Sumy State University

Department

of

Applied Mathematics and Complex Systems Modeling

Report on laboratory work №1

Subject

Computing system

Student: Olena Poroskun

Teacher: Igor A. Knyaz

Sumy, Sumy region

2020

```
%  Пороскун Олена ПМ-81
%       Homework 1

% 1. Scalar variables.
a = 10
b = 2.5*10^23
c = 2 + 3*i
d = exp(j*2*pi/3)

% 2. Vector variables.
aVec = [3.14 15 9 26]
bVec = [2.71; 8; 28; 182]
cVec = 5:-0.2:-5
dVec = logspace(0,1,100)
eVec = "Hello"

% 3. Matrix variables.
aMat = 2*ones(9,9)
v = [1 2 3 4 5 4 3 2 1]
bMat = diag(v)
mas_c = 1:100;
cMat = reshape(mas_c, 10, 10)
dMat = nan (3,4)
eMat = [13, -1, 5;  -22, 10,-87]
fMat = randi([-3 3], 5, 3)

% 4. Scalar equations.
x = 1/(1+exp(-(a-15)/6))
y = (sqrt(a)+ nthroot(b, 21))^pi
z = (log(real((c+d)*(c-d))*sin(a*pi/3)))/(c* conj(c))

% 5. Matrix equations.
xMat = (aVec * bVec)*((aMat).^2)
yMat = (bVec * aVec)
zMat = (det(cMat))*(aMat * bMat)
zMat = det(cMat)

```

```
27      % 4. Scalar equations.
28 -    x = 1/(1+exp(-(a-15)/6))
29 -    y = (sqrt(a)+ nthroot(b, 21))^pi
30 -    z = (log(real((c+d)*(c-d))*sin(a*pi/3)))/(c* conj(c))
31
32      % 5. Matrix equations.
33 -    xMat = (aVec * bVec)*((aMat).^2)
34 -    yMat = (bVec * aVec)
35 -    zMat = (det(cMat))*(aMat * bMat)
36 -    zMat = det(cMat)
37
38      % 6. Common functions and indexing.
39
40 -    cSum = sum(cMat)
41
42 -    eMean = mean(eMat')
43 -    eMean'
44
45 -    eMat(1, :) = 1
46
47 -    cMat
48 -    cSub = cMat(2:9, 2:9)
49
50 -    lin = 1:20
51 -    lin(mod(lin, 2) == 0) = (-1)*lin(mod(lin, 2) == 0)
52
53 -    r = rand(1,5)
54 -    r(find(r<0.5)) = 0
55
56      % 7. Plotting multiple lines and colors.
57
58      % 8. Manipulating variables.
59
```

```
a =

    10


b =

   2.5000e+23


c =

   2.0000 + 3.0000i


d =

  -0.5000 + 0.8660i


aVec =

    3.1400   15.0000    9.0000   26.0000


bVec =

    2.7100
    8.0000
   28.0000
  182.0000
```

```
cVec =

  Columns 1 through 12

    5.0000    4.8000    4.6000    4.4000    4.2000    4.0000    3.8000    3.6000    3.4000    3.2000    3.0000    2.8000

  Columns 13 through 24

    2.6000    2.4000    2.2000    2.0000    1.8000    1.6000    1.4000    1.2000    1.0000    0.8000    0.6000    0.4000

  Columns 25 through 36

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

  Columns 37 through 48

   -2.2000   -2.4000   -2.6000   -2.8000   -3.0000   -3.2000   -3.4000   -3.6000   -3.8000   -4.0000   -4.2000   -4.4000

  Columns 49 through 51

   -4.6000   -4.8000   -5.0000
```

```
dVec =

  Columns 1 through 12

    1.0000    1.0235    1.0476    1.0723    1.0975    1.1233    1.1498    1.1768    1.2045    1.2328    1.2619    1.2915

  Columns 13 through 24

    1.3219    1.3530    1.3849    1.4175    1.4508    1.4850    1.5199    1.5557    1.5923    1.6298    1.6681    1.7074

  Columns 25 through 36

    1.7475    1.7886    1.8307    1.8738    1.9179    1.9630    2.0092    2.0565    2.1049    2.1544    2.2051    2.2570

  Columns 37 through 48

    2.3101    2.3645    2.4201    2.4771    2.5354    2.5950    2.6561    2.7186    2.7826    2.8480    2.9151    2.9836

  Columns 49 through 60

    3.0539    3.1257    3.1993    3.2745    3.3516    3.4305    3.5112    3.5938    3.6784    3.7649    3.8535    3.9442

  Columns 61 through 72

    4.0370    4.1320    4.2292    4.3288    4.4306    4.5349    4.6416    4.7508    4.8626    4.9770    5.0941    5.2140

  Columns 73 through 84

    5.3367    5.4623    5.5908    5.7224    5.8570    5.9948    6.1359    6.2803    6.4281    6.5793    6.7342    6.8926

  Columns 85 through 96

    7.0548    7.2208    7.3907    7.5646    7.7426    7.9248    8.1113    8.3022    8.4975    8.6975    8.9022    9.1116

  Columns 97 through 100

    9.3260    9.5455    9.7701   10.0000
```

```
eVec =

    "Hello"


aMat =

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


v =

     1     2     3     4     5     4     3     2     1
```

```
bMat =

     1     0     0     0     0     0     0     0     0
     0     2     0     0     0     0     0     0     0
     0     0     3     0     0     0     0     0     0
     0     0     0     4     0     0     0     0     0
     0     0     0     0     5     0     0     0     0
     0     0     0     0     0     4     0     0     0
     0     0     0     0     0     0     3     0     0
     0     0     0     0     0     0     0     2     0
     0     0     0     0     0     0     0     0     1

cMat =

     1    11    21    31    41    51    61    71    81    91
     2    12    22    32    42    52    62    72    82    92
     3    13    23    33    43    53    63    73    83    93
     4    14    24    34    44    54    64    74    84    94
     5    15    25    35    45    55    65    75    85    95
     6    16    26    36    46    56    66    76    86    96
     7    17    27    37    47    57    67    77    87    97
     8    18    28    38    48    58    68    78    88    98
     9    19    29    39    49    59    69    79    89    99
    10    20    30    40    50    60    70    80    90   100


dMat =

   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN


eMat =

    13    -1     5
   -22    10   -87


fMat =

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


x =

    0.3029
```

```
x =

    0.3029


y =

   6.2696e+03


z =

    0.1046


xMat =

   1.0e+04 *

    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450
    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450
    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450
    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450
    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450
    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450
    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450
    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450
    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450    2.0450


yMat =

   1.0e+03 *

    0.0085    0.0406    0.0244    0.0705
    0.0251    0.1200    0.0720    0.2080
    0.0879    0.4200    0.2520    0.7280
    0.5715    2.7300    1.6380    4.7320


zMat =

     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     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     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     0     0     0     0     0     0
```

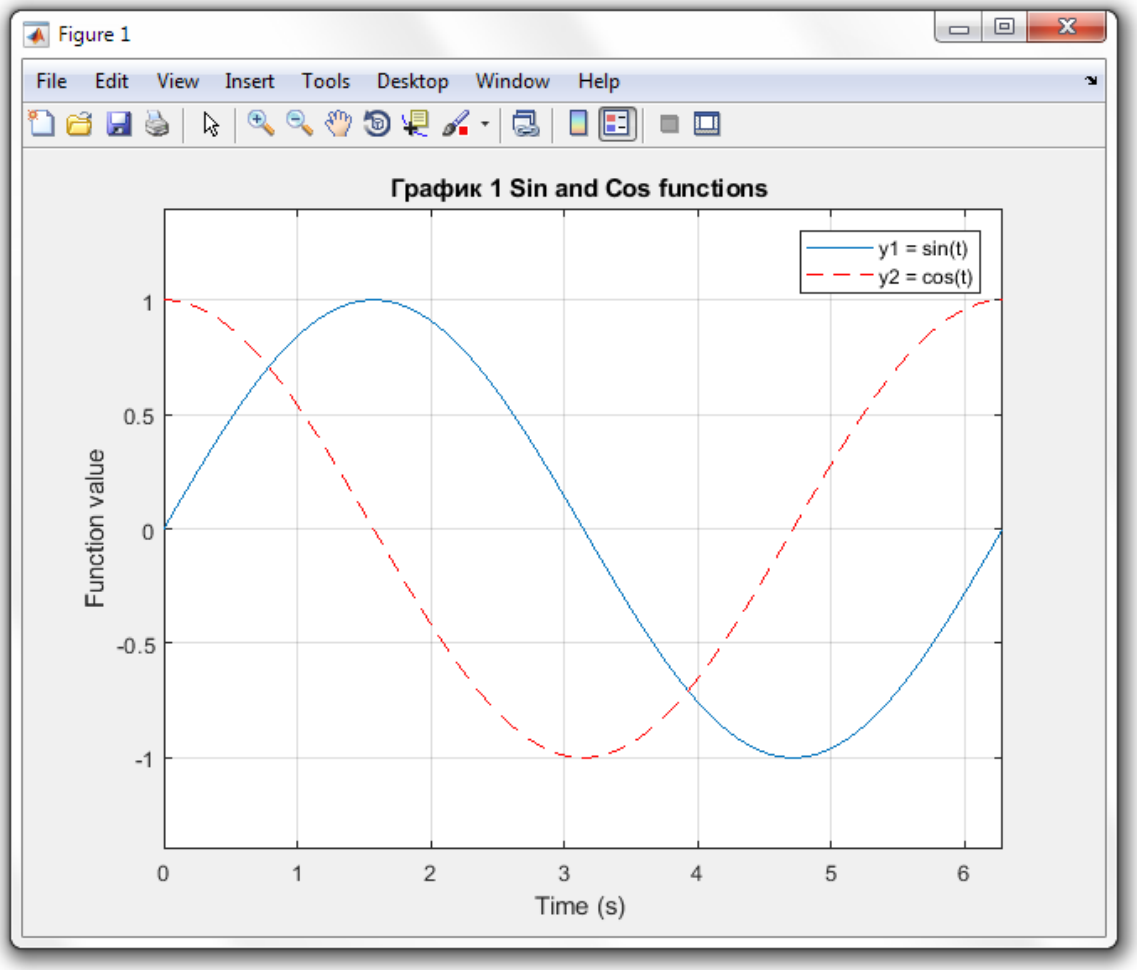
```
zMat =

     0


cSum =

    55   155   255   355   455   555   655   755   855   955


eMean =

    5.6667  -33.0000


ans =

    5.6667
  -33.0000


eMat =

     1     1     1
   -22    10   -87
```

```
cMat =

     1    11    21    31    41    51    61    71    81    91
     2    12    22    32    42    52    62    72    82    92
     3    13    23    33    43    53    63    73    83    93
     4    14    24    34    44    54    64    74    84    94
     5    15    25    35    45    55    65    75    85    95
     6    16    26    36    46    56    66    76    86    96
     7    17    27    37    47    57    67    77    87    97
     8    18    28    38    48    58    68    78    88    98
     9    19    29    39    49    59    69    79    89    99
    10    20    30    40    50    60    70    80    90   100


cSub =

    12    22    32    42    52    62    72    82
    13    23    33    43    53    63    73    83
    14    24    34    44    54    64    74    84
    15    25    35    45    55    65    75    85
    16    26    36    46    56    66    76    86
    17    27    37    47    57    67    77    87
    18    28    38    48    58    68    78    88
    19    29    39    49    59    69    79    89
```

```
lin =

     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15    16    17    18    19    20


lin =

     1    -2     3    -4     5    -6     7    -8     9   -10    11   -12    13   -14    15   -16    17   -18    19   -20


r =

    0.8235    0.6948    0.3171    0.9502    0.0344


r =

    0.8235    0.6948         0    0.9502         0
```

twoLinePlot.m

```
% 7. Plotting multiple lines and colors.
f1 = figure
%t = 0:0.1:2*pi;
t = linspace(0, 2*pi, 1000);
plot (t, sin(t))
grid
hold on
plot (t, cos(t), '--', 'color', [1 0 0])
xlabel('Time (s) ')
ylabel('Function value')
title('График 1 Sin and Cos functions ')
legend('y1 = sin(t)', 'y2 = cos(t)')
xlim([0, 2*pi])
ylim([-1.4, 1.4])
```



calculateGrades.m

```
load('classGrades.mat');

namesAndGrades(1:5,:)
grades = namesAndGrades(:, 2:size(namesAndGrades,2));

meanGrades = mean(grades)
meanGrades = nanmean(grades)
meanMatrix = ones(15,1)* meanGrades(1, :)

curvedGrades = 3.5*(grades./ meanMatrix);
nanmean(curvedGrades)
curvedGrades(curvedGrades > 5) = 5;
totalGrade = nanmean(curvedGrades, 2);
totalGrade = ceil(totalGrade);

letters = 'FDCBA';
letterGrades = letters(totalGrade);
disp(['Grades:  ',   letterGrades])

```

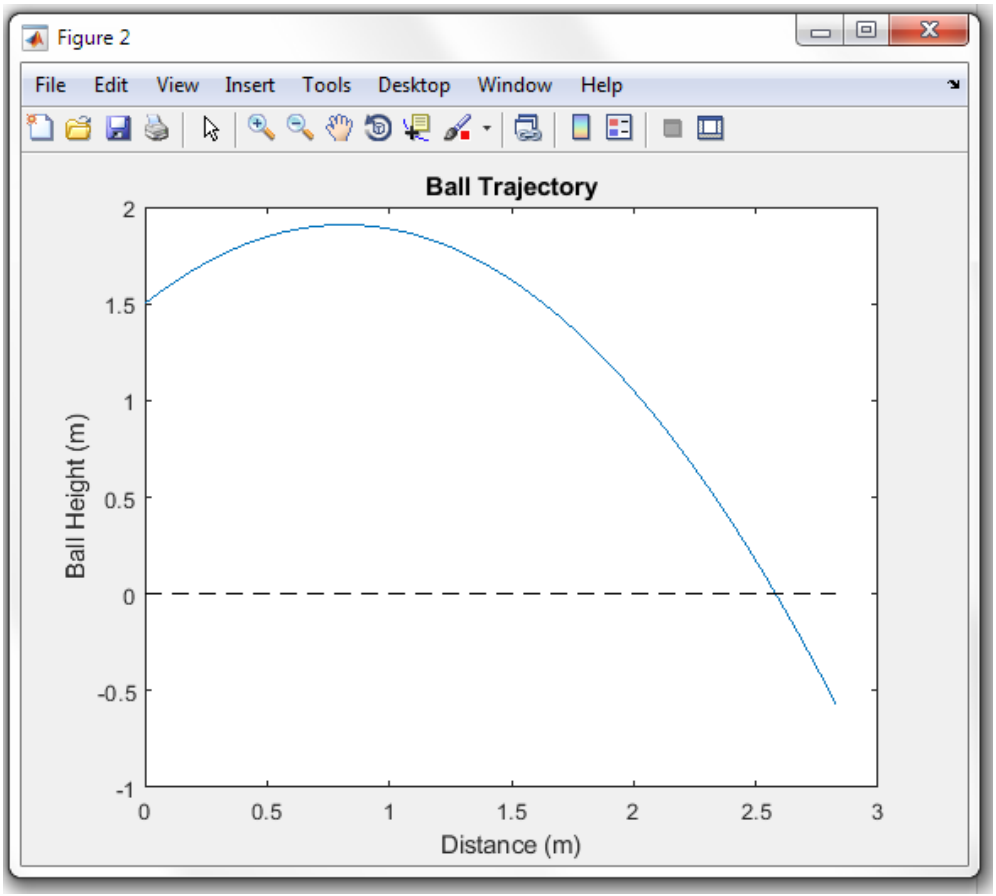
```
Grades:  BCBBBACCBCCCCAB
>> calculateGrades

ans =

    1.0000    2.5064    3.6529    2.4617    3.3022    2.5189    0.0963    4.6502
    2.0000    2.1586    3.2324    3.4737    0.2378    2.4480    0.4194    1.9951
    3.0000    4.9878       NaN    4.8637    1.7439    4.3852    4.8740    0.2370
    4.0000    4.0580    1.9914    1.6388    2.2567    1.7657    3.2567    1.7119
    5.0000    2.4283    3.7491    4.1890       NaN    2.2472    1.1562    3.6798


meanGrades =

       NaN       NaN    2.8361       NaN    2.8540    1.6481       NaN


meanGrades =

    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677


meanMatrix =

    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677


ans =

    3.5000    3.5000    3.5000    3.5000    3.5000    3.5000    3.5000

Grades:  BCBBBACCBCCCCAB
```

throwBall.m

```
% Пороскун Олени ПМ-81
% Homework 1                                  Optional Problems

% 1. Throwing a ball.

HeightAtRel = 1.5;
Gravit = 9.8;
VelocityRel = 4;
AngleVel = 45;

t = linspace(0, 1, 1000);

x = VelocityRel*(cos(AngleVel*(pi/180)))*t;
y = HeightAtRel + VelocityRel*(sin(AngleVel*(pi/180)))*t - 0.5*Gravit*(t.^2);

s = find(y<0);
X = x(s(1));
answer = sprintf('The ball hits the ground at a distance of %g meters.', X);
disp(answer);

figure
plot(x, y);
%grid on;
xlabel('Distance (m)');
ylabel('Ball Height (m)');
title('Ball Trajectory');
hold on

x2 = linspace(0, max(x), 10);
y2(1:length(x2))=0;
plot(x2, y2, 'k--')
```

Command Window

```
>> throwBall
The ball hits the ground at a distance of 2.58211 meters.
>>
```



seriesConvergence.m

```
% Пороскун Олени ПМ-81
% Homework 1          Optional Problems
                                        % 2. Convergence of infinite series.
p = 0.99;
k = 0:1000;
geomSeries = p.^k;
G = 1/(1-p);

figure;
x = 0:max(k);
y(1:length(x)) = G;
plot(x, y, 'r');
hold on;

y2 = cumsum(geomSeries);
plot(x, y2, 'b');
xlabel('Index ');
ylabel('Sum');
title('Convergence of geometric series with p=0.99');
legend('Infinite sum', 'Finite Sum');

p = 2;
n = 1:500;
pSeries = 1./(n.^p);
P = (pi^2)/6;
figure;
x3 = 0:max(n);
y3(1:length(x3)) = P;
plot(x3, y3, 'r')
hold on

x4 = 1:max(n);
y4 = cumsum(pSeries);
plot(x4, y4, 'b')
xlabel('Index');
ylabel('Sum');
ylim([1, 1.8]);
title('Convergence of geometric series with p=2');
legend('Infinite sum', 'Finite Sum');
```

Command Window

```
>> seriesConvergence
fx >>
```

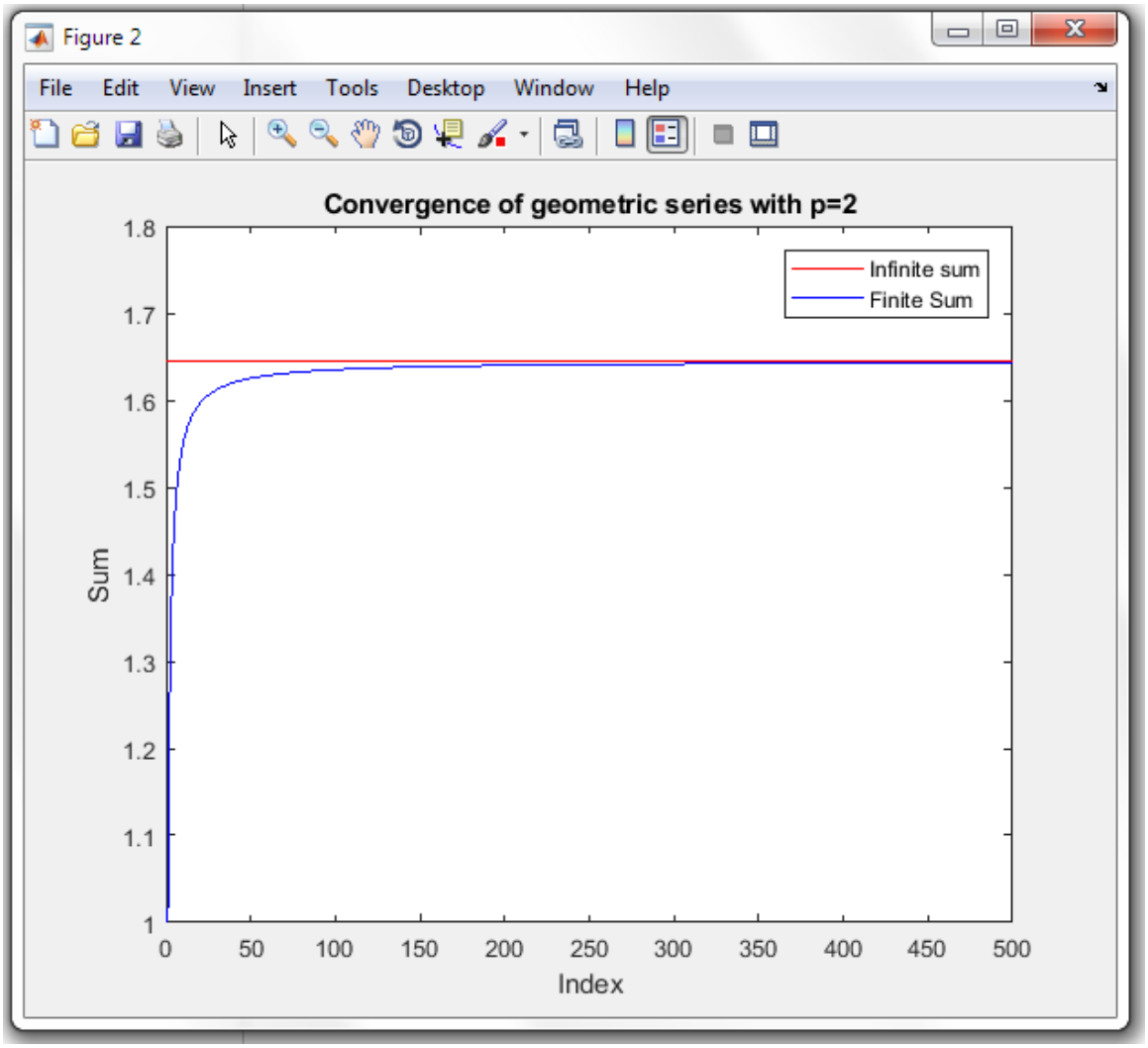


Figure 2
File Edit View Insert Tools Desktop Window Help
Convergence of geometric series with p=2
Infinite sum
Finite Sum
Sum
Index
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1
0 50 100 150 200 250 300 350 400 450 500

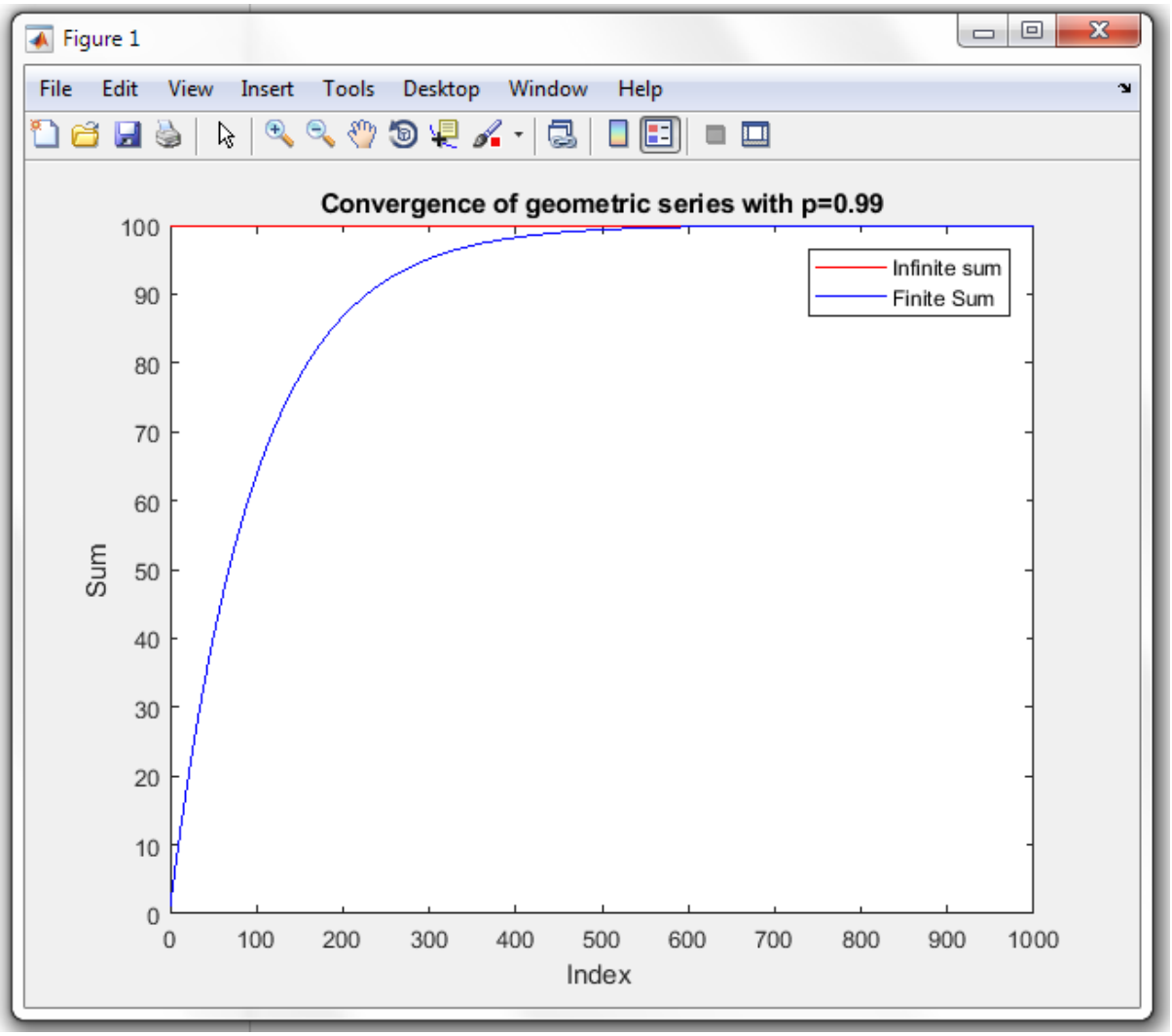


Figure 1
File Edit View Insert Tools Desktop Window Help
Convergence of geometric series with p=0.99
Infinite sum
Finite Sum
Sum
Index
100
90
80
70
60
50
40
30
20
10
0
0 100 200 300 400 500 600 700 800 900 1000

encrypt.m

```
1     % Пороскун Олени ПМ-81
2     % Homework 1          Optional Problems
3                                                  % 3. Encryption Algorithm.
4 -   original = 'This is my top secret message!';
5 -   length(original);
6 -   r = randperm(length(original));
7 -   encoded  = original(r);
8
9 -   mat = ones(length(original),  2);
10 -  mat(:, 1) = r;
11 -  mat(:, 2) = 1:length(original);
12
13 -  mat;
14 -  mat = sortrows(mat);
15 -  decoded = mat(:, 2);
16
17 -  disp(['Original: ', original])
18 -  disp(['Encoded: ',  encoded])
19 -  disp(['Decoded: ',  encoded(decoded)])
20
21 -  correct = strcmp(original, encoded(decoded));
22
23 -  disp (['Decoded correctly (1 true, 0 false): ', num2str(correct)])
24
25
26
```

Command Window

```
>> encrypt
Original: This is my top secret message!
Encoded:  mi m ye es grsicaosshteeT!stp
Decoded: This is my top secret message!
Decoded correctly (1 true, 0 false): 1
>>
```

Homework 1

This homework is designed to teach you to think in terms of matrices and vectors because this is how MATLAB organizes data. You will find that complicated operations can often be done with one or two lines of code if you use appropriate functions and have the data stored in an appropriate structure. The other purpose of this homework is to make you comfortable with using **help** to learn about new functions. The names of the functions you'll need to look up are provided in **bold** where needed. Also, in case, recall we cannot use space in the script's name.
Homework must be submitted on the website before the start of the next class.

What to turn in: Copy the text from your scripts and paste it into a document. If a question asks you to plot or display something to the screen, also include the plot and screen output your code generates. Submit either a *.doc or *.pdf file.

For problems 1-7, write a script called `shortProblems.m` and put all the commands in it. Separate and label different problems using comments.

1. **Scalar variables.** Make the following variables
   a. $a = 10$
   b. $b = 2.5 \times 10^{23}$
   c. $c = 2 + 3i$, where i is the square root of -1
   d. $d = e^{j2\pi/3}$, where j is the square root of -1 and e is Euler's number[1] (use **exp**, **pi**)

2. **Vector variables.** Make the following variables
   a. $aVec = \begin{bmatrix} 3.14 & 15 & 9 & 26 \end{bmatrix}$
   b. $bVec = \begin{bmatrix} 2.71 \\ 8 \\ 28 \\ 182 \end{bmatrix}$
   c. $cVec = \begin{bmatrix} 5 & 4.8 & \cdots & -4.8 & -5 \end{bmatrix}$ (all the numbers from 5 to -5 in increments of -0.2)
   d. $dVec = \begin{bmatrix} 10^{0} & 10^{0.01} & \cdots & 10^{0.99} & 10^{1} \end{bmatrix}$ (Logarithmically spaced numbers between 1 and 10, use **logspace**, make sure you get the length right!)
   e. $eVec = Hello$ ($eVec$ is a string, which is a vector of characters)

3. **Matrix variables.** Make the following variables

   a. $aMat = \begin{bmatrix} 2 & \cdots & 2 \\ \vdots & \ddots & \vdots \\ 2 & \cdots & 2 \end{bmatrix}$ a 9x9 matrix full of 2's (use **ones** or **zeros**)

   b. $bMat = \begin{bmatrix} 1 & 0 & \cdots & & 0 \\ 0 & \ddots & 0 & \ddots & \\ \vdots & 0 & 5 & 0 & \vdots \\ & \ddots & 0 & \ddots & 0 \\ 0 & & \cdots & 0 & 1 \end{bmatrix}$ a 9x9 matrix of all zeros, but with the values $\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 4 & 3 & 2 & 1 \end{bmatrix}$ on the main diagonal (use **zeros**, **diag**).

   c. $cMat = \begin{bmatrix} 1 & 11 & \cdots & 91 \\ 2 & 12 & \ddots & 92 \\ \vdots & \vdots & \ddots & \vdots \\ 10 & 20 & \cdots & 100 \end{bmatrix}$ a 10x10 matrix where the vector 1:100 runs down the columns (use **reshape**).

   d. $dMat = \begin{bmatrix} NaN & NaN & NaN & NaN \\ NaN & NaN & NaN & NaN \\ NaN & NaN & NaN & NaN \end{bmatrix}$ a 3x4 NaN matrix (use **nan**)

   e. $eMat = \begin{bmatrix} 13 & -1 & 5 \\ -22 & 10 & -87 \end{bmatrix}$

   f. Make $fMat$ be a 5x3 matrix of random integers with values on the range -3 to 3 (First use **rand** and **floor** or **ceil.** Now only use **randi**)

4. **Scalar equations.** Using the variables created in 1, calculate x, y, and z.

   a. $x = \dfrac{1}{1+e^{(-(a-15)/6)}}$

   b. $y = \left(\sqrt{a} + \sqrt[21]{b}\right)^{\pi}$, recall that $\sqrt[g]{h} = h^{1/g}$, and use **sqrt.** You can also use **nthroot** (refer to the MATLAB help to understand the difference between **nthroot** and a fractional power)

   c. $z = \dfrac{\log\left(\Re\left[(c+d)(c-d)\right]\sin\left(a\pi/3\right)\right)}{c\bar{c}}$ where $\Re$ indicates the real part of the complex number in brackets, $\bar{c}$ is the complex conjugate of c, and $\log$ is the *natural* log (use **real**, **conj**, **log**).

5. **Matrix equations.** Using the variables created in 2 and 3, find the values of xMat, yMat and zMat below. Use matrix operators.
   a. $\text{xMat} = (\text{aVec} \cdot \text{bVec}) \cdot \text{aMat}^2$
   b. $\text{yMat} = (\text{bVec} \cdot \text{aVec})$, note that this is *not* the same as $(\text{aVec} \cdot \text{bVec})$
   c. $\text{zMat} = |\text{cMat}|(\text{aMat} \cdot \text{bMat})^{\text{T}}$, where $|cMat|$ is the determinant of $cMat$, and T again indicates the transpose (use **det**).

6. **Common functions and indexing.**
   a. Make $cSum$ the column-wise sum of $cMat$. The answer should be a row vector (use **sum**).
   b. Make $eMean$ the mean across the rows of $eMat$. The answer should be a column (use **mean**).
   c. Replace the top row of $eMat$ with $\begin{bmatrix}1 & 1 & 1\end{bmatrix}$.
   d. Make $cSub$ the submatrix of $cMat$ that only contains rows 2 through 9 and columns 2 through 9.
   e. Make the vector $lin = \begin{bmatrix}1 & 2 & \cdots & 20\end{bmatrix}$ (the integers from 1 to 20), and then make every even value in it negative to get $lin = \begin{bmatrix}1 & -2 & 3 & -4 & \cdots & -20\end{bmatrix}$.
   f. Make r a 1x5 vector using **rand**. Find the elements that have values <0.5 and set those values to 0 (use **find**).

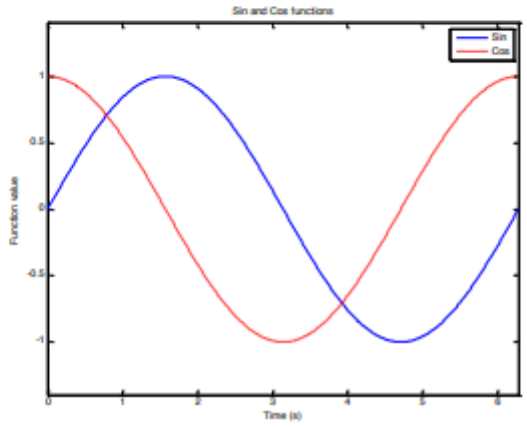
7. **Plotting multiple lines and colors.** In class we saw how to plot a single line in the default blue color on a plot. You may have noticed that subsequent plot commands simply replace the existing line. Here, we'll write a script to plot two lines on the same axes.
   a. Open a script and name it `twoLinePlot.m`. Write the following commands in this script.
   b. Make a new figure using **figure**
   c. We'll plot a sine wave and a cosine wave over one period
      i. Make a time vector t from 0 to 2π with enough samples to get smooth lines
      ii. Plot $\sin(t)$
      iii. Type **hold on** to turn on the 'hold' property of the figure. This tells the figure not to discard lines that are already plotted when plotting new ones. Similarly, you can use **hold off** to turn off the hold property.
      iv. Plot $\cos(t)$ using a red dashed line. To specify line color and style, simply add a third argument to your plot command (see the third paragraph of the **plot** help).

