**Warning: This homework is long, do not wait until the last minute to start or you will not finish. Also, given its length, it is probably a good idea to `clear all` between exercises to free up memory**

## Exercise 1: A Chaotic ODE

The following three-dimensional autonomous chaotic ODE was proposed and analyzed in a recent paper, ***Analysis of a novel three-dimensional chaotic system*** (Li et al., 2013):

$$\begin{cases} \dot{x} &= -ax + fyz \\ \dot{y} &= cy - dxz \\ \dot{z} &= -bz + ey^2 \end{cases}$$

Where $x, y, z$ are the state variables, and $a, b, c, d, e, f$ are positive constant parameters. You will now model this system, reproducing a few of their results (the paper is avaliable at `http://www.sciencedirect.com/science/article/pii/S0030402612002823?np=y`, you should check it out). Much of this work will parallel that done in video lecture 23.

(a) You will begin by creating a cube of initial values. The cube should be oriented in the natural manner; that is, with edges parallel the $x$, $y$, and $z$ axes. Each edge should be .4 long. The spacing between grid points should be .05. The center of the cube is at $(x, y, z) = (.3, 2, 12)$. Using `meshgrid()` create three three-dimensional arrays containing the x-coordinates, y-coordinates, and z-coordinates, respectively. To avoid dealing with high dimensional arrays (as is done in the online video lectures), rescale each of these matrices as a vector (a matrix, `A` , can be rescaled as a vector by `V = A(:)`). Store the resultant vectors as the rows in a three row matrix with the x, y, and z coordinate vectors in the 1st, 2nd, and 3rd rows, respectively. Save the matrix as `A1.dat`.
To check you work, try the command `plot3(x,y,z,'.')` and verify that you have a cube of points spaced as described above.

(b) Let $a = 16,\ b = 5,\ c = 10,\ d = 6,\ e = 18,\ f = .05$. Model the above system with Runge-Kutta 4 (you can reuse some of your code from Homework 4, Problem 3), from time 0 to time 7, using a time step $\Delta t = .01$, and with the initial condition cube created in (a). Structure your solution in the same way as your initial condition, however you should now have a 3-dimensional array (size = `701, 3, 729`). If your output is called `A`, then `A(1,:,:)` will be your initial condition cube, `A(2,:,:)` will correspond to the positions of those initial points after $\Delta t$ seconds, and so on. Be sure to properly vectorize your code, or it will run extremely slowly (and will probably terminate prior to completion on scorelator). Once completed, use the `squeeze()` function to make the 3-D array of solutions into 3 2-D matrices containing the x, y, or z coordinates at the different time steps as the rows of these matrices. Type `help squeeze` to learn how to use this command.

Save the matrix of x-coordinates as `A2.dat`, the matrix of y-coordinates as `A3.dat`, and the matrix of z coordinates as `A4.dat`. The size of each of these should be $701 \times 729$ (the $x$, $y$, and $z$ coordinates of the 729 points from time 0 to time 7 in steps of 0.01).

(c) You will not submit anything more for this problem, but you should construct some cool plots! Plot your solutions at the last time using the code:
`plot3(A2(end,:),A3(end,:),A4(end,:),'.');` where `A2`, `A3`, and `A4` are what you created in part b). Verify that your solution looks similar to the plots of the system found in the paper referenced above (this is also a good way to help you troubleshoot if things are not working out).
Another interesting plot you should make is of the tracks of a few of your initial positions:
`plot3(A2(:,1:5),A3(:,1:5),A4(:,1:5));`
You can increase the number of tracks you plot (you have one for each of the $9^3$ starting positions, but you only), but note that this will quickly become quite taxing on your computer.
Finally, try watching the full evolution of the system by typing:

```
for i=1:length(t)
    plot3(A2(i,:),A3(i,:),A4(i,:),'.');
    axis([-.5,.5,-10,10,0,50])
    drawnow
    pause(h)
end
```

Make sure to comment out your plots before submitting the assignment.

