equations, numerical linear algebra, statistical analysis, and for performing other numerical experiments. It may also be used as a batch-oriented language for automated data processing.

Until recently GNU Octave provided a command-line interface only with graphical plots displayed in separate windows. However, by default the current version runs with a graphical user interface.



Octave can easily be used for basic numerical calculations. Octave knows about arithmetic operations (+, -, *, /), exponentiation (^), natural logarithms/exponents (log, exp), and the trigonometric functions (sin, cos, . . .). Moreover, Octave calculations work on real or imaginary numbers (i,j). In addition, some mathematical constants such as the base of the natural logarithm (e) and the ratio of a circle's circumference to its diameter (pi) are pre-defined.

01) basic arithmetic

```
>> 2*4
ans = 8
>> 2/3
ans = 0.66667
>> 3+4
ans = 7
>> 5-6
ans = -1
```

Ex: verify Euler's Identity,

$$e^{i\pi} = -1$$

type the following which will evaluate to -1 within the tolerance of the calculation.

octave:1> exp (i*pi)

```
>> exp(i*pi)
ans = -1.0000e+00 + 1.2246e-16i
>>
>> |
```

02) Variable and assign Value, print value

(i)

```
>> x = 10
x = 10
>> x
x = 10
>> |
```

(ii)

```
>> name = 'vijani'
name = vijani
>> name
name = vijani
>>
```

03) to find square root of a number

```
>> sqrt(9)
ans = 3
>> sqrt(10)
ans = 3.1623
>> |
```

04) Creating a Matrix

(rXc matrix with random numbers, zeros, ones)

Here r = 3 , c = 4

```
>> rand(3,4)
ans =

    0.161090    0.068445    0.567260    0.891013
    0.770012    0.204334    0.052347    0.848538
    0.918468    0.387937    0.819634    0.617481

>> zeros(3,4)
ans =

     0     0     0
     0     0     0
     0     0     0

>> ones(3,4)
ans =

     1     1     1     1
     1     1     1     1
     1     1     1     1

>> |
```

05) create a matrix and assign it to a variable named A, and print A

```
>> A = ones(3,4)
A =

     1     1     1     1
     1     1     1     1
     1     1     1     1

>> A
A =

     1     1     1     1
     1     1     1     1
     1     1     1     1

>> |
```

06) Matrix Arithmetic

(i) $2*A$ will multiply our matrix A by 2, will result a 3X4 matrix with 2s.

```
>> 2*A
ans =

     2     2     2     2
     2     2     2     2
     2     2     2     2

>> |
```

(ii) multiply 2 matrices

Take a new 4X2 matrix with random numbers into variable B.

Multiply it with A, (3X4 matrix with twos)

$(3 \times 4) \times (4 \times 2)$ will result a (3×2) matrix

Do it manually and see if it gives the correct answer.

```
>> B = rand(4,2)
B =

    0.170671    0.901651
    0.634442    0.838435
    0.074694    0.688073
    0.095320    0.202163


>> A*B
ans =

    0.97513    2.63032
    0.97513    2.63032
    0.97513    2.63032


>> |
```


AI Assignment 1

All .m files that you need to do the assignment are given to you in this folder, this example was done at the class, as **Practical3**. Download [Octave Code](#) from moodle under Practicals:

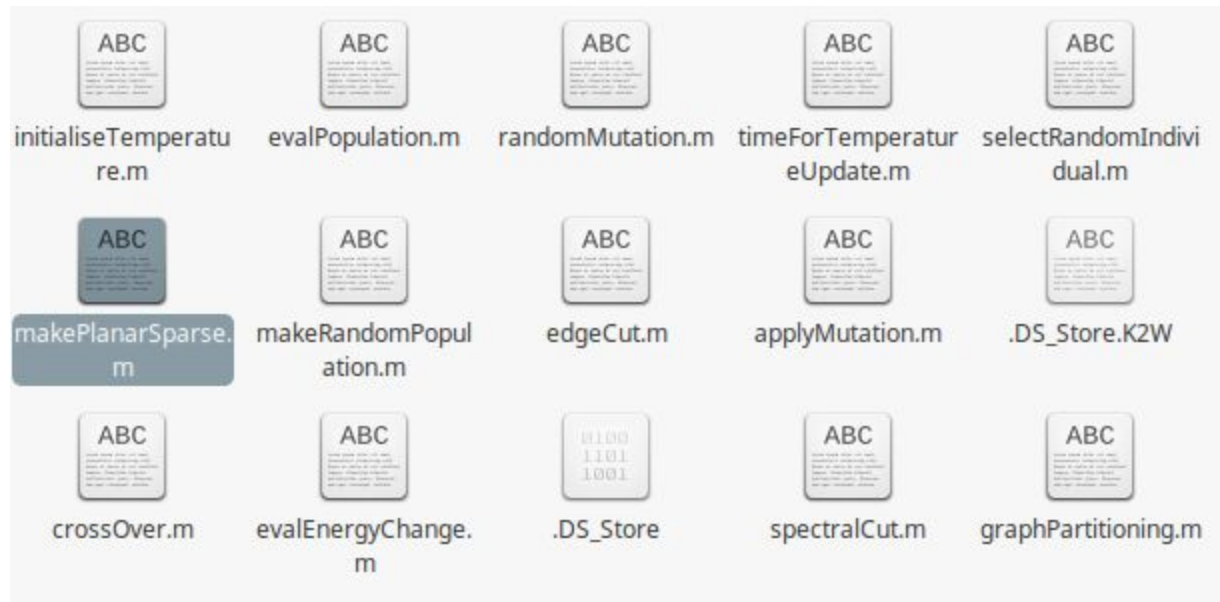
 Octave

I've put a folder at the Dropbox link containing MacOS and Windows versions of Octave, as well as a brief description of how to install them. It may be better to download directly from the SourceForge web site.

 Octave Code for Third Practical session

 Practical Session 3

SAGAGraphPartitioning.zip



You have to have 2 tables like in **Practical Session 3 pdf** , in order to store data to compare 2 algorithms, use Excel sheets or google sheets.

Simulated Annealing

```
N=100; A = makePlanarSparse(N);
[cut,soln]=graphPartitioning(A, 1, 0);
```

N	α	maxGen	chainLen	Quality	Time
100	0.7				
100	0.8				
200	0.9				
...					

Genetic Algorithm

```
P=10; [cut,soln]=graphPartitioning(A, P, 1);
```

N	P	CrossO %	Mut %	numGen	Quality	Time
100	10	50	10			
100	10	50	1			
200	10	10	50			
...						

The goal of this assignment is to **explore Simulated Annealing and Genetic Algorithms by examining their performance on the [Graph Partitioning Problem](#)**. Write a group report that discusses the performance of the algorithms and how that performance is affected by the choice

of parameters. The discussion below will help you to structure your report. Submit the report as a PDF file on the Moodle web-site.

Step 1: Make a matrix to test the algorithms on.

Pick a graph size N , say **N=100**.

Now execute the command

```
>> N=100
N = 100
>> N
N = 100
>>
```

The next command is

`[A,t,x,y] = makePlanarSparse(N)`

The function will return 4 values, to keep them, we have 4 variables

Parameter passing

This function will create a matrix of size N

Execute the next command;

`[A,t,x,y] = makePlanarSparse(N)`

You'll have this error, that's because, makePlanarSparse is not an in-built function of octave, we have to give the place, or go to the place where this function is implemented.

```
>> [A,t,x,y]=makePlanarSparse(N)
error: 'makePlanarSparse' undefined near line 1 column 10
```

This function is available inside the file

/SAGAGraphPartitioning/SAGAGraphPartitioning/[makePlanarSparse.m](#)

So we have to go to the relevant place.

As this is a command line interface, all Linux commands work here.

Linux command to change the drive : [cd folder-path](#)

Give your path to the SAGAGraphPartitioning folder, then you can access files inside that folder.

