Homework 3

This homework is designed to give you practice writing functions to solve problems. The problems in this homework are very common and you will surely encounter similar ones in your research or future classes. As before, the names of helpful functions are provided in **bold** where needed. **Homework must be submitted 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(x) or *.pdf file.

Keep all your code in scripts/functions. If a specific name is not mentioned in the problem statement, you can choose your own script names.

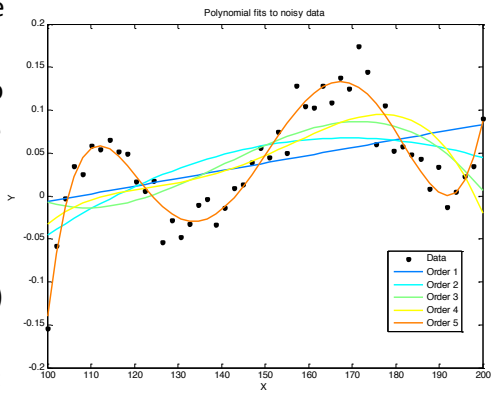
1. **Linear system of equations.** Solve the following system of equations using ****. Compute and display the error vector

$$\begin{aligned} 3a + 6b + 4c &= 1 \\ a + 5b &= 2 \\ 7b + 7c &= 3 \end{aligned}$$

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: $-24e^{-5/3} + 9$.

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.

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:



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

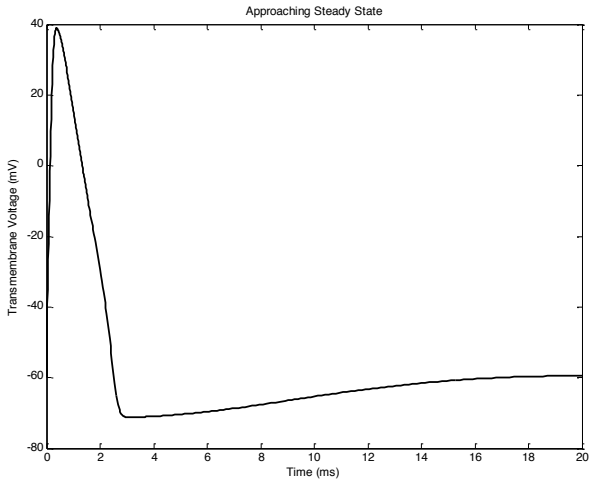
$$\frac{dn}{dt} = (1-n)\alpha_n(V) - n\beta_n(V)$$
$$\frac{dm}{dt} = (1-m)\alpha_m(V) - m\beta_m(V)$$
$$\frac{dh}{dt} = (1-h)\alpha_h(V) - h\beta_h(V)$$
$$\frac{dV}{dt} = -\frac{1}{C}\left(G_K n^4 (V-E_K) + G_{Na} m^3 h (V-E_{Na}) + G_L (V-E_L)\right)$$

    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):

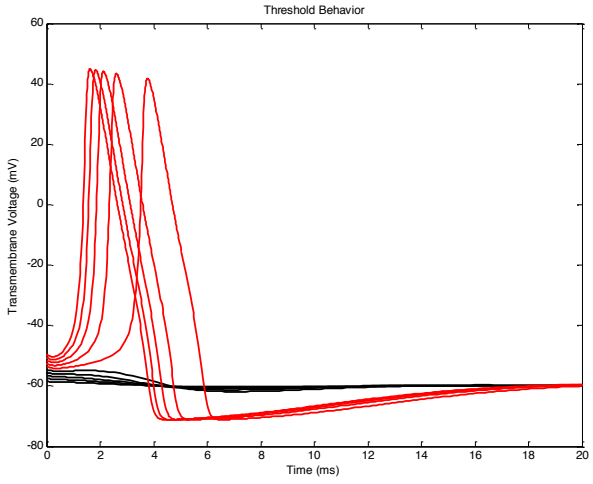
$$C = 1$$
$$G_K = 36$$
$$G_{Na} = 120$$
$$G_L = 0.3$$
$$E_K = -72$$
$$E_{Na} = 55$$
$$E_L = -49.4$$

    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).