## Exercise 2: A Chaotic Map

A map is a system which progresses in discrete steps according to a function $f$, such that we have $x_{n+1} = f(x_n)$ (this is opposed to the ODEs we have studied, which advance continuously according to a differential equation). In this class, we have seen maps in the context of solving ODE's, where we have approximated the solution to the continuous systems with similarly constructed maps (such as the forward Euler or Runge Kutta methods). But maps are worth studying in their own right. In this problem we will be looking at the famous logistic map:

$$x_{n+1} = rx_n(1 - x_n)$$

Despite its simplicity, it exhibits complex and chaotic behavior.

a) With $x_0 = .7$ calculate 30 iterations of the logistic map where $r$ takes on the values 2.5, 3.2, 3.52, and 4. Store these as the columns of a $31 \times 4$ matrix ($x_0$ is included, so the first row of this matrix should be equal to 0.7 in every entry). Each column of the matrix corresponds to one of the $r$ values in the order given.
Save this matrix as `A5.dat`
Try plotting each of these solutions (make sure to comment this out before submitting). For $r = 2.5$, 3.2, and 3.52, you should notice that the solutions eventually begin to repeat themselves (in the language of dynamical systems, they "converge to a periodic orbit"). We will call the "period" of an orbit the number of iterations it takes to return to where it was before. For example, the sequence "1,2,1,2,1,..." would have period two, while the sequence "1,2,6,4,5,1,2,6,4,5,..." would have period five. Define $P_R$ as the period of the orbit the sequence of the logistic map approaches when $r = R$. Looking at the plots of your results, identify the periods when $r = 2.5$ (that is, $P_{2.5}$), when $r = 3.2$, and when $r = 3.52$. Save these in a column vector $[P_{2.5}; P_{3.2}; P_{3.52}]$ as `A6.dat`. Notice that with $r = 4$, the solution does not appear to have any regular behavior.

b) Now find the first 500 iterations of the map with $x_0 = .7$ under a whole range of r values: `r = 2.5:.001:4`. These should be stored in a $501 \times 1501$ matrix, in which each column represents the sequence from $x_0$ to $x_500$ for a different $r$. Make sure your code is properly vectorized (there should only be a single for loop).
Save the last 100 iterations (a $100 \times 1501$ matrix) in `A7.dat`.
To see the nature of the long-term solutions with respect to $r$, plot these last hundred iterations using this code:
`plot(R,A7,'k.','Markersize',1);`
You have made what is known as the bifurcation diagram of the logistic map, a celebrated image from the history of chaos theory. Notice how the map first becomes fractal-like, repeatedly splitting, before eventually giving rise to chaos as $r$ increases. Make sure to comment out the plot before submitting.

## Problem 3: The Heat Equation

We spent two weeks studying ordinary differential equations. This included initial value problems and boundary value problems. We will combine these both to numerically solve a partial differential equation: the heat equation. Consider a metal rod of length $L$. The temperature of the rod at a point $x$ (where $x$ is between 0 and $L$) is given by $u(x,t)$. That is, the temperature changes both along the rod, and with time! A physical model for *how* the temperature of the rod changes with time is the partial differential equation

$$\frac{\partial u}{\partial t} - \alpha^2 \frac{\partial^2 u}{\partial x^2} = 0$$

where $\alpha$ is a constant related to the heat conductance of the bar, and $\frac{\partial u}{\partial t}$ is simply a way of saying "the first derivative of $u$ with respect to time." In order to characterize the behavior, we need both boundary conditions and an initial condition.

For this problem, we assume the ends of the rod is held at a constant zero temperature. Mathematically, this translates to $u(0,t) = 0$ for all time and $u(L,t) = 0$ for all time. Think of it as if the ends of the rod are sitting in ice baths that are kept at a constant temperature, due to the constantly melting ice.

Finally, we need to have an initial condition. That is, we need to know what the temperature of the rod is everywhere along it at the starting time! Obviously, the ends are at zero, as they always are, but the initial temperature of the interior points must be provided as an initial function of $x$ $\phi(x)$. Thus, $u(x,0) = \phi(x)$ for all $x$. Putting all of this together gives us the full problem statement of the heat equation:

$$\begin{aligned}
&\frac{\partial u}{\partial t} - \alpha^2 \frac{\partial^2 u}{\partial x^2} = 0 && 0 < x < L, \quad 0 < t < \infty \\
&u(0,t) = 0 && 0 < t < \infty \\
&u(L,t) = 0 && 0 < t < \infty \\
&u(x,0) = \phi(x) && 0 \le x \le L
\end{aligned}$$