This is how I changed my path:

```
>> cd /home/vijani/Desktop/AI_OOD/AI_assignment_1_2016/SAGAGraphPartitioning/SAGAGraphPartitioning/
>>
```

Linux command to list all files inside current folder : ls

Let's see if we are in the correct folder.

```
>> ls
applyMutation.m      evalPopulation.m      makeRandomPopulation.m  timeForTemperatureUpdate.m
crossOver.m          graphPartitioning.m   randomMutation.m        selectRandomIndividual.m
edgeCut.m            initialiseTemperature.m spectralCut.m
evalEnergyChange.m   makePlanarSparse.m
>>
```

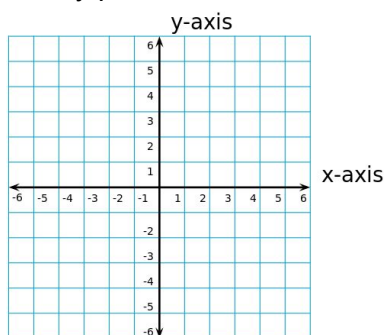
makePlanarSparse

Planer : another name for Plane : තලයක්

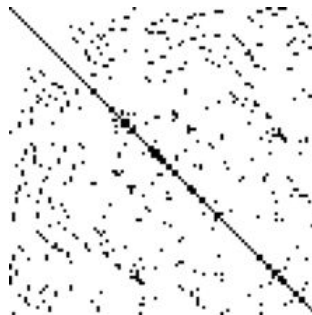
Sparse : scatter diagram - scatter - පැතිරී යනවා/විසිරණය

To examine a graph, first we should represent the graph in a “x y coordinate plane”, in octave, we need to have a matrix to represent our values of x,y coordinates.

Ex: x,y planes



Ex: sparse



Now we are inside the correct folder, we can access the file [makePlanarSparse.m](#) where the function [makePlanarSparse\(N\)](#) is in.

Now execute the command : **[\[A,t,x,y\] = makePlanarSparse\(N\)](#)**

That function will return 4 matrices, which will be assigned to A,t,x,y.

See the output below, it will show you the values of matrix A followed by matrix t, followed by matrix x, followed by matrix y, because it is too large to show all lines (100*100=10000 lines in matrix A, followed by others... how many lines?).

```

>> [A,t,x,y]=makePlanarSparse(N)
A =
Compressed Column Sparse (rows = 100, cols = 100, nnz = 576 [5.8%])

(9, 1) -> 1
(29, 1) -> 1
(49, 1) -> 1
(50, 1) -> 1
(97, 1) -> 1
(31, 2) -> 1
(32, 2) -> 1
(36, 2) -> 1
(51, 2) -> 1
(68, 2) -> 1
(84, 2) -> 1
(100, 2) -> 1
(14, 3) -> 1
(25, 3) -> 1
(28, 3) -> 1
(32, 3) -> 1
(62, 3) -> 1
(75, 3) -> 1
(27, 4) -> 1
(52, 4) -> 1
(56, 4) -> 1
(65, 4) -> 1
(71, 4) -> 1
(72, 4) -> 1
(31, 5) -> 1
(36, 5) -> 1
(70, 5) -> 1
(94, 5) -> 1
(22, 6) -> 1
(41, 6) -> 1
(42, 6) -> 1
(86, 6) -> 1

```

If you need to see other lines enter f , come back to front enter b, to quit enter q.

By pressing f continuously you'll see all lines one day, after that press b continuously, you'll come to the starting point one day. So all lines will be there, simply quit this. Need to quit before giving other command.

```
-- less -- (f)orward, (b)ack, (q)uit
```

Take first line (9,1) --> 1 , that means a value of matrix A. Matrix A is a 100X100 matrix with (x,y) values of the points we use to draw our graph.

r/c	0	1	2	3	4	5	6	7	8	9	.	.	.	99
0	1	1	1	1	1	1	1	1	1	1	.	.	.	1
1	1	1	1	1	1	1	1	1	1	1	.	.	.	1
2	1	1	1	1	1	1	1	1	1	1	.	.	.	1
3	1	1	1	1	1	1	1	1	1	1	.	.	.	1
4	1	1	1	1	1	1	1	1	1	1	.	.	.	1
5	1	1	1	1	1	1	1	1	1	1	.	.	.	1
6	1	1	1	1	1	1	1	1	1	1	.	.	.	1
7	1	1	1	1	1	1	1	1	1	1	.	.	.	1
8	1	1	1	1	1	1	1	1	1	1	.	.	.	1
9	1	1	1	1	1	1	1	1	1	1	.	.	.	1
.
.
.
99	1	1	1	1	1	1	1	1	1	1	.	.	.	1

To check returned matrices A,t,x,y
Just enter and see, like this

```
>> A
A =
Compressed Column Sparse (rows = 100, cols = 100, nnz = 576 [5.8%])

(9, 1) -> 1
(29, 1) -> 1
(49, 1) -> 1
(50, 1) -> 1
(97, 1) -> 1
(31, 2) -> 1
(32, 2) -> 1
(36, 2) -> 1
(51, 2) -> 1
(68, 2) -> 1
(84, 2) -> 1
(100, 2) -> 1
(14, 3) -> 1
(25, 3) -> 1
(28, 3) -> 1
(32, 3) -> 1
(62, 3) -> 1
(75, 3) -> 1
(27, 4) -> 1
(52, 4) -> 1
(56, 4) -> 1
(65, 4) -> 1
(71, 4) -> 1
(72, 4) -> 1
(31, 5) -> 1
(36, 5) -> 1
(70, 5) -> 1
(94, 5) -> 1
(22, 6) -> 1
(41, 6) -> 1
(42, 6) -> 1
(86, 6) -> 1
```

```
>> t
t =
    92    32    87
    92    31    32
    92    46    87
    14     3    32
     2    31    32
    36     2    31
    86    42    54
    86    11    54
    86    11    88
    95    55    54
    76    95    55
    79    95    54
    17    42    54
    17    79    54
    45    33    22
    45    33    38
    41    45    30
    41    45    22
    40    31    94
    47    14    77
    47    24    21
    25    14     3
    25    14    77
    66    24    21
    66    55    21
    68     2    51
     5    31    94
     5    36    31
    70     5    94
    70     5    36
    70    13     8
    70    13    51
    28    75     3
    28    25     3
```

```
>> x
x =
    0.9203402
    0.0250870
    0.4517457
    0.6059691
    0.0640387
    0.5379546
    0.9111393
    0.3766660
    0.8233645
    0.9744637
    0.2690236
    0.3487731
    0.3158094
    0.7125427
    0.6265727
    0.1173806
    0.7663146
    0.4801127
    0.1148727
    0.3358348
    0.9939792
    0.4655044
    0.6875516
    0.9811371
    0.6902241
    0.7302168
    0.5180944
    0.6369755
    0.8249839
    0.4917210
    0.0309333
    0.0108892
    0.2725272
    0.9616987
```

```
>> y
y =
    0.3772625
    0.1387020
    0.0441070
    0.4086074
    0.3424479
    0.8272232
    0.5721001
    0.2618976
    0.4323105
    0.6427166
    0.9882335
    0.4114401
    0.1468472
    0.0071446
    0.6885534
    0.5100475
    0.8357181
    0.5039885
    0.7883355
    0.3570106
    0.0554354
    0.8241990
    0.6863536
    0.1838388
    0.0406047
    0.2706824
    0.4396319
    0.1291855
    0.3433724
    0.5835388
    0.3551843
    0.0363650
    0.8094087
    0.5941681
```

So, what are the matrices `t`, `x` and `y`? Let's read line by line of the code in [makePlanarSparse.m](#)
This is the code:-

```

makePlanarSparse.m
function [A,tri,x,y]=makePlanarSparse(N)

alpha=1.0;
M1=floor(alpha*N);
M2=N-M1;

x=zeros(N,1);
y=zeros(N,1);

x(1:M1)=1.0*rand(M1,1);
y(1:M1)=1.0*rand(M1,1);

x(M1+1:N)=0.5+0.5*rand(M2,1);
y(M1+1:N)=0.5+0.5*rand(M2,1);

tri=delunay(x,y);
A=sparse(zeros(N,N));

for i=1:size(tri,1),
    A(tri(i,1),tri(i,2))=1;
    A(tri(i,1),tri(i,3))=1;
    A(tri(i,3),tri(i,2))=1;
end

A=A+A';
A = 1.0*(A>0);

```

alpha, M1, M2, N, x, y, tri, A are just variable names (here t is known as tri, just another name)

N will be assigned with 100

alpha is 1.0 initially.

M1 = floor(alpha*N)

What is this floor function? You know how to round numbers. See the following table. You can understand what does floor mean. It is a way to round numbers.