This argument is a string specifying the line properties as described in the help file. For example, the string 'k:' specifies a black dotted line.

d. Now, we'll add labels to the plot
   i. Label the x axis using **xlabel**
   ii. Label the y axis using **ylabel**
   iii. Give the figure a title using **title**
   iv. Create a legend to describe the two lines you have plotted by using **legend** and passing to it the two strings 'Sin' and 'Cos'.

e. If you run the script now, you'll see that the x axis goes from 0 to 7 and y goes from -1 to 1. To make this look nicer, we'll manually specify the x and y limits. Use **xlim** to set the x axis to be from 0 to 2π and use **ylim** to set the y axis to be from -1.4 to 1.4.

f. Run the script to verify that everything runs right. You should see something like this:



8. **Manipulating variables.** Write a script to read in some grades, curve them, and display the overall grade. To do this, you'll need to download the file `classGrades.mat` off the class website and put it in the same folder as your script.
   a. Open a script and name it `calculateGrades.m`. Write all the following commands in this script.
   b. Load the `classGrades` file using **load**. This file contains a single variable called *namesAndGrades*
   c. To see how *namesAndGrades* is structured, display the first 5 rows on your screen. The first column contains the students 'names', they're just the integers from 1 to 15. The remaining 7 columns contain each student's score (on a scale from 0 to 5) on each of 7 assignments. There are also some NaNs which indicate that a particular student was absent on that day and didn't do the assignment.
   d. We only care about the grades, so extract the submatrix containing all the rows but only columns 2 through 8 and name this matrix *grades* (to make this work on any size matrix, don't hard-code the 8, but rather use **end** or **size(namesAndGrades,2)**).
   e. Calculate the mean score on each assignment. The result should be a 1x7 vector containing the mean grade on each assignment.
      i. First, do this using **mean**. Display the mean grades calculated this way. Notice that the NaNs that were in the grades matrix cause some of the mean grades to be NaN as well.
      ii. To fix this problem, do this again using **nanmean**. This function does exactly what you want it to do, it computes the mean using only the numbers that are not NaN. This means that the absent students are not considered in the calculation, which is what we want. Name this mean vector *meanGrades* and display it on the screen to verify that it has no NaNs

f. Normalize each assignment so that the mean grade is 3.5 (this is a B- on our 5 point scale). You'll want to divide each column of $grades$ by the correct element of $meanGrades$.

   i. Make a matrix called $meanMatrix$ such that it is the same size as $grades$, and each row has the values $meanGrades$. Do this by taking the outer product of a 15x1 vector of ones and the vector $meanGrades$, which is a row (use **ones**, *****). Display $meanMatrix$ to verify that it looks the way you want.

   ii. To calculate the curved grades, do the following: $curvedGrades = 3.5(grades / meanMatrix)$. Keep in mind that you want to do the division elementwise.

   iii. Compute and display the mean of $curvedGrades$ to verify that they're all 3.5 (**nanmean**).

   iv. Because we divided by the mean and multiplied by 3.5, it's possible that some grades that were initially close to 5 are now larger than 5. To fix this, find all the

elements in $curvedGrades$ that are greater than 5 and set them to 5. Use logical indexing and DO NOT use **find**.

g. Calculate the total grade for each student and assign letter grades

   i. To calculate the $totalGrade$ vector, which will contain the numerical grade for each student, you want to take the mean of $curvedGrades$ across the columns (use **nanmean**, see help for how to specify the dimension). Also, we only want to end up with numbers from 1 to 5, so calculate the ceiling of the $totalGrade$ vector (use **ceil**).

   ii. Make a string called $letters$ that contains the letter grades in increasing order: FDCBA

   iii. Make the final letter grades vector $letterGrades$ by using $totalGrade$ (which after the ceil operation should only contain values between 1 and 5) to index into $letters$.

   iv. Finally, display the following using **disp**: Grades: $letterGrades$

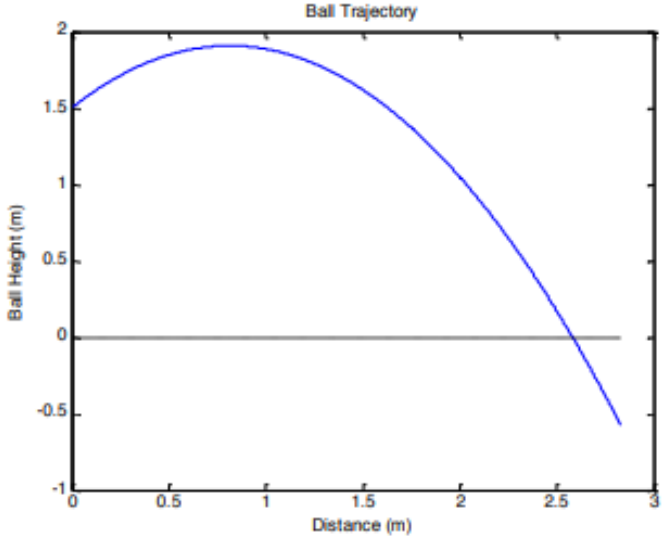
h. Run the script to verify that it works. You should get an output like this

```
>> calculateGrades
ans =
    1.0000    2.5064    3.6529    2.4617    3.3022    2.5189    0.0963    4.6502
    2.0000    2.1586    3.2324    3.4737    0.2378    2.4480    0.4194    1.9951
    3.0000    4.9878       NaN    4.8637    1.7439    4.3852    4.8740    0.2370
    4.0000    4.0580    1.9914    1.6388    2.2567    1.7657    3.2567    1.7119
    5.0000    2.4283    3.7491    4.1890       NaN    2.2472    1.1562    3.6798
meanGrades =
       NaN       NaN    2.8361       NaN    2.8540    1.6481       NaN
meanGrades =
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
meanMatrix =
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
    2.9690    2.9445    2.8361    2.4879    2.8540    1.6481    2.5677
ans =
    3.5000    3.5000    3.5000    3.5000    3.5000    3.5000    3.5000
Grades: BCBBBACCBCCCCAB
```

Optional Problems

1. **Throwing a ball.** Below are all the steps you need to follow, but you should also add your own meaningful comments to the code as you write it.
   a. Start a new file in the MATLAB Editor and save it as `throwBall.m`
   b. At the top of the file, define some constants (you can pick your own variable names)
      i. Initial height of ball at release = 1.5 m
      ii. Gravitational acceleration = 9.8 m/s^2
      iii. Velocity of ball at release = 4 m/s
      iv. Angle of the velocity vector at time of release = 45 degrees
   c. Next, make a time vector that has 1000 linearly spaced values between 0 and 1, inclusive.
   d. If x is distance and y is height, the equations below describe their dependence on time and all the other parameters (initial height h, gravitational acceleration g, initial ball velocity v, angle of velocity vector in degrees θ). Solve for x and y
      i. $x(t) = v\cos\left(\theta\frac{\pi}{180}\right)t$. We multiply θ by $\frac{\pi}{180}$ to convert degrees to radians.
      ii. $y(t) = h + v\sin\left(\theta\frac{\pi}{180}\right)t - \frac{1}{2}gt^2$
   e. Approximate when the ball hits the ground.
      i. Find the index when the height first becomes negative (use **find**).
      ii. The distance at which the ball hits the ground is value of x at that index
      iii. Display the words: *The ball hits the ground at a distance of X m*eters. (where X is the distance you found in part ii above)
   f. Plot the ball's trajectory
      i. Open a new figure (use **figure**)
      ii. Plot the ball's height on the y axis and the distance on the x axis (**plot**)
      iii. Label the axes meaningfully and give the figure a title (use **xlabel**, **ylabel**, and **title**)
      iv. Hold on to the figure (use **hold on**)
      v. Plot the ground as a dashed black line. This should be a horizontal line going from 0 to the maximum value of x (use **max**). The height of this line should be 0. (see **help plot** for line colors and styles)
   g. Run the script from the command window and verify that the ball indeed hits the ground around the distance you estimated in e-ii. You should get something like this:

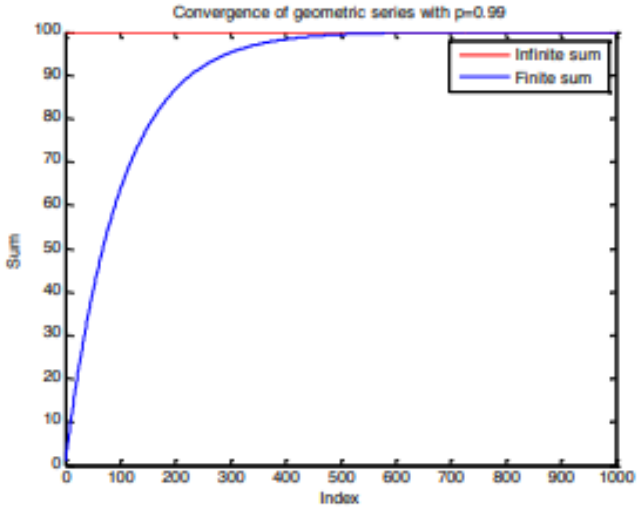
```
>> throwBall
The ball hits the ground at a distance of 2.5821 meters
```

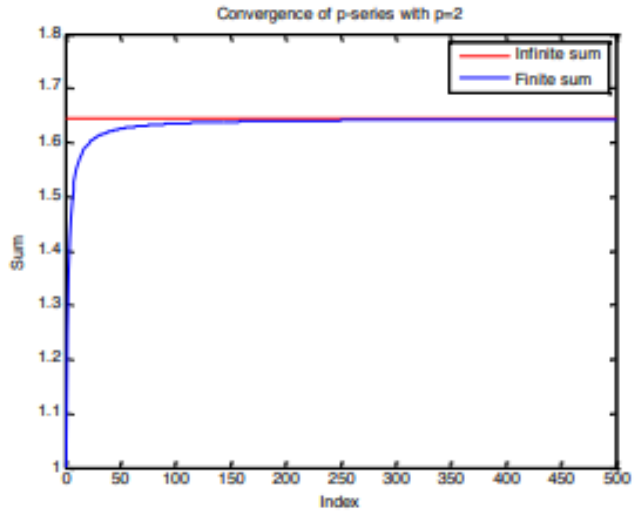


2. **Convergence of infinite series.** We'll look at two series that converge to a finite value when they are summed.
   a. Open a new script in the MATLAB Editor and save it as `seriesConvergence.m`
   b. First, we'll deal with a geometric series $G = \sum_{k=0}^{\infty} p^k$. We need to define the value of p and the values of k
      i. p =0.99.
      ii. k is a vector containing the integers from 0 to 1000, inclusive.
   c. Calculate each term in the series (before summation)
      i. $geomSeries = p^k$ (this should be done elementwise)
   d. Calculate the value of the infinite series
      i. We know that $G = \sum_{k=0}^{\infty} p^k = \frac{1}{1-p}$
   e. Plot the value of the infinite series
      i. Plot a horizontal red line that has x values 0 and the maximum value in k (use **max**), and the y value is constant at G.
   f. On the same plot, plot the value of the finite series for all values of k
      i. Plot the cumulative sum of $geomSeries$ versus k. The cumulative sum of a vector is a vector of the same size, where the value of each element is equal to the sum of all the elements to the left of it in the original vector. (use **cumsum**, and try cumsum([1 1 1 1 1]) to understand what it's doing.) Use a blue line when plotting.
   g. Label the x and y axes, and give the figure a title (**xlabel**, **ylabel**, **title**). Also create a legend and label the first line 'Infinite sum', and the second line 'Finite Sum' (**legend**).
   h. Run the script and note that the finite sum of 1000 elements comes very close to the value of the infinite sum.
   i. Next, we will do a similar thing for another series, the p-series: $P = \sum_{n=1}^{\infty} \frac{1}{n^p}$
   j. At the bottom of the same script, initialize new variables
      i. $p = 2$
      ii. n is a vector containing all the integers from 1 to 500, inclusive.
   k. Calculate the value of each term in the series
      i. $pSeries = \frac{1}{n^p}$
   l. Calculate the value of the infinite p-series. The infinite p-series with $p = 2$ has been proven to converge to $P = \sum_{n=1}^{\infty} \frac{1}{n^p} = \frac{\pi^2}{6}$.

m. Make a new figure and plot the infinite sum as well as the finite sum, as we did for the geometric series
   i. Make a new figure
   ii. Plot the infinite series value as a horizontal red line with x values 0 and the maximum value in n, and the y value is constant at P.
   iii. Hold on to the figure, and plot the cumulative sum of $pSeries$ versus n (use **hold on**, **cumsum**).
   iv. Label the x and y axes, give the figure a title, and make a legend to label the lines as 'Infinite sum', and 'Finite sum' (use **xlabel**, **ylabel**, **title**, **legend**)

n. Run the script to verify that it produces the expected output. It should look something like this:





3. **Encryption Algorithm.** Write a simple shuffling 'encryption' algorithm.
   a. Open a new script and save it as `encrypt.m`
   b. At the top of the script, define the *original* string to be: This is my top secret message!
   c. Next, let's shuffle the indices of the letters. To do this, we need to make a string of encoding indices
      i. Make a vector that has the indices from 1 to the length of the original string in a randomly permuted order. Use **randperm** and **length**
      ii. Encode the original string by using your encoding vector as indices into *original*. Name the encoded message *encoded*.
   d. Now, we need to figure out the decoding key to match the encoding key we just made.
      i. Assemble a temporary matrix where the first column is the encoding vector you made in the previous part and the second column are the integers from 1 to the length of the original string in order. Use **length**, and you may need to transpose some vectors to make them columns using **'**.
      ii. Next, we want to sort the rows of this temporary matrix according to the values in the first column. Use **sortrows**.
      iii. After it's been sorted, extract the second column of the temporary matrix. This is your decoding vector.
      iv. To make the *decoded* message, use the decoding vector as indices into *encoded*.
   e. Display the original, encoded, and decoded messages
      i. Display the following three strings, where : *original*, *encoded*, and *decoded* are the strings you made above. Use **disp**
      Original: *original*
      Encoded: *encoded*
      Decoded: *decoded*
   f. Compare the original and decoded strings to make sure they're identical and display the result
      i. Use **strcmp** to compare the *original* and *decoded* strings. Name the output of this operation *correct*. *correct* will have the value 1 if the strings match and the value 0 if they don't
      ii. Display the following string: Decoded correctly (1 true, 0 false): *correct*
      use **disp** and **num2str**
   g. Run the script a few times to verify that it works well. You should see an output like this:

```
>> encrypt
Original: This is my top secret message!
Encoded : sisrpigct tessaoheem m  yTse!
Decoded : This is my top secret message!
Decoded correctly (1 true, 0 false): 1
```

Sumy State University

Department

of

Applied Mathematics and Complex Systems Modeling

Report on laboratory work №2

Subject

Computing system

Student: Olena Poroskun

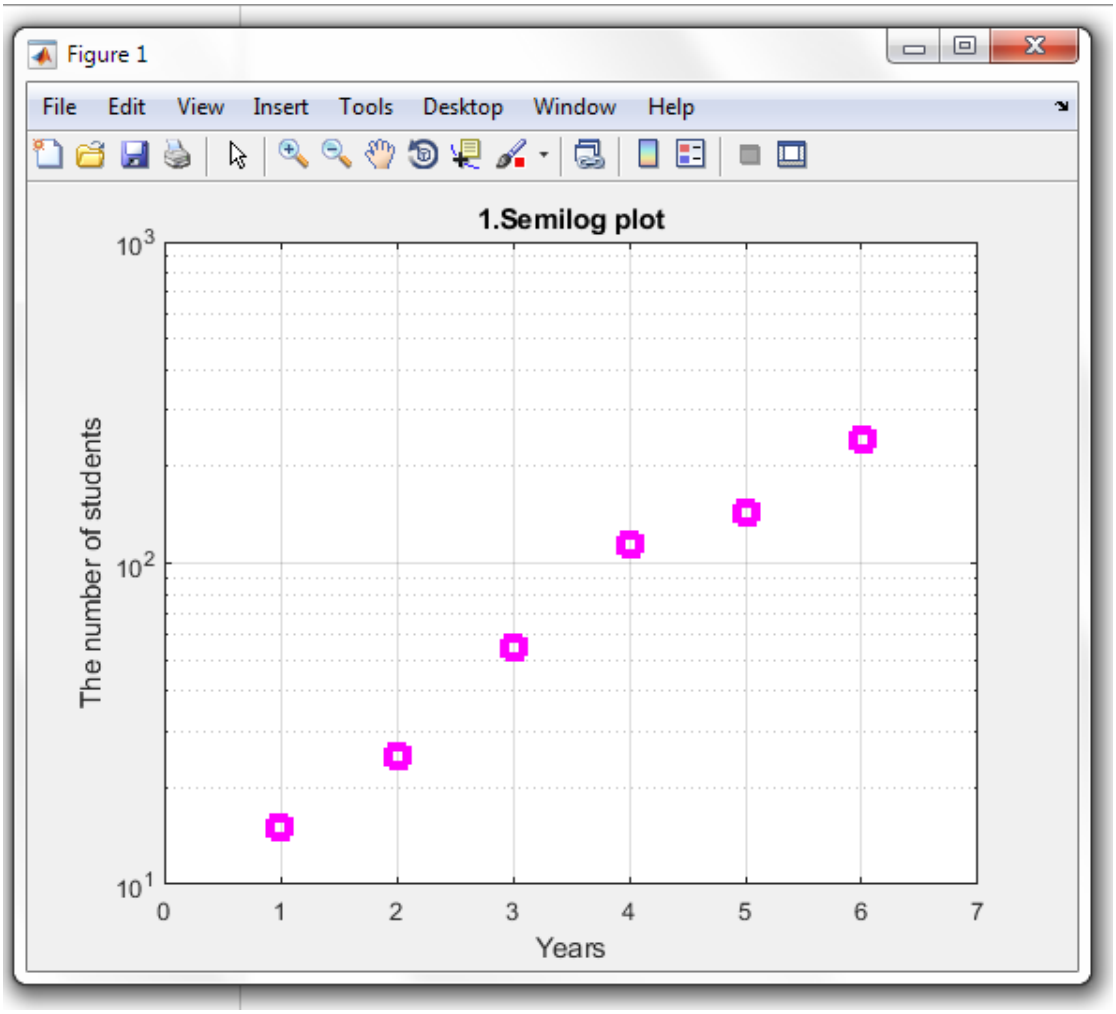
Teacher: Igor A. Knyaz

Sumy, Sumy region

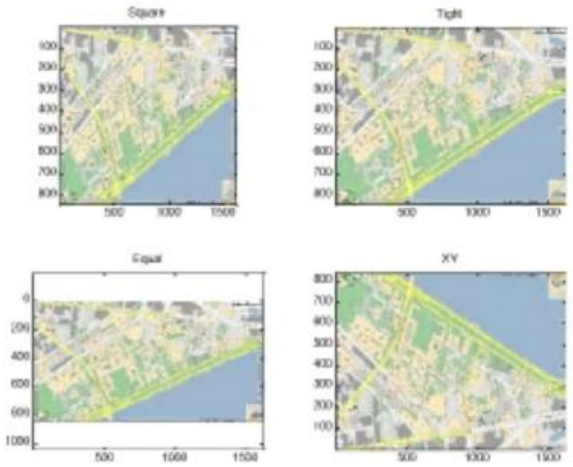
2020

1. **Semilog plot**. Over the past 6 years, the number of students in 6.057 has been 15, 25, 55, 115, 144, 242. Class size seems like it's growing exponentially. To verify this, plot these values on a plot with a log y scale and label it (**semilogy**, **xlabel**, **ylabel**, **title**). Use magenta square symbols of marker size 10 and line width 4, and no line connecting them. You may have to change the x limits to see all 6 symbols (**xlim**). If the relationship really is exponential, it will look linear on a log plot.

```
% Пороскун Олени ПМ-81
% Homework 2

% 1.Semilog plot.
x = 1:6;
y = [15, 25, 55, 115, 144, 242];
semilogy(x, y,  'ms', 'MarkerSize', 10, 'LineWidth',  4);
xlabel('Years');
ylabel('The number of students');
title('1.Semilog plot');
grid on;
xlim([0, 7]);
```

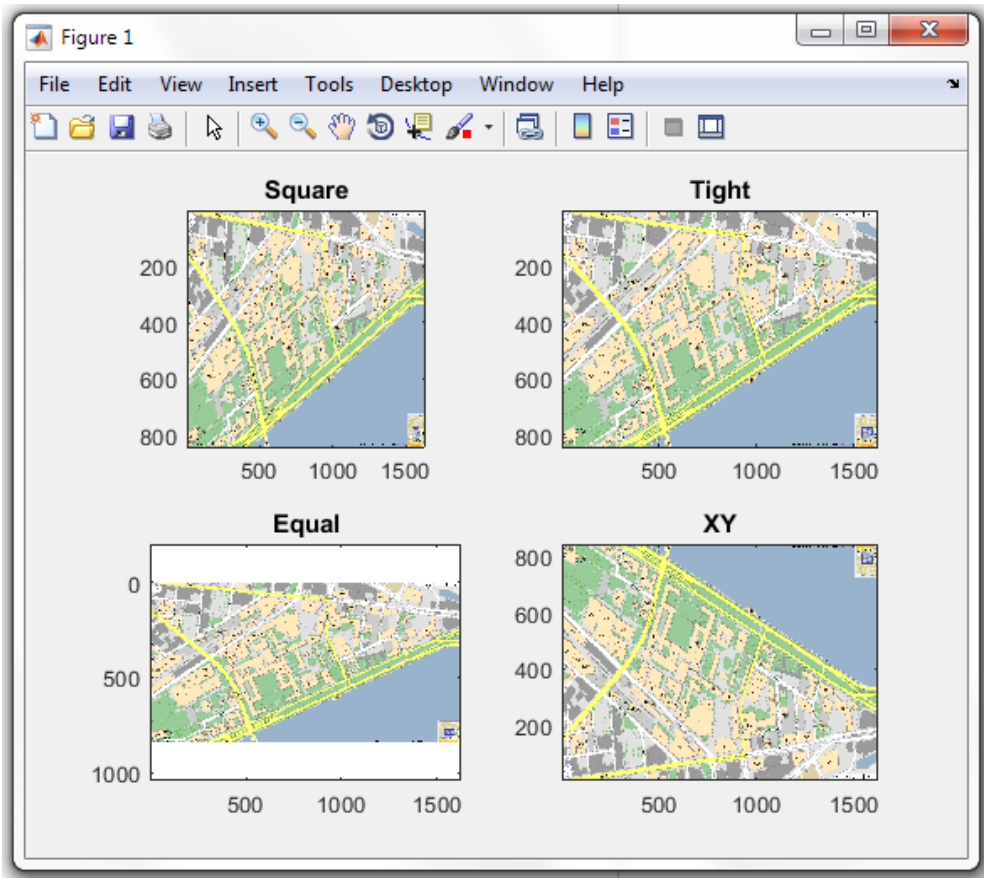


2. **Subplot and axis modes**. Make a new figure that has a 2x2 grid of axes (**subplot**). Load the file mitMap.mat. This file contains an image matrix called `mit` and the appropriate colormap in `cMap`. In each axis display the `mit` image (**image** or **imagesc**) and set the colormap `cMap` (**colormap**). Set the axis of the top left image to be square, the top right to be tight, the bottom left to be equal, and the bottom right to be xy (**axis square**, **axis tight**, etc.). Also add the appropriate title to each axis as below. Note that images are displayed with the origin in the top left corner of an axis (this is mode axis ij), and axis xy moves the origin to the bottom left corner, flipping the image.

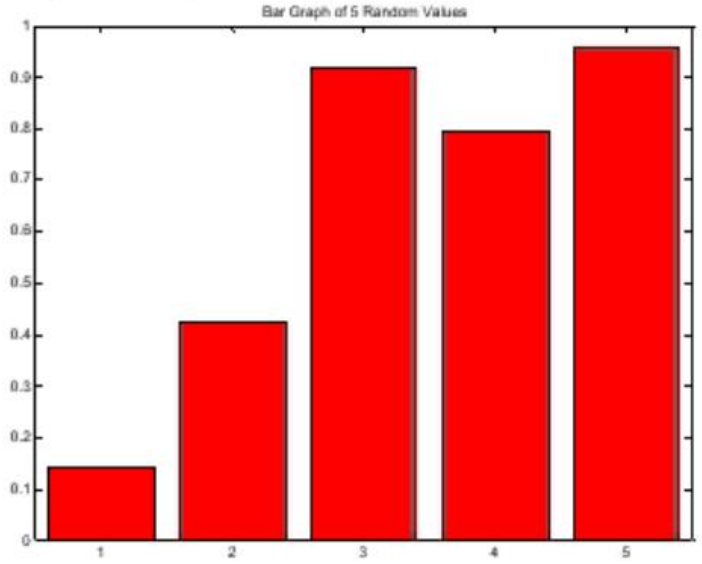


to

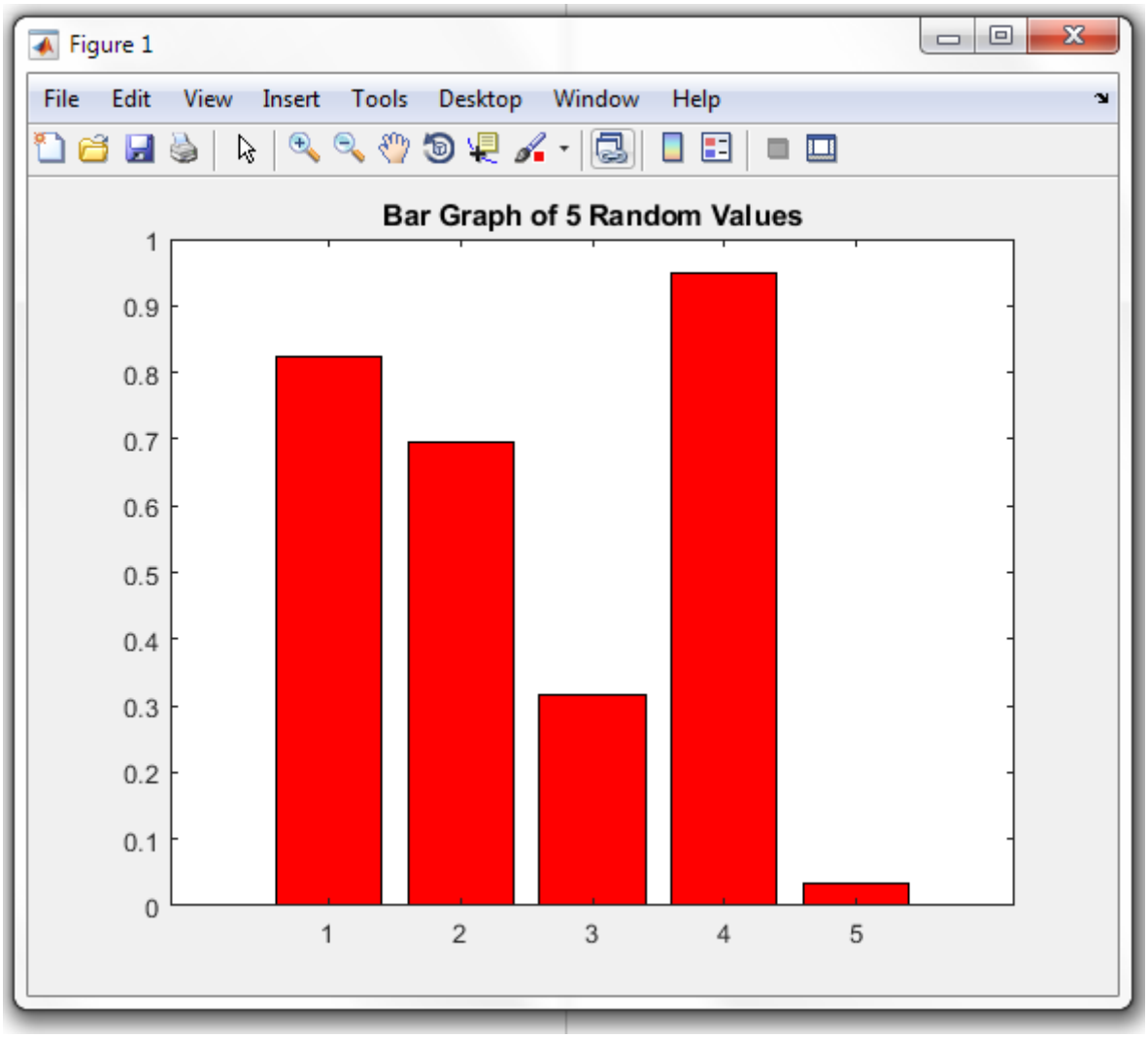
```
% Пороскун Олени ПМ-81
% Homework 2

%2. Subplot and axis modes.
figure;
subplot(2,2,1);
load('mitMap.mat')
image(mit)
colormap(cMap)
axis('square')
title('Square');


subplot(2,2,2);
image(mit)
axis('tight')
title('Tight');

subplot(2,2,3);
image(mit)
axis('equal')
title('Equal');

subplot(2,2,4);
image(mit)
axis('xy')
title('XY');
```



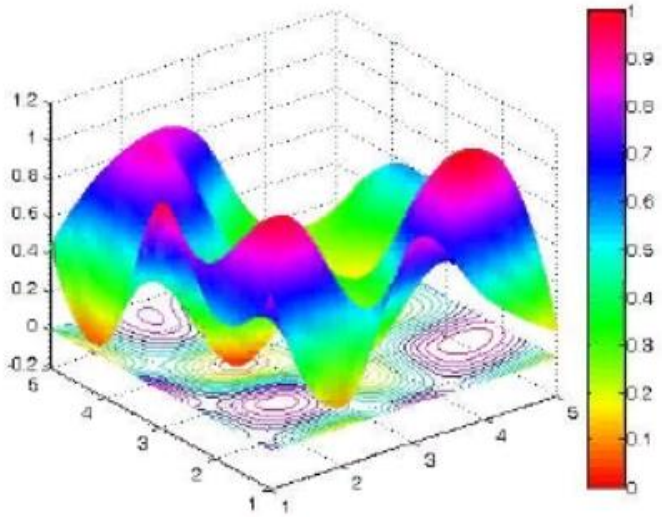
3. **Bar graph**. Make a vector of 5 random values and plot them on a bar graph using red bars, something like the figure below.




```
% Пороскун Олени ПМ-81
% Homework 2

% 3.Bar graph.
x = 1:5;
y = rand(1, 5);
bar(x, y, 'r')
title('Bar Graph of 5 Random Values');
```

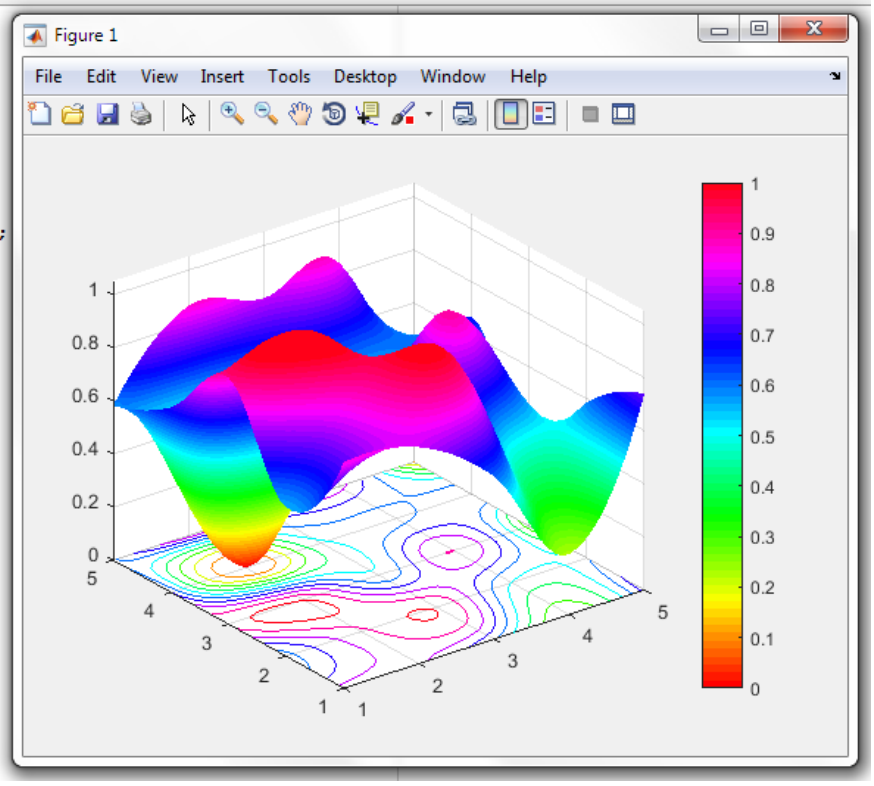


4. **Interpolation and surface plots**. Write a script called `randomSurface.m` to do the following
   a. To make a random surface, make Z0 a 5x5 matrix of random values on the range (0,1) (**rand**).
   b. Make X0 and Y0 using **meshgrid** and the vector 1:5 (use the same vector for both inputs into meshgrid). Now, X0, Y0, and Z0 define 25 points on a surface.
   c. We are going to interpolate intermediate values to make the surface seem smooth. Make X1 and Y1 using **meshgrid** and the vector 1:0.1:5 (again use the same vector for both inputs into meshgrid).
   d. Make Z1 by interpolating X0, Y0, and Z0 at the positions in X1 and Y1 using cubic interpolation (**interp2**, specify cubic as the interpolation method).
   e. Plot a surface plot of Z1. Set the colormap to hsv and the shading property to interp (**surf**, **colormap**, **shading**).
   f. Hold on to the axes and plot the 15-line contour on the same axes (**contour**).
   g. Add a colorbar (**colorbar**).
   h. Set the color axis to be from 0 to 1 (**caxis**). The final figure should look something like this (if you cannot copy/paste the figure into your document appropriately, try changing the figure copy options to use a bitmap format):



randomSurface.m

```
% Пороскун Олени ПМ-81
% Homework 2

% 4.Interpolation and surface plots.

Z0 = rand(5);
[X0, Y0] = meshgrid(1:5, 1:5);
[X1, Y1] = meshgrid(1:0.1:5, 1:0.1:5);
Z1 = interp2(X0, Y0, Z0, X1, Y1, 'cubic');

surf(X1, Y1, Z1)
colormap(hsv)
shading interp


hold on
contour (X1, Y1, Z1)
colorbar
caxis ([0 1])
```



5. **Fun with find**. Write a function to return the index of the value that is nearest to a desired value. The function declaration should be: `ind=findNearest(x, desiredVal).x` is a vector or matrix of values, and `desiredVal` is the scalar value you want to find. Do not assume that `desiredVal` exists in `x`, rather find the value that is closest to `desiredVal`. If multiple values (entries) have the same distance from `desiredVal`, return all of their indices. Test your function to make sure it works on a few vectors and matrices. Useful functions are **abs**, **min**, and **find**. **Hint:** You may have some trouble using `min` when `x` is a matrix. To convert a matrix Q into a vector you can do something like `y=Q(:)`. Then, doing `m=min(y)` will give you the minimum value in `Q`. To find where this minimum occurs in `Q`, do `ind=find(Q==m);`

```
% Пороскун Олени ПМ-81
% Homework 2
% 5. Fun with find.

mas = input('Введіть вектор або масив елементів:');
dis = input('Введіть скалярну величину, яку ви хочете знайти:');
ind = findNearest(mas, dis)

function ind = findNearest(x, desiredVal)
x
X = x(:)
a = abs (desiredVal - x);      % знаходимо відстань між шуканим елеметом та всіма елементами матриці
y = a(:);                      % робимо з цієї матриці стовпець для пошуку індекса
minimal = min (y);             % визначаємо найближче до шуканого елемента число або числа
ind = find(a == minimal);      % знаходимо його(їх) індекс(и)
% x(ind)                       % перевірка по індексу знайденого елемента
end
```

Command Window

```
Введіть вектор або масив елементів:[12 34 56 -2; 32 0 15 -78]
Введіть скалярну величину, яку ви хочете знайти:14

x =

    12    34    56    -2
    32     0    15   -78


X =

    12
    32
    34
     0
    56
    15
    -2
   -78


ind =

     6
```

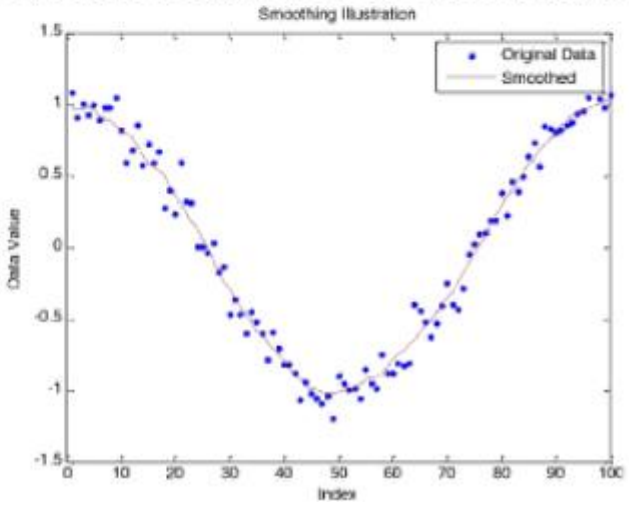
6. **Loops and flow control**. Make function called `loopTest(N)` that loops through the values 1 through N and for each number n it should display 'n is divisible by 2', 'n is divisible by 3', 'n is divisible by 2 AND 3' or 'n is NOT divisible by 2 or 3'. Use a **for** loop, the function **mod** or **rem** to figure out if a number is divisible by 2 or 3, and **num2str** to convert each number to a string for displaying. You can use any combination of **if**, **else**, and **elseif**.

```
1     % Пороскун Олени ПМ-81
2     % Homework 2
3     % 6. Loops and flow control.
4
5 -   N = input('Введіть N:');
6 -   loopTest(N)
7     function loopTest(N)
8 -   A = 1:N;
9 -   for i = 1:N
10 -      a = num2str(A(i),'%g  ');
11 -      mod2 = mod(A(i), 2);
12 -      mod3 = mod(A(i), 3);
13 -      if (mod2 == 0 | mod3 == 0)
14 -          if (mod2 == 0 & mod3 == 0)
15 -          disp([a, ' is divisible by 2 AND 3'])
16 -          elseif (mod2 == 0)
17 -              disp([a, ' is divisible by 2'])
18 -          elseif (mod3 == 0)
19 -              disp([a, ' is divisible by 3'])
20 -          end
21 -      else
22 -          disp([a, ' is NOT divisible by 2 or 3'])
23 -      end
24 -  end
25 -  end
```

Command Window

```
>> Task6
Введіть N:12
1 is NOT divisible by 2 or 3
2 is divisible by 2
3 is divisible by 3
4 is divisible by 2
5 is NOT divisible by 2 or 3
6 is divisible by 2 AND 3
7 is NOT divisible by 2 or 3
8 is divisible by 2
9 is divisible by 3
10 is divisible by 2
11 is NOT divisible by 2 or 3
12 is divisible by 2 AND 3
fx >>
```

7. **Smoothing filter.** Although it is a really useful software, MATLAB doesn't contain an easy to use smoothing filter[1]! Write a function with the declaration: `smoothed=rectFilt(x,width)` The filter should take a vector of noisy data (`x`) and smooth it by doing a symmetric moving average with a window of the specified `width`. For example if `width=5`, then `smoothed(n)` should equal `mean(x(n-2:n+2))`. Note that you may have problems around the edges: when `n<3` and `n>length(x)-2`.
   a. The lengths of `x` and `smoothed` should be equal.
   b. For symmetry to work, make sure that `width` is odd. If it isn't, increase it by 1 to make it odd and display a warning, but still do the smoothing.
   c. Make sure you correct edge effects so that the smoothed function doesn't deviate from the original at the start or the end[2]. Also make sure you don't have any horizontal offset between the smoothed function and the original (since we're using a symmetric moving average, the smoothed values should lie on top of the original data).
   d. You can do this using a loop and **mean** (which should be easy but may be slow), or more efficiently by using **conv** (if you are familiar with convolution).
   e. Load the mat file called noisyData.mat. It contains a variable `x` which is a noisy line. Plot the noisy data as well as your smoothed version, like below (a width of 11 was used):



Task7Funct.m

```
1     % Пороскун Олени ПМ-81
2     % Homework 2
3     % 7. Smoothing filter.
4
5 -   load('noisyData.mat')
6 -   Width = input('Width is ')
7 -   smoothed = rectFilt(x, Width);
8
9     function smoothed = rectFilt(X, width)
10 -  if (mod(width, 2) == 0)
11 -      width = width+1;
12 -      disp ('WARNING! Width is not odd.')
13 -  end
```