The main differential equation in the first line has two derivatives in it: a first derivative in time and a second derivative in space. We can discretize them as we learned to several weeks ago. Suppose we discretize time from 0 to some end time $T$ in steps of $\Delta t$, and we discretize space from 0 to $L$ in steps of $\Delta x$. We write the solution at position $x_i$ and time $t_n$ as $u(x_i, t_n)$. Let's use the forward difference for the time derivative and the centered difference for the space derivative:

$$\begin{aligned}
\frac{\partial u(x_i,t_n)}{\partial t} &\approx \frac{u(x_i,t_{n+1}) - u(x_i,t_n)}{\Delta t} \\
\frac{\partial^2 u(x_i,t_n)}{\partial^2 x} &\approx \frac{u(x_{i+1},t_n) - 2u(x_i,t_n) + u(x_{i-1},t_n)}{(\Delta x)^2}.
\end{aligned}$$

For every interior point (that is every $x_i$ that is not on either end), we can define an equation using these two approximations:

$$\frac{u(x_i,t_{n+1})-u(x_i,t_n)}{\Delta t}-\alpha^2\left(\frac{u(x_{i+1},t_n)-2u(x_i,t_n)+u(x_{i-1},t_n)}{(\Delta x)^2}\right)\approx 0.$$

Suppose we know the temperature all along the rod at time $t_i$ and we want to find what it is at that point the next time step $t_{n+1}$ (recall that this is $u(x_i,t_{n+1})$). We can solve the equation above for this unknown quantity:

$$u(x_i,t_{n+1})=\left(1-2\frac{\alpha^2\Delta t}{(\Delta x)^2}\right)u(x_i,t_n)+\frac{\alpha^2\Delta t}{(\Delta x)^2}(u(x_{i+1},t_n)+u(x_{i-1},t_n)). \tag{1}$$

The only complication arises near the boundaries, but recall that we fixed $u(0,t_n)=0$ and $u(L,t_n)=0$ for all $t_n$. Thus the equations we have for $u(x_1,t_{n+1})$ and $u(x_{L/\Delta x-1},t_{n+1})$ are:

$$u(x_1,t_{n+1})=\left(1-2\frac{\alpha^2\Delta t}{(\Delta x)^2}\right)u(x_1,t_n)+\frac{\alpha^2\Delta t}{(\Delta x)^2}u(x_2,t_n) \tag{2}$$

and

$$u(x_{L/\Delta x-1},t_{n+1})=\left(1-2\frac{\alpha^2\Delta t}{(\Delta x)^2}\right)u(x_{L/\Delta x},t_n)+\frac{\alpha^2\Delta t}{(\Delta x)^2}u(x_{L/\Delta x-1},t_n). \tag{3}$$

At a time $t_n$, define the vector $U_n$ the temperature at all of our grid points at that time. If we lop off the top and bottom entries, which are always zero, we are left with only the interior points. Call this snipped vector $U_n^{int}$:

$$U_n=\begin{bmatrix}u(0,t_n)\\u(x_1,t_n)\\\vdots\\u(x_{L/\Delta x},t_n)\\u(L,t_n)\end{bmatrix}=\begin{bmatrix}0\\u(x_1,t_n)\\\vdots\\u(x_{L/\Delta x},t_n)\\0\end{bmatrix},\qquad U_n^{int}=\begin{bmatrix}u(x_1,t_n)\\\vdots\\u(x_{L/\Delta x},t_n)\end{bmatrix}.$$

a) Using equations (1), (2), and (3) for each interior point $x_i$, we can find a matrix $M$ such that

$$U_{n+1}^{int} = MU_n^{int}.$$

You may need to sketch this out by hand to get an idea of how this will all work. You should find that the matrix $M$ has a particular tridiagonal form:

$$M = \begin{bmatrix} A & B & 0 & \cdots & 0 \\ B & A & B & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & \cdots & B & A \end{bmatrix}$$

Given $\Delta t = .004, \Delta x = .01, \alpha = .1, L = 1$ find $A$ and $B$, and find the matrix $M$. Save $A$ as `A8.dat`, $B$ as `A9.dat`, and $M$ as `A10.dat`.

b) Use `eig()` to find the eigenvalues of $M$. Find the absolute value of the eigenvalue with the largest magnitude. Save this value as `A11.dat`. Will the system be stable?

c) If we let the initial condition be $\phi(x) = \exp(-200(x - 0.5)^2)$, then $u(x_i, t_0) = \exp(-200(x_i - 0.5)^2)$. Calculate $U_0^{int}$ and save it as `A12.dat`. It should be a column vector that is $99 \times 1$.