Number	Ceil	Round	Floor
To the nearest integer			
7.1	8	7	7
7.6	8	8	7
7.8	8	8	7
To the nearest ten			
91	100	90	90
96	100	100	90
98	100	100	90

- If the number you are rounding is followed by 5, 6, 7, 8, or 9, round the number up. Example: 38 rounded to the nearest ten is 40. ...
- If the number you are rounding is followed by 0, 1, 2, 3, or 4, round the number down. Example: 33 rounded to the nearest ten is 30.
- Floor and ceil also ways of rounding,
 - floor : always round the number down
 - ceil : always round the number up

So M1 is 100

$$M2 = N - M1$$

M2 is 0

x = zeros(N,1)

y = zeros(N,1)

Built-in Function: **zeros(r, c)**

Return a matrix whose elements are all 0.

r : no of rows, c : no of columns

So x, and y are 100x1 matrices with all zeros

Built-in Function: **rand(r,c)**

Return a matrix with random elements

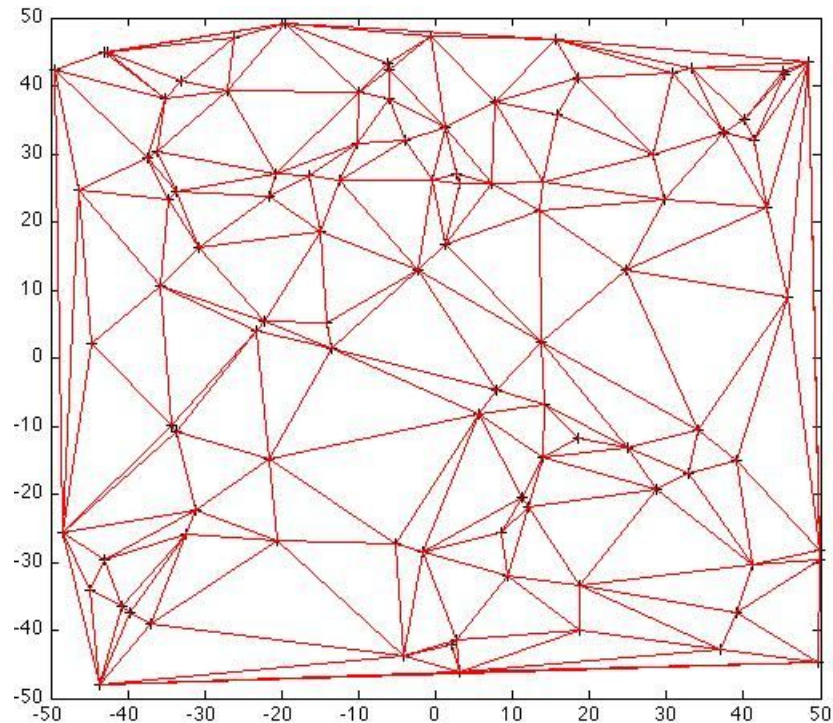
tri = delaunay(x,y)

Built-in Function: delaunay(r,c)

For 2-D sets, the return value `tri` is a set of triangles which satisfies the Delaunay circum-circle criterion, i.e., only a single data point from $[x, y]$ is within the circum-circle of the defining triangle. The set of triangles `tri` is a matrix of size $[n, 3]$. Each row defines a triangle and the three columns are the three vertices of the triangle. The value of $tri(i,j)$ is an index into x and y for the location of the j -th vertex of the i -th triangle.

[\(delaunay\)](#)

2D Example :



[\(3D example for delaunay\)](#)

So, `tri` or `t` is a 100×3 matrix, 100 rows, 3 columns, each row has 3 points of a triangle. Press `t` to see the matrix, and press `f` to see all 100 lines.

A = `sparse(zeros(N,N))`

`sparse(A)` converts a full matrix into sparse form by squeezing out any zero elements. If a matrix contains many zeros, converting the matrix to sparse [storage saves memory](#).
Compress the matrix.

Example:

```

>> p=5
p = 5
>> H = zeros(p,p)
H =

    0    0    0    0    0
    0    0    0    0    0
    0    0    0    0    0
    0    0    0    0    0
    0    0    0    0    0

>> H(2,3)
ans = 0
>> sparse(H)
ans =

Compressed Column Sparse (rows = 5, cols = 5, nnz = 0 [0%])

>> H(2,3)
ans = 0
>> H
H =

    0    0    0    0    0
    0    0    0    0    0
    0    0    0    0    0
    0    0    0    0    0
    0    0    0    0    0

>> |

```

size(tri,1) : gives no of rows of tri

size(tri,2) : gives no of columns of tri

A' : transpose of matrix A

Example :

```

K =

    2    3    5
    2    3    1
    9    1   10

>> K'
ans =

    2    2    9
    3    3    1
    5    1   10

>>

```

This function `makePlanarSparse(N)` will return 4 matrices which will be used to create our graph.

`A` : 100X100 matrix with (x,y) values of points of our graph

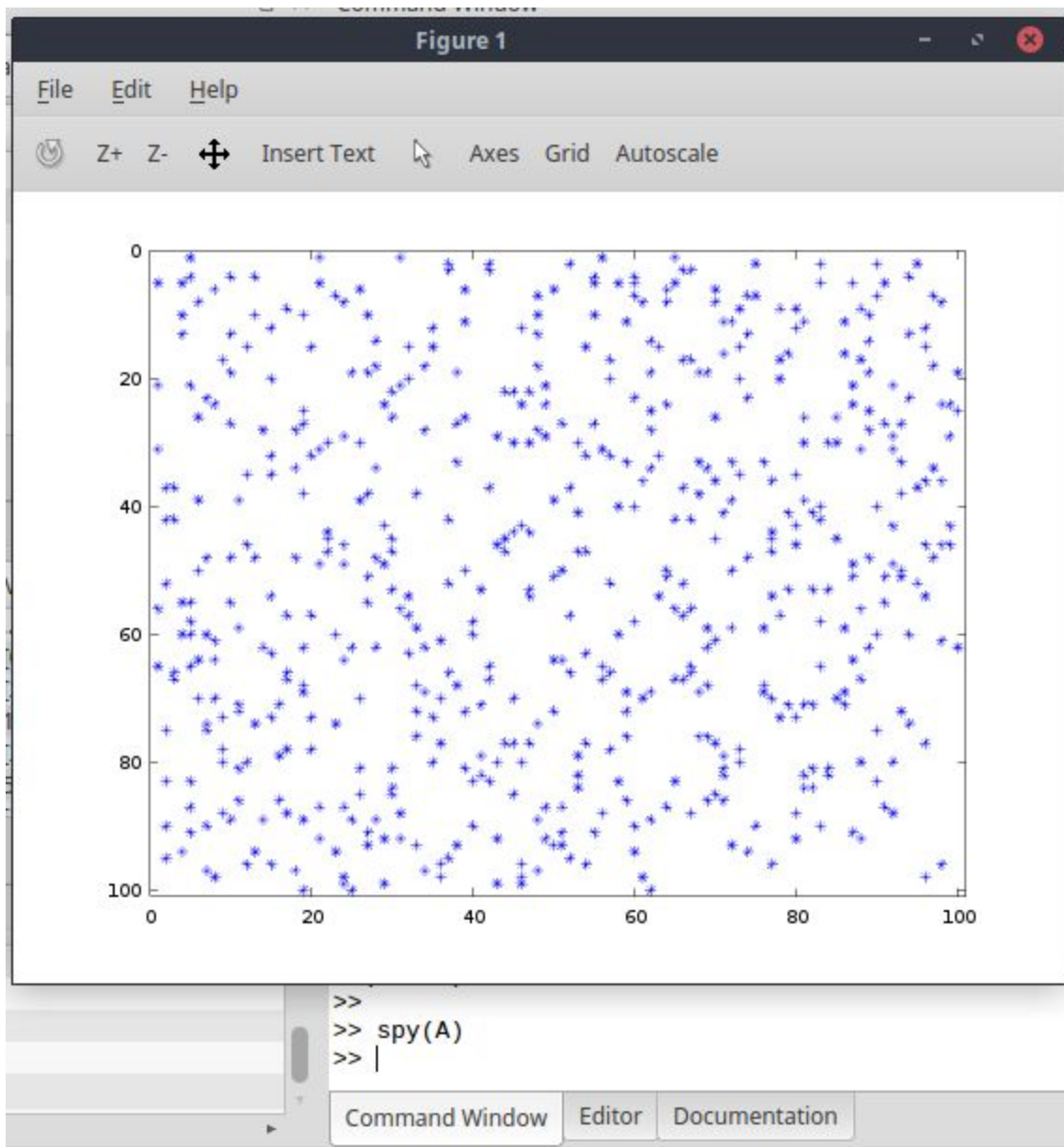
`t` or `tri` : 100X3 matrix with points of triangles