Task7Funct.m

```
    smoothed = rectFilt(x, Width);

function smoothed = rectFilt(X, width)
if (mod(width, 2) == 0)
    width = width+1;
    disp ('WARNING! Width is not odd.')
end

lim = floor(width/2);
lim1 = 1 + max(1, lim);
lim2 = length(X) - min(length(X),lim);

for i = 1:length(X)
    if(i < lim1)
        smoothed(i) = mean(X(i:i+1));
    elseif(i > lim2)
        smoothed(i) = mean(X(i-1:i));
    else %(i >= lim1 || i <= lim2)
        smoothed(i) = mean(X(i - lim : i + lim));
    end
end

if (length(smoothed) == length(smoothed))
    disp ('The lengths of x and smoothed are equal.')
else
    disp ('The lengths of x and smoothed are NOT equal.')
end

X
smoothed

figure;
x0 = 1:length(X);
plot(x0, X, 'b.', 'MarkerSize', 11);
hold on
x1 = 1:length(smoothed);
plot(x1, smoothed, 'r', 'LineWidth', 1);
xlabel('Index');
ylabel('Data value');
legend('Original Data','Smoothed');
title('Smoothing ilolustration');
end
```

Command Window

```
>> Task7Funct
Width is 11

Width =

    11

The lengths of x and smoothed are equal.

X =

  Columns 1 through 17

    1.0840    0.9092    1.0020    0.9275    0.9983    0.8900    0.9774    0.9769    1.0450    0.8218    0.5914    0.6821    0.8592    0.5713    0.7266    0.5925    0.6709

  Columns 18 through 34

    0.2762    0.3956    0.2361    0.5877    0.3183    0.3115    0.0050    0.0007   -0.0431    0.0306   -0.1701   -0.1347   -0.4717   -0.3625   -0.4687   -0.6018   -0.4492

  Columns 35 through 51

   -0.5257   -0.6023   -0.7882   -0.5887   -0.7102   -0.8160   -0.8214   -0.8842   -1.0639   -0.9447   -1.0228   -1.0574   -1.0911   -1.0408   -1.1957   -0.9031   -0.9475

  Columns 52 through 68

   -0.9975   -0.9909   -1.0552   -0.8576   -0.9530   -0.9876   -0.7537   -0.8805   -0.8826   -0.8154   -0.8301   -0.8135   -0.4023   -0.4401   -0.5232   -0.6257   -0.5306

  Columns 69 through 85

   -0.4040   -0.2479   -0.3997   -0.4378   -0.2872   -0.0459    0.0233    0.0927    0.0978    0.1920    0.1881    0.3831    0.2207    0.4609    0.3874    0.4937    0.6353

  Columns 86 through 100

    0.7345    0.5667    0.8498    0.8321    0.7985    0.8217    0.8521    0.8726    0.9307    0.9552    1.0506    1.1346    1.0386    0.9770    1.0625

smoothed =

  Columns 1 through 17

    0.9966    0.9556    0.9647    0.9629    0.9442    0.9294    0.8929    0.8883    0.8492    0.8309    0.7940    0.7741    0.7104    0.6575    0.5840    0.5627    0.5379
```

```
smoothed =

  Columns 1 through 17

    0.9966    0.9556    0.9647    0.9629    0.9442    0.9294    0.8929    0.8883    0.8492    0.8309    0.7940    0.7741    0.7104    0.6575    0.5840    0.5627    0.5379

  Columns 18 through 34

    0.5042    0.4265    0.3747    0.3047    0.2536    0.1771    0.1398    0.0610    0.0065   -0.0895   -0.1731   -0.2423   -0.2906   -0.3454   -0.4131   -0.4694   -0.5185

  Columns 35 through 51

   -0.5804   -0.6122   -0.6597   -0.7138   -0.7449   -0.7971   -0.8454   -0.8899   -0.9128   -0.9680   -0.9855   -0.9975   -1.0135   -1.0232   -1.0224   -1.0145   -1.0082

  Columns 52 through 68

   -1.0018   -0.9711   -0.9566   -0.9281   -0.9201   -0.9095   -0.8927   -0.8392   -0.7833   -0.7529   -0.7231   -0.6816   -0.6498   -0.5923   -0.5484   -0.5141   -0.4647

  Columns 69 through 85

   -0.3949   -0.3563   -0.3078   -0.2514   -0.1770   -0.1117   -0.0401    0.0025    0.0807    0.1557    0.2267    0.2887    0.3533    0.3964    0.4648    0.5229    0.5784

  Columns 86 through 100

    0.6183    0.6757    0.7131    0.7625    0.8045    0.8422    0.8786    0.9215    0.9331    0.9540    1.0029    1.0926    1.0866    1.0078    1.0198

fx >>
```

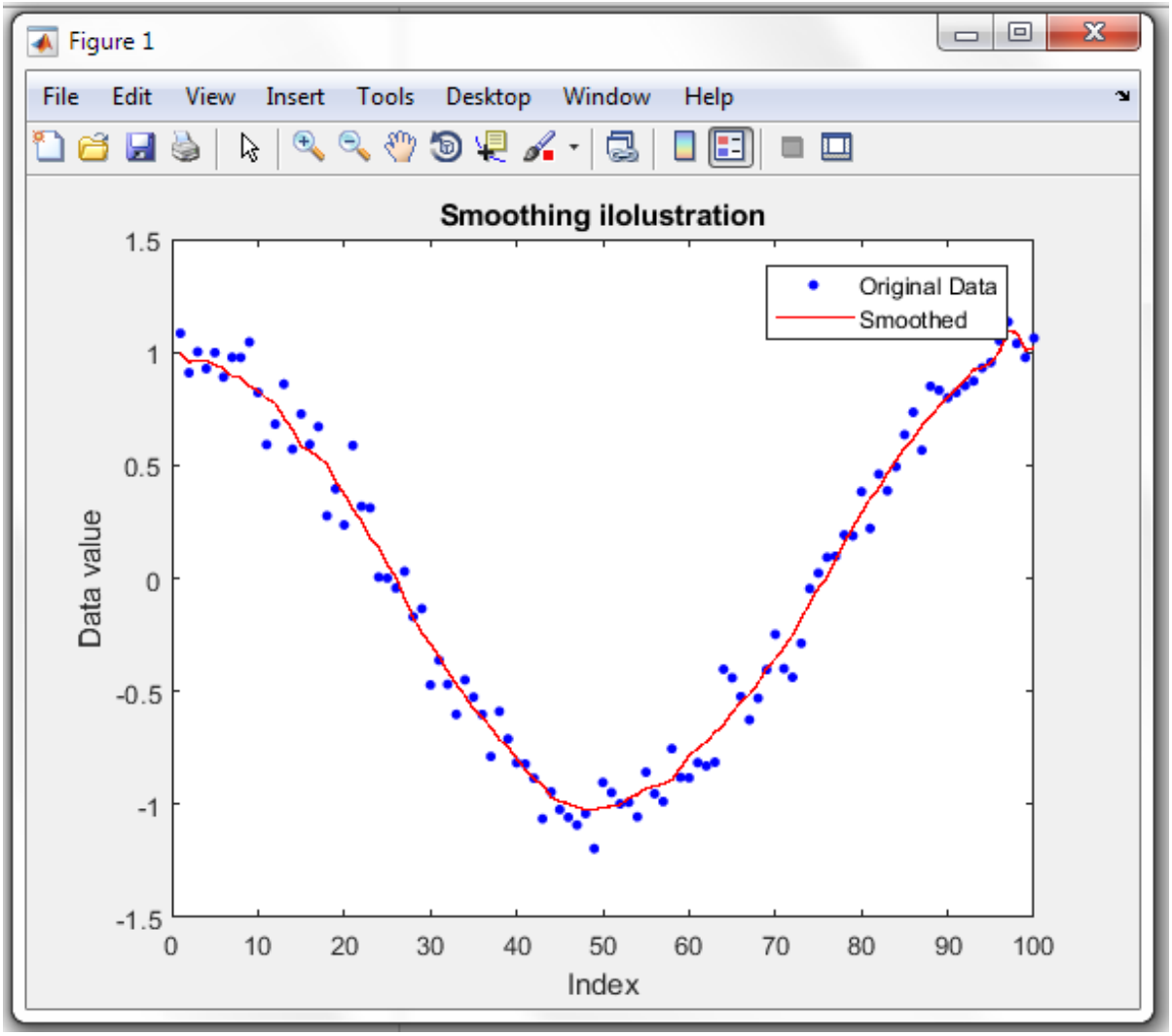


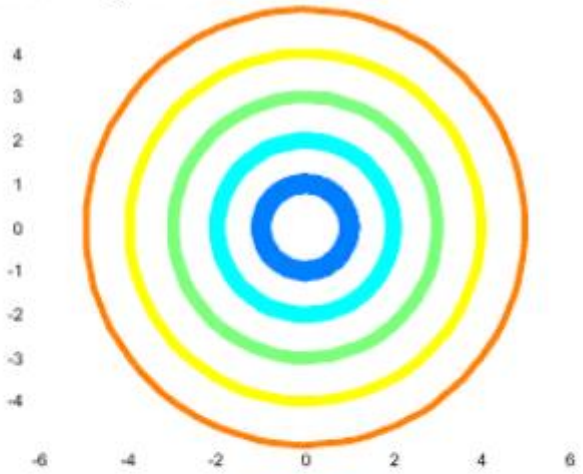
Figure 1
File Edit View Insert Tools Desktop Window Help
Smoothing ilolustration
Original Data
Smoothed
Data value
Index
1.5
1
0.5
0
-0.5
-1
-1.5
0 10 20 30 40 50 60 70 80 90 100

8. **Plot a circle.** It's not immediately obvious how to plot a circle in MATLAB. Write the function `[x,y]=getCircle(center,r)` to get the x and y coordinates of a circle. The circle should be centered at `center` (2-element vector containing the x and y values of the center) and have radius `r`. Return `x and y` such that `plot(x,y)` will plot the circle.

   a. Recall that for a circle at the origin (0,0), the following is true: , for t on $x\,t(\,) = \cos(\,)t$ $y\,t(\,) = \sin(\,)t$

   the range $[0, 2p\,]$. Now, you just have to figure out how to scale and translate it.

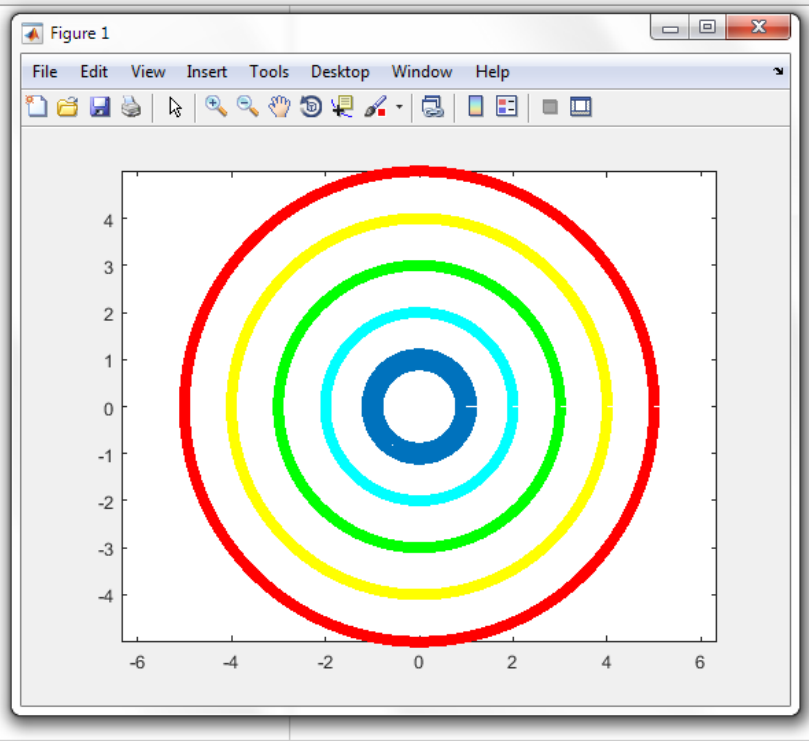
   b. Write a script called `concentric.m`. In this script, open a new figure and plot five circles, all centered at the origin and with increasing radii. Set the line width for each circle to something thick (at least 2 points), and use the colors from a 5-color `jet` colormap (**jet**). The property names for line width and color are `'LineWidth'` and `'Color'`, respectively. Other useful function calls are **hold on** and **axis equal**. It should look something like this:



   c. Make a script called `olympic.m`. This script should use your `getCircle` function to draw the Olympic logo, as shown below. Don't worry about making the circles overlap in the same way as the official logo (it's possible but is too complicated for now). Also, when specifying colors, you can use the same color codes as when plotting lines ('b' for blue, 'k' for black, etc.) instead of providing an RGB vector.

concentric.m

```
% Пороскун Олени ПМ-81
% Homework 2
% Task 8.Plot a circle.

center = [0, 0];
r = 1;
[x,y] = getCircle(center,r);

colors = ['b', 'c', 'g', 'y', 'r'];

for rad = 2:5
    t1 = 0:0.01:2*pi;
    x1 = rad*cos(t1);
    y1 = rad*sin(t1);
    plot(x1, y1, colors(rad), 'LineWidth', 6);
end


function [x,y] = getCircle(center,r)
figure;
t = 0:0.001:2*pi;
x = r*cos(t);
y = r*sin(t);
plot(center(1)+x, center(2)+y, 'LineWidth', 12);
hold on
axis('equal')
end

```

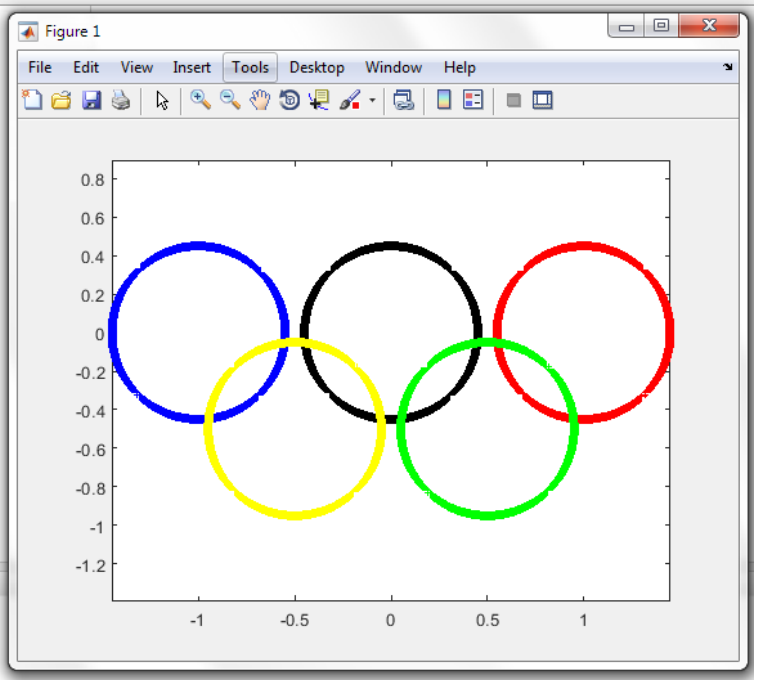


Command Window

```
>> concentric
>>
```

olympic.m

```
% Пороскун Олени ПМ-81
% Homework 2
% Task 8.Plot a circle.

r = 0.45;
center_x = [-1, 0, 1, -0.5, 0.5];
center_y = [0, 0, 0, -0.5, -0.5];
colours = ['b', 'k', 'r', 'y', 'g'];

figure;

for i = 1:5
    [x,y] = getCircles([center_x(i), center_y(i)], r, colours(i));
end

function [x,y] = getCircles(center, r, color)
t = 0:0.001:2*pi;
x = r*cos(t);
y = r*sin(t);
plot(center(1)+x, center(2)+y, color, 'LineWidth', 5);
hold on
axis('equal')
end
```



Command Window

```
>> olympic
>>
```

Optional Problems

9. **Logical indexing and piecewise plots. Transistor I-V curves**. A transistor is an electrical component which has 3 terminals[3]. The terminals are named Gate, Source and Drain. The name transistor derives from the words trans-resistor, meaning a variable resistor. We can control the current flowing through the transistor (namely between the drain and source nodes - I_{DS}) by changing the relative voltage applied between the gate and the source (V_{GS}). The following equations describes this current which depends also on the voltage between the drain and source nodes (V_{DS}).

$$I_{DS} = \begin{cases} 0 & V_{GS} - V_T < 0 \le V_{DS} \quad \text{cutoff} \\ K\left[\left(V_{GS} - V_T\right)V_{DS} - \frac{1}{2}V_{DS}^2\right] & 0 \le V_{DS} \le V_{GS} - V_T \quad \text{linear} \\ \frac{1}{2}K\left(V_{GS} - V_T\right)^2 & 0 < V_{GS} - V_T < V_{DS} \quad \text{saturation} \end{cases}$$

K is a constant, dependent on the transistor technology and dimensions, with units of amperes/volts^2. V_T is a characteristic threshold voltage, below which the transistor is regarded as "off". The three current regions are designated as cutoff (no current flowing), linear (current is linear with respect to V_{DS}) and saturation (current is independent of V_{DS}).

In this example we will assume a transistor with a threshold voltage V_T=1V and K=50μA/V^2.

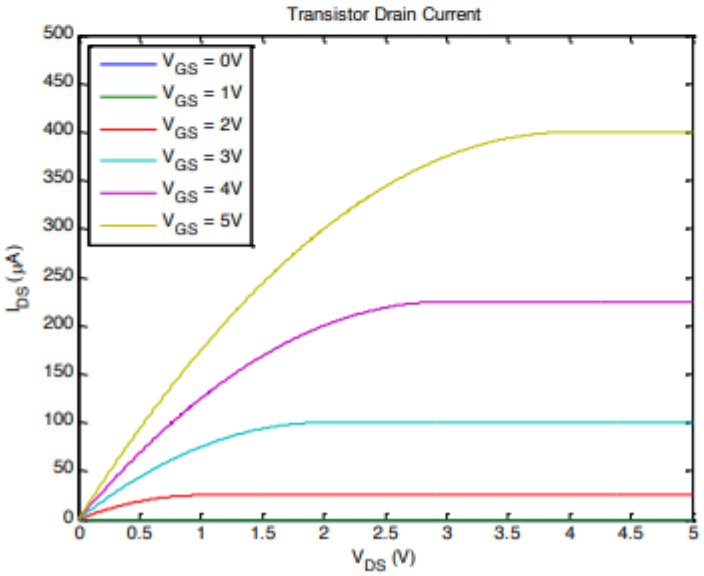
We wish to plot the characteristic I-V curve for this transistor, i.e. a plot showing the drain-source current as a function of the drain-source voltage for several gate-source voltages. In other words, we wish to have a plot where V_{DS} is the independent variable and V_{GS} will be a parameter. The curve of current vs. drain voltage will be plotted for several gate voltages.

We will assume that V_{DS} can change between 0 and 5V, and that we wish to explore the curve for V_{GS} in the set [0, 1, 2, 3, 4, 5] V.

One way to plot these graphs would be to loop through all possible values of V_{GS}, calculate the current, plot, hold and repeat the process. Alternatively, if you wish to practice the power of vectorization in MATLAB, we can actually complete the entire parameterized calculation without any loops. We may construct a **meshgrid** of V_{DS} and V_{GS} and calculate I_{DS} concurrently. I_{DS} will be a matrix where each column corresponds to a different V_{GS} parameterization of I_{DS} vs. V_{DS}. Feel free to implement either or both solutions.

a. Create the desired vectors/matrices for `Vds` and `Vgs` (`Vgs only has 6 values!`). Remember, we wish to have the plot of I_{DS} vs. V_{DS} for every V_{GS} value, what is the appropriate command to create these variables?
b. Based on the variables created, we can create logical vectors (or matrices) indicating the index locations for each operating mode. Create a logical variable `lin` which is logical 1 when the transistor is in the linear region and zero elsewhere. Similarly, create a logical variable `sat` which is logical 1 in the saturation region and zero elsewhere.
c. Create the variable `Ids` according to the equation given previously and your constructed logical variables for each domain.
d. Plot the various curves on a single plot.

The end result should look similar to this:

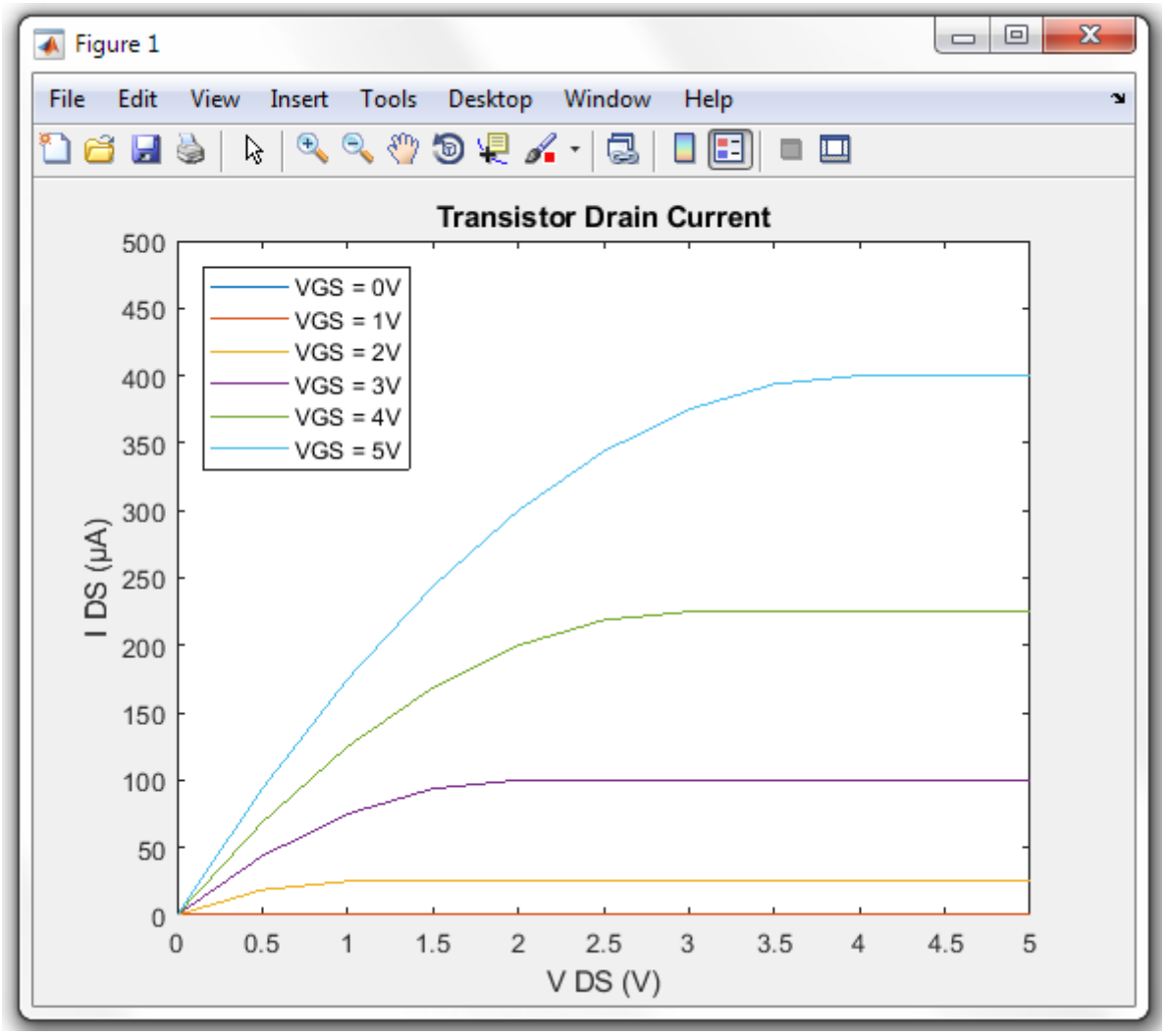


Task_9.m

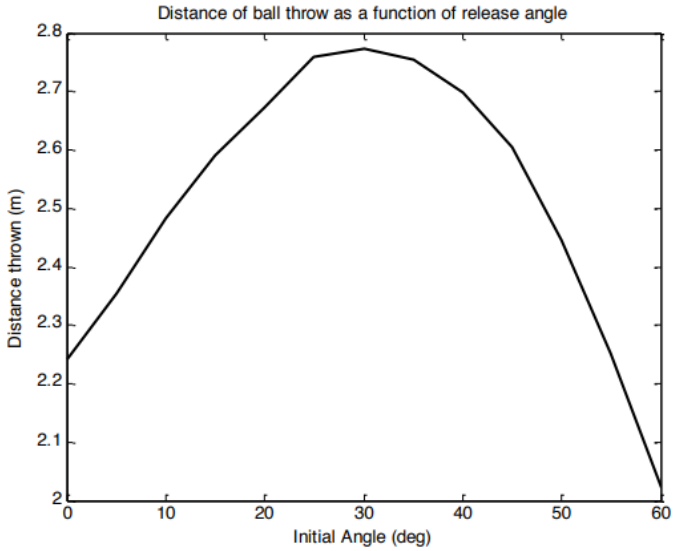
```
% Пороскун Олени ПМ-81
% Homework 2                                Optional Problems
% 9 Logical indexing and piecewise plots. Transistor I-V curves.

Vt = 1;
K = 50;
Vds = 0:0.5:5;
Vgs = [0, 1, 2, 3, 4, 5];

for i = 1:6
    for j = 1:length(Vds)
        lin(j) = (Vds(j) >= 0 &  Vgs(i) - Vt >= Vds(j));
        sat(j) = (Vds(j) > Vgs(i) - Vt & Vgs(i) - Vt > 0);
        if (Vds(j) >= 0 & Vgs(i) - Vt < 0)                    % cutof
            Ids(j) = 0;
        elseif (lin(j) == 1)                                  % linear
            Ids(j) = K*((Vgs(i) - Vt)*Vds(j) - 0.5*(Vds(j)^2));
        elseif (sat(j) == 1)                                  % saturation
            Ids(j) = 0.5*K*((Vgs(i) - Vt)^2);
        end
    end
    Ids
    plot(Vds, Ids)
    hold on
end

title('Transistor Drain Current');
ylim([0 500]);
xlabel('V DS (V)');
ylabel('I DS (μA)');
legend('VGS = 0V', 'VGS = 1V', 'VGS = 2V', 'VGS = 3V', 'VGS = 4V', 'VGS = 5V', 'Location','northwest');
```

Command Window

```
>> Task_9

Ids =

     0     0     0     0     0     0     0     0     0     0     0

Ids =

     0     0     0     0     0     0     0     0     0     0     0

Ids =

         0   18.7500   25.0000   25.0000   25.0000   25.0000   25.0000   25.0000   25.0000   25.0000   25.0000

Ids =

         0   43.7500   75.0000   93.7500  100.0000  100.0000  100.0000  100.0000  100.0000  100.0000  100.0000

Ids =

         0   68.7500  125.0000  168.7500  200.0000  218.7500  225.0000  225.0000  225.0000  225.0000  225.0000

Ids =

         0   93.7500  175.0000  243.7500  300.0000  343.7500  375.0000  393.7500  400.0000  400.0000  400.0000

fx >>
```

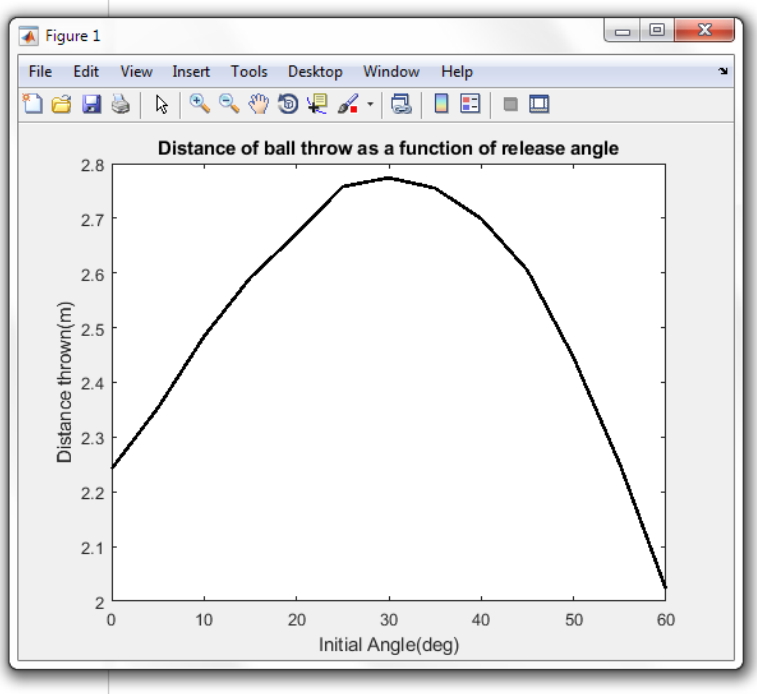


9. **Functions**. If you wrote the ball throwing script for Homework 1, turn it into a function. Add a function declaration that takes `v` and `theta` as inputs and returns the `distance` at which the ball hits the ground: `distance=throwBall(v,theta)`. We generally don't want functions to plot things every time they run, so remove the figure command and any plotting commands from the function. To be able to simulate a wide range of v and theta, make the time go until 10sec and add an `if` statement that will display the warning 'The ball does not hit the ground in 10 seconds' if that turns out to be the case. Also, if the ball doesn't hit the ground in 10 seconds, you should return `NaN` as the `distance`. To test your function, write a script `testBall.m` to throw the ball with the same velocity `v` but different angles and plot the distance as a function of angle `theta`. The plot should look something like the figure below. You will need to run `throwBall` within a loop in order to calculate the distances for various `thetas`.



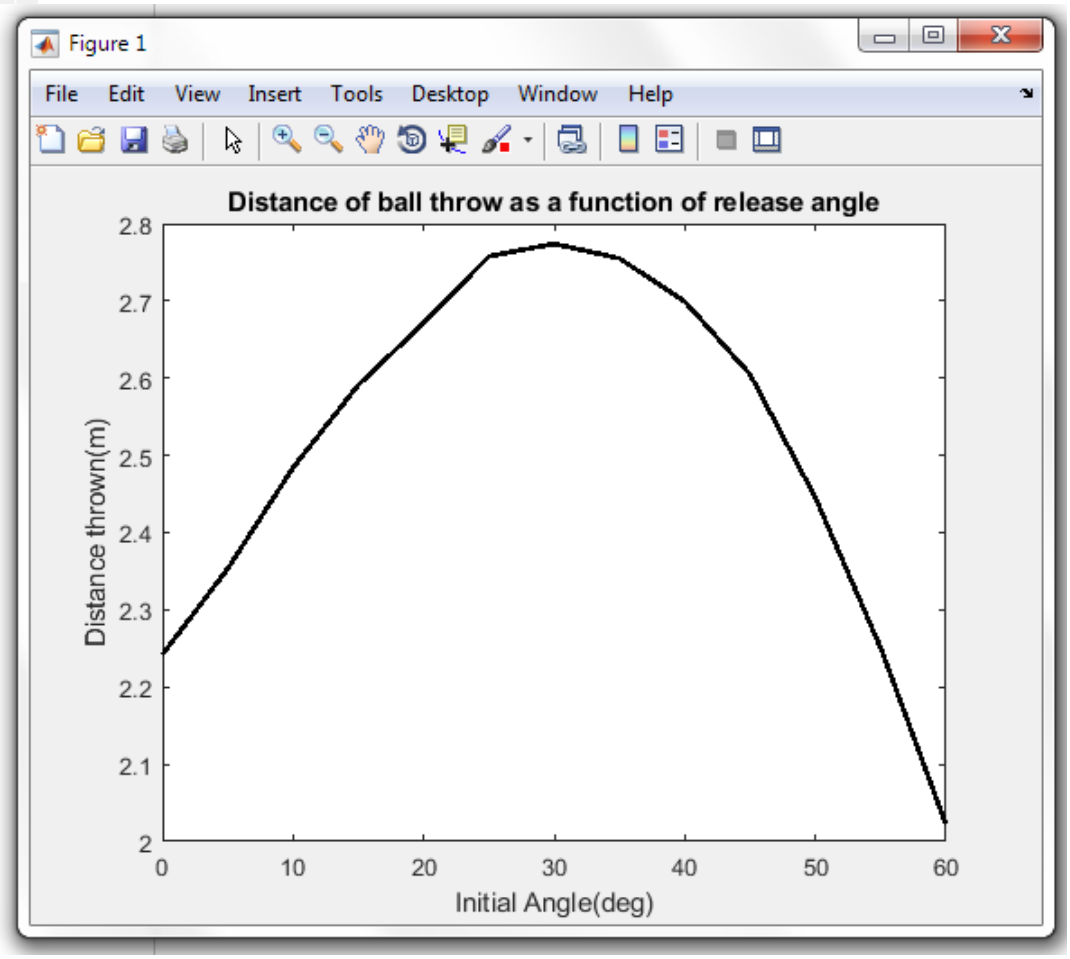
testBall.m

```
% Пороскун Олени ПМ-81
% Homework 2                              Optional Problems
% 9 Functions.

v = 4;
theta = 0:5:60;
for i = 1:length(theta)
    distance(i) = throwBall(v,theta(i));
end

plot(theta, distance, 'k', 'Linewidth', 2);

xlabel('Initial Angle(deg)');
ylabel('Distance thrown(m)');
title('Distance of ball throw as a function of release angle');

function distance = throwBall(v,theta)

HeightAtRel = 1.5;
Gravit = 9.8;

t = linspace(0, 10, 1000);
x = v *(cos(theta*(pi/180)))*t;
y = HeightAtRel + v*(sin(theta*(pi/180)))*t - 0.5*Gravit*(t.^2);

if (isempty(find(y<0)) == 1)
    disp('The ball does not hit the ground in 10 seconds.')
    distance = NaN;
else
    dis = x(find(y <= 0));
    distance = dis(1);
end
%answer = sprintf('distance = %g', distance);
%disp(answer);
end
```

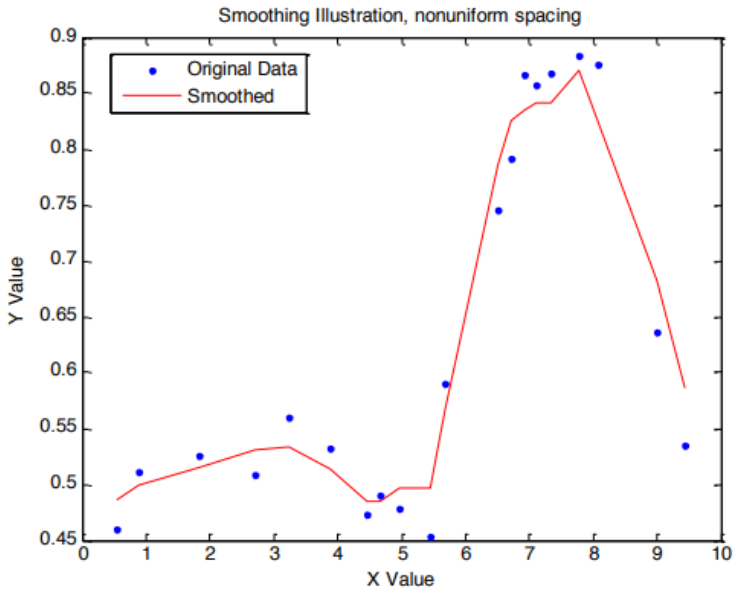


testBall.m

```
% Пороскун Олени ПМ-81
% Homework 2                                  Optional Problems
% 9 Functions.

v = 4;
theta = 0:5:60;
for i = 1:length(theta)
    distance(i) = throwBall(v,theta(i));
end

plot(theta, distance, 'k', 'Linewidth', 2);

xlabel('Initial Angle(deg)');
ylabel('Distance thrown(m)');
title('Distance of ball throw as a function of release angle');

function distance = throwBall(v,theta)

HeightAtRel = 1.5;
Gravit = 9.8;

t = linspace(0, 10, 1000);
x = v *(cos(theta*(pi/180)))*t;
y = HeightAtRel + v*(sin(theta*(pi/180)))*t - 0.5*Gravit*(t.^2);

if (isempty(find(y<0)) == 1)
    disp('The ball does not hit the ground in 10 seconds.')
    distance = NaN;
else
    dis = x(find(y <= 0));
    distance = dis(1);
end
%answer = sprintf('distance = %g', distance);
%disp(answer);
end
```



10. **Smoothing nonuniformly sampled data.** Modify your smoothing filter above to also work on data that are not sampled at a constant rate. Modify the program to also accept `x` if it is an `Nx2` matrix, where the first column is the x values and the second column is the y values of the data points (such that doing `plot(x(:,1),x(:,2),'.')` would plot the data). In this case, `width` should be on the same scale as the scale of the x values, so for example if the x(:,1) values are on the range [0,1], a `width` of 0.2 is acceptable. Also, in this case the `width` doesn't have to be odd as before. Assume that the x values are in increasing order. The output should also be an `Nx2` matrix, where the first column is the same as the first column in `x`, and the second column contains the smoothed values. The most efficient way to do this is unclear; you may find the interpolating function **interp1** helpful, but you can definitely do this without using it. The file optionalData.mat contains a matrix `x` that you can test your function on. When smoothed with a width of 2, you should get a result like:

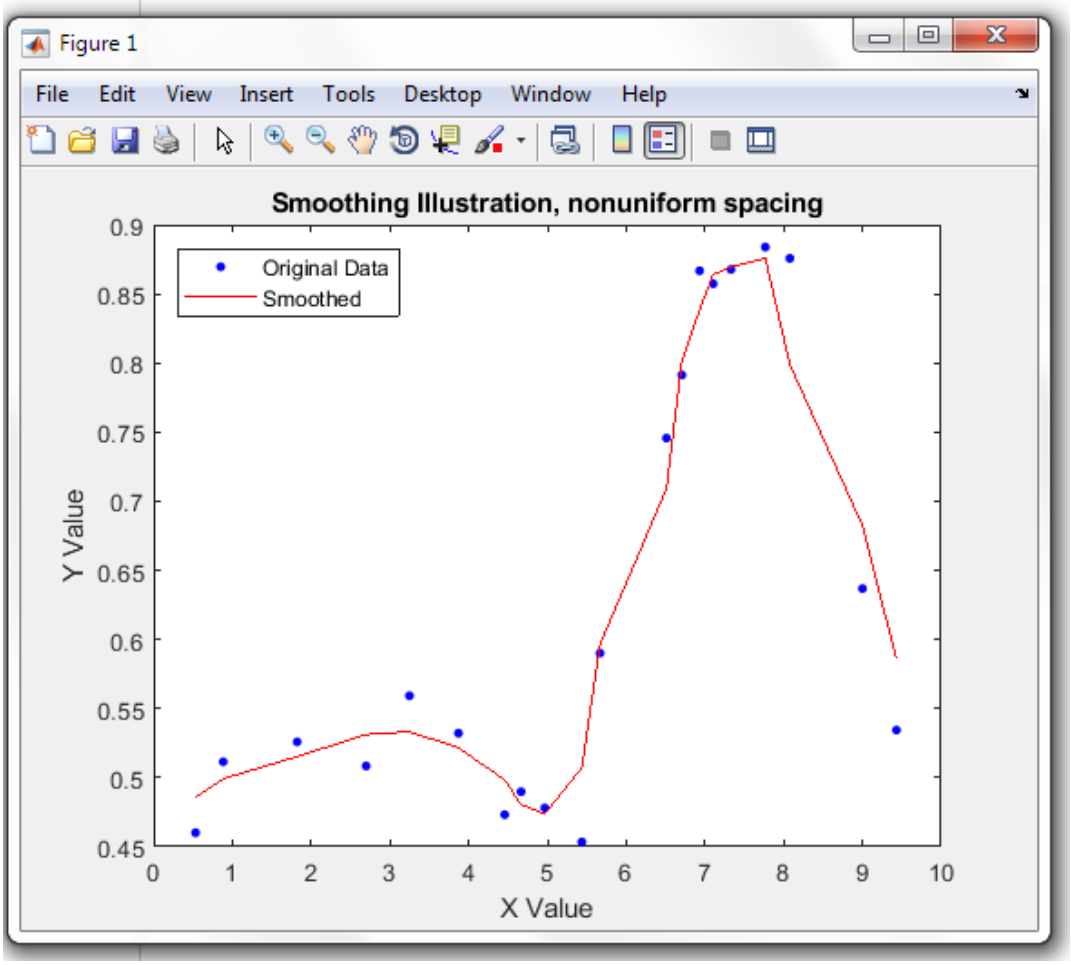


Task_10_NewFunct.m

```
% Пороскун Олени ПМ-81
% Homework 2                                    Optional Problems
% 10. Smoothing nonuniformly sampled data.

load('optionalData.mat');
x
Width = input('Width is ')
smoothed = rectFilt(x, Width);

function smoothed = rectFilt(x, width)

figure;
plot(x(:,1), x(:,2), 'b.', 'MarkerSize', 11);
hold on

X = x(:, 1);
Y = x(:, 2);

lim = floor(width/2);
lim1 = 1 + max(1, lim);
lim2 = length(x) - min(length(x),lim);

for i = 1:length(x)
    smoothed(i, 1) =  X(i);
    if(i < lim1)
        smoothed(i,2) = mean(Y(i:i+1));
    elseif(i > lim2)
        smoothed(i,2) = mean(Y(i-1:i));
    else %(i >= lim1 || i <= lim2)
        smoothed(i,2) = mean(Y(i - lim : i + lim));
    end
end
sm_2 = smoothed(:,2)
plot(smoothed(:, 1), smoothed(:, 2), 'r')
xlabel('X Value');
ylabel('Y Value');
legend('Original Data','Smoothed', 'Location','northwest');
title('Smoothing Illustration, nonuniform spacing');
end
```

```
x =

    0.5328    0.4602
    0.8836    0.5115
    1.8226    0.5259
    2.6985    0.5084
    3.2487    0.5592
    3.8711    0.5322
    4.4582    0.4732
    4.6659    0.4898
    4.9672    0.4781
    5.4381    0.4535
    5.6686    0.5901
    6.5129    0.7457
    6.7076    0.7914
    6.9336    0.8668
    7.1043    0.8574
    7.3319    0.8678
    7.7676    0.8839
    8.0787    0.8757
    9.0021    0.6368
    9.4323    0.5344

Width is 2

Width =

     2


sm_2 =
```

```
sm_2 =

    0.4858
    0.4992
    0.5153
    0.5312
    0.5333
    0.5215
    0.4984
    0.4804
    0.4738
    0.5072
    0.5964
    0.7091
    0.8013
    0.8385
    0.8640
    0.8697
    0.8758
    0.7988
    0.6823
    0.5856
```



11. **Buy and sell a stock**. Write the following function:

    `endValue=tradeStock(initialInvestment, price, buy, sell)`

    The weekly `price` of Google stock from 8/23/2004 until 1/11/2010 is saved in the file googlePrices.mat[4]. This file also contains two other vectors: `peaks` and `lows`. The `peaks` vector contains indices into the `price` vector when the stock peaked, and the `lows` vector contains indices into the `price` vector when the stock hit a low. Run your program so that it buys stock when it's low, and sell it when it's high. Below is a list of specifications.

    a. The inputs are: `initialInvestment` – the amount of money you have to invest at the beginning, in dollars; `price` – a vector of stock prices over time; `buy` – a vector of times when to buy (should just be integers that index into the price vector); `sell` – a vector of times when to sell (similar to the buy vector). `endValue` is the end value of your investment. If all of your stock isn't sold at the end, use the last price of the stock to calculate how much it's worth and add it to your available cash. Make the function general so that it will work with any given `price`, `buy` and `sell` vectors.

    b. You can only buy integer numbers of shares of stock (you can't buy 3.23 shares). You also can't buy more stock than you can afford (if the current price is $100 and you have $599 in cash, you can only buy 5 shares). When deciding how much you can buy, factor in the transaction cost (see next section) so that you don't go negative. When buying, always buy as many shares as you can, and when selling, sell all your shares.

    c. Each buy or sell transaction costs you $12.95, make sure you include this in your program. If your initial investment is small, you may not be able to carry out all the buy and sell orders. It may also not make sense to sell at each specified time: for example if you bought 10 shares of stock and the price increases by $1, it doesn't make sense to sell it since the transaction cost would eat up all your profit and cost you $2.95 extra. However you can't make the decision of whether to buy or not since you don't know where the price is going to go (you can't look forward in time). You don't have to be super clever about deciding whether to sell or not, but you can if you're so inclined.

    d. After it's complete, run your program with various initial investments. Load googlePrices.mat into your workspace and then run the function using the `price`, `peaks`, and `lows` vectors. To check that you did it right, try an initial investment of $100. This should give you an end value of $100. Also try an initial investment of $100000, which should result in a total value of about $61,231,407 (with a $100,000 initial investment, it turns out that you don't even have to decide whether it's a good idea to sell because the transaction cost becomes negligible compared to the number of shares you have).

Task11Function.m

```
1     % Пороскун Олени ПМ-81
2     % Homework 2                                    Optional Problems
3     % 11. Buy and sell a stock.
4
5     load ('googlePrices.mat');
6
7     buy = lows;
8     sell = peaks;
9
10    initialInvestment = 100;
11    a = sprintf('initialInvestment = %10.3f', initialInvestment);
12    disp(a)
13    endValue = tradeStock(initialInvestment, price, buy, sell);
14
15    initialInvestment = 100000;
16    a = sprintf('initialInvestment = %10.3f', initialInvestment);
17    disp(a)
18    endValue = tradeStock(initialInvestment, price, buy, sell);
19
20    function endValue = tradeStock(initialInvestment, price, buy, sell)
21
22    endValue = initialInvestment;
23    for i = 1:length(buy)
24        k = floor(endValue/price(buy(i)));
25        if (k > 0)
26            endValue = endValue - k*price(buy(i)) - 12.95;
27            endValue = endValue + k*price(sell(i)) - 12.95;
28        end
29    end
30    endValue = sprintf('endValue = %10.3f', endValue);
31    disp(endValue)
32    end
```

Command Window

```
>> Task11Function
initialInvestment =    100.000
endValue =    100.000
initialInvestment = 100000.000
endValue = 61233814.730
fx >>
```

Sumy State University

Department

of

Applied Mathematics and Complex Systems Modeling

Report on laboratory work №3

Subject

Computing system

Student: Olena Poroskun

Teacher: Igor A. Knyaz

Sumy, Sumy region

2020

1. **Linear system of equations.** Solve the following system of equations using ****. Compute and display the error vector

$$3a+6b+4c=1$$
$$a+5b=2$$
$$7b+7c=3$$

```
% Пороскун Олени ПМ-81
% Homework 3

% 1. Linear system of equations.
A = [3, 6, 4; 1, 5, 0; 0, 7, 7];
B = [1; 2; 3];
x=A\B
```

Command Window

```
x =

   -0.5824
    0.5165
   -0.0879
```

2. **Numerical integration.** What is the value of: $\int_0^5 xe^{-x/3}dx$? Use **trapz** or **quad**. Compute and display the difference between your numerical answer and the analytical answer: $-24e^{-5/3}+9$.

```
% Пороскун Олени ПМ-81
% Homework 3

% 2. Numerical integration.

x = 0:5/100:5;
y = x.*exp(-x/3);
I = trapz(x, y)

analit = -24*exp(-5/3)+9

difference = abs(I - analit);
dif = sprintf('difference = %.4f', difference);
disp(dif)
```

Command Window

```
I =

    4.4668


analit =

    4.4670

difference = 0.0002
fx >>
```

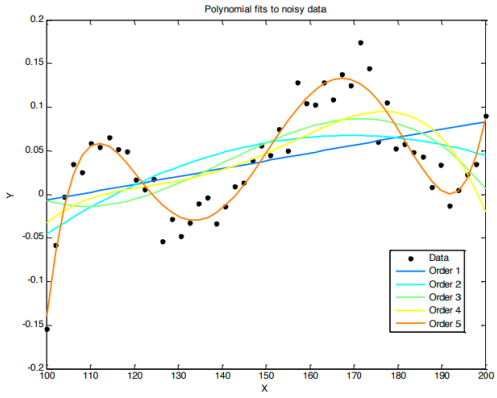
3. **Computing the inverse.** Calculate the inverse of $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and verify that when you multiply the original matrix by the inverse, you get the identity matrix (**inv**). Display the inverse matrix as well as the result of the multiplication of the original matrix by its inverse.

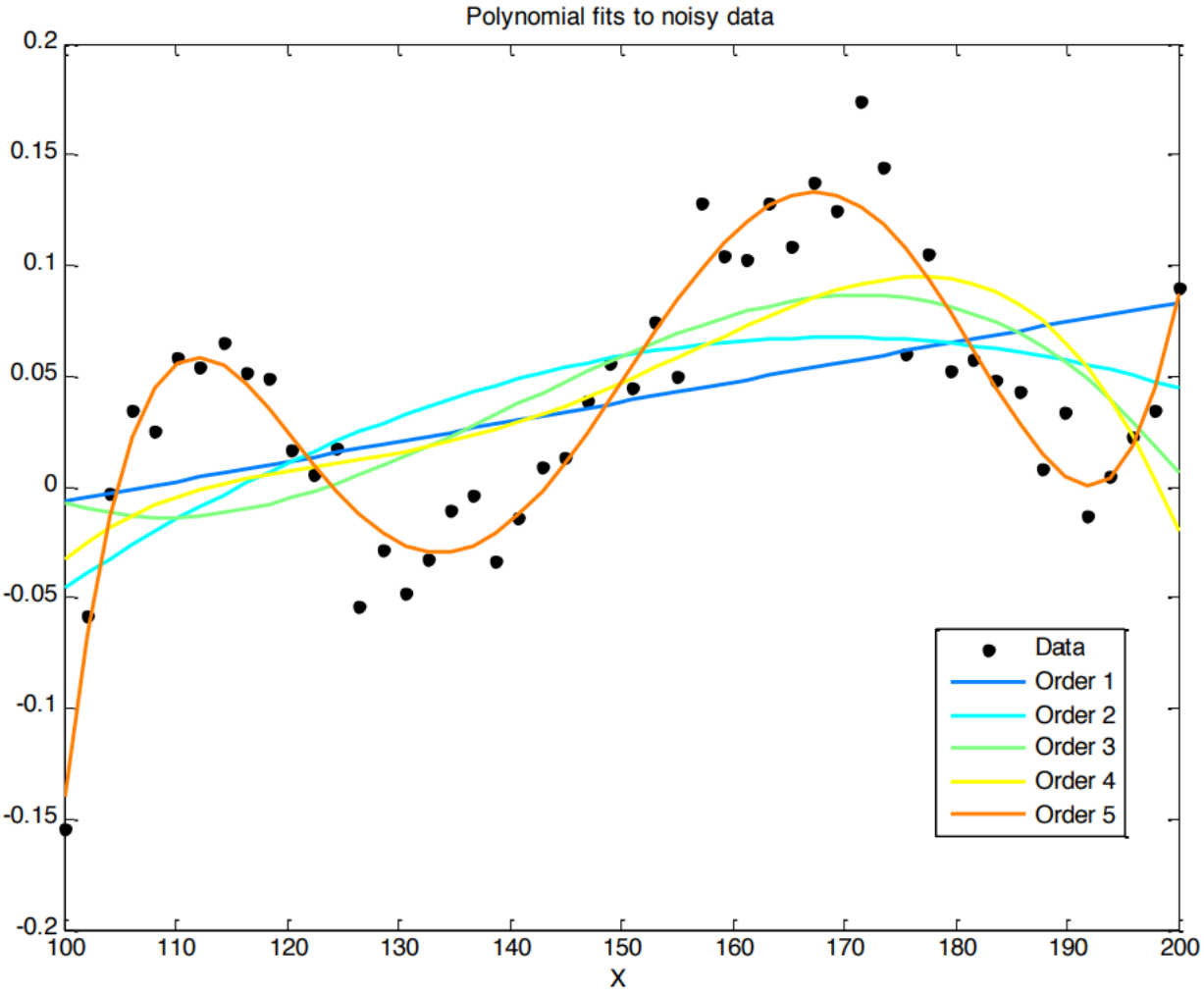
```
% Пороскун Олени ПМ-81
% Homework 3

% 3. Computing the inverse.

A = [1, 2; 3, 4]
Inv_A = inv(A)
B = A*Inv_A  %одинична матриця
```

Command Window

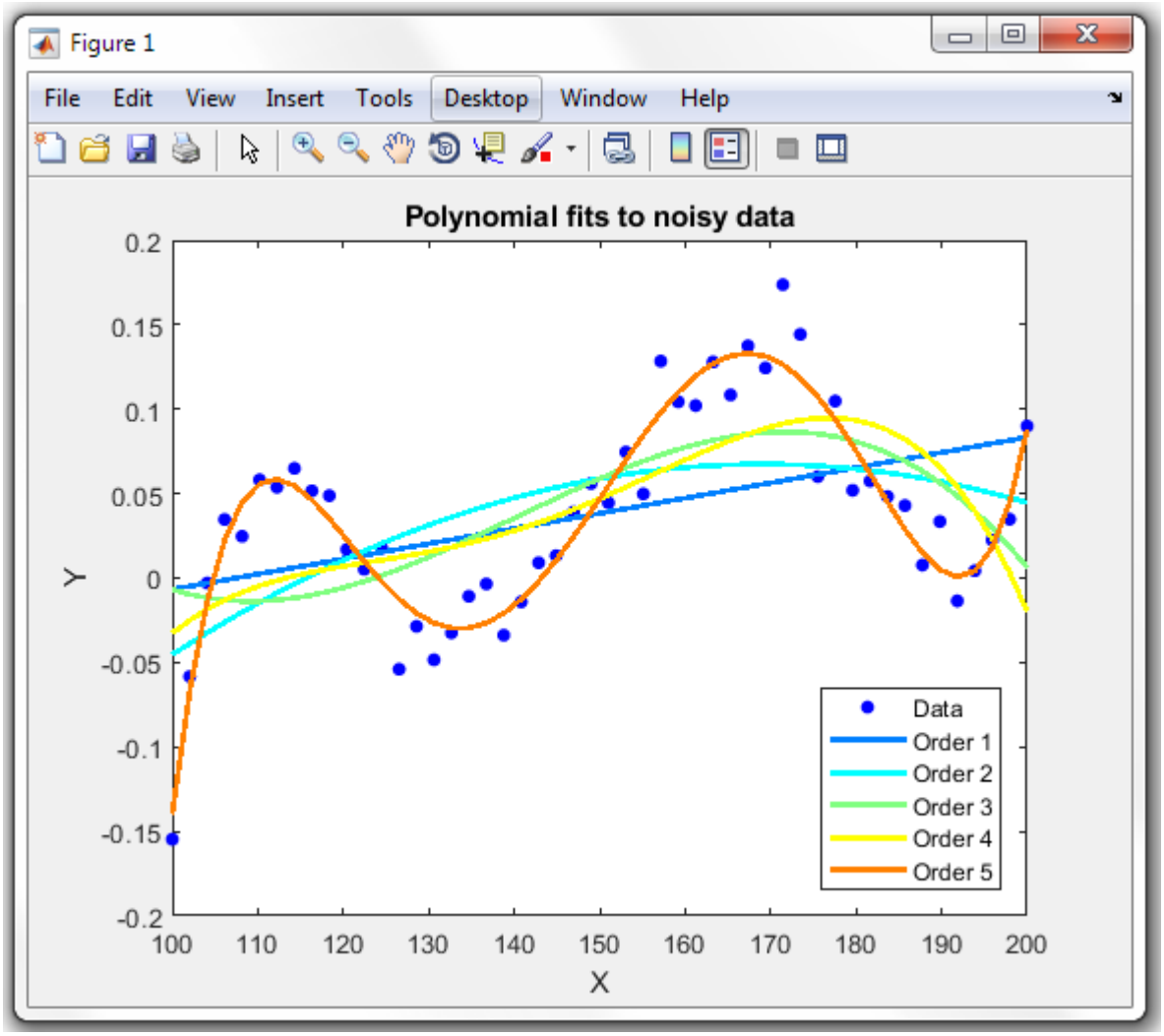
```
A =

     1     2
     3     4


Inv_A =

   -2.0000    1.0000
    1.5000   -0.5000


B =

    1.0000         0
    0.0000    1.0000
```

4. **Fitting polynomials**. Write a script to load the data file `randomData.mat` (which contains variables `x` and `y`) and fit first, second, third, fourth, and fifth degree polynomials to it. Plot the data as blue dots on a figure, and plot all five polynomial fits using lines of different colors on the same axes. Label the figure appropriately. To get good fits, you'll have to use the centering and scaling version of **polyfit** (the one that returns three arguments, see **help**) and its counterpart in **polyval** (the one that accepts the centering and scaling parameters). It should look like this:





```
% Пороскун Олени ПМ-81
% Homework 3

% 4. Fitting polynomials.

figure;
load('randomData.mat');
plot(x, y, '.b', 'MarkerSize', 15);
hold on

col = jet(5);
for i = 1:5
    [P,S,MU] = polyfit(x,y,i);
    f1 = polyval(P,x,[],MU);
    plot(x, f1,'Color', col(i,:), 'LineWidth', 2);
end

xlabel('X')
ylabel('Y')
title('Polynomial fits to noisy data')
legend('Data','Order 1','Order 2','Order 3','Order 4','Order 5','Location','southeast')
```



5. **Hodgkin-Huxley model of the neuron.** You will write an ODE file to describe the spiking of a neuron, based on the equations developed by Hodgkin and Huxley in 1952 (they received a Nobel Prize for this work). The main idea behind the model is that ion channels in the neuron's membrane have voltage-sensitive gates that open or close as the transmembrane voltage changes. Once the gates are open, charged ions can flow through them, affecting the transmembrane voltage. The equations are nonlinear and coupled, so they must be solved numerically.

   a. Download the HH.zip file from the class website and unzip its contents into your homework folder. This zip folder contains 6 m-files: `alphah.m`, `alpham.m`, `alphan.m`, `betah.m`, `betam.m`, `betan.m`. These functions return the voltage-dependent opening ($\alpha(V)$) and closing ($\beta(V)$) rate constants for the h, m, and n gates.

   b. Write an ODE file to return the following derivatives (you don't need to understand what they mean):

$$\frac{dn}{dt} = (1-n)\alpha_n(V) - n\beta_n(V)$$

$$\frac{dm}{dt} = (1-m)\alpha_m(V) - m\beta_m(V)$$

$$\frac{dh}{dt} = (1-h)\alpha_h(V) - h\beta_h(V)$$

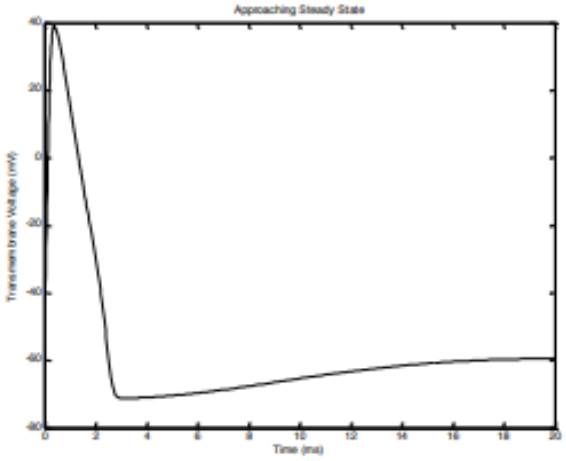
$$\frac{dV}{dt} = -\frac{1}{C}\left(G_K n^4 (V - E_K) + G_{Na} m^3 h (V - E_{Na}) + G_L (V - E_L)\right)$$

   and the following constants (C is membrane capacitance, G are the conductances and E are the reversal potentials of the potassium (K), sodium (Na), and leak (L) channels):

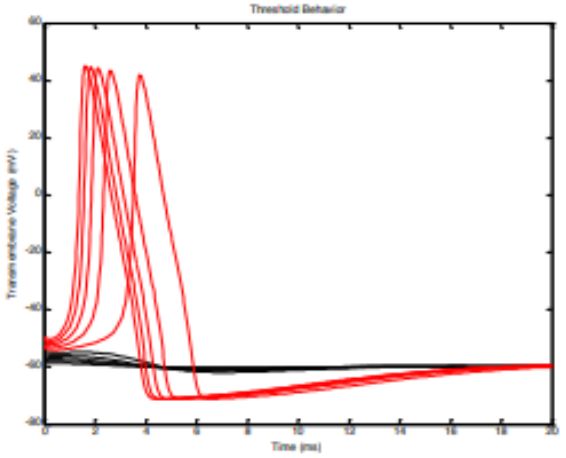
$$C = 1$$
$$G_K = 36$$
$$G_{Na} = 120$$
$$G_L = 0.3$$
$$E_K = -72$$
$$E_{Na} = 55$$
$$E_L = -49.4$$

   c. Write a script called `HH.m` which will solve this system of ODEs and plot the transmembrane voltage. First, we'll run the system to steady state. Run the simulation for 20ms (the timescale of the equations is ms) with initial values: $n = 0.5; m = 0.5; h = 0.5; V = -60$ (**ode45**). Store the steady-state value of all 4 parameters in a vector called `ySS`. Make a new figure and plot the timecourse of $V(t)$

to verify that it reaches steady state by the end of the simulation. It should look something like this:



d. Next, we'll explore the trademark feature of the system: the all-or-none action potential. Neurons are known to 'fire' only when their membrane surpasses a certain voltage threshold. To find the threshold of the system, solve the system 10 times, each time using `ySS` as the initial condition (**ode45** will take this as an input argument) but increasing the initial value of V by 1, 2, … 10 mV from its steady state value. After each simulation, check whether the peak voltage surpassed 0mV, and if it did, plot the voltage with a red line, but if it didn't, plot it with a black line. Plot all the membrane voltage trajectories on the same figure, like below. We see that if the voltage threshold is surpassed, then the neuron 'fires' an action potential; otherwise it just returns to the steady state value. You can zoom in on your figure to see the threshold value (the voltage that separates the red lines from the black lines).



t_5.m | ODE.m | HH.m

```m
% Пороскун Олени ПМ-81
% Homework 3
% 5. Hodgkin-Huxley model of the neuron.

function F = ODE(t,y)
C = 1;
G_K = 36;
G_Na = 120;
G_L = 0.3;
E_K = -72;
E_Na = 55;
E_L = -49.4;

dn = (1-y(1))*alphan(y(4)) - y(1)*betan(y(4));
dm = (1-y(2))*alpham(y(4)) - y(2)*betam(y(4));
dh = (1-y(3))*alphah(y(4)) - y(3)*betah(y(4));
dV = -(1/C)*(G_K*(y(1)^4)*(y(4)-E_K)+G_Na*(y(2)^3)*y(3)*(y(4)-E_Na)+G_L*(y(4)-E_L));

F = [dn; dm; dh; dV];
end
```

ODE.m | HH.m

```m
% Пороскун Олени ПМ-81
% Homework 3
% 5. Hodgkin-Huxley model of the neuron.

figure(1);
[t, y] = ode45(@ODE, [0 20],[0.5 0.5 0.5 -60]);
V = y(:, 4);
plot(t, V, 'k');
title('Approaching Steady State')
xlabel('Time (ms)');
ylabel('Transmembrane Voltage (mV)');

figure(2);
yss = y(end, :);
for i = 1:10
    y_increas = [0, 0, 0, i];
    y_0 = yss + y_increas;
    [t, y] = ode45(@ODE, [0 20], y_0);
    V = y(:, 4);
    if(max(V)>0)
        plot(t, V, 'r');
    else
        plot(t, V, 'k');
    end
    hold on
end

title('Threshold Behavior')
xlabel('Time (ms)');
ylabel('Transmembrane Voltage (mV)');
```

Command Window

```
>> HH
>>
```

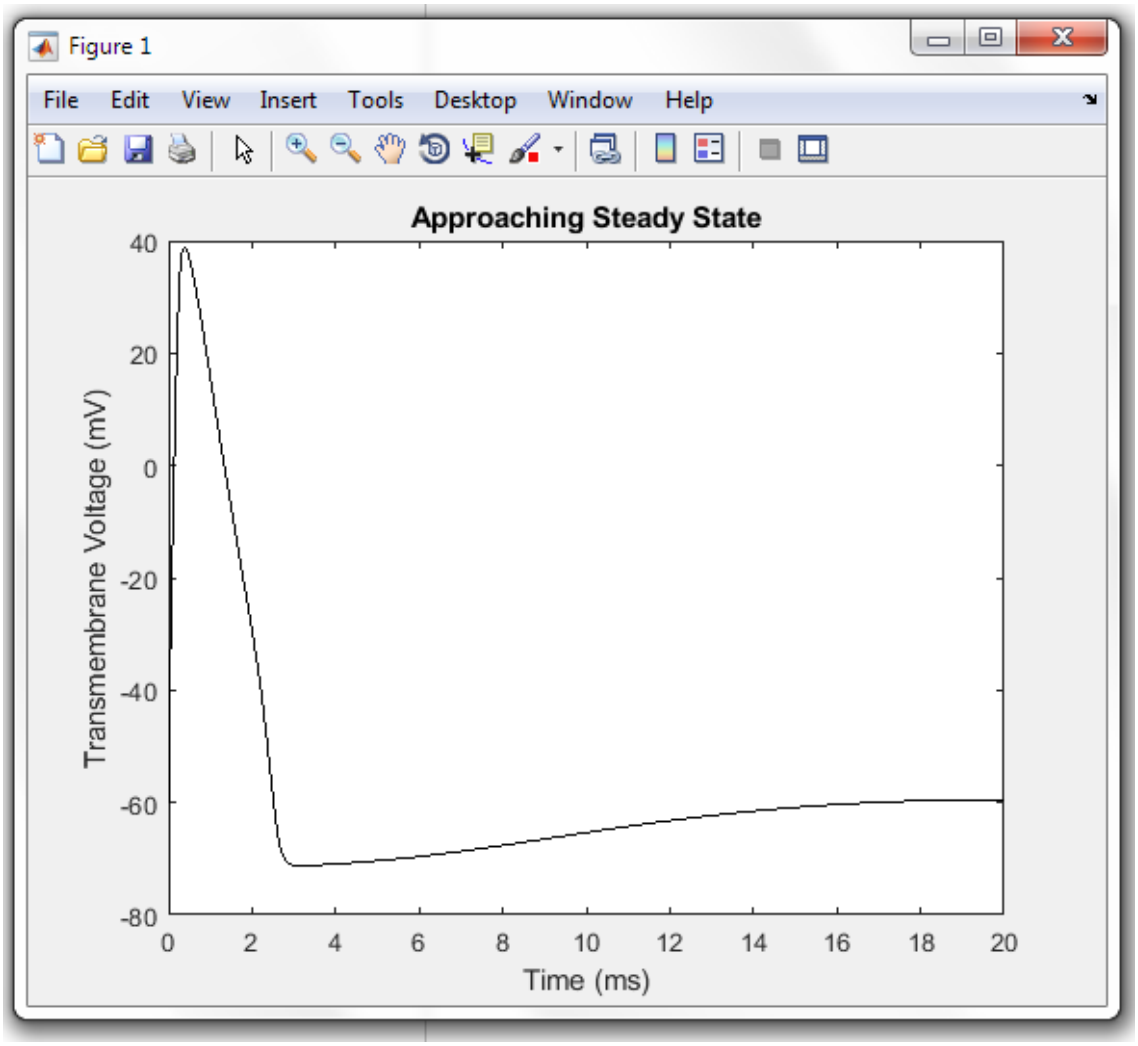


Figure 1
File Edit View Insert Tools Desktop Window Help
Approaching Steady State
Transmembrane Voltage (mV)
Time (ms)
40
20
0
-20
-40
-60
-80
0 2 4 6 8 10 12 14 16 18 20

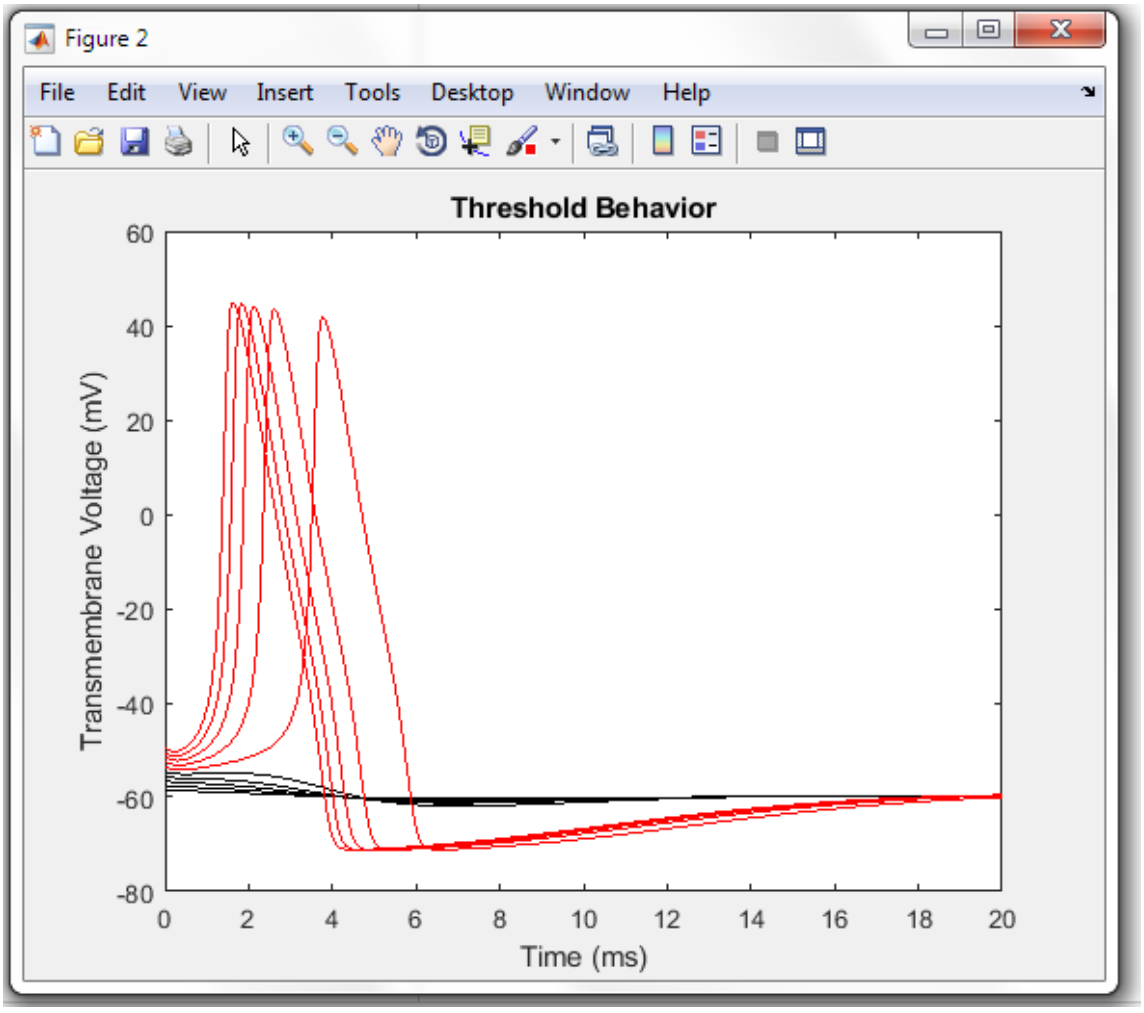


Figure 2
File Edit View Insert Tools Desktop Window Help
Threshold Behavior
Transmembrane Voltage (mV)
Time (ms)
60
40
20
0
-20
-40
-60
-80
0 2 4 6 8 10 12 14 16 18 20

Optional Problem

1. **Linear regression, in multiple ways.** Linear regression is a widely-used class of statistical models that attempts to fit a relationship between a scalar dependent variable y and one or more independent variables x. Suppose you run an experiment with p independent variables, with values $\boldsymbol{x} = [x_1, \dots, x_p]$, and as a result you measure a real number y. Repeating this n times, you can assemble the following matrices:

$$\boldsymbol{X} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,p} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,p} \end{bmatrix} \; ; \; \boldsymbol{Y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

The goal of linear regression is the fit each y_m to its corresponding $\boldsymbol{x_m} = [x_{m,1}, \dots, x_{m,p}]$, by finding the appropriate $\boldsymbol{\theta} = [\theta_1, \dots, \theta_p]^{\mathrm{T}}$ and b that fits the following set of n equations:

$$y_1 = \theta_1 x_{1,1} + \theta_2 x_{1,2} + \cdots + \theta_p x_{1,p} + b = \boldsymbol{x_1} * \boldsymbol{\theta} + b$$
$$\vdots$$
$$y_n = \theta_1 x_{n,1} + \theta_2 x_{n,2} + \cdots + \theta_p x_{n,p} + b = \boldsymbol{x_n} * \boldsymbol{\theta} + b$$

In other words, we would like to model the relationship as follows:

$$y = \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_p x_p + b = \boldsymbol{x} * \boldsymbol{\theta} + b$$

You might be familiar with the case where $p = 1$: finding the line of best fit. In particular, least-squares is the most well-known approach. In this exercise we will consider multiple ways to perform least squares, each with its own merits, along with some natural generalizations.

   a. Download the file `regression.mat`
   b. from the class website and load it into your workspace. This will provide you with data matrices `X` (data) and `Y` (labels,) where in this case they are actually column vectors ($n = 100, p = 1$).
   c. The first thing you should do when you have data is to visualize it! Fortunately, the given matrices `X` and `Y` are 1-D, so a 2-D **scatter** plot suffices. Plot the data as black dots.
   d. Examining the plot, the data mostly indicates a linear relationship, with some regional outliers. Let's try fitting a line using least-squares regression. We want to find $\boldsymbol{\theta}$ and b that fits the data well by minimizing the sum-of-squared-errors (hence "least squares"):

$$J(\boldsymbol{\theta}, b) = \sum_{m=1}^{n} (y_m - (\boldsymbol{x_m} * \boldsymbol{\theta} + b))^2$$

      Intuitively, if each $(\boldsymbol{x_m} * \boldsymbol{\theta} + b)$ matches closely to y_m, then we have found a good fit. We will minimize this cost function $J(\boldsymbol{\theta}, b)$ by numerical optimization, using **fminsearch**. Before we so, it will be more convenient to concatenate $\boldsymbol{\theta}$ and b into a single parameter vector $\boldsymbol{\beta} = [\boldsymbol{\theta}; b]$. Notice that $(\boldsymbol{x_m} * \boldsymbol{\theta} + b) = [\boldsymbol{x_m}, 1] * \boldsymbol{\beta}$.
      Create an augmented data matrix `XAug` by concatenating a column of ones to `X`.
   e. Recall how **fminsearch** is used: We provide it with a function that evaluates, for any $\boldsymbol{\beta}$, the objective function being optimized ($J(\boldsymbol{\beta})$ in this case). Write (in a separate file) a function `cost = squaredCost(beta, dataAug, labels)` that computes $J(\boldsymbol{\beta})$ using the given augmented data points in `dataAug` and labels in `labels`.

