Homework 3 Solutions

1-3. The following script solves problems 1-3:

```
% shortProblems

% problem 1
disp('Problem 1');
% make the matrix and right hand side vector, then solve using \
mat=[3 6 4;1 5 0;0 7 7];
y=[1;2;3];
abc=mat\y;

% check that the solution is correct, if it is, this should be close to 0
errorVector=mat*abc-y


% problem 2
disp('--------------------')
disp('Problem 2');
% make the x vector
x=linspace(0,5,1000);
approximate=trapz(x,x.*exp(-x/3));
actual=-24*exp(-5/3)+9;
disp(['Difference between approximate and actual integral is ' ...
    num2str(abs(approximate-actual))]);


% problem 3
disp('--------------------')
disp('Problem 3')
temp=[1 2;3 4];
i=inv(temp)
identity=i*temp
```

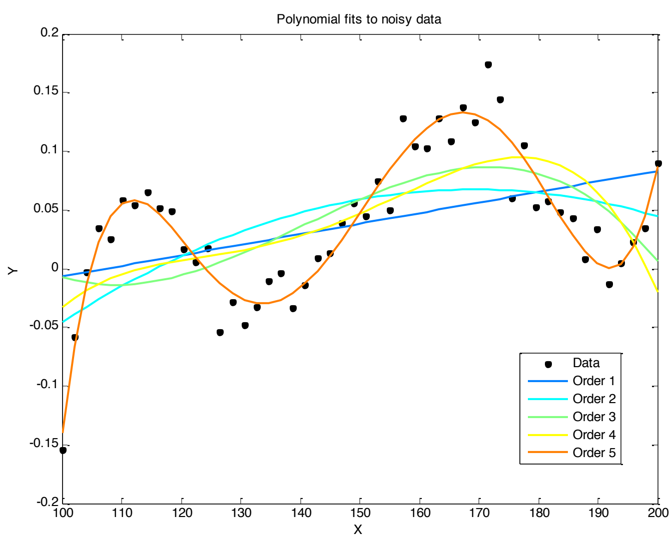
The script displays the following to the screen:

```
>> shortProblems
Problem 1

errorVector =

  1.0e-015 *

   -0.1110
         0
         0

-------------------
Problem 2
Difference between approximate and actual integral is 2.3504e-006
-------------------
Problem 3

i =

   -2.0000    1.0000
    1.5000   -0.5000


identity =

    1.0000         0
    0.0000    1.0000
```

4. The polynomial fitting script is pasted below

```
% polynomialFitting

% load the data
load randomData

% make a figure and plot the data
figure
plot(x,y,'k.','markersize',15);
hold on

% fit polynomials of various degrees and plot them on the figure
degree=1:5;
colors=jet(5);
for n=1:length(degree)
    % fit the data, centering and scaling it
    [P,S,MU]=polyfit(x,y,degree(n));

    % plot the data
    plot(x,polyval(P,x,S,MU),'color',colors(n,:),'linewidth',2);
end

% add labels
xlabel('X');
ylabel('Y');
title('Polynomial fits to noisy data');
legend('Data','Order 1','Order 2','Order 3','Order 4','Order 5');
```

This script generates the figure below:



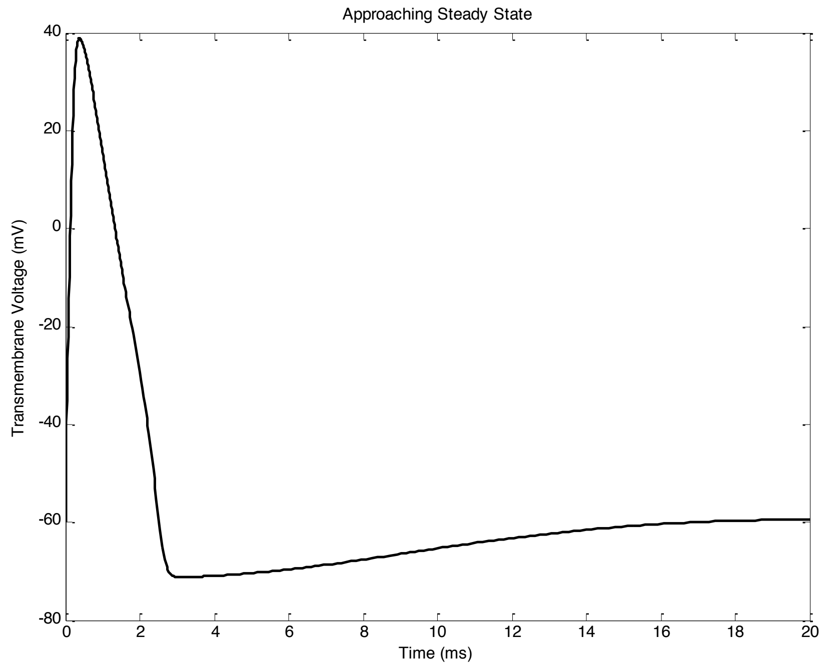
5. The HHODE.m file is pasted below:

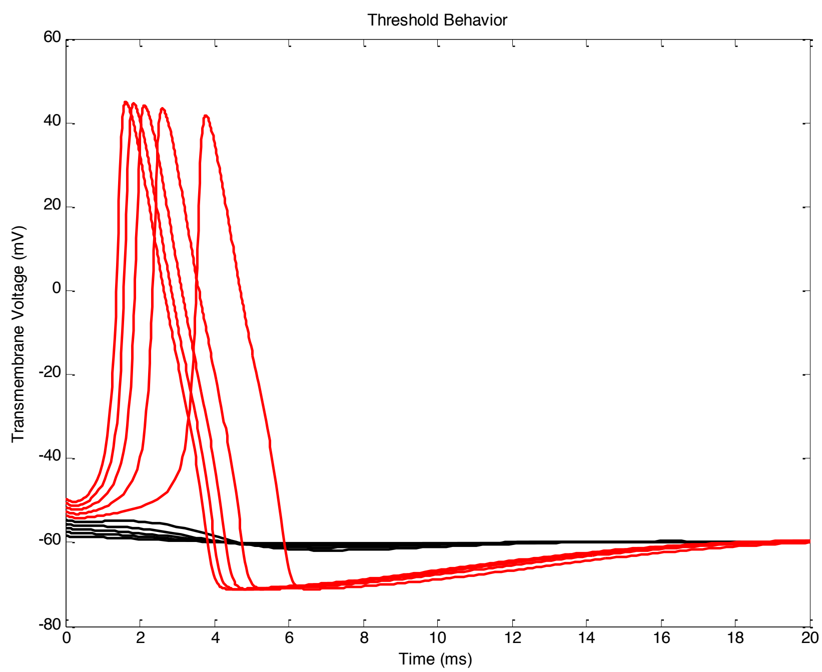
```
% dydt=HHODE(t,y)
%
% ode file for the hodgkin-huxley equations. the input vector y contains
% the state variables n,m,h,V in that order
function dydt=HHODE(t,y)

% define the constants
C=1;
gK=36;
gNa=120;
gL=0.3;
EK=-72;
ENa=55;
EL=-49.4;

% pull out the current values of the variables
n=y(1);
m=y(2);
h=y(3);
V=y(4);

% evaluate the derivatives
dndt=(1-n)*alphan(V)-n*betan(V);
dmdt=(1-m)*alpham(V)-m*betam(V);
dhdt=(1-h)*alphah(V)-h*betah(V);
dVdt=-1/C*(gK*n^4*(V-EK)+gNa*m^3*h*(V-ENa)+gL*(V-EL));

% put the derivatives in a column vector
dydt=[dndt;dmdt;dhdt;dVdt];
```

Below is the script HH.m, which solves the differential equations in HHODE a few times and plots the results.

```
% HH
%
% solves the Hodgkin Huxley equations in file HHODE and plots the result

% run the simulation to get steady state values
[t,y]=ode45('HHODE',[0 20],[.5 .5 .5 -60]);
ySS=y(end,:);

% plot it going to steady state
figure
plot(t,y(:,end),'k','linewidth',1.5);
xlabel('Time (ms)');
ylabel('Transmembrane Voltage (mV)');
title('Approaching Steady State');

% displace the initial voltage to find the threshold
figure
for n=1:10
    % run the simulation, each time incrementing the initial voltage by n mV
    [t,y]=ode45('HHODE',[0 20],ySS+[0 0 0 n]);
    v=y(:,end);
    if max(v)>0 % if peak V >0, plot with red line
        plot(t,v,'r','linewidth',1.5);
    else % if didn't surpass threshold, plot with black line
        plot(t,v,'k','linewidth',1.5);
    end
    hold on;
end
xlabel('Time (ms)');
ylabel('Transmembrane Voltage (mV)');
title('Threshold Behavior');
```