`x` : 100X1 with all zeros ("0"s)

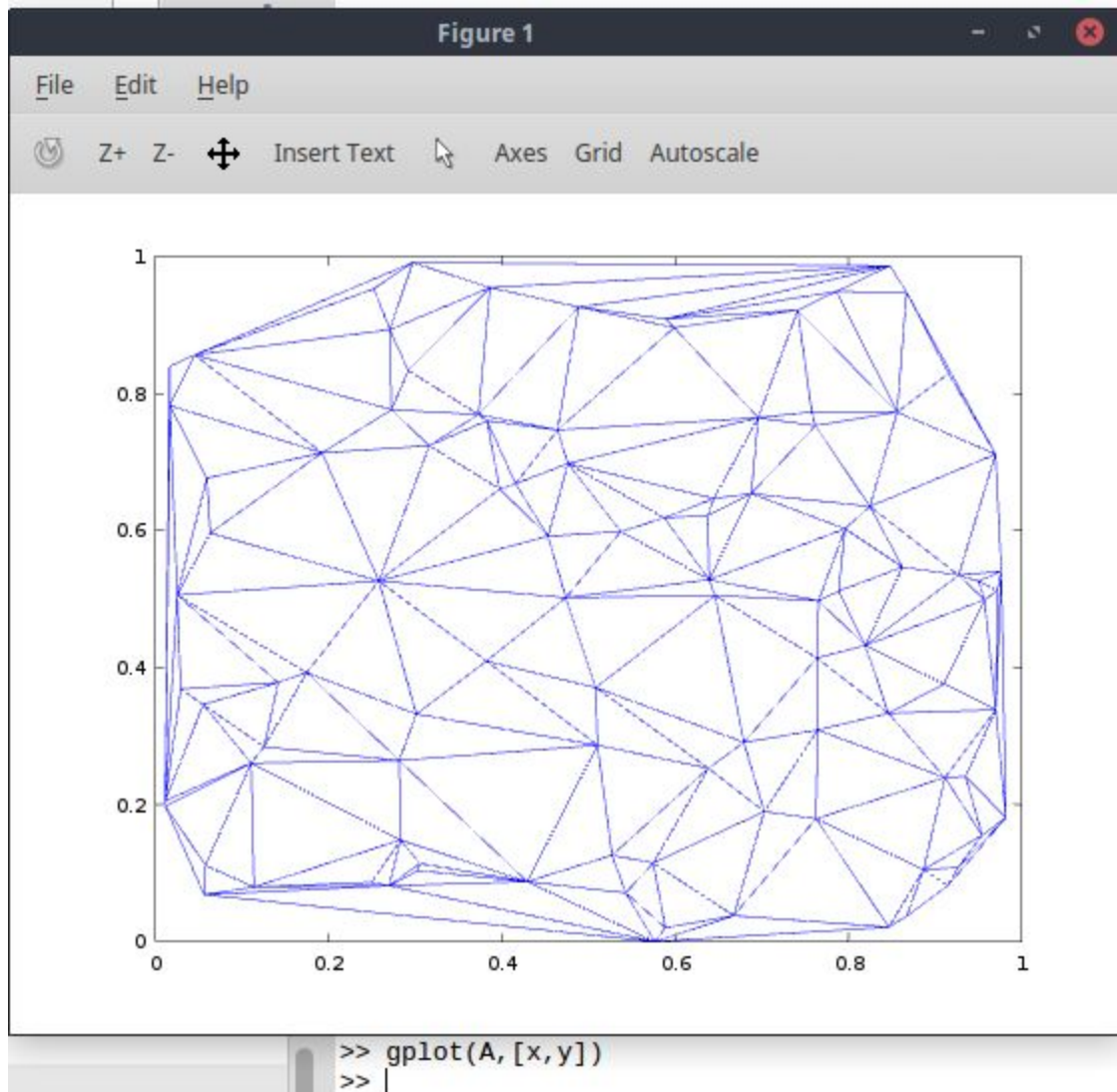
`y` : 100X1 with all zeros ("0"s)

(this `x` and `y` matrices will denote the range of our graph in the 2D planar)

`spy(A)` : by running a command, you can examine the adjacency matrix of the graph. You will get a picture like below.



`gplot(A,[x,y])` : by running this command you can examine the graph itself



Now we have a graph, let's partition the graph now.

There are two algorithms of partitioning a graph.

- Simulated Annealing
- Genetic Algorithm

We can examine the performance of both algorithms using this graph.

Read all comments and understand the code of graphPartitioning.m file with the following guide.

01) Simulated Annealing

Run the Simulated Annealing algorithm to partition the graph into two clusters by using this command:

> **[cut, soln] = graphPartitioning(A, 1, 0);** (this will take some time to run and put back to >>)

This function will be at [graphPartitioning.m](#) file. Open and see.

First few lines of the file :-

```
function [bestEval,bestSoln] = graphPartitioning(A, psize,ga)

% A is the adjacency matrix of the graph to be partitioned
% psize is the size of the population (set psize=1 if using simulated
% annealing)
% ga is set to 1 if a genetic algorithm is to be used and set to 0 if
% simulated annealing is to be used
```

Still we don't know what happened with the above command. Let's see.

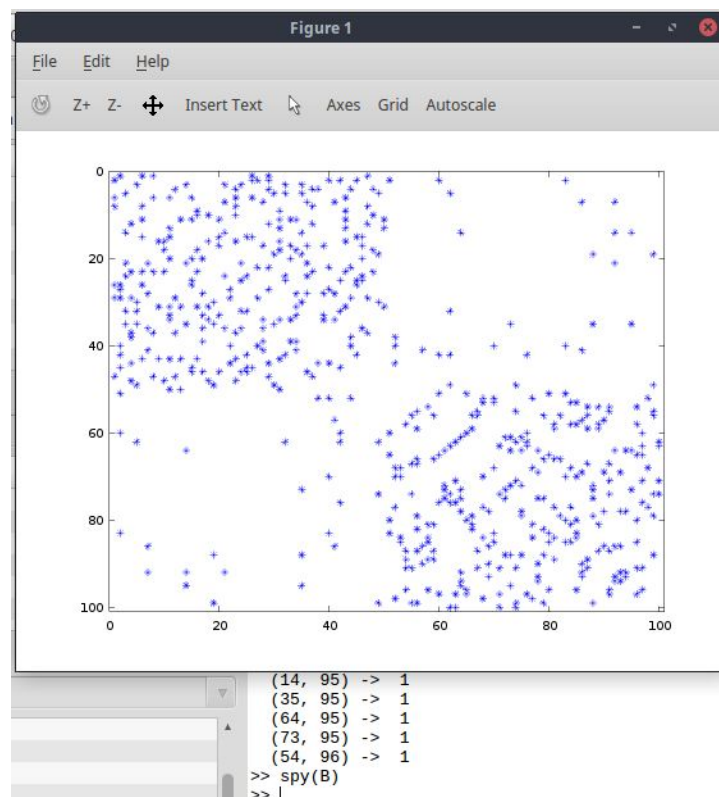
If we reorganize the matrix according to the partition just computed, we can **see the effect of the partitioning**:

> **[c,indx]=sort(soln);** (this will output c and indx matrices, quit it and give next command)

> **B = A(indx, indx);** (this will output B matrix, quit it and run;)

> **spy(B) ;**

The picture may look something like:



B is also an adjacency matrix of the graph, but the rows and columns of B have been organised so that the first $N/2$ rows and columns correspond to partition 0 and the rest to partition 1. If the soln is a good partition of A, then we can expect to see two dense blocks of nodes on the diagonal of B, with relatively fewer nodes on the two off-diagonal blocks.

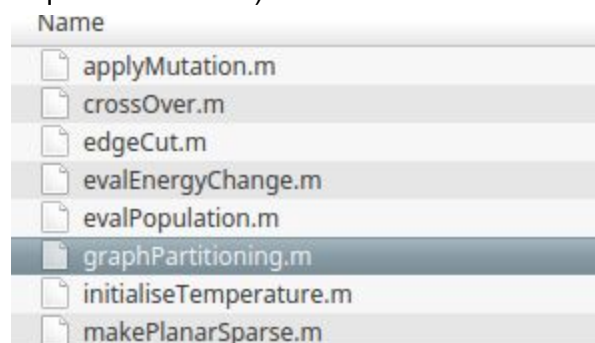
Examining the impact of the parameters to the efficiency of the SA algorithm.