      Note: Do not use `XAug` and `Y`! The function cannot access variables in the workspace; they will be passed in through the arguments of `squaredCost`.
      Optional: Can you do this without using a for-loop? Optimization will be much faster.
   f. Note that our function has three input arguments, however we only wish to minimize the cost function with respect to `beta`, while keeping `dataAug` and `labels` as parameters which are constant in our minimization. In order to use **fminsearch** we will need to define a new, anonymous function which only has one parameter (`beta`). This function will call our previously declared `squaredCost` function and pass on our workspace variables `XAug` and `Y` as parameters. Complete the following command to declare the new anonymous function `squaredCostReduced = @(beta)…`

g. Now call **fminsearch** by using `squaredCostReduced` and an initial value. See **help** to find out what the first two arguments should be (they're the same as introduced in lecture). Use zeros for the initial value, noting that it needs have the same size as $\boldsymbol{\beta}$. Display `beta`, and compute the cost $J(\boldsymbol{\beta})$ using `squaredCost` (or `squaredCostReduced`) and `beta`. Verify by plotting the fitted line (with coefficients in `beta`) with the data points.

h. Since there is only a single independent variable, we can try **polyfit** on the data as well. Call **polyfit** on `X` and `Y` to fit a straight line, and verify that the coefficients returned match those from **fminsearch** (since **polyfit** does least-squares regression too).

i. It turns out that there is an analytical solution for least-squares regression, known as the normal equations (which you can find by differentiating $J(\boldsymbol{\beta})$ and setting equal to zero):

$$\boldsymbol{\beta} = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y}$$

Verify that this gives the same set of coefficients (use `XAug`).

j. In fact, because we have an over-determined system of linear equations, we can solve for $\boldsymbol{\beta}$ in the same way you did in Q1. Verify your answer.

k. At this point, you may think that we've wasted our time trying to re-invent the wheel. Well, not quite! First, notice that **polyfit** cannot handle more than one independent variable. Second, the analytical solutions only work for the beautiful quadratic cost function we've seen above, the sum-of-squares error. This cost function is motivated by a simple probabilistic modeling assumption, that the sampled outputs `Y` follow the linear relationship with added normally-distributed (Gaussian) noise.[1] However, this noise distribution is unrealistic in many cases, and modifications are needed.

Looking at the plots you've made, you will notice that the fitted line does not fit the majority of data well because of a few outliers. It turns out that, instead of using the sum-of-squares error, using absolute-deviation error significantly improves robustness against outliers (Optional: Why is that? What does the cost function mean?):

$$J_{abs}(\boldsymbol{\theta}, b) = \sum_{m=1}^{n} abs(\, y_m - (\boldsymbol{x_m} * \boldsymbol{\theta} + b)\,)$$

Repeats parts (d)-(f), this time writing a different function `cost = absoluteCost(beta, dataAug, labels)`. Note that it's not valid to compare the values from the two different cost functions. Plot the data, the least-squares line (in blue), and the least-absolute-deviations line (in red) together.

There's no analytical solution for this cost function![2]

l. Optional: Investigate what happens when you use other p-norms in the cost function:

$$J_{abs}(\boldsymbol{\theta}, b) = \sum_{m=1}^{n} [\, abs(\, y_m - (\boldsymbol{x_m} * \boldsymbol{\theta} + b)\,)\,]^p$$

The case of $p = 1$ corresponds to least-absolute-deviations, $p = 2$ to least-squares.

Try modifying your loss function to take in p as well so that you only need to write one!

[1] Incidentally, Gauss, the "Prince of Mathematics", is widely credited with developing and advancing the theory behind the least-squares method in the 18th century. Along with many other achievements: http://www.gaussfacts.com

OP_task1.m | squaredCost.m | absoluteCost.m

```
% Пороскун Олени ПМ-81
% Homework 3          Optional Problem
% 1. Linear regression, in multiple ways.

load('regression.mat');
figure;
plot(X, Y, '.k');
hold on

XAug = ones(size(X,1), 1);
XAug(:,2) = X;
squaredCostReduced = @(beta)squaredCost(beta, XAug(:,2), Y);
beta_sqr = fminsearch(@(beta) squaredCostReduced(beta), [0;0])

f1 = polyval(beta_sqr,X);
plot(X, f1, 'b');

p = polyfit(X,Y,1)  % коефіцієнти для порівняння с функцією polyfit
%f = polyval(p,X);
%plot(X, f, 'b')

beta_analit = (((XAug')* XAug)^(-1))*(XAug')*Y

squaredCostReduced = @(beta)absoluteCost(beta, XAug(:,2), Y);
beta = zeros(2,1);
beta_abs = fminsearch(@(beta) squaredCostReduced(beta), [0;0])
f2 = polyval(beta_abs,X);
plot(X, f2, 'r');

title('Task 1 OP HW3');
xlabel('data (X)');
ylabel('labels (Y)');
legend('data','squaredCost','absoluteCost','Location','northwest');

```

OP_task1.m | squaredCost.m | absoluteCost.m

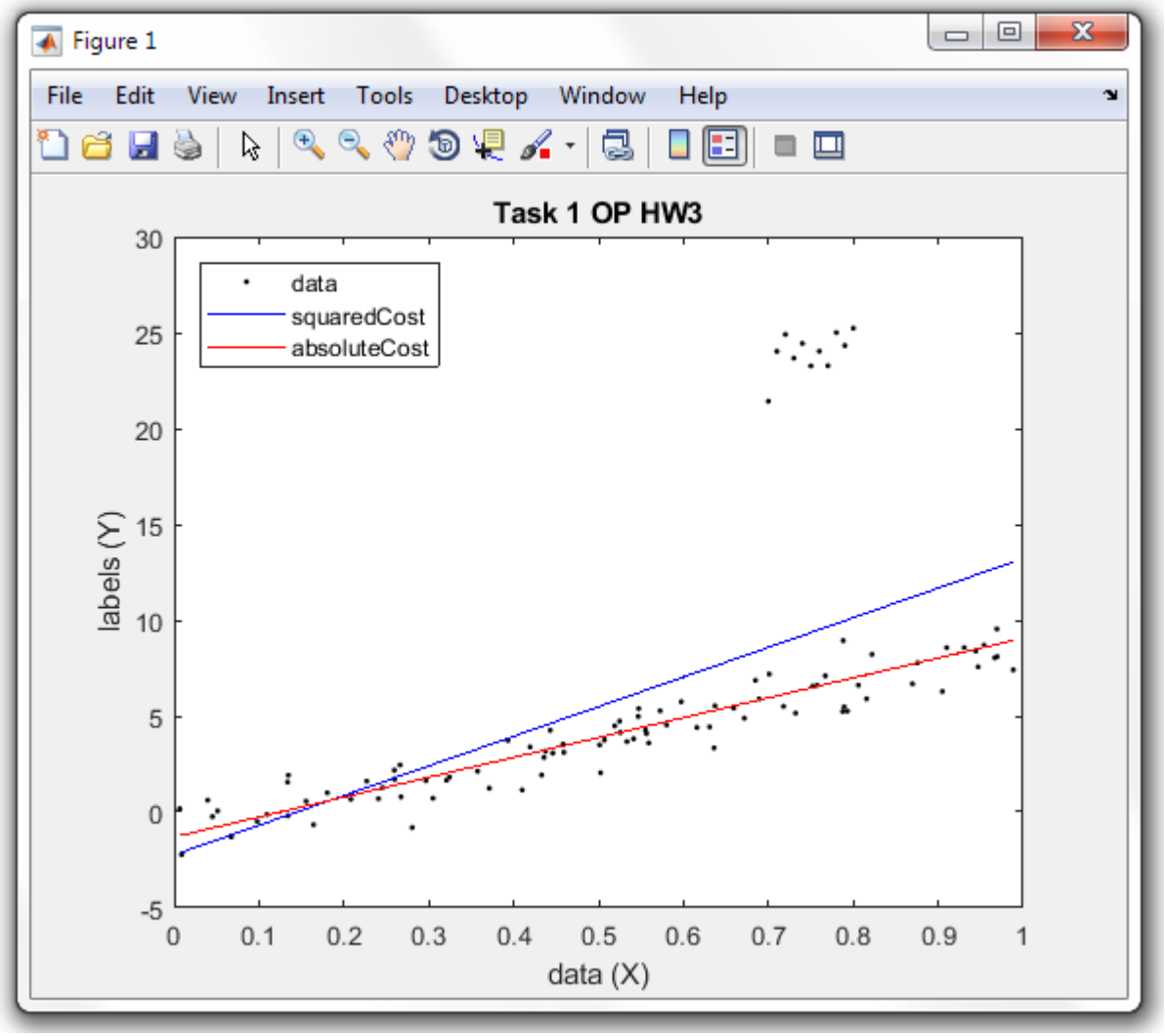
```
% Пороскун Олени ПМ-81
% Homework 3            Optional Problem

% 1. Linear regression, in multiple ways.

% X (dataAug),  Y (labels)
% n = 100, p = 1

function cost = squaredCost(beta, dataAug, labels)
cost = 0;
for m = 1:100
    cost = cost + (labels(m)-[dataAug(m),1]*beta)^2;
end
end
```

OP_task1.m | squaredCost.m | absoluteCost.m

```
% Пороскун Олени ПМ-81
% Homework 3            Optional Problem

% 1. Linear regression, in multiple ways.

% X (dataAug)  Y (labels)
% n = 100, p = 1

function cost = absoluteCost(beta, dataAug, labels)
cost = 0;
for m = 1:100
    cost = cost + abs(labels(m)-[dataAug(m),1]*beta);
end
end
```

Command Window

```
>> OP_task1

beta_sqr =

   15.4616
   -2.2396


p =

   15.4616   -2.2396


beta_analit =

   -2.2396
   15.4616


beta_abs =

   10.3676
   -1.3021

fx >>
```

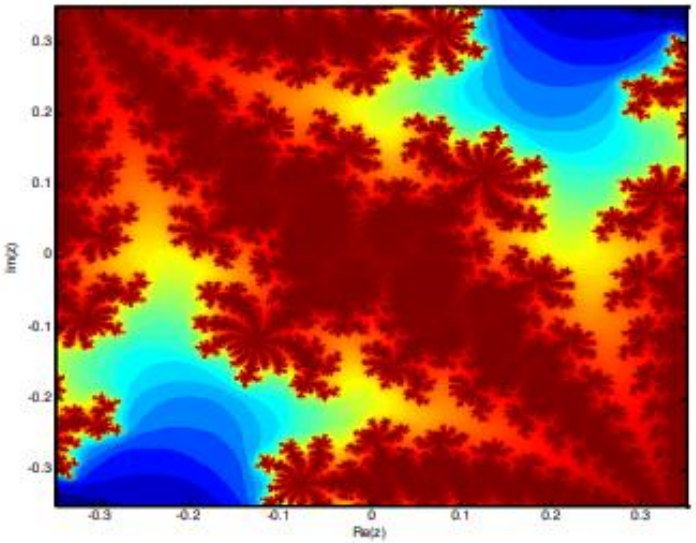


2. **Optional, but highly recommended: Julia Sets.** In this problem you will generate quadratic Julia Sets. The following description is adapted from Wikipedia at http://en.wikipedia.org/wiki/Julia_Sets. For more information about Julia Sets please read the entire article there.

Given two complex numbers, c and z_0, we define the following recursion:

$$z_n = z_{n-1}^2 + c$$

This is a dynamical system known as a quadratic map. Given a specific choice of c and z_0, the above recursion leads to a sequence of complex numbers $z_1, z_2, z_3 \ldots$ called the orbit of z_0. Depending on the exact choice of c and z_0, a large range of orbit patterns are possible. For a given fixed c, most choices of z_0 yield orbits that tend towards infinity. (That is, the modulus $|z_n|$ grows without limit as n increases.) For some values of c certain choices of z_0 yield orbits that eventually go into a periodic loop. Finally, some starting values yield orbits that appear to dance around the complex plane, apparently at random. (This is an example of chaos.) These starting values, z_0, make up the Julia set of the map, denoted J_c. In this problem, you will write a MATLAB script that visualizes a slightly different set, called the filled-in Julia set (or Prisoner Set), denoted K_c, which is the set of all z_0 with orbits which do not tend towards infinity. The "normal" Julia set J_c is the edge of the filled-in Julia set. The figure below illustrates a Julia Set for one particular value of c. You will write MATLAB code that can generate such fractals in this problem.



a. It has been shown that if the modulus of z_n becomes larger than 2 for some n then it is guaranteed that the orbit will tend to infinity. The value of n for which this becomes true is called the 'escape velocity' of a particular z_0. Write a function that returns the escape velocity of a given z_0 and c. The function declaration should be: `n=escapeVelocity(z0,c,N)` where `N` is the maximum allowed escape velocity (basically, if the modulus of z_n does not exceed 2 for `n<N`, return `N` as the escape velocity. This will prevent infinite loops). Use **abs** to calculate the modulus of a complex number

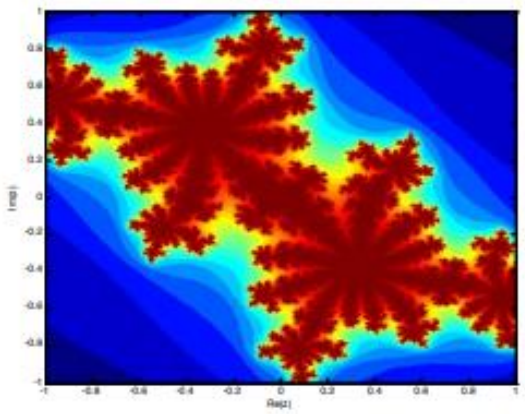
b. To generate the filled in Julia Set, write the following function `M=julia(zMax,c,N)`. `zMax` will be the maximum of the imaginary and complex parts of the various z_0's for which we will compute escape velocities. `c` and `N` are the same as defined above, and M is the matrix that contains the escape velocity of various z_0's.

   i. In this function, you first want to make a 500x500 matrix that contains complex numbers with real part between `-zMax` and `zMax`, and imaginary part between `-zMax` and `zMax`. Call this matrix `Z`. Make the imaginary part vary along the y axis of this matrix. You can most easily do this by using **linspace** and **meshgrid**, but you can also do it with a loop.

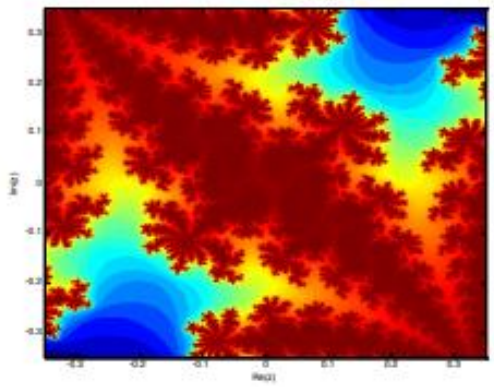
   ii. For each element of `Z`, compute the escape velocity (by calling your `escapeVelocity`) and store it in the same location in a matrix `M`. When done, the matrix `M` should be the same size as `Z` and contain escape velocities with values between 1 and `N`.

   iii. Run your `julia` function with various `zMax`, `c`, and `N` values to generate various fractals. To display the fractal nicely, use **imagesc** to visualize `atan(0.1*M)`, (taking the arctangent of `M` makes the image look nicer; you may also want to use **axis xy** so the y values aren't flipped). WARNING: this function may take a while to run.

The figure below was created by running:
`M=julia(1,-.297491+i*0.641051,100);` and visualizing it as described above.



The figure below was generated by running the same `c` parameter as above, but on a smaller range of `z` values and with a larger `N`:
`M=julia(.35,-.297491+i*0.641051,250);`



OP_task2_final.m

```
% Пороскун Олени ПМ-81
% Homework 3           Optional Problem
% 2. Optional, but highly recommended: Julia Sets.

M1 = julia(1, -.297491+i*0.641051, 100);
figure(1);
imagesc(atan(0.1*M1));
xlabel('Re(z)');
ylabel('Im(z)');
axis xy;
colormap(jet);

M2 = julia(.35, -.297491+i*0.641051, 250);
figure(2);
imagesc(atan(0.1*M2));
xlabel('Re(z)');
ylabel('Im(z)');
axis xy;
colormap(jet);

function n = escapeVelocity(z0, c, N)
    clear i;  % очищуємо i для того щоб вона не використовувалась як змінна
    z = z0;
    n = 0;
    while (((abs(z)<2) && (n ~= N)))
        z_last = z;       % попередній номер для швидкості
        z = z^2 + c;      % динамічна рекурсія
        n = n + 1;
        if (z == z_last)
            n = N;
        end
    end
end

function M = julia(zMax, c, N)
    clear i;
    if nargin
    X = linspace(-zMax,zMax,500);    % дійсна частина
    Y = 1i*X;                        % уявна частина, і та 1i одне і теж
    [Re,Im] = meshgrid(X,Y);
    Z = Re + Im;


    M = zeros(size(Z));
    for x = 1:size(Z,1)
        for y = 1:size(Z,2)
            M(x,y) = escapeVelocity(Z(x,y), c, N);
        end
    end
    end
end

```

Command Window

```
>> OP_task2_final
>>
```

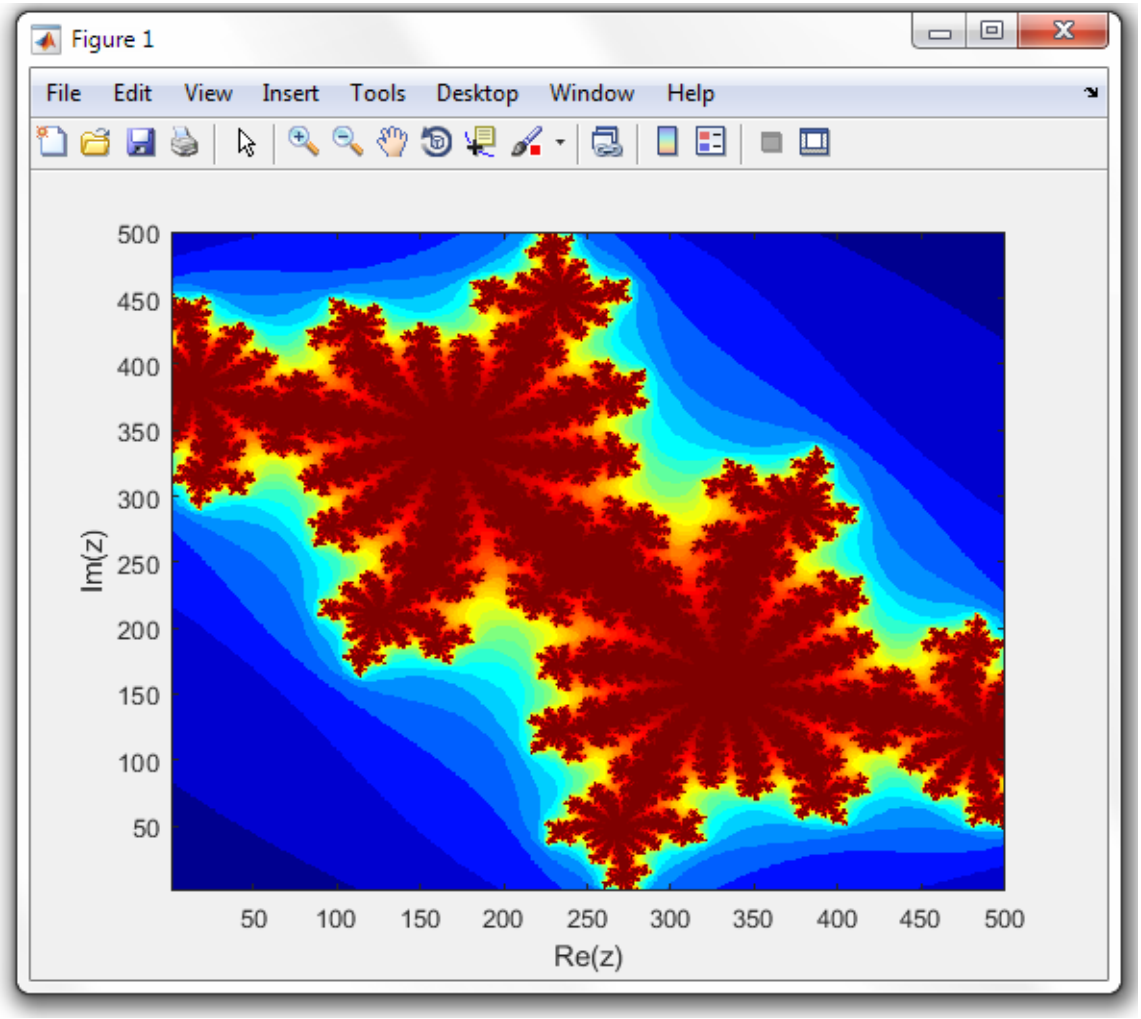


Figure 1
File Edit View Insert Tools Desktop Window Help
500
450
400
350
300
250
200
150
100
50
Im(z)
50 100 150 200 250 300 350 400 450 500
Re(z)

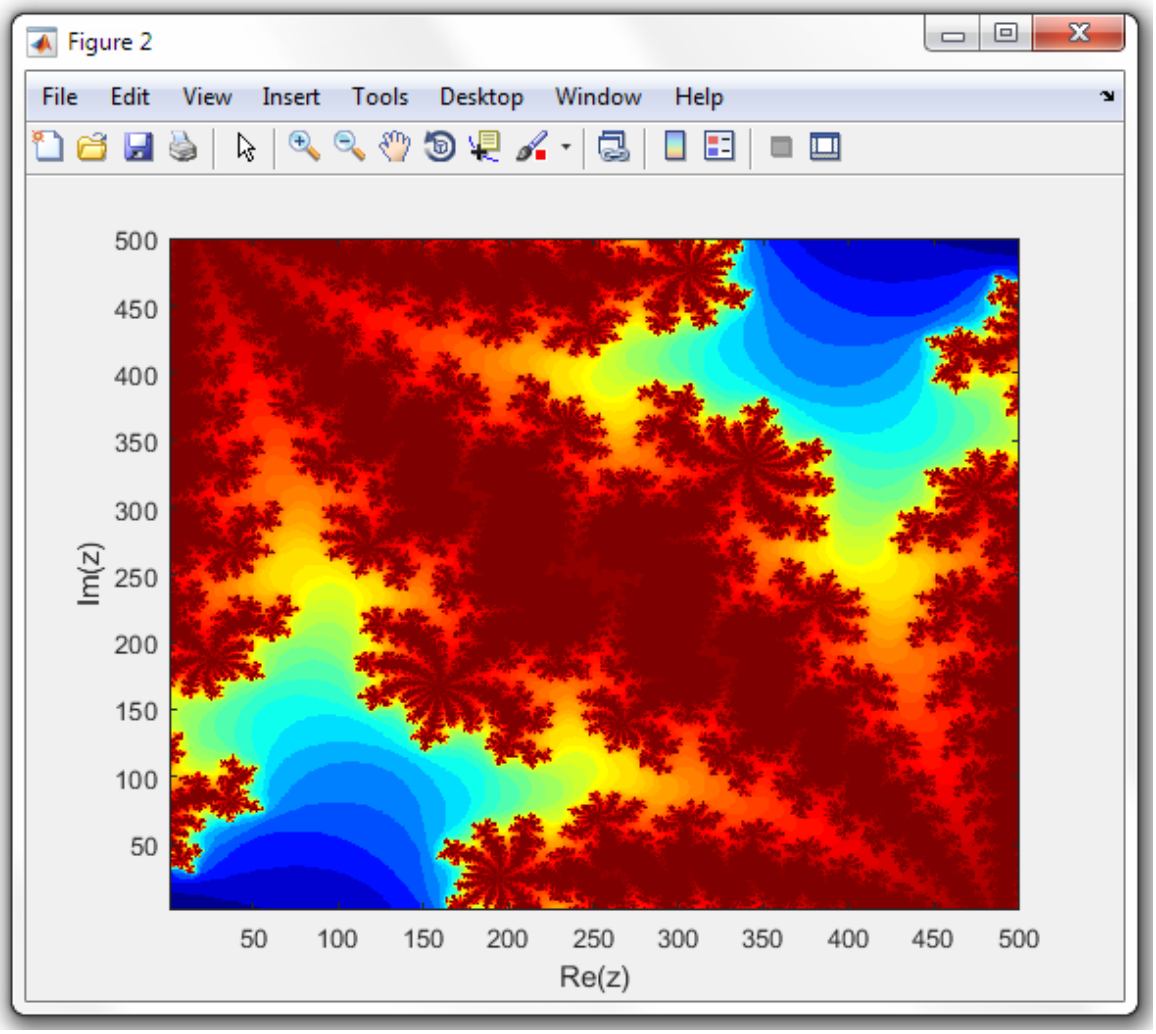


Figure 2
File Edit View Insert Tools Desktop Window Help
500
450
400
350
300
250
200
150
100
50
Im(z)
50 100 150 200 250 300 350 400 450 500
Re(z)

3. **Optional: Solve a Sudoku game.** Sudoku is a popular number placement puzzle game. The objective of the game is to fill a 9x9 grid with 9 sets of the numbers from 1 to 9, such that each number (from 1 to 9) occurs only once every row, column and 3x3 sub-block. Below is an example of a Sudoku board before and after solving.

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

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

Figure 1 Unsolved (left) and solved (right) Sudoku boards

In this exercise we will write a simple Matlab function that receives a 9x9 matrix input representing an incomplete Sudoku game and returns the correctly solved board. There are several approaches for solving such a problem; we will implement a simple backtracking algorithm.

The concept of backtracking involves guessing a certain solution which does not violate the game rules and proceeding until we reach a point where we cannot satisfy the rules. At this point, we reverse our steps (backtrack) to the last known working solution and make a different choice among the available possibilities until the entire game is solved.

a. Download the file `sudoku.mat` from the class website and load it into your workspace. You will now have an example for an unsolved Sudoku board in your workspace named `unsolvedBoard`.

b. Write a function with the declaration `safe = checkSudoku(board, row, col, num)`

`board` is a 9x9 matrix which contains the Sudoku board to check. Unfilled board cells are indicated by a NaN value.

`row` and `col` are indexes to the check if the number `num` (between 1 and 9) can be inserted without causing a violation. i.e. does `num` already exist in the given row, column or 3x3 sub-block. Note that we do not need to check the entire board's validity (although you could try to do so as practice) since we will be filling our board one element at a time.

The function returns a logical `true` or `false` value in the output `safe` depending on the outcome of the check.

There are several ways to accomplish this functionality. Possible functions you may want to use are **sort**, **unique**, **ismember, any, all**. Note that NaN values are unequal to one another!

Test your function with various inputs to see it is operating as expected!

For example:

```
>> checkSudoku(unsolvedBoard, 3, 1, 1)
ans =
     1
```

```
>> checkSudoku(unsolvedBoard, 3, 1, 5)
ans =
     0
```

c. Write a function with the declaration `solvedBoard = solveSudoku(board)`. The input to this function will be an incomplete Sudoku board and the output will be the solved board.

Below are hints and suggestions for steps to help you implement this function:

i. Find the indices of all the missing values in the table and store them in the variable `emptyInd` (**find**, **isnan**)

ii. Create a variable called `ind` which will indicate our current location within the `emptyInd` array

iii. Write a loop that will traverse the `emptyInd` array. At each location check what number can be placed in that location using your function from part (b) (Which numbers do you need to check?). If it's not valid, try a different number, if none work, replace it with a NaN, and go back to the previous empty index and try a different number than before. (Which loop type would be most suitable for this scenario?) (**ind2sub**, **checkSudoku**)

d. Test your function with the given example `unsolvedBoard` and compare the result to the image shown above.

e. **Optional**: Write a function `displaySudoku(board)` for displaying the Sudoku game board in a more "human friendly" form. See an example for such a display below.

```
>> load sudoku
>> displaySudoku(unsolvedBoard)
+=============================+
| 5  3     |     7     |           |
| 6        | 1   9   5 |           |
|     9  8 |           |     6     |
|=============================|
| 8        |     6     |        3  |
| 4        | 8       3 |        1  |
| 7        |     2     |        6  |
|=============================|
|     6    |           | 2   8     |
|          | 4   1   9 |        5  |
|          |     8     |     7   9 |
+=============================+

>> solvedBoard = solveSudoku(unsolvedBoard);
>> displaySudoku(solvedBoard)
+=============================+
| 5  3  4  | 6  7  8  | 9  1  2  |
| 6  7  2  | 1  9  5  | 3  4  8  |
| 1  9  8  | 3  4  2  | 5  6  7  |
|=============================|
| 8  5  9  | 7  6  1  | 4  2  3  |
| 4  2  6  | 8  5  3  | 7  9  1  |
| 7  1  3  | 9  2  4  | 8  5  6  |
|=============================|
| 9  6  1  | 5  3  7  | 2  8  4  |
| 2  8  7  | 4  1  9  | 6  3  5  |
| 3  4  5  | 2  8  6  | 1  7  9  |
+=============================+
```

Task_3_OP_HW3.m

```
% Пороскун Олени ПМ-81
% Homework 3           Optional Problem
% 3. Optional: Solve a Sudoku game.

load('sudoku.mat');
unsolvedBoard

s1 = checkSudoku(unsolvedBoard, 3, 1, 1)  % ans = 1
s0 = checkSudoku(unsolvedBoard, 3, 1, 5)  % ans = 0
s2 = checkSudoku(unsolvedBoard, 3, 5, 9)  % ans = 0
s3 = checkSudoku(unsolvedBoard, 4, 6, 1)  % ans = 1

solvedBoard = solveSudoku(unsolvedBoard)

function solvedBoard = solveSudoku(board)
    emptyInd = find(isnan(board));               % порожні індекси в board
    solvedBoard = solverec(board, emptyInd, 1);  % для рекурсії
end

function safe = checkSudoku(board, row, col, num)
    subrow = board(row, :);   % відділяємо рядок з номером  row
    subcol = board(:, col);   % відділяємо стовпець з номером  col

    subSquareRow = (1:3) + 3*(ceil(row/3)-1); % вибераємо тільки рядки(та стовпці), які потрапили до квадратика 3x3
    subSquareCol = (1:3) + 3*(ceil(col/3)-1); % в якому знаходиться число num з координатами (row,col)

    subBoard = board(subSquareRow, subSquareCol);  % виділяємо квадратик 3x3, де є число num
    subBoard = subBoard(:);                        % перетворюємо у вектор-стовпець для зручності порівняння

    if any(subrow == num) || any(subcol == num) || any(any(subBoard == num))  % any(a) = 1 якщо хоча б один елемент
        safe = false;                                                         % з а ненульовий
    else
        safe = true;
    end
end

function candidates = getCandidates(board, row, col)
    subrow = board(row, :);
    subcol = board(:, col);
```

Task_3_OP_HW3.m

```
37      function candidates = getCandidates(board, row, col)
38          subrow = board(row, :);
39          subcol = board(:, col);
40
41          subSquareRow = (1:3) + 3*(ceil(row/3)-1);
42          subSquareCol = (1:3) + 3*(ceil(col/3)-1);
43
44          subBoard = board(subSquareRow, subSquareCol);
45          subBoard = subBoard(:);
46
47          % Отримати різницю кожного масиву в порівнянні з опорною лінією
48          refval = 1:9;
49          cdrow = setdiff(refval,subrow);                 % setdiff (a, b) - повертає різницю множин,
50          cdcol = setdiff(refval,subcol);                 % тобто, ті елементи вектора a,
51          cdsqr = setdiff(refval,subBoard);               % які не містяться у векторі b.
52
53                                                          % intersect(a,b)- повертає перетин множин
54          candidates = intersect(intersect(cdrow,cdcol),cdsqr);   % для двох векторів a і b,
55                                                          % тобто, загальні елементи векторів a і b.
56      end
57
58      function [res, solved, noSolutionFound] = solverec(board,emptyInd,ind,solved)
59
60      % перший випадок виклику фукнції для повернення
61      if (nargin < 4)                                     % nargin - повертає число вхідних параметрів
62          solved  = false;                                % даної функції.
63      end
64
65      % другий випадок
66      noSolutionFound = false;
67
68      % перевіряємо чи закінчили повністю з  EmptyInd
69      if (ind > numel(emptyInd))                          % numel(f)  -  повертає число елементів
70          solved = true;                                  % масиву f
71      end
72
73      if (solved)
74          res = board;
75          return;
76      end
```

Task_3_OP_HW3.m

```
    end

    % Якщо не задовольнили попередні умови , то потрібно знайти новий emptyInd
    % Підготуємо наступні індекси (рядок, стовпець і лінійний індекс)
    num = emptyInd(ind);
    col = ceil(num/9);
    row = num - ((col-1)*9);

    cd  = getCandidates(board, row, col);   % можливі варінти чисел
    ncd = numel(cd);                        % число варінатів

    if (ncd == 0)
        % немає варіанта для цієї комірки, то повертаємося
        noSolutionFound = true;
    else
        for k = 1:ncd                % перекираємо можливі варінти по одному
            board(num) = cd(k);      % пробуємо одний варінт
            % переходимо до настпної пустої комірки
            [res, solved, noSolutionFound] = solverec(board,emptyInd,ind+1,solved);

            % виходимо якщо є рішення
            if (solved)
                return;
            end

            % в іншому випадку, скинути цей emptyInd, перш ніж переходити до
            % наступного варіанта
            if (noSolutionFound)
                board(num) = NaN;
            end
        end
    end

    if (noSolutionFound)
        % Ми перебрали всі можливі варіанти для emptyInd
        % Тепер відступаємо ще далі
        board(num) = NaN;
        res = board;
        return  % це насправді необов'язково, функція все одно
                % "повернеться" в кінці блоку if
    end
    end
```

Command Window

```
>> Task_3_OP_HW3

unsolvedBoard =

     5     3   NaN   NaN     7   NaN   NaN   NaN   NaN
     6   NaN   NaN     1     9     5   NaN   NaN   NaN
   NaN     9     8   NaN   NaN   NaN   NaN     6   NaN
     8   NaN   NaN   NaN     6   NaN   NaN   NaN     3
     4   NaN   NaN     8   NaN     3   NaN   NaN     1
     7   NaN   NaN   NaN     2   NaN   NaN   NaN     6
   NaN     6   NaN   NaN   NaN   NaN     2     8   NaN
   NaN   NaN   NaN     4     1     9   NaN   NaN     5
   NaN   NaN   NaN   NaN     8   NaN   NaN     7     9


s1 =

  logical

   1


s0 =

  logical

   0


s2 =

  logical

   0


s3 =

  logical

   1
```

Command Window

```
s1 =

  logical

   1


s0 =

  logical

   0


s2 =

  logical

   0


s3 =

  logical

   1


solvedBoard =

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

>>
```

Sumy State University

Department

of

Applied Mathematics and Complex Systems Modeling

Report on laboratory work № 4

Subject

Computing system

Student: Olena Poroskun

Teacher: Igor A. Knyaz

Sumy, Sumy region

2020

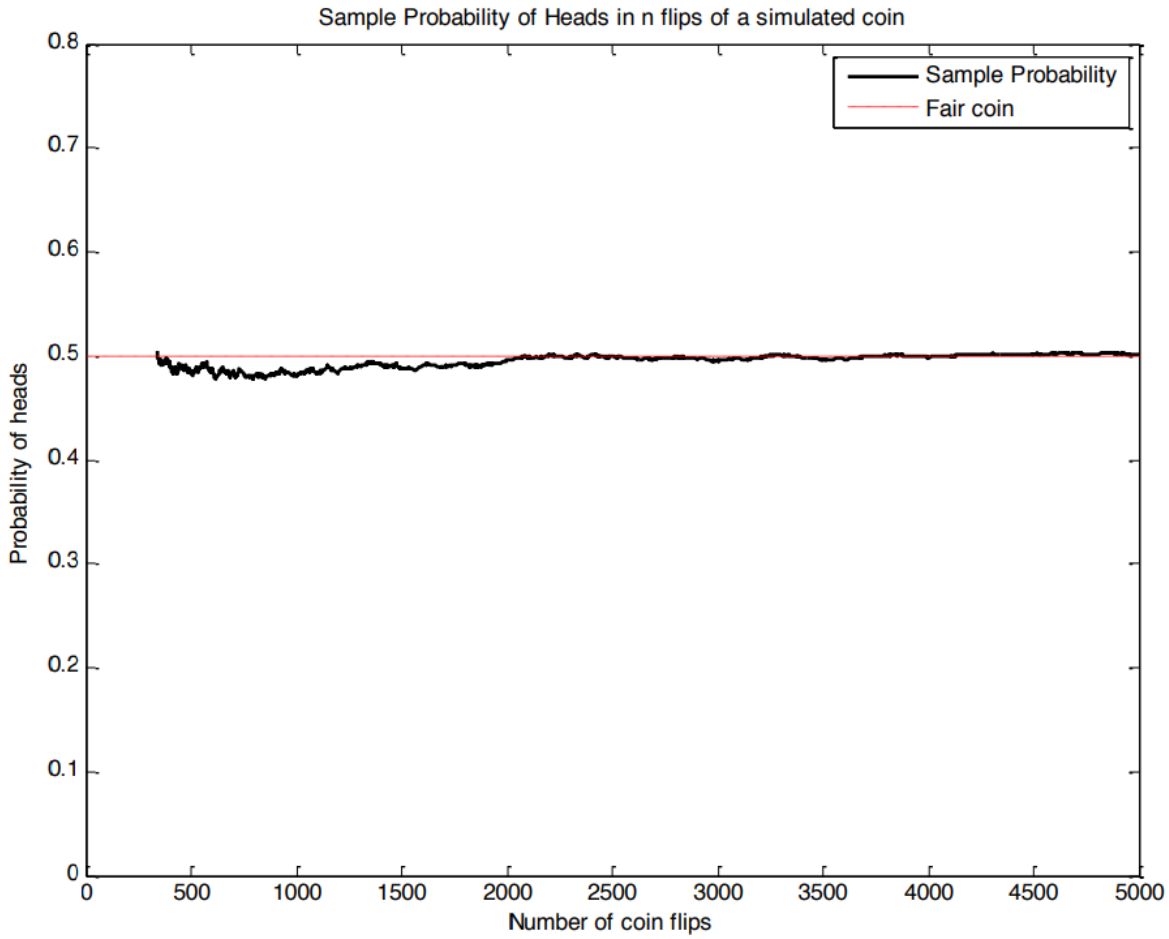
1. **Random variables.** Make a vector of 500 random numbers from a Normal distribution with mean 2 and standard deviation 5 (**randn**). After you generate the vector, verify that the sample mean and standard deviation of the vector are close to 2 and 5 respectively (**mean**, **std**).

```
1      % Пороскун Олени ПМ-81
2      % Homework 4
3      %
4      % 1.Random variables.
5
6 -    vec = 5*(randn(1,500)) + 2;
7 -    mean_vec = mean(vec)
8 -    standev_vec = std(vec)
```

Command Window

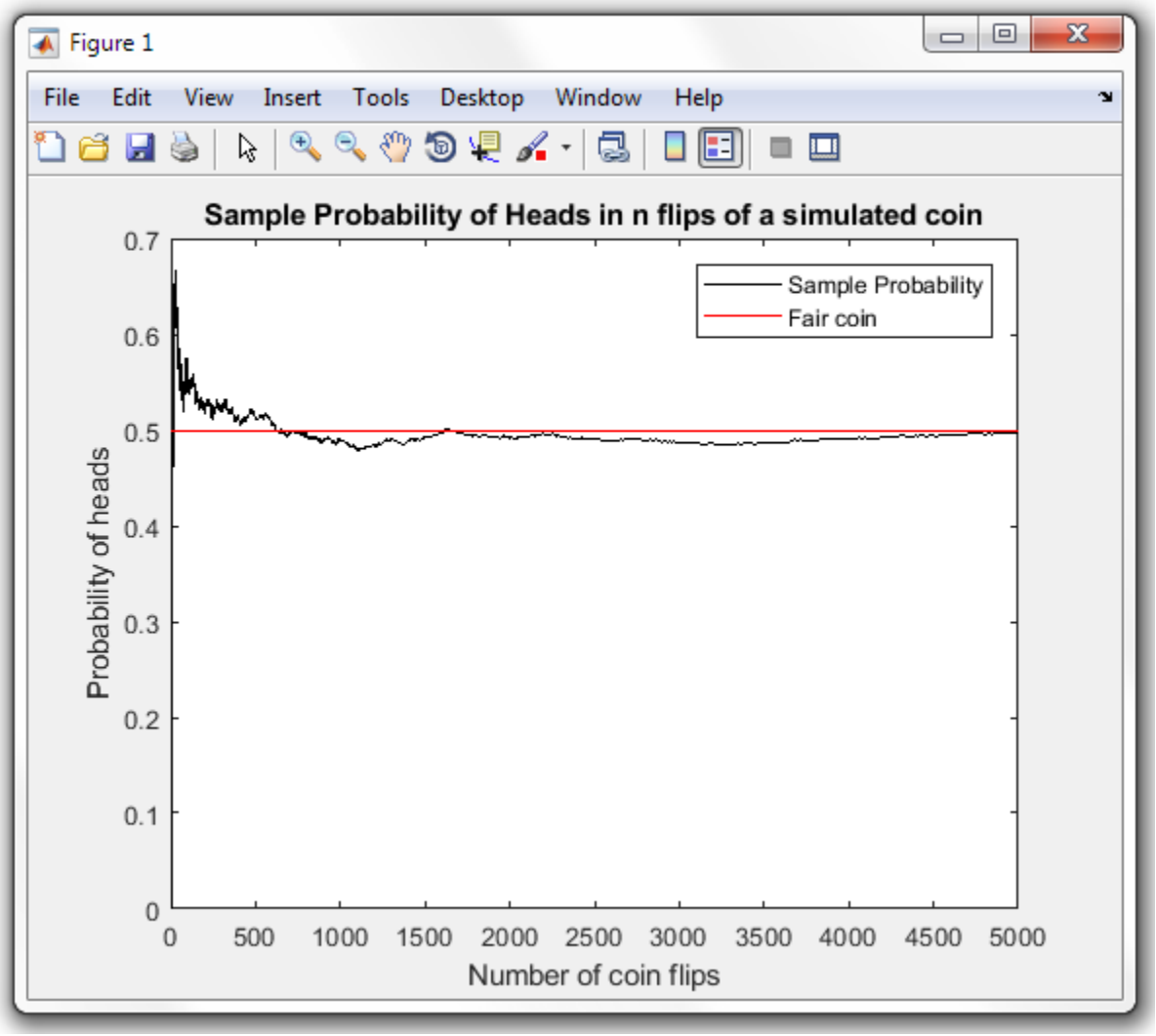
```
mean_vec =

    2.0410


standev_vec =

    4.8514
```

2. **Flipping a coin.** Write a script called `coinTest.m` to simulate sequentially flipping a coin 5000 times. Keep track of every time you get 'heads' and plot the running estimate of the probability of getting 'heads' with this coin. Plot this running estimate along with a horizontal line at the expected value of 0.5, as below. This is most easily done without a loop (useful functions: **rand**, **round**, **cumsum**).