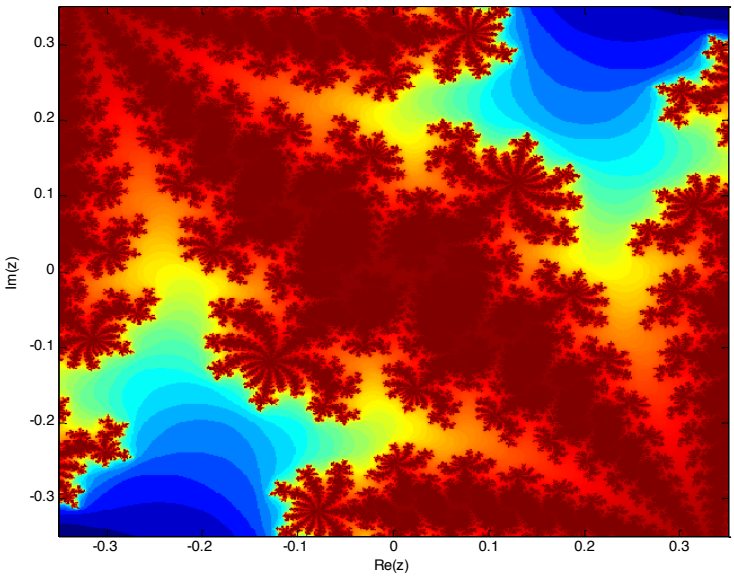
Optional Problem

6. **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 $z_1, z_2, z_3 \mathrm{K}$ 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

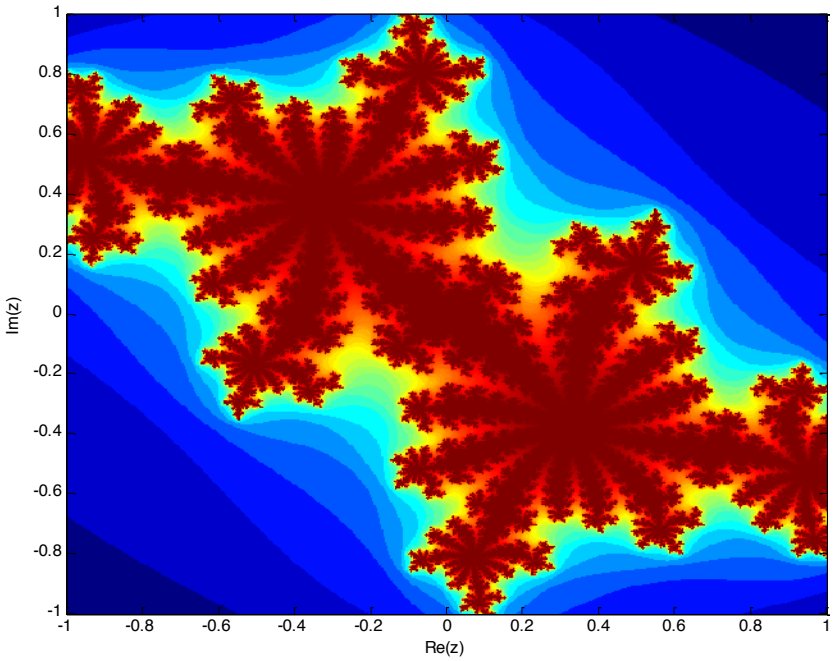
b. To generate the filled in Julia Set, write the following function `M=julia(zMax,c,N)`. `zMax` will be the maximum of the imaginary and complex parts of the various z_0's for which we will compute escape velocities. `c` and `N` are the same as defined above, and M is the matrix that contains the escape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