1) No of generations

As written, the Simulated Annealing is run for a total number of generations specified by the **maxGenerations** variable on line 61 of the **graphPartitioning.m** code. We would like to know how this parameter affects the run-time of the code and the quality of the solution obtained.

```
60  
61     maxGenerations=100000;  
62     maxChainLength=1000;  
63     alpha=0.8;
```

(Open all files on Octave , so you can see line numbers clearly. As you are in the correct folder, all its files will be shown on the **left side of octave GUI**, right click the file you want so it will open, you can do any changes required and save.)



At line 129 of the code there is a statement to write the current best solution out to the console. It is written out once for each generation.

Let's write this into a file instead, so that we can plot how the best solution improves from one generation to the next.

```
124         itersWithoutImprovement=0;  
125         markovChainLength=0;  
126     end  
127  
128  
129     % fprintf('%d\n',bestEval);  
130 end  
131  
132
```

What you have to do is, Insert at line 28 a statement to open a file called **besteval.txt**.

fp = fopen('besteval.txt','w');

```

22
23 % evalEnergyChange : a function to evaluate the change in
24 % change in the evaluation function) induced by the given
25
26 % applyMutation : a function to apply a random mutation
27
28 fp = fopen('besteval.txt','w');
29
30
31 N = size(A,1);
32
33 % For the GA, the crossover operator will create new cl

```

Now change line 129 so that it writes the best quality solution found after each generation to the file: **fprintf(fp,'%d\n',bestEval);**

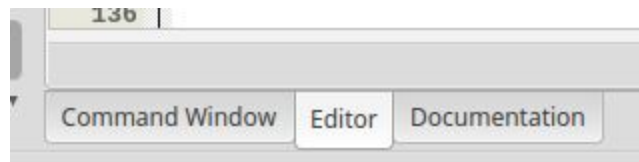
Finally, at line 131 make sure the file is closed when the function exits:
fclose(fp);

```

124         itersWithoutImprovement=0;
125         markovChainLength=0;
126     end
127
128
129     fprintf(fp,'%d\n',bestEval);
130 end
131 fclose(fp);
132
133

```

(you can swap to the command window from the tab below.)



Now in command window, when the function is executed:

> [cut, soln] = graphPartitioning(A, 1, 0); (this will take some time, wait for >>)

it generates the besteval.txt file in your working folder.

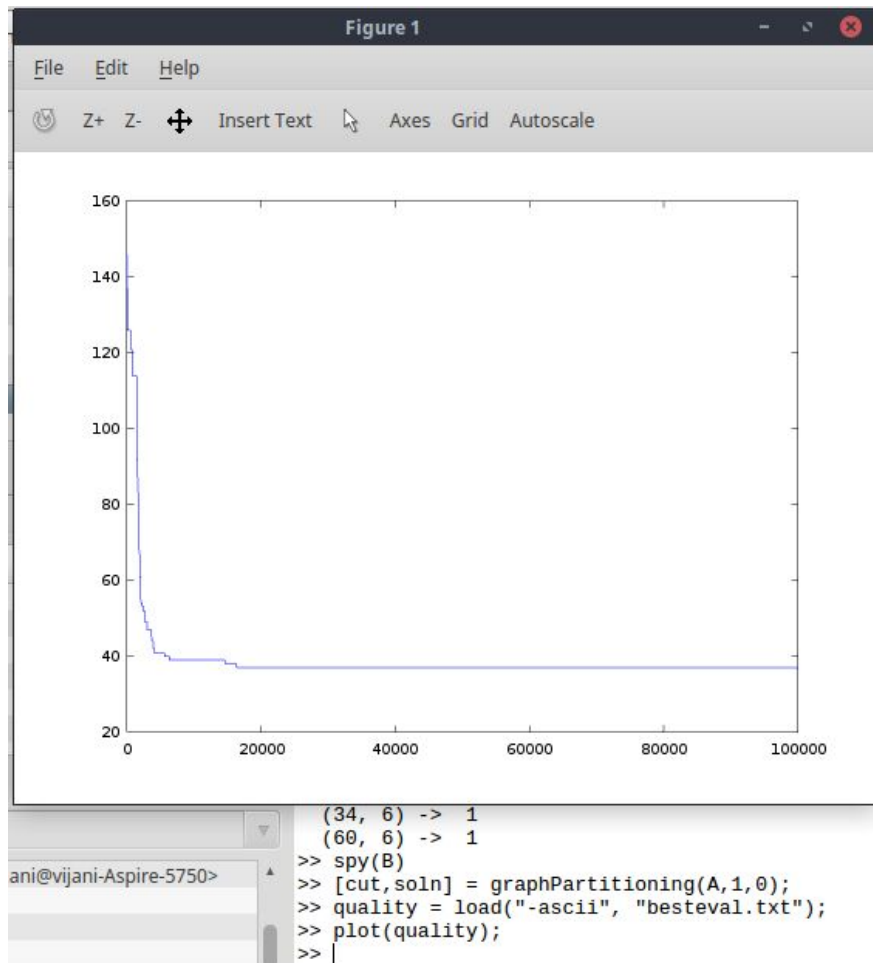
(in this file, it will show every quality values for each generation of the range where maxGenerations is 100000 as given at line 61)

Let's plot what is contained in that file:

> quality = load("-ascii", "besteval.txt");

> plot(quality) ;

We now see a plot of the different quality obtained after each generation:



→ Here you can see that, the quality is decreased as the no of generations is increased, describe how it has decreased. (at first decreases rapidly then looks like it stays constant in large no of generations)

Now, let's change the code so that it will record the time it takes for each generation.

At line 29 in `graphPartitioning.m` insert a line to open a file in which times will be recorded:

`fp2=fopen('time.txt','w');`

```

26 % applyMutation : a function to ap
27
28 fp = fopen('besteval.txt', 'w');
29 fp2 = fopen('time.txt', 'w');
30
31 N = size(A,1);

```

At line 80, insert a line to start a timer:

`tic;`

```

78 % generations
79 % used instea
80 tic;
81 bestEval = in
82 for i=1:maxGe
83

```

At line 128 insert a line to record the time each generation took:

fprintf(fp2,'%f\n',toc);

```
125         markovChainLength=0;
126     end
127
128     fprintf(fp2, '%f\n', toc);
129     fprintf(fp, '%d\n', bestEval);
130 end
131 fclose(fp);
132 fclose(fp2);
133
```

Save the changes. Now run the code again: (switch back to command window)

> [cut, soln] = graphPartitioning(A, 1, 0); (wait till it run and gives back >>)

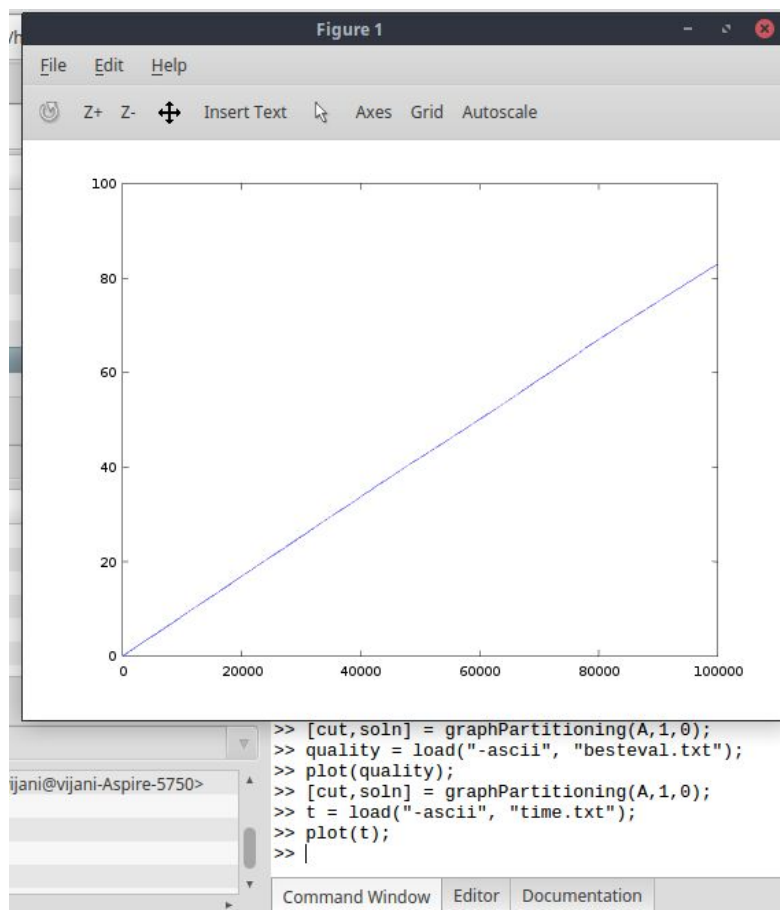
(this will create time.txt in your working folder)