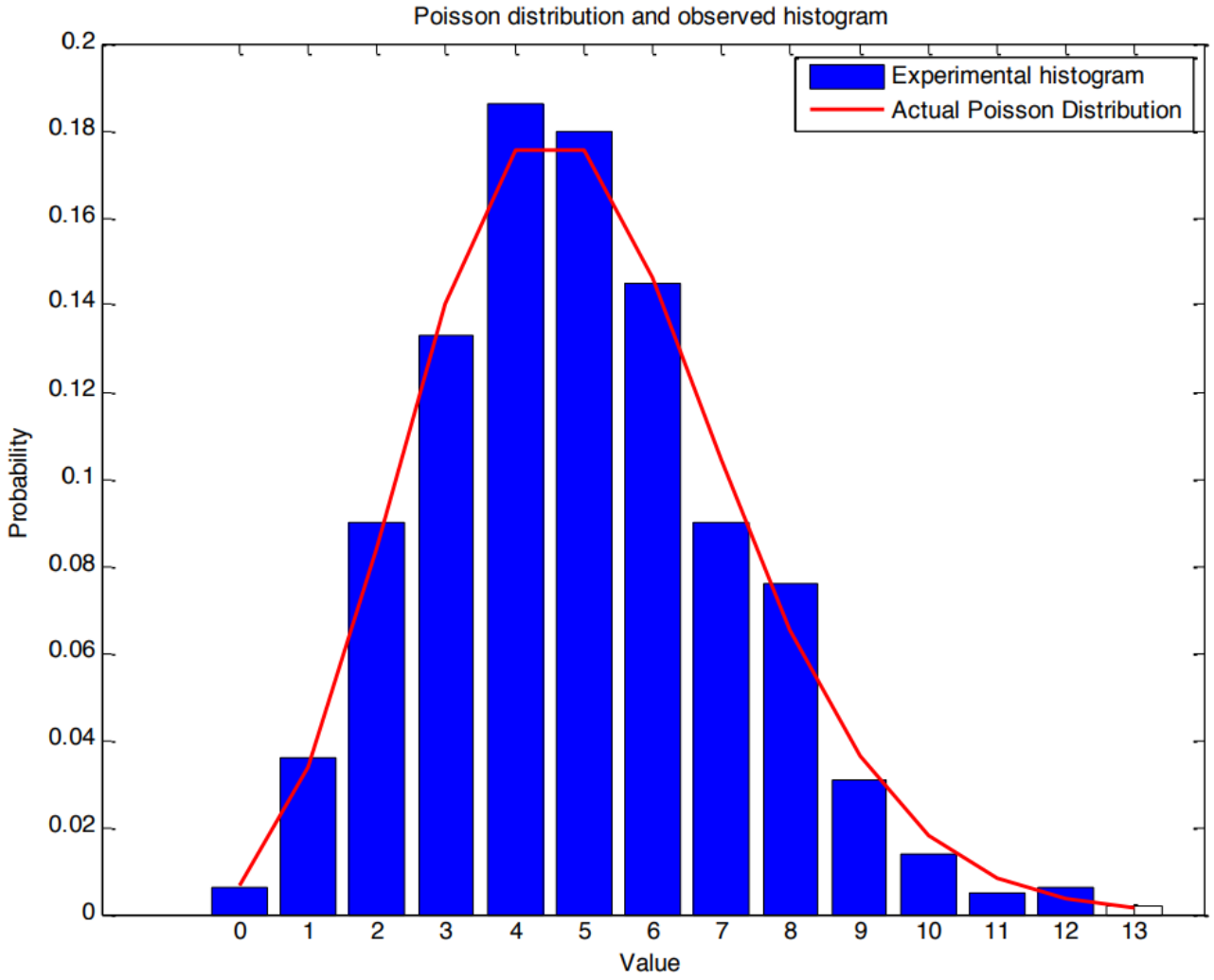


coinTest.m

```
% Пороскун Олени ПМ-81
% Homework 4
% 2. Flipping a coin.

figure;
mas = rand(1,5000);
mas_r = round(mas);
suma = cumsum(mas_r);
i = 1:5000;
p = suma./i;
x1 = [0, 5000];
y1 = [0.5, 0.5];
plot(i, p,'k', x1, y1, 'r')
xlabel('Number of coin flips');
ylabel('Probability of heads');
legend('Sample Probability', 'Fair coin');
title('Sample Probability of Heads in n flips of a simulated coin');
```

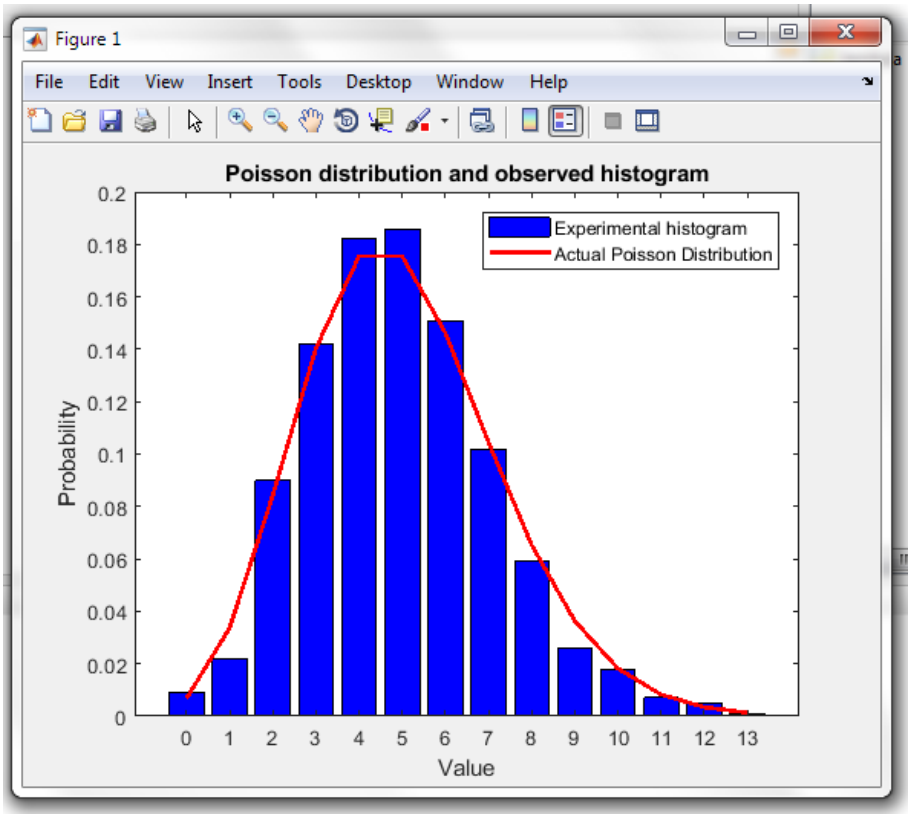


3. **Histogram.** Generate 1000 Poisson distributed random numbers with parameter $\lambda = 5$ (**poissrnd**)[1]. Get the histogram of the data and normalize the counts so that the histogram sums to 1 (**hist** – the version that returns 2 outputs N and X, **sum**). Plot the normalized histogram (which is now a probability mass function) as a bar graph (**bar**). Hold on and also plot the actual Poisson probability mass function with $\lambda = 5$ as a line (**poisspdf**). You can try doing this with more than 1000 samples from the Poisson distribution to get better agreement between the two. **Hint:** By default **hist** gives 10 equally-spaced bins; we want one bar for each non-negative integer.



Task_3.m

```
% Пороскун Олени ПМ-81
% Homework 4
% 3. Histogram

figure;
lambda = 5;
X = poissrnd(lambda, 1, 1000);
[X, N] = hist(X,14)
X = X./ sum(X)
bar([0:13], X,'b')
hold on
N1 = 0:13;
Y = poisspdf(N1,lambda)
plot(N1, Y, 'r-', 'LineWidth', 2);
xlabel('Value');
ylabel('Probability');
title('Poisson distribution and observed histogram');
legend('Experimental histogram', 'Actual Poisson Distribution');
```




```
>> Task_3
X =
     9    22    90   142   182   186   151   102    59    26    18     7     5     1
N =
    0.4643    1.3929    2.3214    3.2500    4.1786    5.1071    6.0357    6.9643    7.8929    8.8214    9.7500   10.6786   11.6071   12.5357
X =
    0.0090    0.0220    0.0900    0.1420    0.1820    0.1860    0.1510    0.1020    0.0590    0.0260    0.0180    0.0070    0.0050    0.0010
Y =
    0.0067    0.0337    0.0842    0.1404    0.1755    0.1755    0.1462    0.1044    0.0653    0.0363    0.0181    0.0082    0.0034    0.0013
fx >>
```

4. **Practice with cells.** Usually, cells are most useful for storing strings, because the length of each string can be unique.
    a. Make a 3x3 cell where the first column contains the names: 'Joe', 'Sarah', and 'Pat', the second column contains their last names: 'Smith', 'Brown', 'Jackson', and the third column contains their salaries: $30,000, $150,000, and $120,000. Display the cell using **disp**.
    b. Sarah gets married and decides to change her last name to 'Meyers'. Make this change in the cell you made in a. Display the cell using **disp**.
    c. Pat gets promoted and gets a raise of $50,000. Change his salary by adding this amount to his current salary. Display the cell using **disp**.

    The output to parts a-c should look like this:

```
>> cellProblem
    'Joe'       'Smith'       [ 30000]
    'Sarah'     'Brown'       [150000]
    'Pat'       'Jackson'     [120000]

    'Joe'       'Smith'       [ 30000]
    'Sarah'     'Meyers'      [150000]
    'Pat'       'Jackson'     [120000]

    'Joe'       'Smith'       [ 30000]
    'Sarah'     'Meyers'      [150000]
    'Pat'       'Jackson'     [170000]
```

sellProblem4.m*

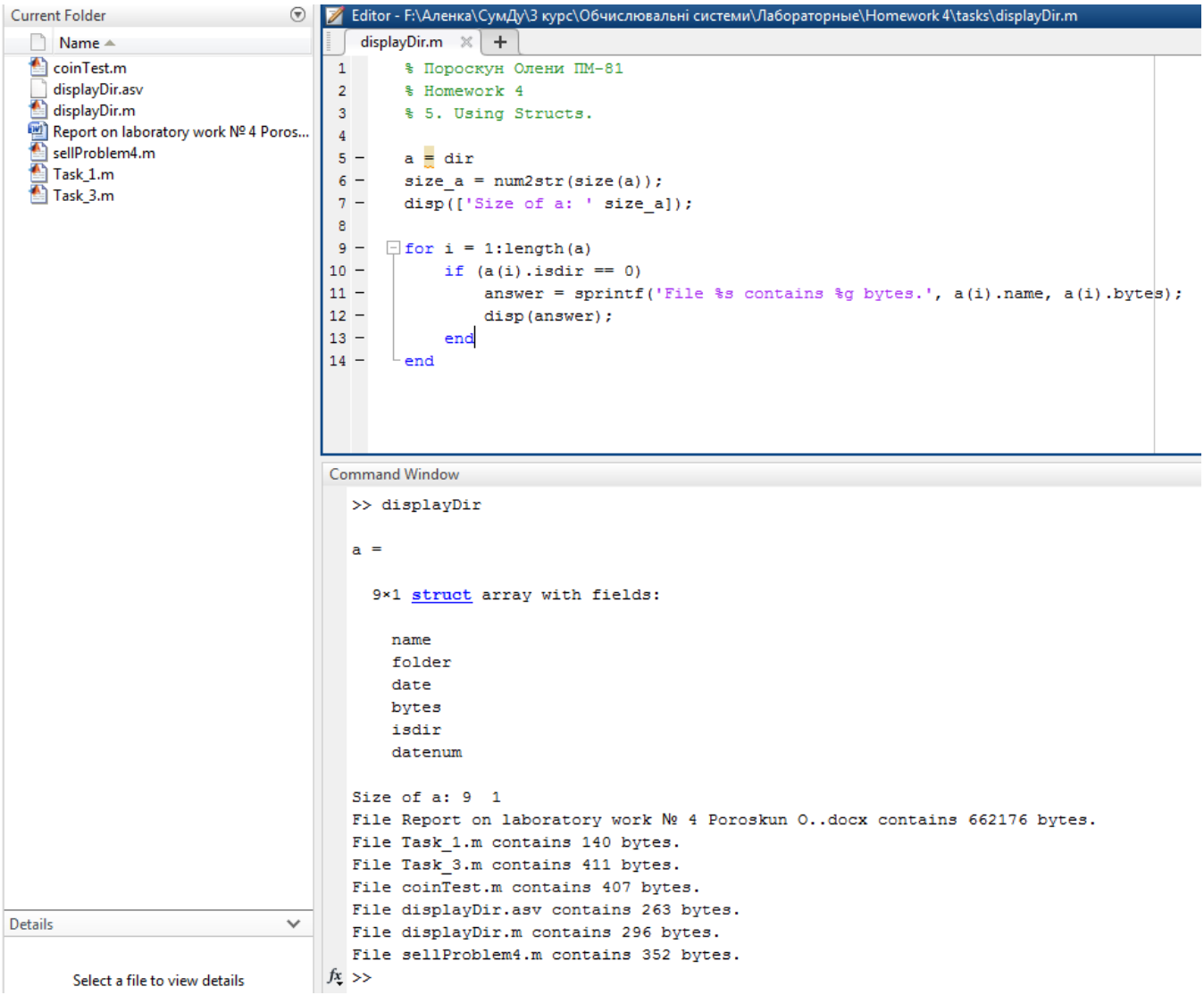
```
1   % Пороскун Олени ПМ-81
2   % Homework 4
3   % 4. Practice with cells.
4
5   arr_cell = cell(3,3);
6   arr_cell(:, 1) =  {'Joe', 'Sarah', 'Pat'};
7   arr_cell(:, 2) = {'Smith', 'Brown', 'Jackson'};
8   arr_cell(:, 3) = {30000, 150000, 120000};
9   disp(arr_cell);
10
11  arr_cell(2, 2) = {'Meyers'};
12  disp(arr_cell);
13
14  arr_cell{3, 3} = arr_cell{3, 3} + 50000;
15  disp(arr_cell);
```

Command Window

```
>> sellProblem4
    'Joe'       'Smith'       [ 30000]
    'Sarah'     'Brown'       [150000]
    'Pat'       'Jackson'     [120000]

    'Joe'       'Smith'       [ 30000]
    'Sarah'     'Meyers'      [150000]
    'Pat'       'Jackson'     [120000]

    'Joe'       'Smith'       [ 30000]
    'Sarah'     'Meyers'      [150000]
    'Pat'       'Jackson'     [170000]

fx >>
```

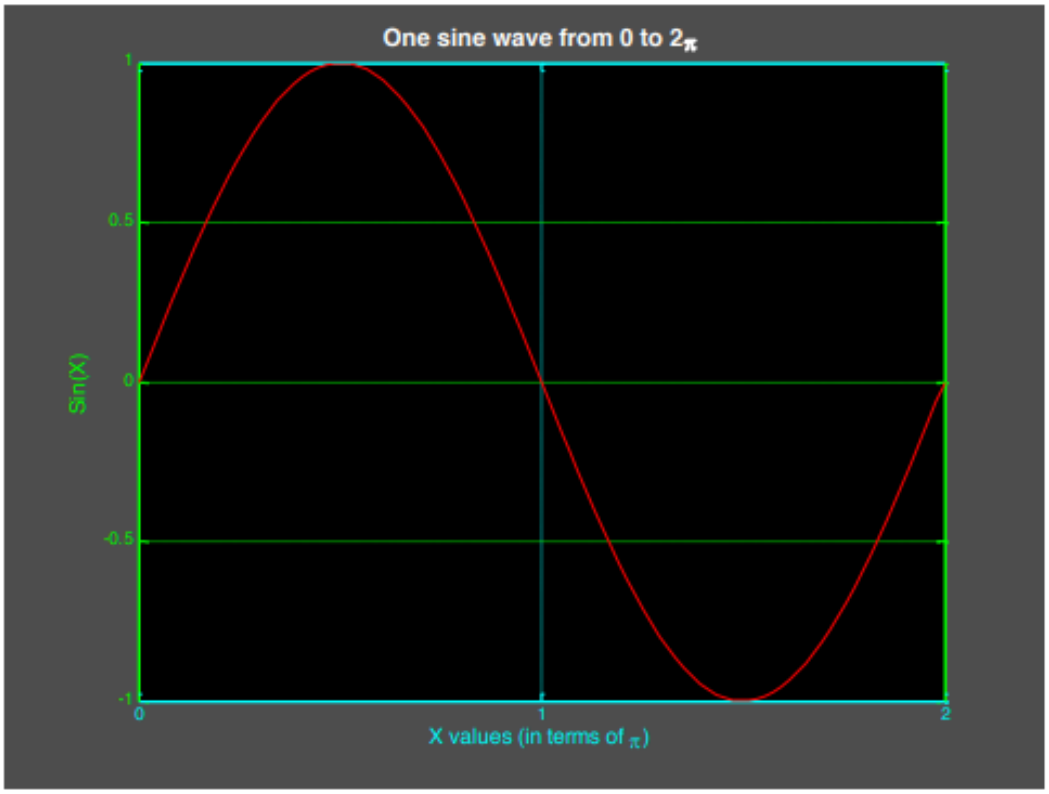
5. **Using Structs**. Structs are useful in many situations when dealing with diverse data. For example, get the contents of your current directory by typing `a=dir;`
   a. `a` is a struct array. What is its size? What are the names of the fields in `a`?
   b. Write a loop to go through all the elements of a, and if the element is not a directory, display the following sentence 'File *filename* contains X bytes', where *filename* is the name of the file and X is the number of bytes.
   c. Write a function called `displayDir.m`, which will display the sizes of the files in the current directory when run, as below. Your output may have different filenames.

```
>> displayDir
File brown2D.m contains 417 bytes.
File coinTest.m contains 524 bytes.
File displayDir.m contains 304 bytes.
File plotPoisson.m contains 543 bytes.
File someData.txt contains 66 bytes.
```



Optional Homework Assignments

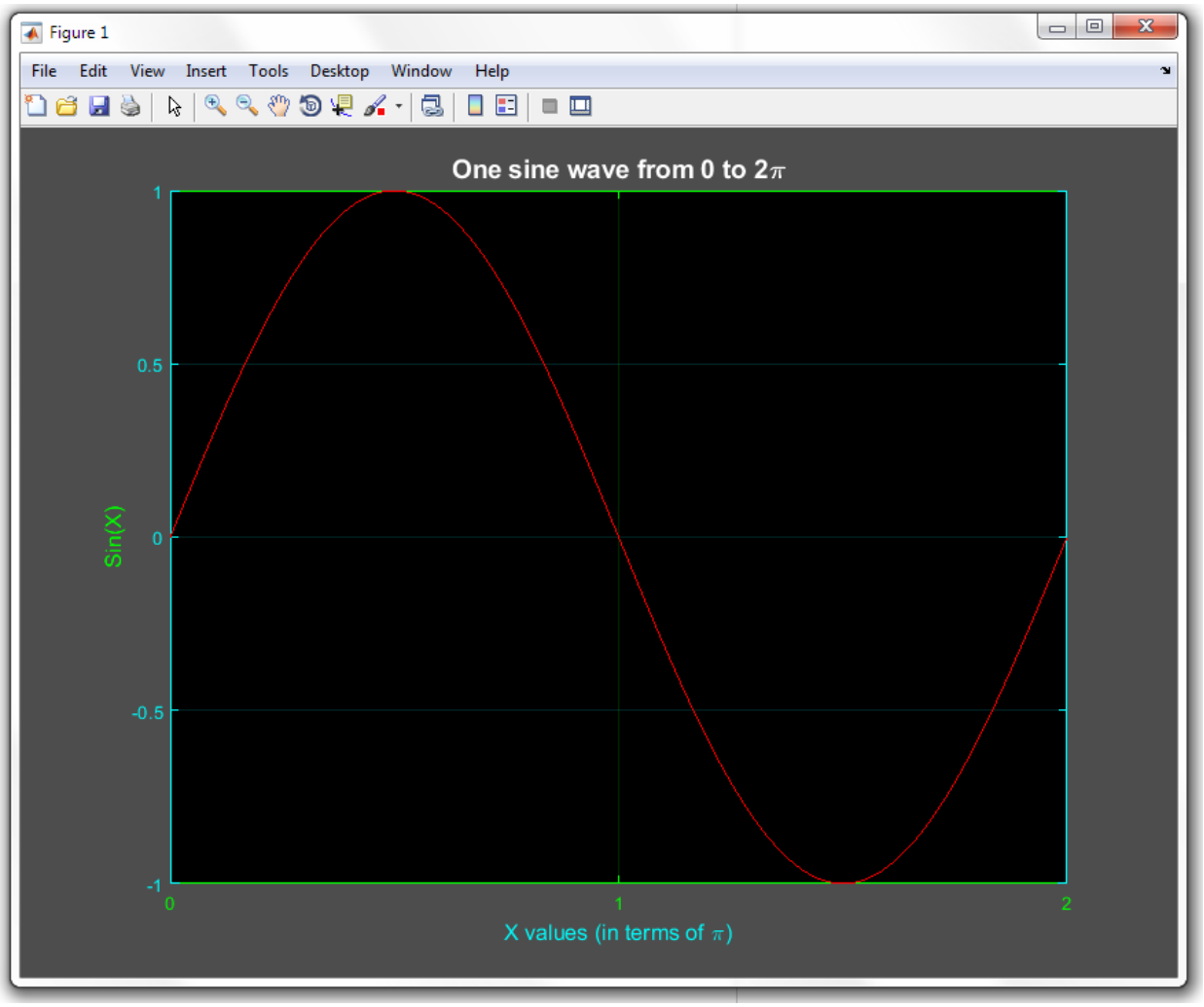
6. **Handles.** We'll use handles to set various properties of a figure in order to make it look like this:



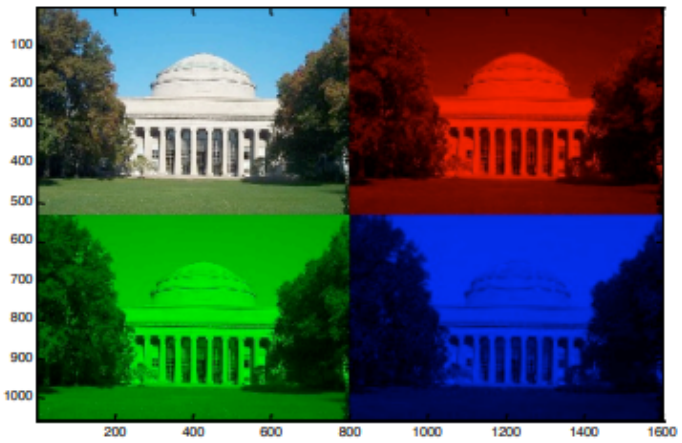
a. Do all the following in a script named `handlesPractice.m`
b. First, make a variable `x` that goes from 0 to 2π, and then make `y=sin(x)`.
c. Make a new `figure` and do `plot(x,y,'r')`
d. Set the x limit to go from 0 to 2π (**xlim**)
e. Set the `xtick` property of the axis to be just the values `[0 pi 2*pi]`, and set `xticklabel` to be `{'0','1','2'}`. Use **set** and **gca**
f. Set the `ytick` property of the axis to be just the values `-1:.5:1`. Use **set** and **gca**
g. Turn on the grid by doing `grid on`.
h. Set the `ycolor` property of the axis to green, the `xcolor` property to cyan, and the `color` property to black (use **set** and **gca**)
i. Set the `color` property of the figure to a dark gray (I used [.3 .3 .3]). Use **set** and **gcf**
j. Add a title that says 'One sine wave from 0 to 2π' with `fontsize` 14, `fontweight` bold, and `color` white. Hint: to get the π to display properly, use \pi in your string. Matlab uses a Tex or Latex interpreter in xlabel, ylabel, and title. You can do all this just by using **title**, no need for handles.
k. Add the appropriate x and y labels (make sure the π shows up that way in the x label) using a `fontsize` of 12 and `color` cyan for x and green for y. Use **xlabel** and **ylabel**

handlesPractice.m

```
1     % Пороскун Олени ПМ-81
2     % Homework 4
3     % 6. Handles.
4
5 -   x = 0:pi/100:2*pi;
6 -   y = sin(x);
7 -   plot(x, y, 'r');
8 -   xlim([0, 2*pi])
9 -   set(gca, 'XTick', [0 pi 2*pi]);
10 -  set(gca, 'XTickLabel', {'0', '1', '2'});
11 -  set(gca, 'YTick', -1:.5:1);
12 -  grid on;
13 -  set(gca, 'XColor', 'g');
14 -  set(gca, 'YColor', 'c');
15 -  set(gca, 'Color',  'k');
16 -  set(gcf, 'Color', [.3 .3 .3]);
17 -  title('One sine wave from 0 to 2\pi','FontSize',14,'FontWeight','Bold','Color', 'w');
18 -  xlabel ('X values (in terms of \pi)', 'FontSize',12, 'Color','c');
19 -  ylabel('Sin(X)', 'FontSize', 12, 'Color', 'g');
```



7. **Image processing**. Write a function to display a color image, as well as its red, green, and blue layers separately. The function declaration should be `im=displayRGB(filename)`. `filename` should be the name of the image (make the function work for *.jpg images only). `im` should be the final image returned as a matrix. To test the function, you should put a jpg file into the same directory as the function and run it with the filename (include the extension, for example `im=displayRGB('testImage.jpg')`). You can use any picture you like, from your files or off the internet. Useful functions: **imread**, **meshgrid**, **interp2**, **uint8**, **image**, **axis equal**, **axis tight**.
    a. To make the program work efficiently with all image sizes, first interpolate each color layer of the original image so that the larger dimension ends up with 800 pixels. The smaller dimension should be appropriately scaled so that the length:width ratio stays the same. Use **interp2** with cubic interpolation to resample the image. **Hint:** The image is an MxNx3 matrix; you need to interpolate each of the 3 color layers separately. **Note:** If you have difficulty doing this section, try part (b) first. (But if you get part (b) working, try to do this and use the interpolated image for part (b).)
    b. Create a composite image that is 2 times as tall as the original, and 2 times as wide. Place the original image in the top left, the red layer in the top right, the green layer in the bottom left, and the blue layer in the bottom right parts of this composite image. The function should return the composite image matrix in case you want to save it as a jpg again (before displaying or returning, convert the values to unsigned 8-bit integers using **uint8**). **Hint:** To get just a single color layer, all you have to do is set the other two layers to zero. For example if `X` is an MxNx3 image, then `X(:,:,2)=0; X(:,:,3)=0;` will retain just the red layer. Include your code and the final image in your homework writeup. It should look something like this:

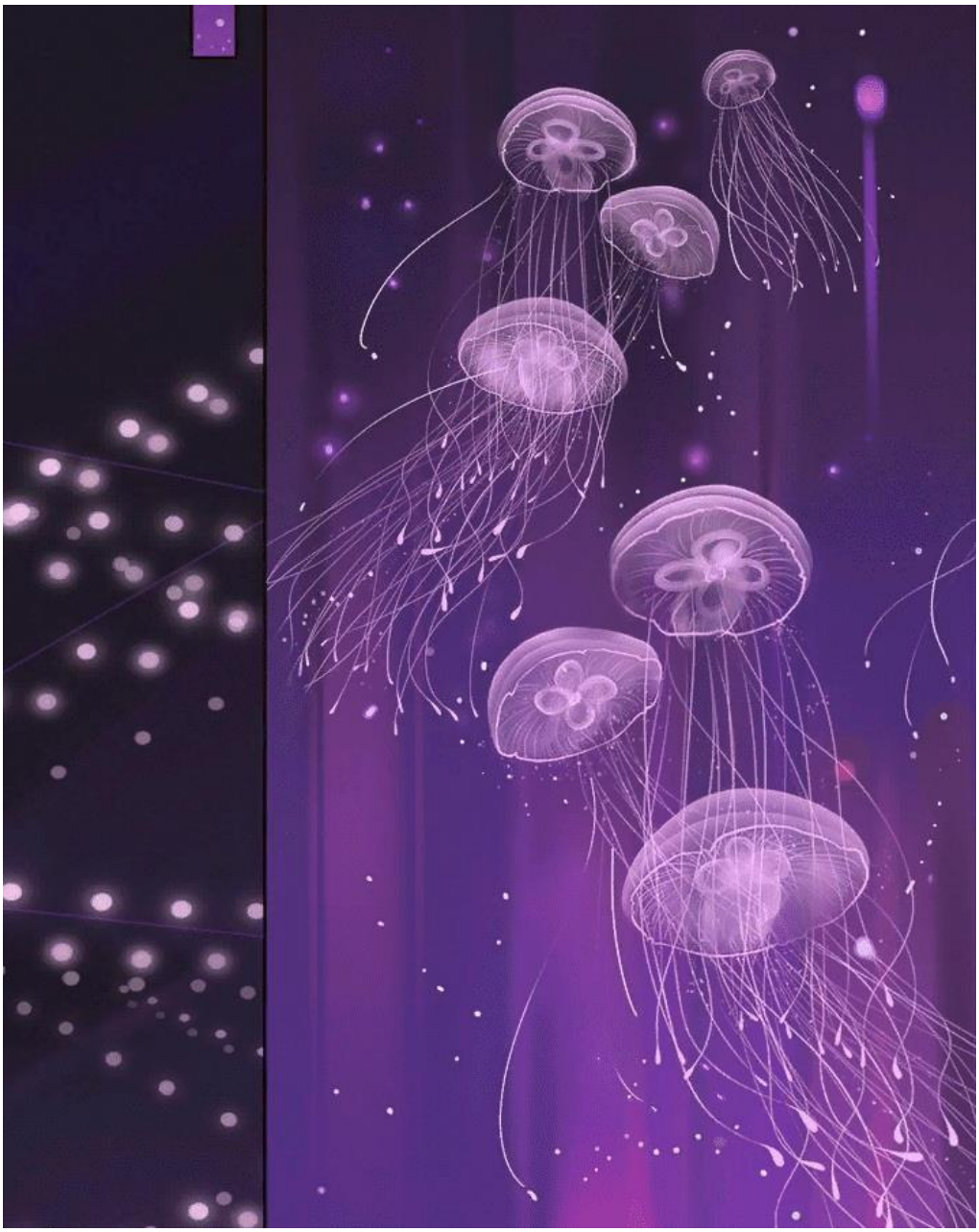


Task_7.m

```m
% Пороскун Олени ПМ-81
% Homework 4
% 7. Image processing.

im = displayRGB('M1.jpg');

function im = displayRGB(filename)

[PATHSTR,NAME,EXT] = fileparts(filename);
if (EXT == '.jpg')                                          % функція має працювати тільки для зображень *.jpg
    myImage = imread(filename);                             % imread - читання зображення з файлу
    myImageSize = size(myImage);
    myImage = double(myImage);                              % double - представлення елементів масиву у форматі double
    newLength = 800;

    if (max(myImageSize) ~= newLength)
        myImageSize = size(myImage);
        x = 1:myImageSize(2);
        y = 1:myImageSize(1);
        [X,Y] = meshgrid(x,y);

        newSize = floor(myImageSize / max(myImageSize)*newLength);
        xn = linspace(1, myImageSize(2), newSize(2));
        yn = linspace(1, myImageSize(1), newSize(1));
        [Xn,Yn] = meshgrid(xn,yn);

        for i = 1:3                                         % ZI = interp2(X,Y,Z,XI,YI,method) - повертає матрицю ZI, що містить
            newImage(:,:,i) = interp2(X,Y,myImage(:,:,i),Xn,Yn,'cubic');   % значення функції в точках, заданих аргументами XI і YI, отримані
        end                                                 % шляхом інтерполяції двовимірної залежності, заданої матрицями X,Y і Z.

    else
        newImage = myImage;
    end
```

Task_7.m

```m
% Пороскун Олени ПМ-81
% Homework 4
% 7. Image processing.

im = displayRGB('M1.jpg');

function im = displayRGB(filename)

[PATHSTR,NAME,EXT] = fileparts(filename);
if (EXT == '.jpg')                                          % функція має працювати тільки для зображень *.jpg
    myImage = imread(filename);                             % imread - читання зображення з файлу
    myImageSize = size(myImage);
    myImage = double(myImage);                              % double - представлення елементів масиву у форматі double
    newLength = 800;

    if (max(myImageSize) ~= newLength)
        myImageSize = size(myImage);
        x = 1:myImageSize(2);
        y = 1:myImageSize(1);
        [X,Y] = meshgrid(x,y);

        newSize = floor(myImageSize / max(myImageSize)*newLength);
        xn = linspace(1, myImageSize(2), newSize(2));
        yn = linspace(1, myImageSize(1), newSize(1));
        [Xn,Yn] = meshgrid(xn,yn);

        for i = 1:3                                         % ZI = interp2(X,Y,Z,XI,YI,method) - повертає матрицю ZI, що містить
            newImage(:,:,i) = interp2(X,Y,myImage(:,:,i),Xn,Yn,'cubic');   % значення функції в точках, заданих аргументами XI і YI, отримані
        end                                                 % шляхом інтерполяції двовимірної залежності, заданої матрицями X,Y і Z.

    else
        newImage = myImage;
    end

    myImage = uint8(myImage);               % B = uint8(A) - функція uint8 використовується для перетворення елементів масиву
    newImage = uint8(newImage);             % A в цілі невід'ємні числа в діапазоні [0, 255] і поміщає їх в новий масив B.

    imageRed = newImage;                    % створюємо зображенння в червоному кольорі замінюючи значення в палітрі RGB
    imageRed(:,:,2) = 0;                    % зеленого та блакитного на 0
    imageRed(:,:,3) = 0;
    imageGreen = newImage;                  % створюємо зображенння в зеленому кольорі замінюючи значення в палітрі RGB
```