The figures generated by HH.m are below:





6. a) The `escapeVelocity` function is pasted below:

```
% n=escapeVelocity(z0,c,N)
%
% calculates the escape velocity of a julia set with parameters z0 and c. N
% is the maximum number of allowed iterations (if abs(z_n) does not surpass
% 2 for n<N, N is returned.
%
% inputs:
% z0 - julia set parameter z0 (complex number)
% c - julia set parameter c (complex number)
% N - max number of iterations of z_n+1=z(n)^2+c
%
% output:
% n - the escape velocity of the given z0 and c

function n=escapeVelocity(z0,c,N)

n=0; % initialize counter
z=z0; % initialize z_n
% iterate the loop until you reach N or abs(z)>2

while abs(z)<=2 && n<N
    % calculate the next z
    z=z^2+c;
    % increment counter
    n=n+1;
end
```

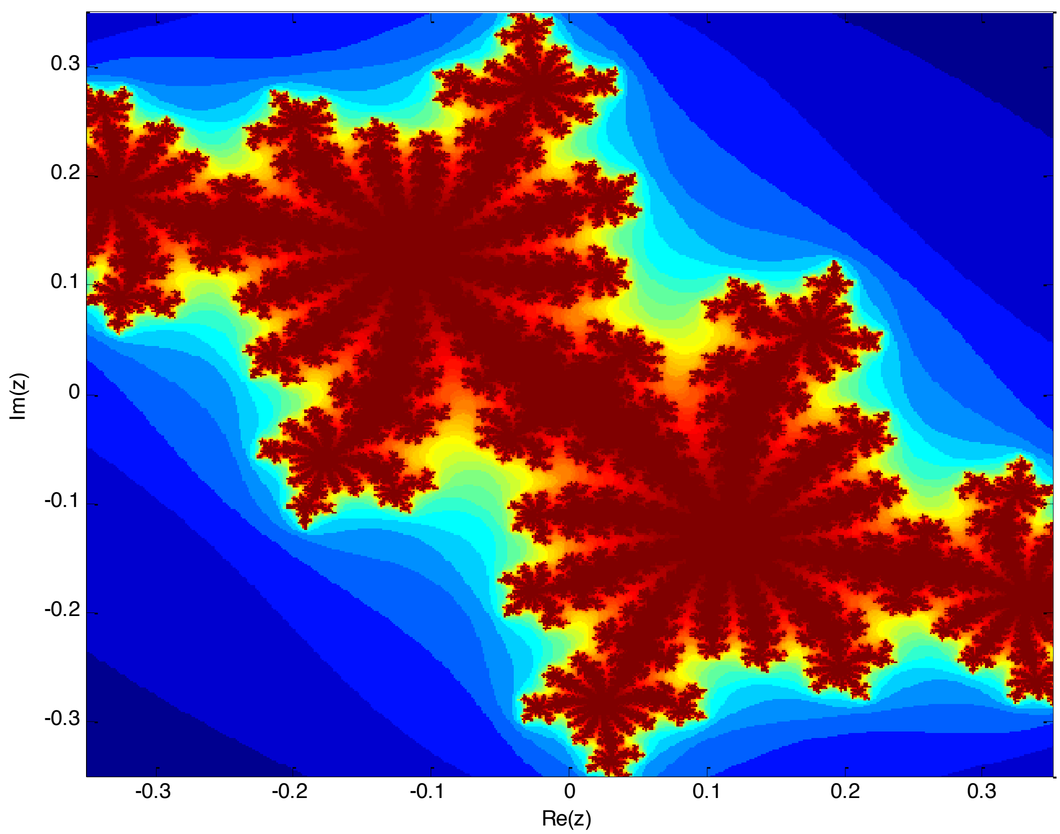
b) The `julia` function is pasted below:

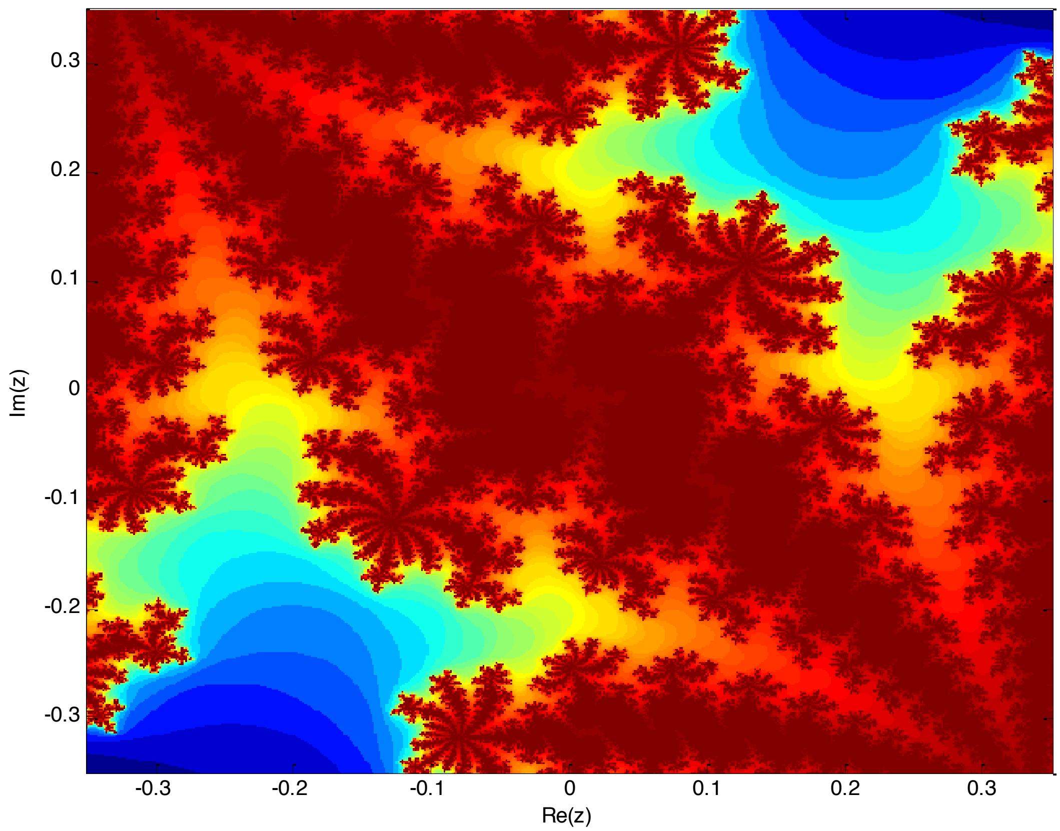
```
% M=julia(zMax,c,N)
%
% returns the matrix of escape velocities for a grid of complex numbers
% with real and imaginary value between -zMax and zMax (500 values)
%
% inputs:
% zMax - the complex numbers for which escape velocities are computed will
%        be 500 values between -zMax and zMax for both real and imaginary
%        part
% c - the complex number c, which is a parameter in the Julia set
% N - the maximum number of iterations to use when computing escape
%        velocities
%
% output:
% M - matrix of escape velocities for the 500x500 imaginary numbers with
%        real and imaginary part between -zMax and zMax
%
% EXAMPLE
%            M=julia(.35, -.297491+i*0.641051,250);
%            figure; x=linspace(-.35,.35,500);
%            imagesc(x,x,atan(.1*M)); axis xy
%            xlabel('Re(z)'); ylabel('Im(z)');

function M=julia(zMax,c,N)

% make the complex grid
temp=linspace(-zMax,zMax,500);
[R,I]=meshgrid(temp,temp);
Z=R+1i*I;

% initialize an empty M
M=zeros(size(Z));

% loop through all the Z's and calculate escape velocity
for row=1:size(Z,1)
    for col=1:size(Z,2)
        % pull out the current z
        z0=Z(row,col);
        % calculate escape velocity
        velocity=escapeVelocity(z0,c,N);
        % store escape velocity in M
        M(row,col)=velocity;
    end
end
```

This script generates two Julia Sets

```
% runJulia
% runs the julia set with a nice value of z0 and c and plots the images

% make one figure
M=julia(1, -.297491+i*0.641051,100);
figure; x=linspace(-.35,.35,500);
imagesc(x,x,atan(.1*M)); axis xy
xlabel('Re(z)'); ylabel('Im(z)');

% make the zoomed in version
M=julia(.35, -.297491+i*0.641051,250);
figure; x=linspace(-.35,.35,500);
imagesc(x,x,atan(.1*M)); axis xy
xlabel('Re(z)'); ylabel('Im(z)');
```

The figures below are generated by the above script:




0.3
0.2
0.1
0
-0.1
-0.2
-0.3
Im(z)
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
Re(z)