Read in the times that have been gathered:

> t = load("-ascii", "time.txt");

We can plot the time that had elapsed by the end of each generation:

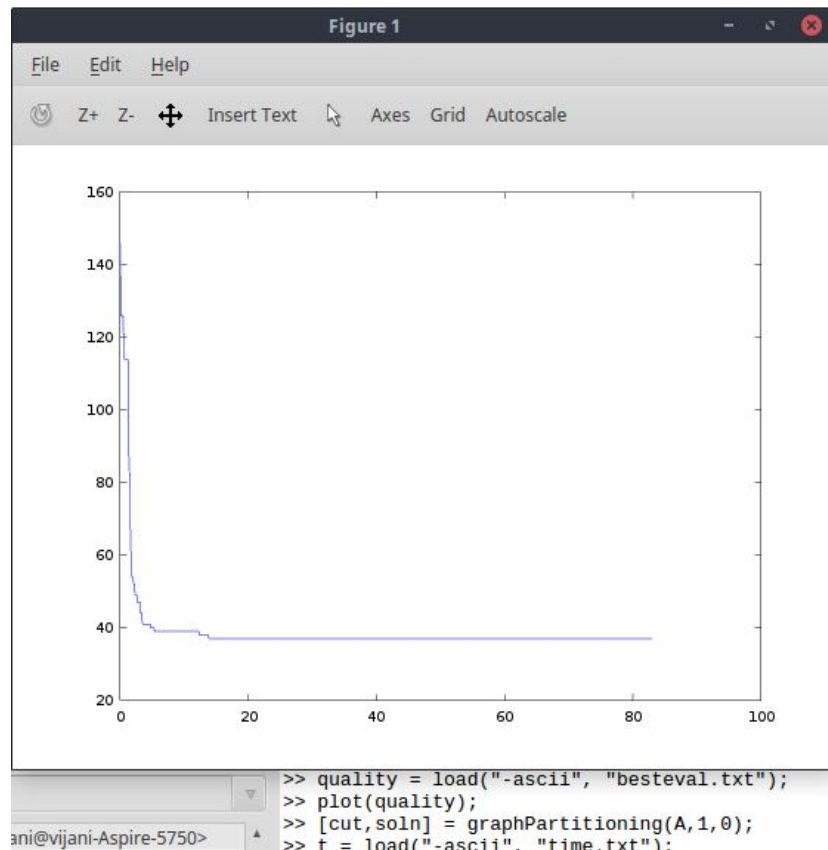
> plot(t) ;



→ You can see that, the time taken is increased as the no of generations is increased. It is like a x=y graph. (Explain in your words.)

It is more interesting to plot the time against the quality, as this shows us the trade-off directly – by how much does the quality improve the longer the algorithm is run:

```
> plot(t, quality) ;
```



Here time is x-axis, quality is y-axis. Quality is decreased as time taken is increased. (at first decreases rapidly then looks like it stays constant when time taken is high)

Dealing with Randomness

If you carry all the above steps out another time, you will get different answers.

This is because the algorithm has randomness in it. It is better therefore to get the average performance over a number of runs (say, 10, but the more the better). Below shows what to do to average over 3 runs (this could be done much more easily by writing an Octave script).

```
> [cut, soln] = graphPartitioning(A, 1, 0);  
> t1 = load("-ascii", "time.txt");  
> q1 = load("-ascii", "besteval.txt");  
> [cut, soln] = graphPartitioning(A, 1, 0);  
> t2 = load("-ascii", "time.txt");  
> q2 = load("-ascii", "besteval.txt");
```

```

> [cut, soln] = graphPartitioning(A, 1, 0);
> t3 = load("-ascii", "time.txt");
> q3 = load("-ascii", "besteval.txt");
> q = (q1+q2+q3)/3;
> t = (t1+t2+t3)/3;
> plot(t, q);

```

1) effect of α (alpha)

All of the above steps were carried out for a particular value of α that is set at **line 63 in the code** of the file graphPartitioning.m

```
alpha=0.8;
```

Change the value of α and carry out the analysis of the number of generations again.

Pick a value for alpha **[0.3 , 0.5, 0.7, 0.8 , 0.99 , 1.0]**

N = 100, alpha = x, MaxGen = 100000, chainLen(Proportion value)=1000(0.01)

I am going to run 3 times for each alpha value and take average to analysis. Use following commands;

Run this only once, because you are going to apply algorithm to the same graph.

```

N=100;
[A,t,x,y]=makePlanarSparse(N);

```

To run SA algorithm for different alpha values

```

[cut,soln]=graphPartitioning(A,1,0);
t1=load("-ascii", "time.txt");
q1=load("-ascii", "besteval.txt");
[cut,soln]=graphPartitioning(A,1,0);
t2=load("-ascii", "time.txt");
q2=load("-ascii", "besteval.txt");
[cut,soln]=graphPartitioning(A,1,0);
t3=load("-ascii", "time.txt");
q3=load("-ascii", "besteval.txt");
t=(t1+t2+t3)/3;
q=(q1+q2+q3)/3;
plot(t);
plot(q);
plot(t,q);

```

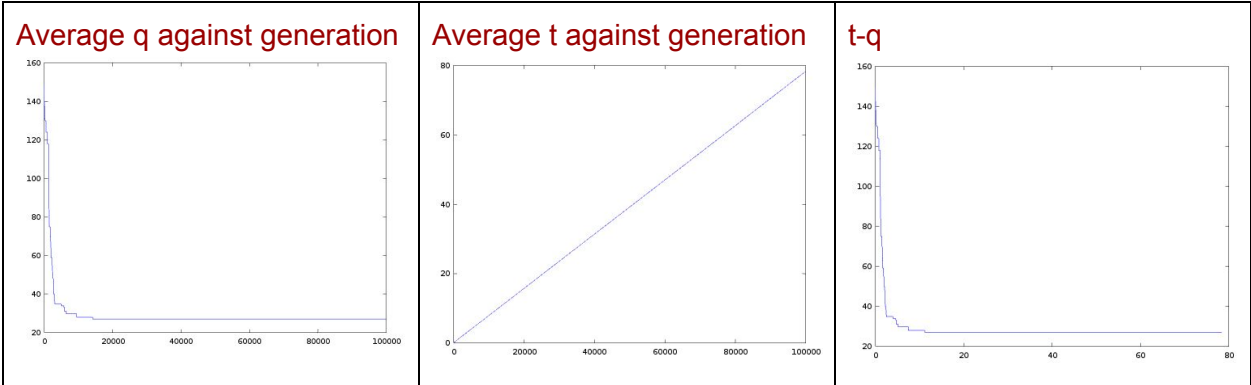
N = 100, **alpha = 0.3**, MaxGen = 100000, chainLen(Proportion value)=1000(0.01)

Average q against generation	Average t against generation	t-q
------------------------------	------------------------------	-----

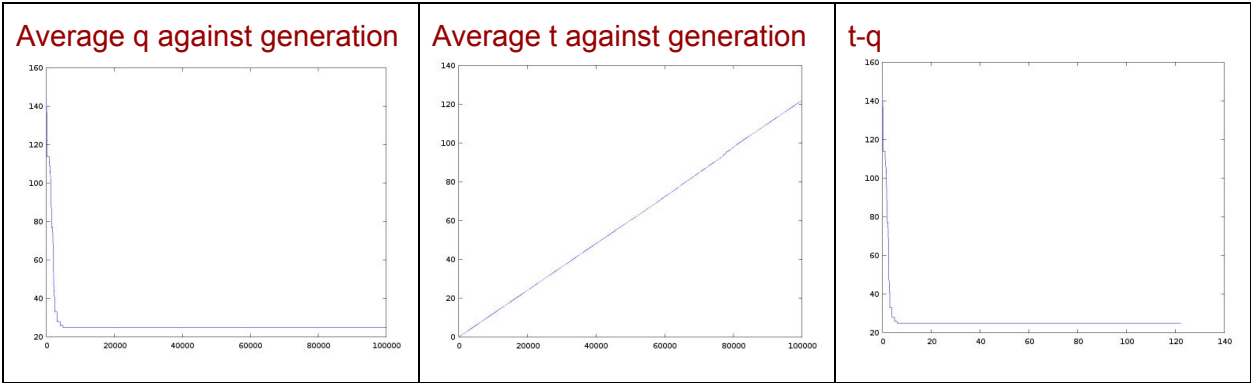
N = 100, **alpha = 0.5**, MaxGen = 100000, chainLen(Proportion value)=1000(0.01)