M1.jpg

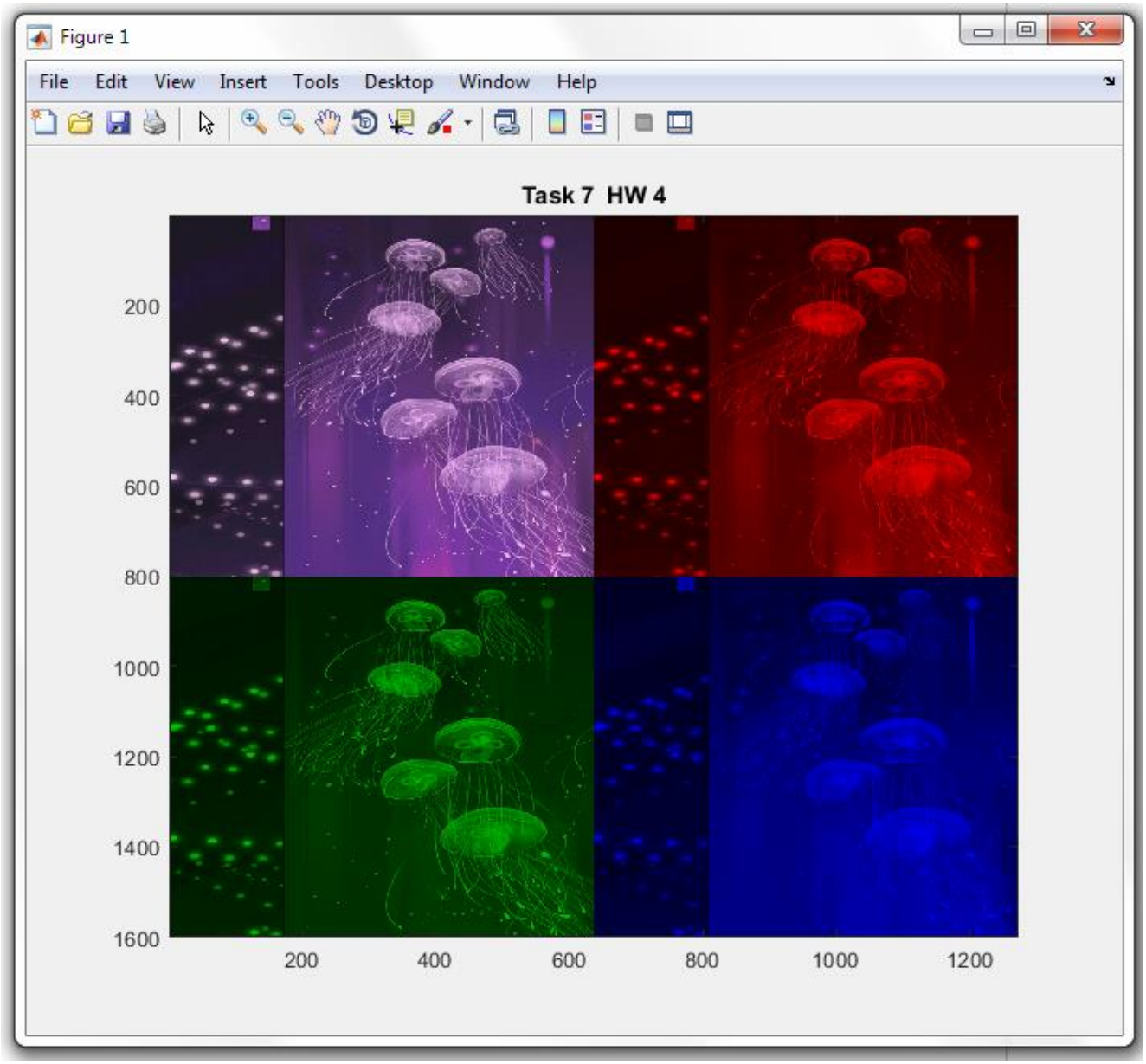


Figure 1
File Edit View Insert Tools Desktop Window Help
Task 7 HW 4
200
400
600
800
1000
1200
1400
1600
200
400
600
800
1000
1200

Task_7.m

```
% Пороскун Олени ПМ-81
% Homework 4
% 7. Image processing.

im = displayRGB('M1.jpg');

function im = displayRGB(filename)

[PATHSTR,NAME,EXT] = fileparts(filename);
if (EXT == '.jpg')                                          % функція має працювати тільки для зображень *.jpg
    myImage = imread(filename);                             % imread - читання зображення з файлу
    myImageSize = size(myImage);
    myImage = double(myImage);                              % double - представлення елементів масиву у форматі double
    newLength = 800;

    if (max(myImageSize) ~= newLength)
        myImageSize = size(myImage);
        x = 1:myImageSize(2);
        y = 1:myImageSize(1);
        [X,Y] = meshgrid(x,y);

        newSize = floor(myImageSize / max(myImageSize)*newLength);
        xn = linspace(1, myImageSize(2), newSize(2));
        yn = linspace(1, myImageSize(1), newSize(1));
        [Xn,Yn] = meshgrid(xn,yn);

        for i = 1:3                                         % ZI = interp2(X,Y,Z,XI,YI,method) - повертає матрицю ZI, що містить
            newImage(:,:,i) = interp2(X,Y,myImage(:,:,i),Xn,Yn,'cubic');   % значення функції в точках, заданих аргументами XI і YI, отримані
        end                                                 % шляхом інтерполяції двовимірної залежності, заданої матрицями X,Y і Z.

    else
        newImage = myImage;
    end

    myImage = uint8(myImage);                   % B = uint8(A) - функція uint8 використовується для перетворення елементів масиву
    newImage = uint8(newImage);                 % A в цілі невід'ємні числа в діапазоні [0, 255] і поміщає їх в новий масив B.

    imageRed = newImage;                        % створюємо зображенння в червоному кольорі замінюючи значення в палітрі RGB
    imageRed(:,:,2) = 0;                        % зеленого та блакитного на 0
    imageRed(:,:,3) = 0;
    imageGreen = newImage;                      % створюємо зображенння в зеленому кольорі замінюючи значення в палітрі RGB
```

Task_7.m

```
        x = 1:myImageSize(2);
        y = 1:myImageSize(1);
        [X,Y] = meshgrid(x,y);

        newSize = floor(myImageSize / max(myImageSize)*newLength);
        xn = linspace(1, myImageSize(2), newSize(2));
        yn = linspace(1, myImageSize(1), newSize(1));
        [Xn,Yn] = meshgrid(xn,yn);

        for i = 1:3                                                 % ZI = interp2(X,Y,Z,XI,YI,method) - повертає матрицю ZI, що містить
            newImage(:,:,i) = interp2(X,Y,myImage(:,:,i),Xn,Yn,'cubic');    % значення функції в точках, заданих аргументами XI і YI, отримані
        end                                                         % шляхом інтерполяції двовимірної залежності, заданої матрицями X,Y і Z.

    else
        newImage = myImage;
    end

    myImage = uint8(myImage);                   % B = uint8(A) - функція uint8 використовується для перетворення елементів масиву
    newImage = uint8(newImage);                 % A в цілі невід'ємні числа в діапазоні [0, 255] і поміщає їх в новий масив B.

    imageRed = newImage;                        % створюємо зображенння в червоному кольорі замінюючи значення в палітрі RGB
    imageRed(:,:,2) = 0;                        % зеленого та блакитного на 0
    imageRed(:,:,3) = 0;
    imageGreen = newImage;                      % створюємо зображенння в зеленому кольорі замінюючи значення в палітрі RGB
    imageGreen(:,:,1) = 0;                      % червоного та блакитного на 0
    imageGreen(:,:,3) = 0;
    imageBlue = newImage;                       % створюємо зображенння в блакитному кольорі замінюючи значення в палітрі RGB
    imageBlue(:,:,1) = 0;                       % червоного та зеленого на 0
    imageBlue(:,:,2) = 0;
    imageDecomposition = [newImage,imageRed; imageGreen,imageBlue];     % створюємо матрицю з 4 уже заданих раніше  зображень

    figure;
    im = imageDecomposition;                    % im повинен бути кінцевим зображенням, що повертається у вигляді матриці
    image(im);                                  % image - відображення растрових зображень у графічному вікні
    title('Task 7  HW 4');
    axis tight                                  % axis tight - встановлює діапазони координат по осях відповідно до діапазонів зміни даних.
else
    disp('The file received is not jpg.');
    im = 0;
end
end

```

Sumy State University

Department

of

Applied Mathematics and Complex Systems Modeling

Report on laboratory work № 5

Subject

Computing system

Student: Olena Poroskun

Teacher: Igor A. Knyaz

Sumy, Sumy region

2020

Lab 5

Problem 1: Component Analysis

An electrical engineer supervises the production of three types of electrical components. Three kinds of material -- metal, plastic, and rubber -- are required for production. The amounts needed to produce each component are:

Component	Metal (g/component)	Plastic (g/component)	Rubber (g/component)
1	15	0.30	1.0
2	17	0.40	1.2
3	19	0.55	1.5

Suppose that totals of 3.89, 0.095, and 0.282 kg of metal, plastic, and rubber, respectively, are used each day. We wish to find how many of each component has been produced per day.

a. **Give the equations for this linear system**, making sure to identify what the unknowns are.
b. **Solve your system from part a using MATLAB** by creating the coefficient matrix and data vector and solving the system using the backslash operator.

Task_1.m

```
% Пороскун Олени ПМ-81
% Homework 5
% Problem 1: Component Analysis

% Components: 1, 2, 3.

Material = [15,   17,   19;    % Metal (g/component)
            0.3,  0.4, 0.55;   % Plastic (g/component)
            1.0,  1.2,  1.5]   % Rubber (g/component)
Total_component = [3.89;
                   0.095;
                   0.282];
Total_component = Total_component*10^3  % переводимо з кілограмів в грами

answer = Material \ Total_component;
answer = sprintf('Component 1 = %g \nComponent 2 = %g \nComponent 3 = %g',+...
+ answer(1), answer(2), answer(3));
disp(answer)

```

Command Window

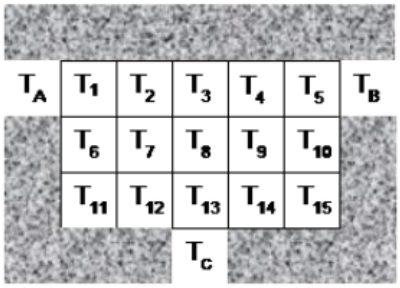
```
>> Task_1

Material =

   15.0000   17.0000   19.0000
    0.3000    0.4000    0.5500
    1.0000    1.2000    1.5000


Total_component =

        3890
          95
         282

Component 1 = 90
Component 2 = 60
Component 3 = 80
>>
```

Problem 2: Heat Plate Linear System

Solve a linear system to determine the temperature at various locations across a heat plate. We have simplified the problem to focus on the general concepts used when modeling such a problem as a system of linear equations.

Imagine that you wish to know the temperature at all points on a plate that is insulated except for three positions on the edge where heating elements of known temperature are applied. At some point when the temperature at all positions on the plate have stabilized, what will the temperature at each point be?

To solve such a problem, we imagine the plate surface divided into equal sized "tiles" and restate the problem to find the temperature at the center of each tile. We know that the temperature of each tile depends upon its neighboring temperatures. In fact, it is a weighted average of those temperatures based on the proportion of our tile's edge that is shared with its neighbor. Here is a rectangular "plate" divided into 15 equal area tiles. Assume that each edge of each tile has identical length, so that the temperature of any one tile is a simple average of the temperatures of all neighboring tiles that share a side with the tile in question. This creates a system of 15 equations, one for each "tile" area.



T_A, T_B, and T_C are the non-insulated positions along the edge where a heat source of known temperature will be applied. The temperatures at T_1 through T_{15} are the unknowns that you wish to find.

a. **Define a function named `heatPlate`** that accepts three temperature values `TA`, `TB`, and `TC` (representing T_A, T_B, and T_C, respectively) and returns a single matrix result that is a 3x5 matrix of temperatures for the unknowns T_1 through T_{15} in the order labeled above. If you can not figure out how to return the values as a 3x5 matrix, return them as a column vector for a minor (1/2 pt) deduction.

   To solve this problem, identify 15 equations. Each equation represents the temperature value for one tile based upon the temperature values of each of its neighboring tiles. For example, the temperature at T_7 is equal to the average of the temperatures: T_2, T_6, T_8, and T_{12}.

   What about the corners? Well, since the gray area is insulation, the only effect is from the non-insulating tiles. Thus, T_{11} is equal to the average of only T_6 and T_{12}.

   How do T_A, T_B, and T_C factor into the equations? They will be in the equations, but since their values will be known when the function is called, the terms with T_A, T_B, and T_C should be on the right-hand side of the equations once the equations are put into general form.

   DO NOT SOLVE FOR ANY OF THE TEMPERATURES OF THIS PROBLEM BY HAND. Work out the equations for each tile and then reorder the terms in general form, with the unknown terms on the left-hand side and the known terms on the right-hand side of the equation. Once you have the equations, rewrite the linear system as a matrix equation and put the commands to solve this problem in your `heatPlate(TA,TB,TC)` function definition.

b. **Write code to call your `heatPlate` function to compute the values of `T1` through `T15` for each of the following conditions.** Do not suppress the output of your function calls.
   i. Find `T1` through `T15` when `TA` = 50, `TB` = 50, and `TC` = 50
   ii. Find `T1` through `T15` when `TA` = 20, `TB` = 20, and `TC` = 80
   iii. Find `T1` through `T15` when `TA` = 15, `TB` = 95, and `TC` = 40

Task_2.m

```
% Пороскун Олени ПМ-81
% Homework 5
% Problem 2: Heat Plate Linear System

T1 = heatPlate(50, 50, 50) % 1 випадок
T2 = heatPlate(20, 20, 80) % 2 випадок
T3 = heatPlate(15, 95, 40) % 3 випадок

function T = heatPlate(TA, TB, TC)
TA
TB
TC
                % 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
Temperature = [ 3,-1, 0, 0, 0,-1, 0, 0, 0, 0, 0, 0, 0, 0, 0;  % 1
               -1, 3,-1, 0, 0, 0,-1, 0, 0, 0, 0, 0, 0, 0, 0;  % 2
                0,-1, 3,-1, 0, 0, 0,-1, 0, 0, 0, 0, 0, 0, 0;  % 3
                0, 0,-1, 3,-1, 0, 0, 0,-1, 0, 0, 0, 0, 0, 0;  % 4
                0, 0, 0,-1, 3, 0, 0, 0, 0,-1, 0, 0, 0, 0, 0;  % 5
               -1, 0, 0, 0, 0, 3,-1, 0, 0, 0,-1, 0, 0, 0, 0;  % 6
                0,-1, 0, 0, 0,-1, 4,-1, 0, 0, 0,-1, 0, 0, 0;  % 7
                0, 0,-1, 0, 0, 0,-1, 4,-1, 0, 0, 0,-1, 0, 0;  % 8
                0, 0, 0,-1, 0, 0, 0,-1, 4,-1, 0, 0, 0,-1, 0;  % 9
                0, 0, 0, 0,-1, 0, 0, 0,-1, 3, 0, 0, 0, 0,-1;  % 10
                0, 0, 0, 0, 0,-1, 0, 0, 0, 0, 2,-1, 0, 0, 0;  % 11
                0, 0, 0, 0, 0, 0,-1, 0, 0, 0,-1, 3,-1, 0, 0;  % 12
                0, 0, 0, 0, 0, 0, 0,-1, 0, 0, 0,-1, 4,-1, 0;  % 13
                0, 0, 0, 0, 0, 0, 0, 0,-1, 0, 0, 0,-1, 3,-1;  % 14
                0, 0, 0, 0, 0, 0, 0, 0, 0,-1, 0, 0, 0,-1, 2]; % 15

              % 1  2  3  4  5  6  7  8  9  10 11  12 13 14 15
Temp_conct = [TA; 0; 0; 0; TB; 0; 0; 0; 0; 0; 0; 0; TC; 0; 0];
T = Temperature \ Temp_conct;
end
```

Command Window

```
>> Task_2

TA =

    50

TB =

    50

TC =

    50

T1 =

   50.0000
   50.0000
   50.0000
   50.0000
   50.0000
   50.0000
   50.0000
   50.0000
   50.0000
   50.0000
   50.0000
   50.0000
   50.0000
   50.0000
   50.0000

TA =

    20

TA =

    20

TB =

    20

TC =

    80

T2 =

   32.6839
   38.6481
   40.7952
   38.6481
   32.6839
   39.4036
   42.4652
   45.0895
   42.4652
   39.4036
   43.0616
   46.7197
   54.6322
   46.7197
   43.0616

TA =

    15

TA =

    15

TB =

    95

TC =

    40

T3 =

   31.7243
   41.5422
   49.8012
   59.1338
   71.9337
   38.6308
   43.1010
   48.7276
   55.6664
   61.6674
   41.0671
   43.5033
   46.3419
   53.1368
   57.4021
```

Problem 3: Interpolation

The table below lists values for dissolved oxygen concentration in water (with a chloride concentration of 20 g/L) as a function of temperature.

***T* (C)**	0	5	10	15	20	25	30
Dissolved oxygen (mg/L)	11.4	10.3	8.96	8.08	7.35	6.73	6.20

We ultimately wish to estimate the dissolved oxygen concentration at a temperature of 2.5 C.

a. **Interpolate the data** in two ways:
   1. using a polynomial that goes through all the data points
   2. using a polynomial of the appropriate degree that goes through the first 3 data points

   Display the coefficients of each interpolating polynomial.

b. **Plot the data points and the interpolating polynomials on a single figure.** Use black circles to represent the data points. Plot the polynomial through all the data points over the range T = 0 to 30 C and as a magenta dashed line. Plot the polynomial through just the first 3 data points over the range T = 0 to 10 C and as a solid cyan line.

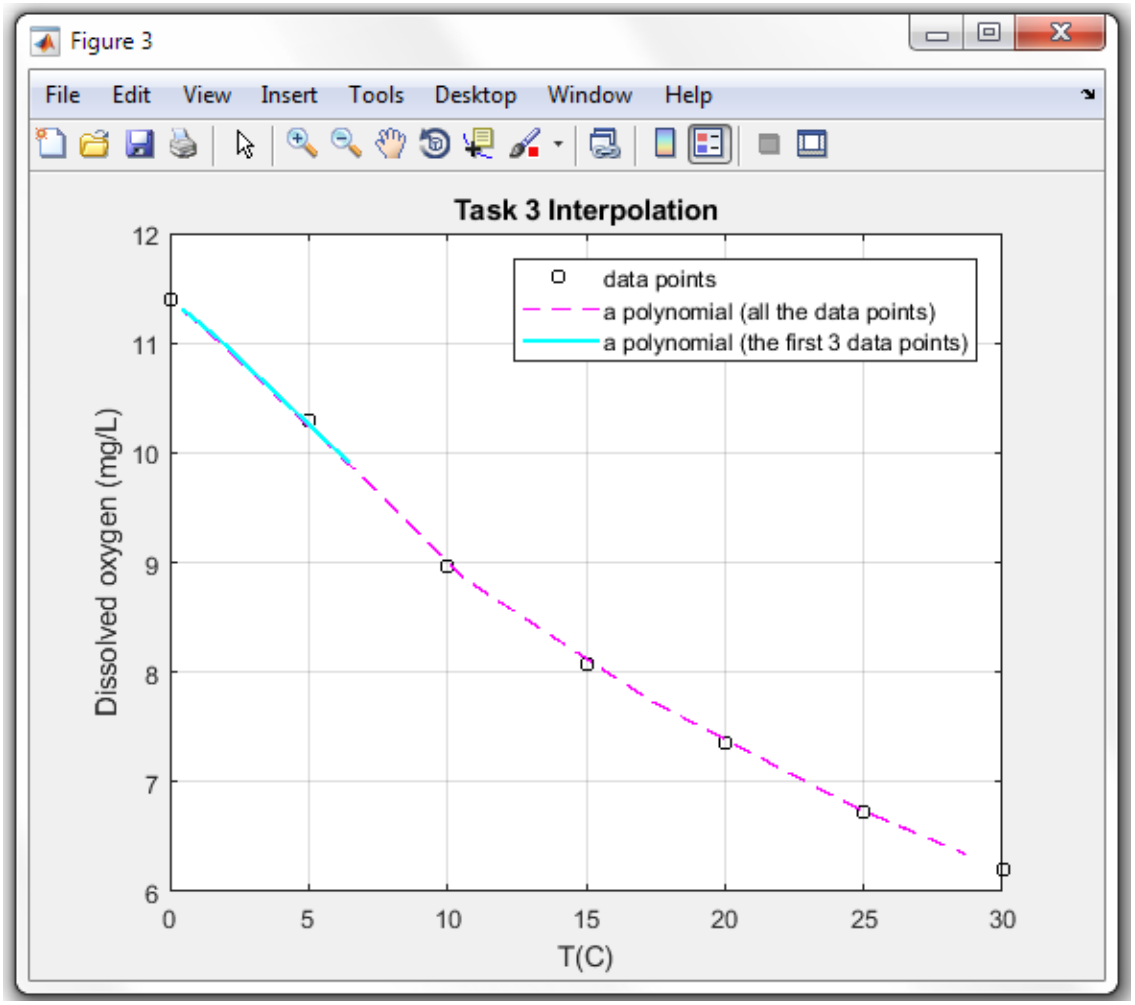
c. **Estimate the dissolved oxygen concentration at a temperature of 2.5 C in three ways:**

i. using your polynomial interpolation through all the data points

ii. using your polynomial interpolation through the first 3 data points

iii. using a spline interpolation

d. **Which of your estimates from part c is the best? Which is the worst?** Briefly justify your answers.



Task_3.m

```
% Пороскун Олени ПМ-81
% Homework 5
% Problem 3: Interpolation

figure(3);
T = 0:5:30;
Oxygen = [11.4, 10.3, 8.96, 8.08, 7.35, 6.73, 6.20];
plot(T, Oxygen, 'ko', 'MarkerSize', 5);
grid on; hold on;

T11 = 30*rand(1,7);
T11 = sort(T11)
O11 = interp1(T, Oxygen, T11)
plot(T11, O11, 'm--', 'LineWidth', 1);

T22 = 10*rand(1,3);
T22 = sort(T22)
O22 = interp1(T, Oxygen, T22, 'PCHIP')  % 'pchip' — интерполяция кубическим эрмитовым сплайном
plot(T22, O22, 'c', 'LineWidth', 1.5);

Temp = 2.5;
O1 = interp1(T, Oxygen, Temp)
O2 = interp1(T(1:3), Oxygen(1:3), Temp, 'PCHIP')
O3 = interp1(T, Oxygen, Temp, 'spline')           % spline' - интерполяция кубическим сплайном

title('Task 3 Interpolation');
xlabel('T(C)');
ylabel('Dissolved oxygen (mg/L)');
legend('data points','a polynomial (all the data points)','a polynomial (the first 3 data points)');
```

Command Window

```
>> Task_3

T11 =

    0.4621    1.7934    7.0434   10.5948   17.2563   24.6358   28.6840


O11 =

   11.2983   11.0055    9.7524    8.8553    7.7506    6.7752    6.3395


T22 =

    0.4302    1.6899    6.4912


O22 =

   11.3147   11.0542    9.9024


O1 =

   10.8500


O2 =

   10.8785


O3 =

   10.9418
```

Problem 4: Approximation

For the data in the following table, the *y* data obviously decays as *x* increases:

***x* data**	1.00	2.12	3.09	5.23	7.64	9.14	11.2	14.3	16.1	19.3	22.7	25.0
***y* data**	50.6	44.1	40.3	31.2	28.7	24.8	23.2	18.3	16.2	14.8	12.9	12.6

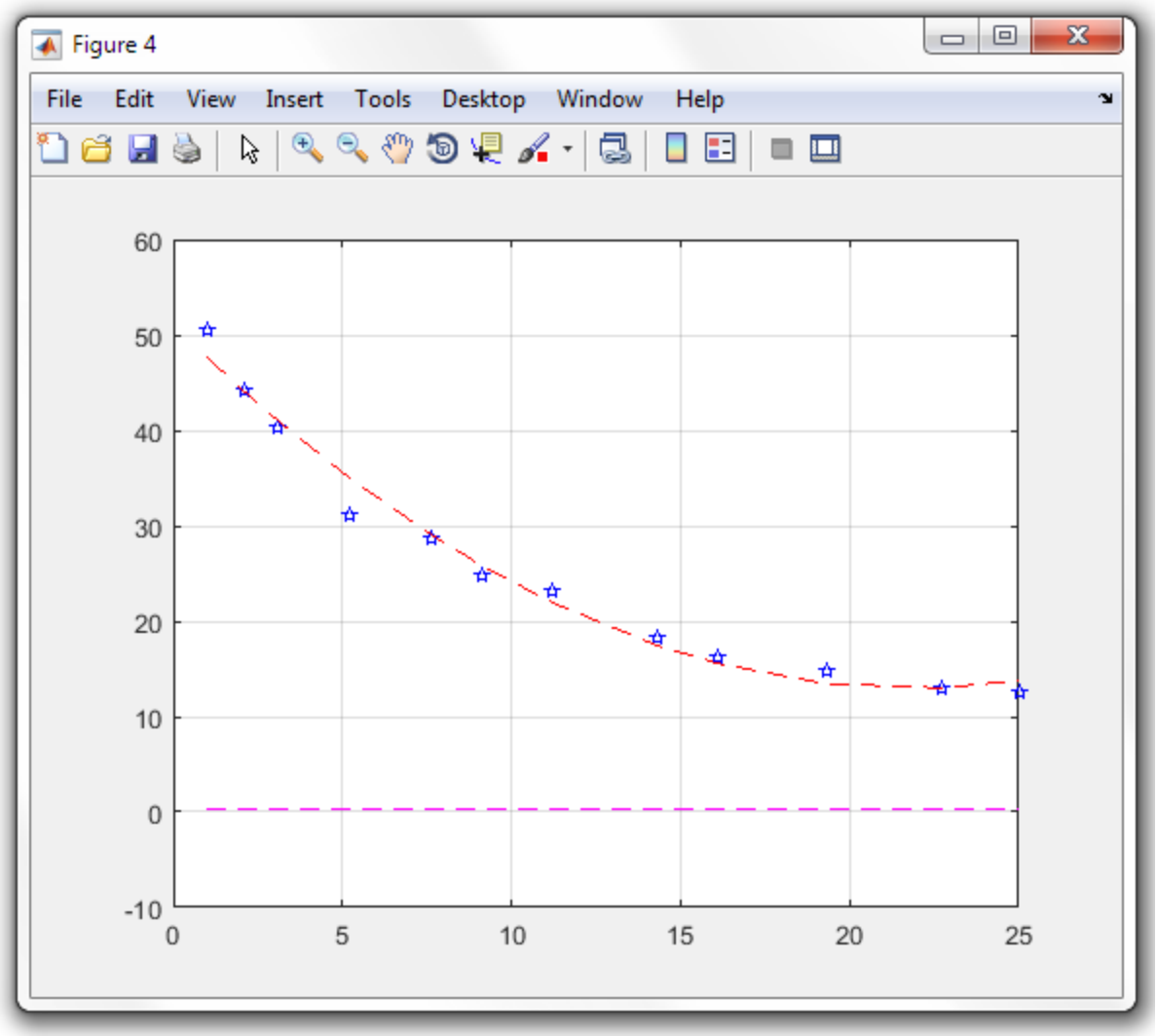
a. **Fit this data to a curve of the form:**

$$y = \frac{1}{ln(ax^2 + bx + c)}$$

Make sure to display and identify the values you get for *a*, *b*, and *c*. (*Hint: transform the curve to a polynomial.*)

b. **Plot the data points and the approximating function** over the range *x* = 1 to 25 on a single figure. Use blue pentagrams to represent the data points and a red dash-dot line for the approximating curve.

c. **Estimate the value of *y* at *x* = 8** using your approximating function.



Task_4.m

```
% Пороскун Олени ПМ-81
% Homework 5
% Problem 4: Approximation

x_data = [1.00 2.12 3.09 5.23 7.64 9.14 11.2 14.3 16.1 19.3 22.7 25.0];
y_data = [50.6 44.1 40.3 31.2 28.7 24.8 23.2 18.3 16.2 14.8 12.9 12.6];

% y = 1/(log(a*x^2+b*x+c));
% log(a*x^2+b*x+c) = 1/y;
% a*x^2 + b*x + c = exp(1/y);

figure(4);
plot(x_data, y_data, 'bp')
grid on
hold on

p = polyfit(x_data, y_data, 2)
f = polyval(p, x_data)
plot(x_data, f, 'r--')

y = 1./(log(f))
plot(x_data, y, 'm--')
ylim([-10 60]);

x1 = 8;
f1 = polyval(p, x1);
disp(['The value of y at x = 8 is equal to ', num2str(f1),'.']);
```

Command Window

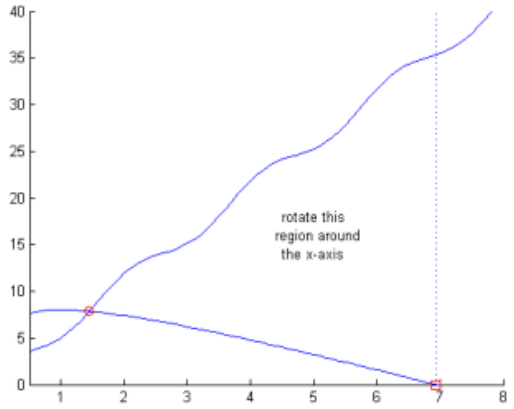
```
>> Task_4

p =

    0.0801   -3.4954   51.0598


f =

   47.6445   44.0097   41.0241   34.9708   29.0326   25.8067   21.9642   17.4638   15.5575   13.4509   13.0111   13.7643


y =

    0.2588    0.2642    0.2692    0.2813    0.2969    0.3076    0.3237    0.3496    0.3644    0.3848    0.3897    0.3814

The value of y at x = 8 is equal to 28.2255.
>>
```

Problem 5: Volume of Revolution

This problem deals with the two curves:

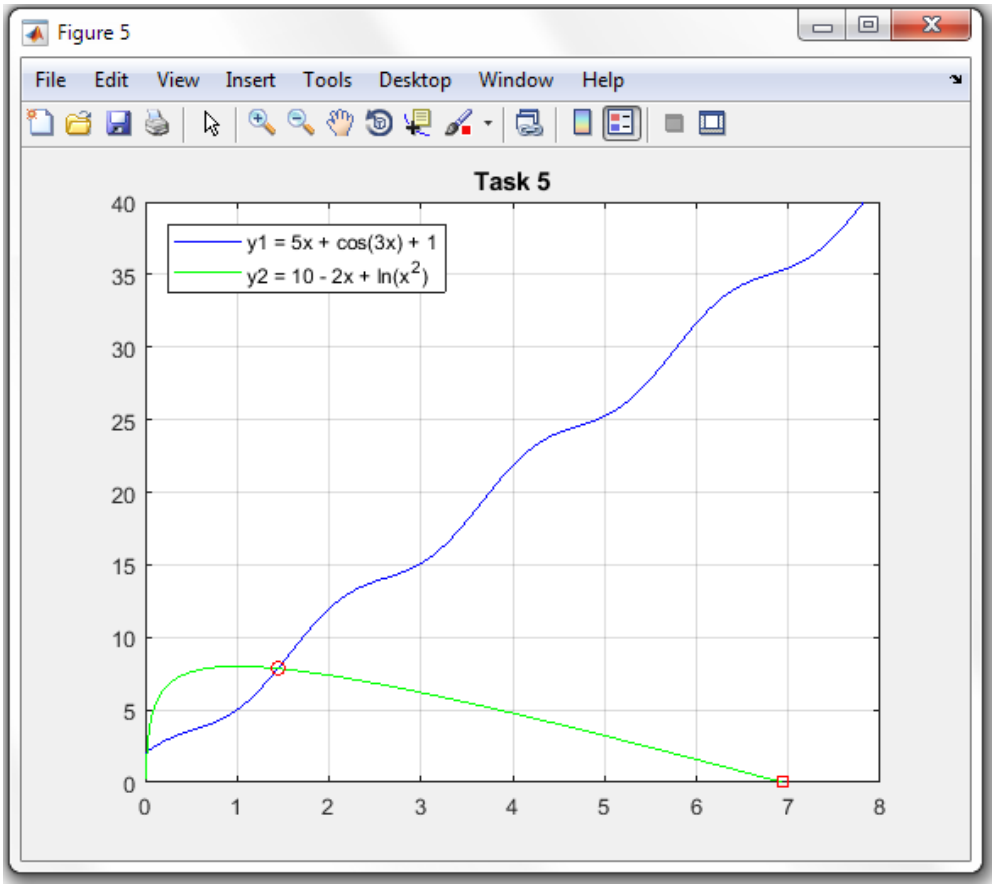
$$y = 5x + \cos(3x) + 1$$
$$y = 10 - 2x + \ln(x^2)$$

These curves are displayed in the figure below:



a. **Determine the *x*- and *y*-coordinates of the point where the two curves intersect** (indicated with a red circle in the figure above). Make sure your script displays the coordinates of the intersection point.
b. **Determine where the second curve intersects the *x*-axis** (indicated with a red square in the figure above). Make sure your script displays this value.

Determine the volume of the solid formed from rotating around the *x*-axis the region between the two curves from where they intersect until where the second curve intersects the *x*-axis. Note that you will need to use your points from parts a and b.