Average q against generation	Average t against generation	t-q
------------------------------	------------------------------	-----

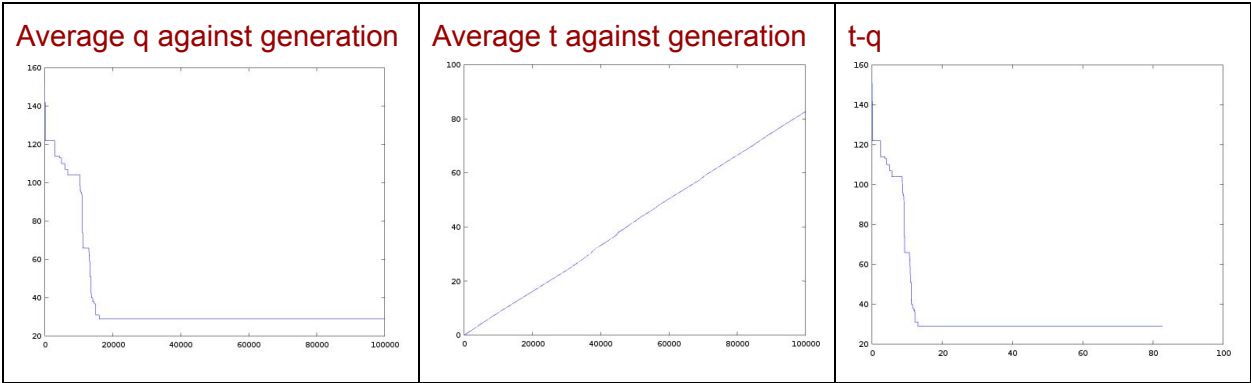
N = 100, **alpha = 0.7**, MaxGen = 100000, chainLen(Proportion value)=1000(0.01)



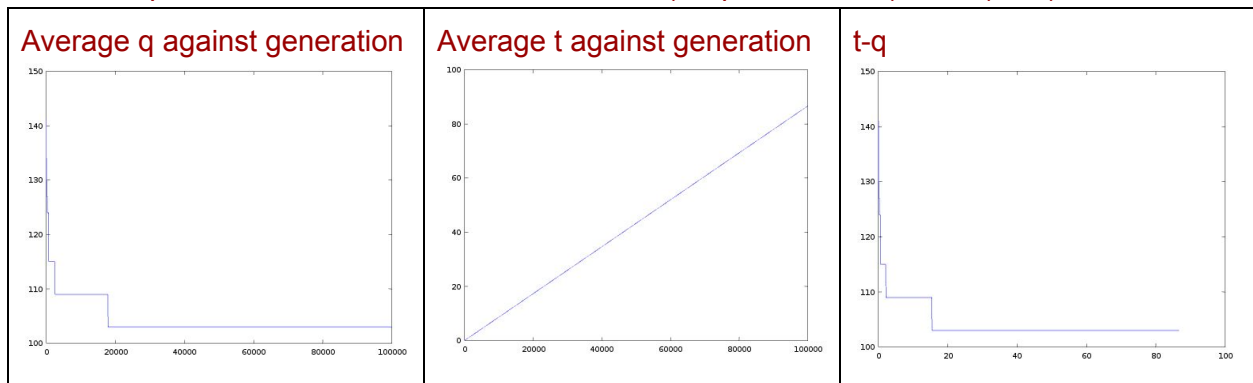
N = 100, **alpha = 0.8**, MaxGen = 100000, chainLen(Proportion value)=1000(0.01)



N = 100, **alpha = 0.99**, MaxGen = 100000, chainLen(Proportion value)=1000(0.01)



N = 100, alpha = 1.0, MaxGen = 100000, chainLen(Proportion value)=1000(0.01)



→ According to the above charts when Alpha increases, time taken decreases for larger no of generations and quality is also increases when having higher alpha values.

Try to answer the following questions:

1. Is there an α value that you would recommend as clearly the best for this Problem?

Higher alpha value, because our objective is to increase the performance of the algorithm.

2. Is the convergence profile different for different values of α ?

i.e. for some values of α , does the quality reduce more rapidly over the first few iterations or settle down to a fixed value earlier/later than other values of α ?

Yes, it is shown in above charts, for lower values of alpha, the quality reduces more rapidly over the first iterations and settles down to a fixed value earlier than the higher values of alpha.

3. What happens when α is small e.g. $\alpha < 0.5$

This should be “alpha<0.8” as I think, because if alpha is 0.5 it gives an error “dividing by zero” because not going with other parameter values.

For lesser values of alpha, the quality reduces more rapidly. Time taken is high for larger no of generations.

4. What happens when α is large e.g. $\alpha > 0.99$

the quality does not reduce more rapidly (reduces slowly) over the first iterations and settles down to a fixed value for some time and again reduces and finally settles down to a final fixed value later than the lower values of alpha. Time taken reduces for large no of generations.

2) effect of MarkovChainLength

The Markov Chain Length is the number of iterations that the Simulated Annealing algorithm spends at any particular value of the temperature.

Choose the best α from your above analysis. (choose a higher value for alpha, I will take 0.99)

Now consider different values of the chain length, which can be set at line 62 of the code.

Rather than set the chain length a particular number, set it as a proportion of the overall maximum number of generations e.g.

```
maxChainLength=ceil(0.1*maxGenerations);
```

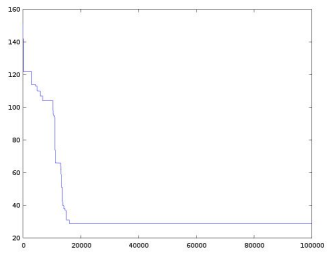
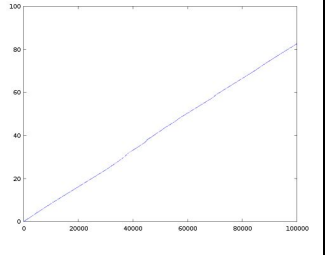
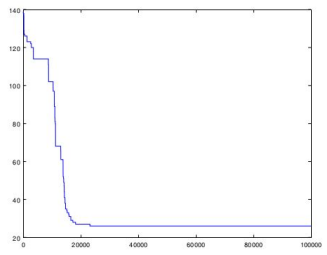
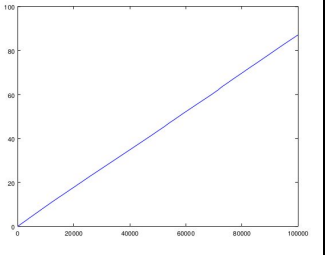
Try different values for the proportion from the list [0.01, 0.05, 0.1, 0.2, 0.3, 0.4.]
 (If maxGenerations=100000 and maxChainLength=1000; line is uncommented, 0.01 is used)

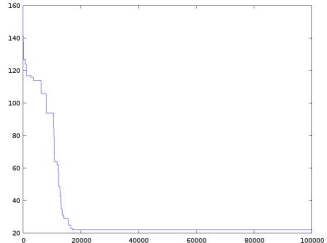
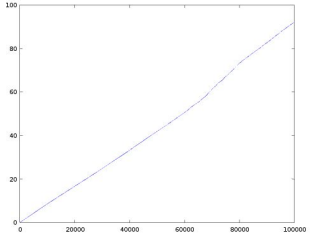
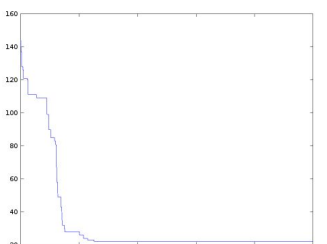
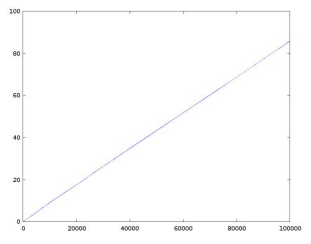
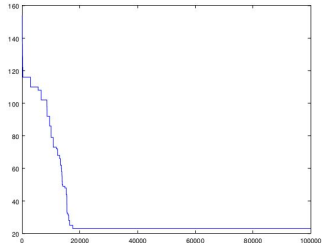
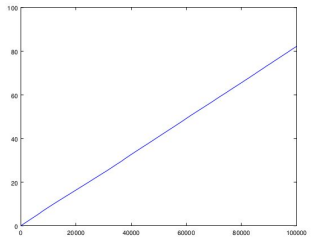
```

61 maxGenerations=100000;
62 %maxChainLength=1000;
63 maxChainLength=ceil(0.05*maxGenerations);
64 alpha=0.99;
65

```

N=100, alpha=0.99, MaxGen=100000, chainLen(Proportion value) =x

N	α (alpha)	MaxGen	chainLen (Proportion value)	Quality	Time
100	0.99	100000	1000 (0.01)		
			5000 (0.05)		

			10000 (0.1)		
			20000 (0.2)		
			30000 (0.3)		
			40000 (0.4)		

Answer the following question:

1. Is there a best proportion for the Markov Chain length, taking into account that we are interested in the time as well as the quality of the solution.

I can see no any improvement (no new best solution/proportion) being found in entire Markov chain at one alpha value (temperature).

3) effect of N

Carry out all the above analysis again for different values of N . The range of N that **you choose depends on the speed and memory of your computer**, but it would be interesting to try values of N that are much bigger than N = 100

e.g. N = 1000, N = 10000 or N = 100, 000.

Pick at least 5 different values of N .

You may need to run the problem for a much greater number of generations when N is large. Use the spectralCut function to give you an idea of whether the algorithm has converged to a good solution i.e. run

```
> cut = spectralCut(A);
```

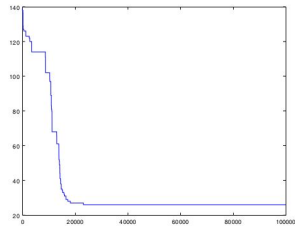
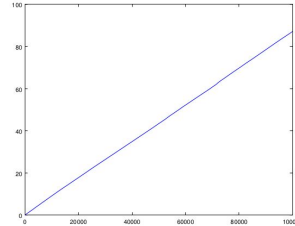
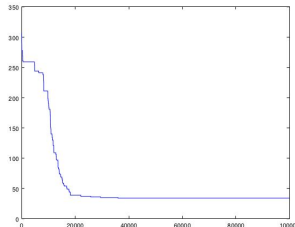
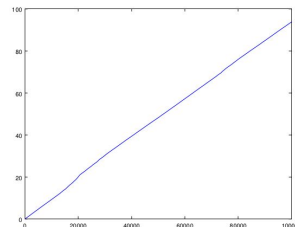
```
> cut
```

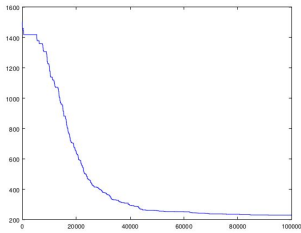
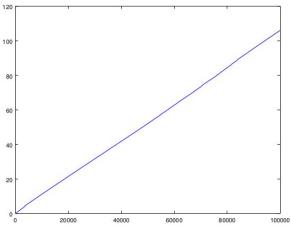
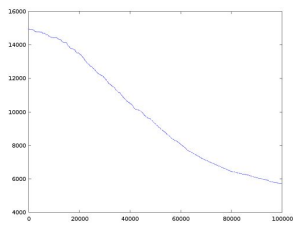
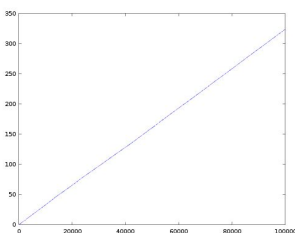
If the cut value obtained by the simulated annealing algorithm is much greater than the cut value obtained by the spectralCut algorithm, then you need to run for a bigger number of generations.

Suggestion for the table

MaxGen = 100,000 and

chainLen(Propotion value) = 5000(0.05)

N	α	Quality	Time
100	0.7		
	0.8		
	0.99		
200	0.7		
	0.8		
	0.99		
300	0.7		
	0.8		
	0.99		

1000	0.7		
	0.8		
	0.99		
10,000	0.7		
	0.8		
	0.99		
100,000	0.7		
	0.8		
	0.99		

1. Record for each N the best value of α found in your analysis.
2. Discuss the question of whether the best value of α depends on N .

Answer these two questions from the above table, if you need, take another alpha value for each N and take output.

02) Genetic Algorithm

1) Cross over and Mutation

From the beginning, we fixed $N=100$.

Now fix a population size, say

> **P = 10;**

Now create a graph for size N

To run a genetic algorithm with this population size, do

> **[cut, soln] = graphPartitioning(A, P, 1);**

For this population size, try to work out the best crossover rate and mutation rate to use for the genetic algorithm.

The crossover rate and mutation rate are set at **lines 57 and 58 in the code** of the file **graphPartitioning.m**

Like this :

```
53 % Some more parameters
54
55 % here a fixed number of crossovers is chosen. Alternatively a
56 % probability of crossover could be set using roulette wheel approach
57 numCrossOvers=floor(psize*0.5);
58 numMutations=floor(psize*0.1);
59 numMutations = max(numMutations, 1);
```

Pick from these 5 crossover rates **[0.0 , 0.2 , 0.4 , 0.5 , 0.6]**

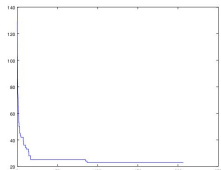
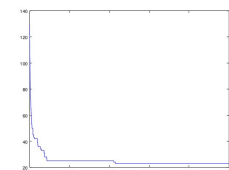
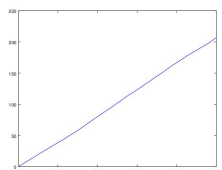
Pick from these 2 mutation rates **[0.0 , 0.1]**

This gives in total **10 different parameter settings** for the algorithm.

As was previously done for simulated annealing, **plot the quality obtained against the time** for each combination of parameter setting.

Table suggestion

N=100 and P=10

Mutation %	CrossOver %	q against t graph	Quality	Time (s)
0.0	0.0			
	0.2			
	0.4			
	0.5			
	0.6			
0.1	0.0			
	0.2			

	0.4			
	0.5			
	0.6			

Discuss the following questions:

- 1. What is the best combination of crossover and mutation rate?**
- 2. Which is more effective on this problem – crossover or mutation?**
- 3. How do the results compare with simulated annealing? (Remember to take into account not just the final quality obtained but also the time it took to get it)**

2) Population Size

Choose P from this list : [10, 2, 50]

First we fixed population size as

>P=10;

Repeat the analysis of the previous section for two other population sizes : P=2 and P=50.

Fix your crossover rate and mutation rate according to the best values that you obtained previously.

Table suggestion

Best Mutation % is “....best value you found.....”

Best Crossover % is “....best value you found.....”

N =100

P	q against t graph	Quality	Time
10			
2			
50			

Try to answer the following questions:

1. Does the best crossover rate and mutation rate change for different population sizes?
2. Is it better to have a small population or a large population?

3) Graph Size

Fix your crossover rate and mutation rate **according to the best values that you obtained previously.**

Now, let's examine whether the **best population size, P** depends on the size of the graph N.

Run the program for the three different population sizes and three different values of N.

Pick N from this list : [100, 1000, 10,000]

(depending on speed of your computer).

Table Suggestion

Best Mutation % is “....best value you found.....” and

Best Crossover % is “....best value you found.....”

P	N	q against t graph	Quality	Time
10	100			
	1000			
	10,000			
2	100			
	1000			
	10,000			
50	100			
	1000			
	10,000			

Answer the question:

1. Does the best population size P depend on the size of the problem N.

My machine is Core i3, so I selected N values, 100, 200, 300 because it takes 6min max to execute one round. Try N = 100,000 if you have core i5 machine.

-

[Recommended study material](#)

There is a pdf, google for it, this will give you a complete understanding of Octave.

GNU Octave

A high-level interactive language for numerical computations

Edition 4 for Octave version 4.2.0

November 2016

By;

John W. Eaton

David Bateman

Søren Hauberg

Rik Wehbring