Task_5.m

```
% Пороскун Олени ПМ-81
% Homework 5
% Problem 5: Volume of Revolution

syms x;
y1 = 5*x + cos(3*x) + 1;
y2 = 10 - 2*x + log(x^2);

x_inter1 = vpasolve(y1-y2);     % vpasolve - чисельно розв'язує алгебраїчні рівняння
x_inter1 = vpa(x_inter1, 4);    % vpa -  задає число значущих цифр
y_inter1 = 10 - 2*x_inter1 + log(x_inter1^2);
y_inter1 = vpa(y_inter1, 4);
answer1 = sprintf('The point where the two curves intersect: \nx = %g, y = %g.', x_inter1, y_inter1);
disp(answer1);    % друкуємо координати точки перетину функції y1 та y2

x_inter2 = vpasolve(y2);
x_inter2 = vpa(x_inter2, 4);
y_inter2 = 0;
answer2 = sprintf('\nThe second curve intersects the x-axis in the point: \nx = %g, y = %g.\n', x_inter2, y_inter2);
disp(answer2);    % друкуємо координати точки перетину функції y2 та осі X

xx = linspace(0, 8, 1000);
y11 = 5*xx + cos(3*xx) + 1;
y22 = 10 - 2*xx + log(xx.^2);
figure(5);
plot(xx, y11, 'b', xx, y22, 'g');   % будуємо криві y1 та y2
hold on
grid on
plot(x_inter1, y_inter1, 'ro', x_inter2, y_inter2, 'rs'); % будуємо точки перетину, якы ми знайшли раніше
ylim([0 40]);

title('Task 5');
legend('y1 = 5x + cos(3x) + 1', 'y2 = 10 - 2x + ln(x^2)', 'Location', 'northwest');

V = int(pi*(y1^2 - y2^2), x_inter1, x_inter2);  % об'єм фігури обмежена двома кривими y1 та y2
V = vpa(V, 6);

fprintf('Volume of the solid formed from rotating around the x-axis the region between the two curves\n')
fprintf('from where they intersect until where the second curve intersects the x-axis: %g \n', V)
```

Command Window

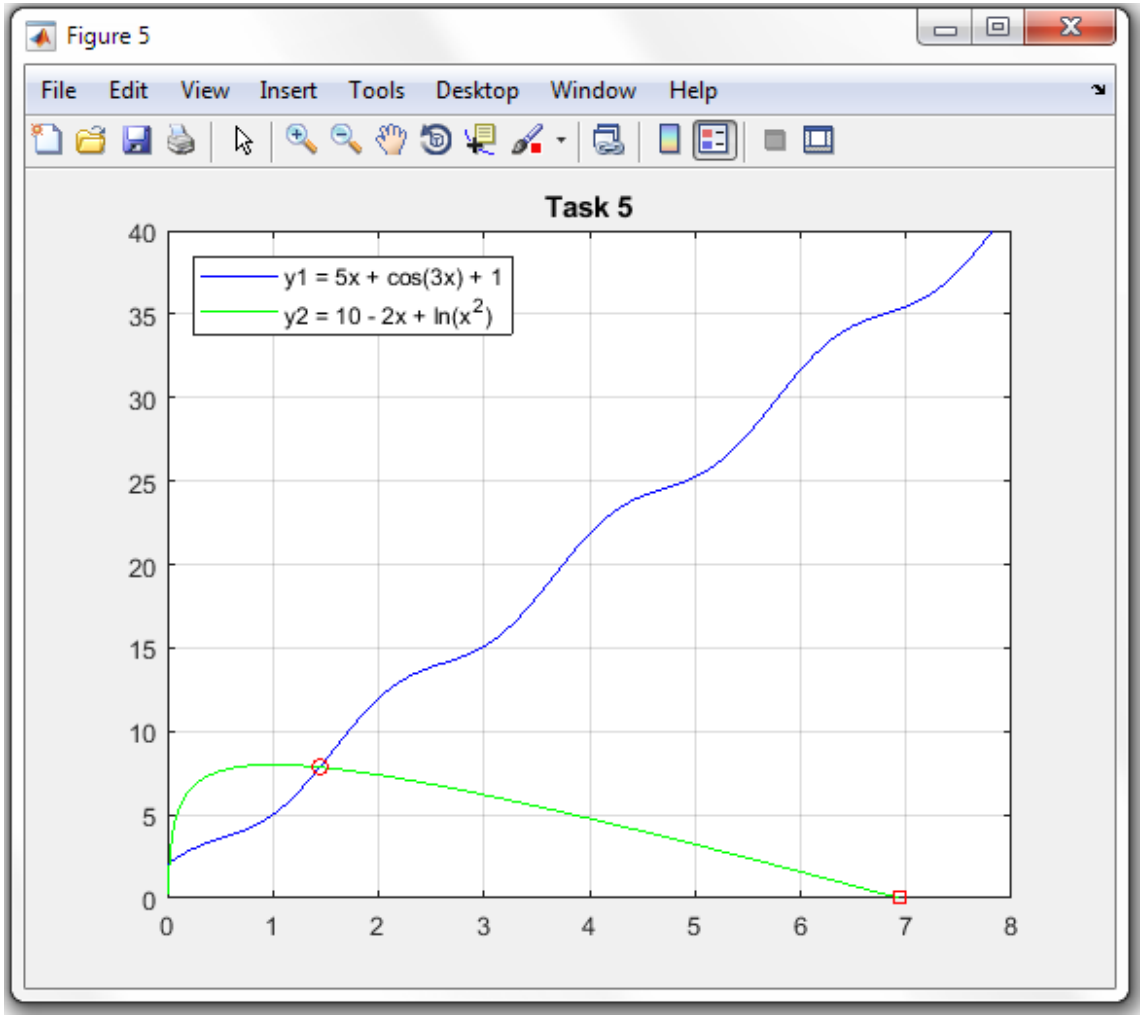
```
>> Task_5
The point where the two curves intersect:
x = 1.44378, y = 7.84697.

The second curve intersects the x-axis in the point:
x = 6.93685, y = 0.

Volume of the solid formed from rotating around the x-axis the region between the two curves
from where they intersect until where the second curve intersects the x-axis: 9077.56
>>
```

Task_5.m

```
% Пороскун Олени ПМ-81
% Homework 5
% Problem 5: Volume of Revolution

syms x;
y1 = 5*x + cos(3*x) + 1;
y2 = 10 - 2*x + log(x^2);

x_inter1 = vpasolve(y1-y2);     % vpasolve - чисельно розв'язує алгебраїчні рівняння
x_inter1 = vpa(x_inter1, 4);    % vpa -  задає число значущих цифр
y_inter1 = 10 - 2*x_inter1 + log(x_inter1^2);
y_inter1 = vpa(y_inter1, 4);
answer1 = sprintf('The point where the two curves intersect: \nx = %g, y = %g.', x_inter1, y_inter1);
disp(answer1);    % друкуємо координати точки перетину функції y1 та y2

x_inter2 = vpasolve(y2);
x_inter2 = vpa(x_inter2, 4);
y_inter2 = 0;
answer2 = sprintf('\nThe second curve intersects the x-axis in the point: \nx = %g, y = %g.\n', x_inter2, y_inter2);
disp(answer2);    % друкуємо координати точки перетину функції y2 та осі X

xx = linspace(0, 8, 1000);
y11 = 5*xx + cos(3*xx) + 1;
y22 = 10 - 2*xx + log(xx.^2);
figure(5);
plot(xx, y11, 'b', xx, y22, 'g');   % будуємо криві y1 та y2
hold on
grid on
plot(x_inter1, y_inter1, 'ro', x_inter2, y_inter2, 'rs'); % будуємо точки перетину, яки ми знайшли раніше
ylim([0 40]);

title('Task 5');
legend('y1 = 5x + cos(3x) + 1', 'y2 = 10 - 2x + ln(x^2)', 'Location', 'northwest');

V = int(pi*(y1^2 - y2^2), x_inter1, x_inter2);  % об'єм фігури обмежена двома кривими y1 та y2
V = vpa(V, 6);

fprintf('Volume of the solid formed from rotating around the x-axis the region between the two curves\n')
fprintf('from where they intersect until where the second curve intersects the x-axis: %g \n', V)
```

Command Window

```
>> Task_5
The point where the two curves intersect:
x = 1.44378, y = 7.84697.

The second curve intersects the x-axis in the point:
x = 6.93685, y = 0.

Volume of the solid formed from rotating around the x-axis the region between the two curves
from where they intersect until where the second curve intersects the x-axis: 9077.56
>>
```



Sumy State University

Department

of

Applied Mathematics and Complex Systems Modeling

Report on laboratory work № 6

Subject

Computing system

Student: Olena Poroskun

Teacher: Igor A. Knyaz

Sumy, Sumy region

2020

LAB 6

Dealing with digital images in MATLAB

MATLAB has a toolbox (i.e., library) that is useful for dealing with digital images. The Image Processing Toolbox contains many functions that can be used to read in, modify, and write images in different formats. The functions needed for this problem are:

- `imread(filename, format)` - reads in a digital image file with the given `filename` in the specified `format`. Both `filename` and `format` are strings and for this assignment, the format will always be `'jpg'`. A matrix representation of the image is returned. Example:
  - `filename = '….jpg';`
  - `image = imread(filename, 'jpg');`
- `imshow(image)` - displays the image in a separate image window. Example:
  - `imshow(image)`

The main program

The main program (which has been partially completed for you) first reads in the digital image file ….jpg (bmp) and displays it. The program then repeatedly presents the user with a menu of choices (numbered from 1 to 6), prompts to enter a choice and reads it in, and performs the desired actions, until the user enters `0` to quit. The menu options are as follows:

1. `new image` - prompts the user to enter the name of jpg image file, reads in the file and makes the image the new current image, and displays the new current image. You may assume the user always enters the name of a file which can be read in as a jpg image.
2. `original` - displays the current image in its original (un-filtered) state
3. `negative` - applies the `negative` filter to the original current image and displays the filtered image
4. `sharpen` - applies the `sharpen` filter to the original current image and displays the filtered image
5. `posterize` - applies the `posterize` filter to the original current image and displays the filtered image
6. `blur` - applies the `blur` filter to the original current image and displays the filtered image

If the user enters in something other than an integer between 0 and 6 (inclusive), the program displays `invalid choice`, redisplays the menu, and prompts the user for a new choice.

The image filters

Each image filter is a function which takes one argument, the image matrix. The function returns a new image matrix which results from applying the desired filter to the image given in the argument. The filters all have the same basic algorithm:

```
for each row r in the image matrix
    for each column c in row r in the image matrix
        assign a value to the pixel at location (r, c) in the
new image based on the filter
```

The `negative` filter

The `negative` filter creates a "negative" image: black pixels become white and white pixels become black. To achieve this effect, each pixel's value is subtracted from 255.

The `negative` filter

The `negative` filter creates a "negative" image: black pixels become white and white pixels become black. To achieve this effect, each pixel's value is subtracted from 255.

The `sharpen` filter

The `sharpen` filter makes an image "sharper" by make dark pixels darker and light pixels lighter. More precisely, the pixel values in the new image are determined as follows: if the pixel value in the original image is less than 128, the pixel value in the new image is obtained by subtracting 16 from the pixel value in the original image; otherwise, the pixel value in the new image is obtained by adding 16 to the pixel value in the original image.

The `posterize` filter

The `posterize` filter creates a poster-like effect by reducing the number of possible pixel values to 4 (i.e., black, white, light gray, and dark gray) using the following conversion table:

original pixel value	new pixel value
0 - 71	0
72 - 127	72
128 - 183	183
184 - 255	255

The `blur` filter

The `blur` filter creates a blurred image. A pixel value in the new image is obtained by taking an average (mean) of the pixels around the original image. More specifically, to find the pixel value at row r, column c in the blurred image, find the average of the pixel values in the original image in the 5 x 5 square matrix centered on the pixel at (r, c), rounded to the nearest integer. Note that around the edges of the matrix, it will not be possible to have an entire 5 x 5 square centered on a given pixel; for those pixels, you will find the average of as much of the 5 x 5 square as exists within the image matrix.

For example, suppose our image has the following pixel matrix:

```
10   10   80   80  200  200
10   10   80   80  200  200
30   30  120  120  220  220
30   30  120  120  220  220
50   50  160  160  240  240
50   50  160  160  240  240
```

Then, the value of the pixel in the blurred image at row 3, column 4 would be 136, which is computed by taking the average of the pixels shown in **bold red** below:

10	**10**	**80**	**80**	**200**	**200**
10	**10**	**80**	**80**	**200**	**200**
30	**30**	**120**	**120**	**220**	**220**
30	**30**	**120**	**120**	**220**	**220**
50	**50**	**160**	**160**	**240**	**240**
50	50	160	160	240	240

and the value of the pixel in the blurred image at row 1, column 2 would be 55, which is computed by taking the average of the pixels shown in **bold blue** below:

10	**10**	**80**	**80**	200	200
10	**10**	**80**	**80**	200	200
30	**30**	**120**	**120**	220	220
30	30	120	120	220	220
50	50	160	160	240	240
50	50	160	160	240	240

Hints:

- To implement the calculation of the average, think about creating a doubly-nested `for`-loop to iterate over a 5x5 square centered a pixel (r, c) (without worrying about whether the entire square is inside the image or not). Then, inside the body of the nested `for`-loops, check to see if the location being considered in that iteration is inside the image matrix; if it is, use that pixel in the calculation of the average (and if it isn't, just ignore that location).
- Remember that the average = (sum of pixel values)/(number of values in sum). Since not every square is entirely within the bounds of the image matrix, the number of values in the sum will not always be 25.

The mirror filter

The `mirror` filter creates a mirror image of the original image. You can get up to 3 points in extra credit by writing a `mirror` filter and adding a menu option `7` to your main program.

HW_6.m

```
% Пороскун Олени ПМ-81
% Homework 6

menu()
M = input('Select a number and enter it: ');
i=0;
while (i == 0)
    switch M
        case 1
            disp('You choose 1 - new image');
            filename = input('Enter name of file: ', 's');
            image = imread(filename, 'jpg');
        case 2
            disp('You choose 2 - original');
            figure;   imshow(image);
        case 3
            disp('You choose 3 - negative filter');
            negative(image);
        case 4
            disp('You choose 4 - sharpen filter');
            sharpen(image);
        case 5
            disp('You choose 5 - posterize filter');
            posterize(image);
        case 6
            disp('You choose 6 - blur filter');
            blur(image);
        case 7
            disp('You choose 7 - mirror filter');
            mirror(image)
        case 0
            break;
        otherwise
            disp(' ');
            disp('invalid selection');
    end
    disp(' ');
    menu();
    M = input('Select a number and enter it: ');
end

```

HW_6.m

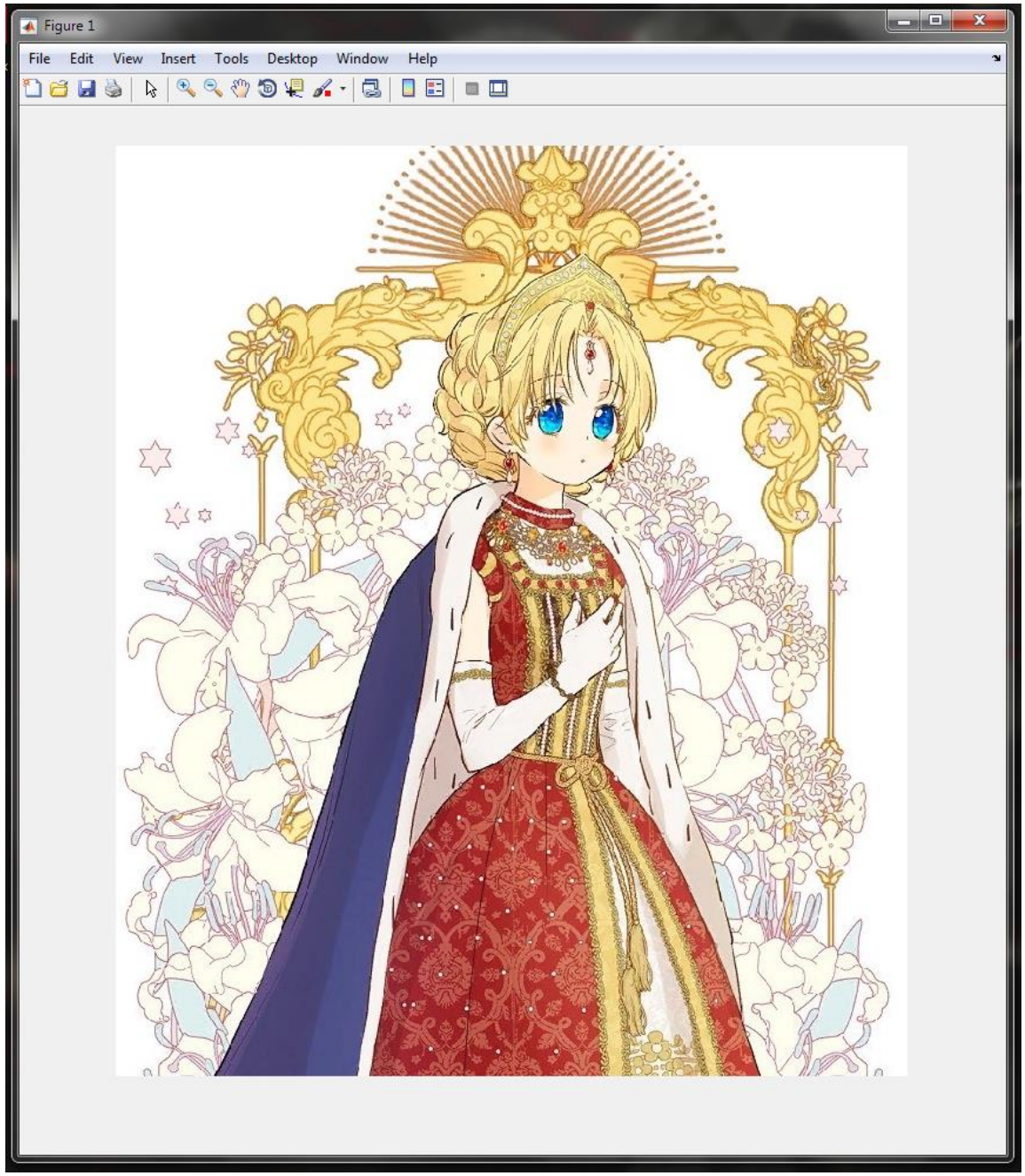
```
function menu()
disp('A menu of choices:');
disp('1. new image');
disp('2. original');
disp('3. negative');
disp('4. sharpen');
disp('5. posterize');
disp('6. blur');
disp('7. mirror');
disp('Enter 0 to quit.');
end

function negative(image)
image = 255 - image;
figure;  imshow(image)
end

function sharpen(image)
size_image = size(image);
rows = size_image(1);
columns = size_image(2);

for i = 1:rows
    for j = 1:columns
        if (image(i,j) < 128)
            image(i,j,:) = image(i,j, :) - 16;
        else
            image(i,j,:) = image(i,j, :) + 16;
        end
    end
end
figure;  imshow(image)
end

```

HW_6.m

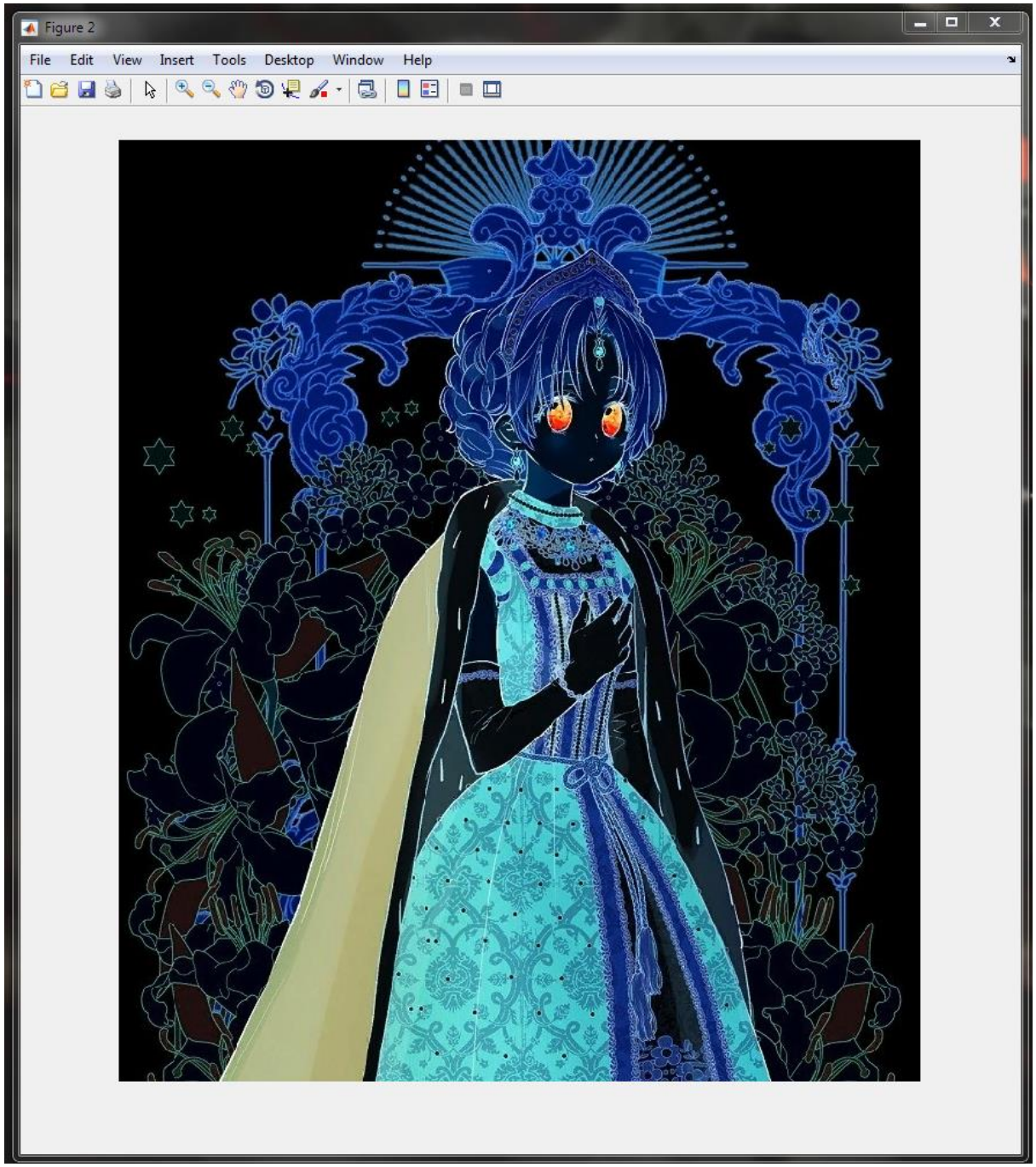
```
function posterize(image)
size_image = size(image);
for i = 1:size_image(1)
    for j = 1:size_image(2)
        if (image(i,j) >= 0  && image(i,j) <= 71)
            image(i,j,:) = 0;
        end
        if (image(i,j) >= 72 && image(i,j) <= 127)
            image(i,j,:) = 72;
        end
        if (image(i,j) >= 128 && image(i,j) <= 183)
            image(i,j,:) = 183;
        end
        if (image(i,j) >= 184 && image(i,j) <= 255)
            image(i,j,:) = 255;
        end
    end
end
figure;  imshow(image)
end

function blur(image)
[row, col] = size(image);
for i = 1+2:row-2
    for j = 1+2:col-2
        im = image(i-2:i+2, j-2:j+2);
        image(i,j) = sum(im(:))/25;
    end
end
figure;  imshow(image)
end

function mirror(image)
size_image = size(image);
rows = size_image(1);
columns = size_image(2);

med_c = ceil(columns/2);
for i = 1:rows
    for j = 1:med_c
        key = image(i, j, :);
        image(i, j,:) = image(i, columns-j+1, :);
        image(i, columns-j+1, :) = key;
    end
end
figure;  imshow(image)

med_r = ceil(rows/2);
for i = 1:med_r
    for j = 1:columns
        key = image(i, j, :);
        image(i, j,:) = image(rows-i+1, j, :);
        image(rows-i+1, j, :) = key;
    end
end
figure;  imshow(image)
end
```

```
>> HW_6
A menu of choices:
1. new image
2. original
3. negative
4. sharpen
5. posterize
6. blur
7. mirror
Enter 0 to quit.
Select a number and enter it: 1
You choose 1 - new image
Enter name of file: Manhwa

A menu of choices:
1. new image
2. original
3. negative
4. sharpen
5. posterize
6. blur
7. mirror
Enter 0 to quit.
Select a number and enter it: 2
You choose 2 - original
```

Original:

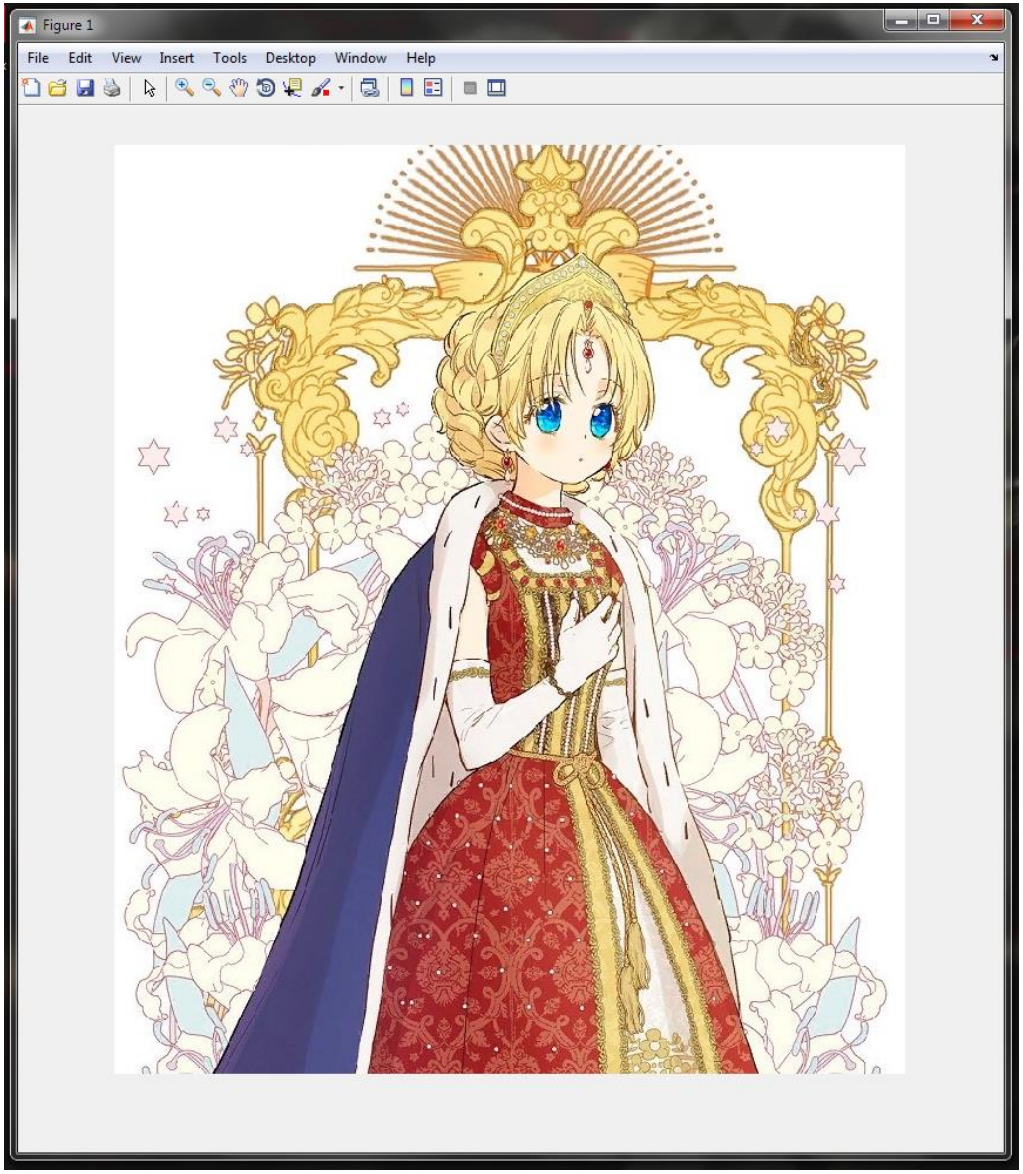


Negative filter:

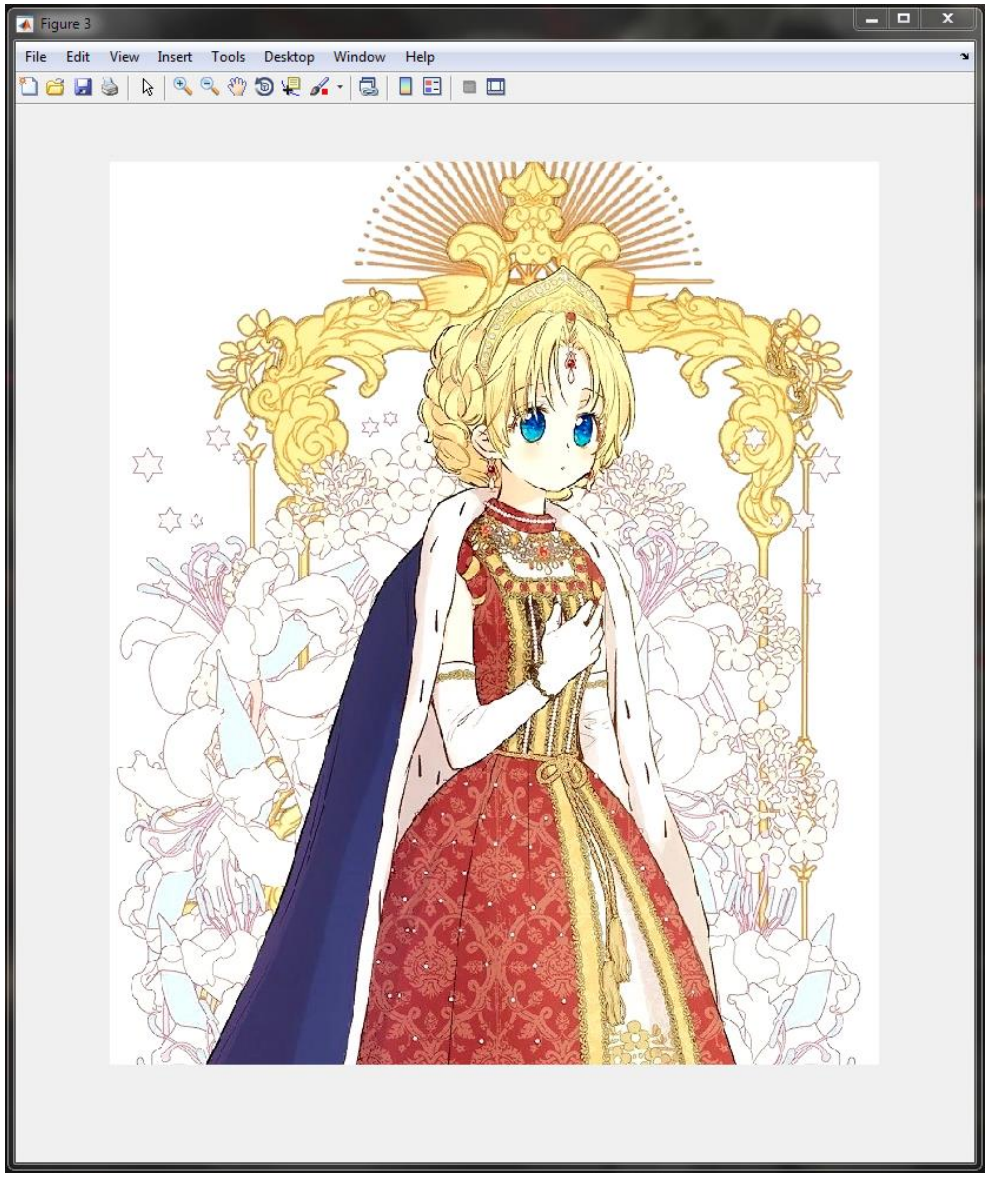



```
You choose 2 - original

A menu of choices:
1. new image
2. original
3. negative
4. sharpen
5. posterize
6. blur
7. mirror
Enter 0 to quit.
Select a number and enter it: 3
You choose 3 - negative filter

A menu of choices:
1. new image
2. original
3. negative
4. sharpen
5. posterize
6. blur
7. mirror
Enter 0 to quit.
Select a number and enter it: 4
You choose 4 - sharpen filter
```

Original:



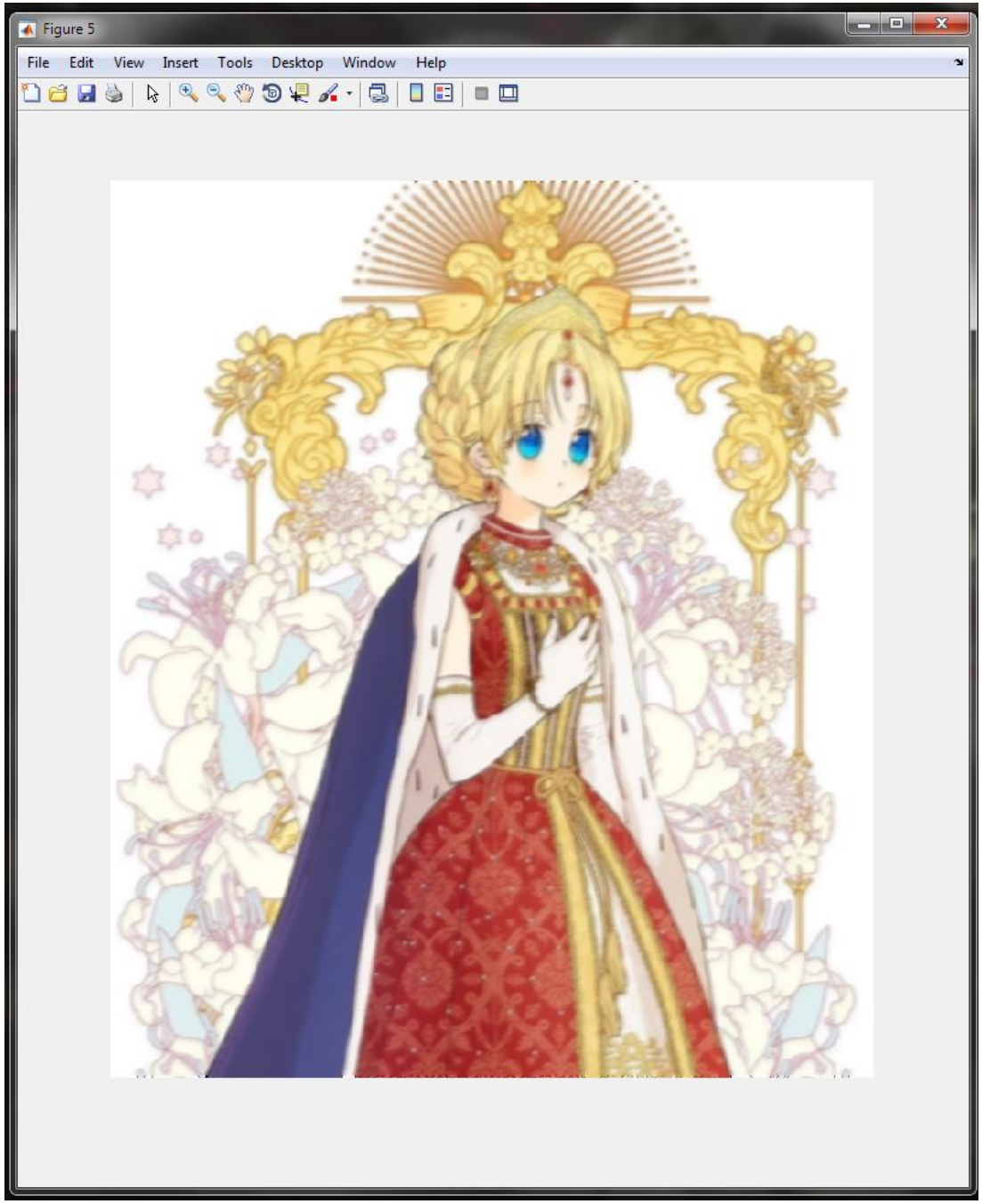
Sharpen filter:



Pasterize filter:

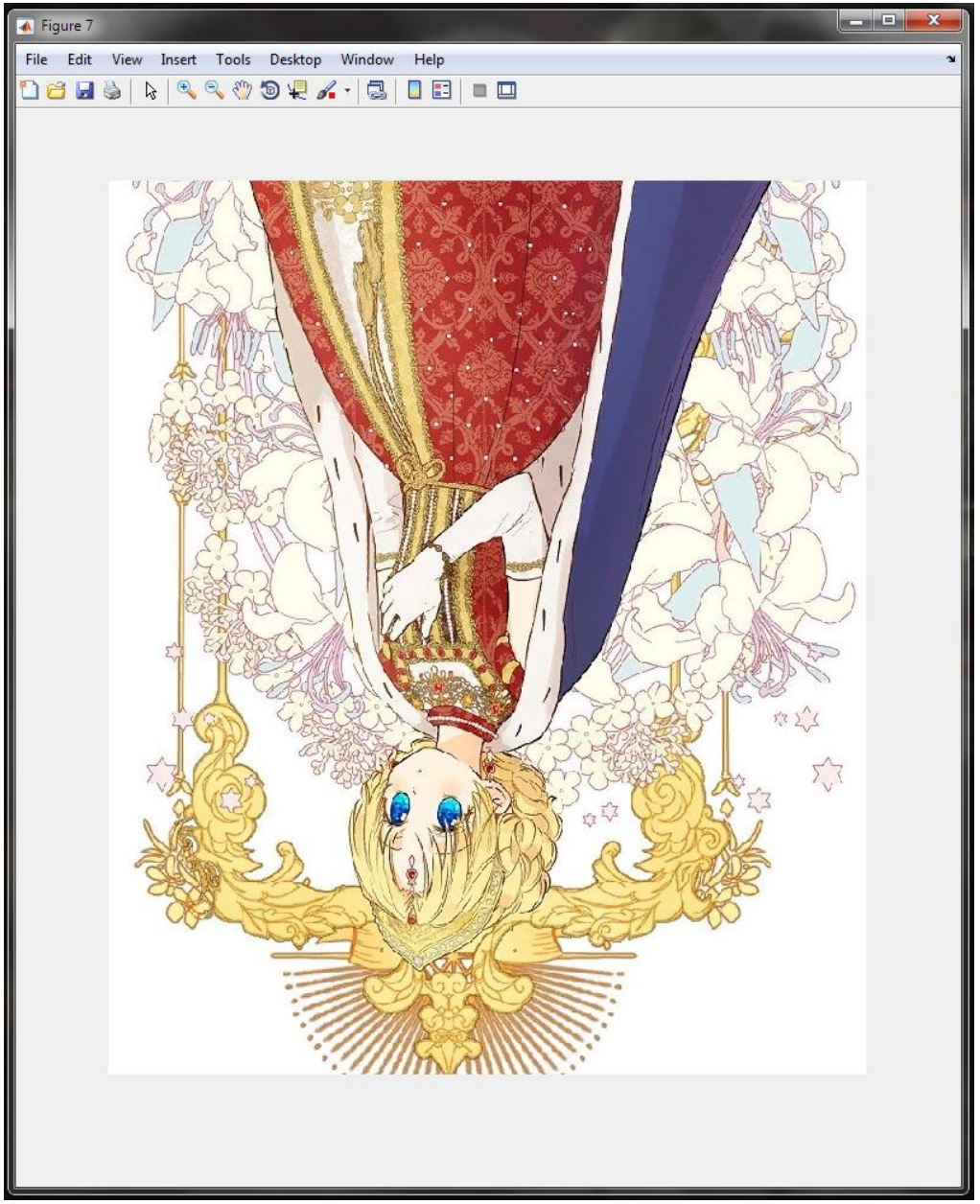

```
You choose 4 - sharpen filter

A menu of choices:
1. new image
2. original
3. negative
4. sharpen
5. posterize
6. blur
7. mirror
Enter 0 to quit.
Select a number and enter it: 5
You choose 5 - posterize filter
```

Blur filter:




```
You choose 5 - posterize filter

A menu of choices:
1. new image
2. original
3. negative
4. sharpen
5. posterize
6. blur
7. mirror
Enter 0 to quit.
Select a number and enter it: 6
You choose 6 - blur filter

A menu of choices:
1. new image
2. original
3. negative
4. sharpen
5. posterize
6. blur
7. mirror
Enter 0 to quit.
Select a number and enter it: 7
You choose 7 - mirror filter
```

Mirror filter:



```
You choose 7 - mirror filter

A menu of choices:
1. new image
2. original
3. negative
4. sharpen
5. posterize
6. blur
7. mirror
Enter 0 to quit.
Select a number and enter it: 464564

invalid selection

A menu of choices:
1. new image
2. original
3. negative
4. sharpen
5. posterize
6. blur
7. mirror
Enter 0 to quit.
Select a number and enter it: 0
>>
```