d) Calculate the solutions $U_n^{int}$ between times 0 and 2, and save it in a matrix in which the columns are the $U_n^{int}$ at the different time steps. Then add a row of zeros at the top and bottom. These are there because the two endpoints always have value 0. With those rows appended, you have a matrix of $U_n$.
Save this matrix as `A13.dat` (it should have dimensions $101 \times 501$).

e) Watch your solution evolve with time using the code below (make sure to comment it out when before you submit)You do not need to submit anything more. Simply revel in the fact that you numerically solved a partial differential equation! Wow!

```
for j=1:(2/dt+1)
    plot(linspace(0,L,L/dx+1)',A13(:,j));
    axis([0,L,0,1]);
    pause(dt);
end
```

## Exercise 4: Facial Recognition

In this problem we'll improve upon Professor Kutz's facial recognition approach shown in video lecture 27. You have two options when starting this problem: you can either generate the matrices of picture data on your own or you can download the premade matrices. In order to make your own you will need the Image Processing Toolbox (it came with MATLAB if you bought the $100 version). If you are interested in processing the images on your own go to **Step 1**. If not, proceed to **Step 2**.

**Step 1**

Download the `make_face_dat.m` and `faces.zip` files from the course website. Unzip the latter into the same folder as `make_face_dat.m`. Run `make_face_dat.m`, it will make a file `imag_data.mat`, put this file in the same folder as your homework file. Proceed to **Step 3**

**Step 2**

Download the file `imag_data.mat` from the course website, and put it in the same folder as your homework file. Proceed to **Step 3**.

**Step 3**

Use `load imag_data.mat` to load a matrix `B` and a vector `u` into your workspace. You will **not** need to upload `imag_data.mat` to scorelator.

a) The columns of `B` each store that pixel measurements for a separate face, with columns 1 through 20 being pictures of Sylvester Stallone and columns 21 through 40 being pictures of Taylor Swift. Each picture is $200 \times 175$ pixels, but has been rewritten as a long column vector. These vectors have been placed side by side to form the matrix $B$. Taking the average over all these vectors will give you a vector which represents an "average" face (at least a Stallone/T-Swift average).
Save this column vector as `A14.dat` If you have the Image Processing Toolbox, you can see this spooky average face by using the command `imshow(uint8(reshape(A14,200,175)))`. Note that the vector had to be reshaped as a matrix in order to be displayed. Make sure to comment this out before you submit.

b) Since we are interested in the deviation from the average face, remove the average face from the vectorized image data (subtract the average face from each column of B). Call this new matrix `A`. Compute the reduced SVD of `A` using `svd()` with the `'economy'` flag.
Output the first 10 singular values of $A$ as a column vector and save it as `A15.dat`

c) Plot the singular values and note the decay. We can see from the values that there are more than one important directions, so we will use the first 3 principal directions: $\vec{u}_1, \vec{u}_2, \vec{u}_3$. Project the data in matrix `A` onto these three principal directions, where here we take the projection of a vector $v$ onto a vector $u$ to be $u^T v$. We can project all the data in a matrix $A$ onto a vector $u$ by computing $u^T A$. Store the projection

in the first principal direction as `x`, the second as `y`, and the third as `z`.
Output the row vector `x` as `A16.dat`.

d) Plot the first 20 columns of `x`,`y`,`z` using `plot3` and the flag `'bo'` to give blue circles. In the same figure plot the last 20 columns of `x`,`y`,`z` using `plot3` and the flag `'ro'` to give red circles. Notice that when the images are plotted in their principal-directions, the Stallone photos (blue dots) are mostly separated from the T-Swift photos (the red dots). We can use this to classify images.

e) We will now look at the unknown image `u`, and attempt to figure out who it is using the principal directions. Subtract the average face from the mystery image `u`, and then project the result onto the three principal directions.
Store the result as a column vector in `A17.dat`

Add this point to the previous plot with the flag `'gs'` to produce a green square, does it appear to be more in Stallone's blue region or T-Swift's red region? If you have the image processing toolbox you can check by displaying the image using `imshow(uint8(reshape(u,200,175)))`. Make sure to comment out all plots and images before submitting your assignment.

## 1 Problem 5: Noise reduction

In this problem we will attempt to use `fft` to filter noise out of an audio file. Download the file `noisy_message.wav` to the same folder as your homework file. You will **not** need to upload `noisy_message.wav` to Scorelator. To load `noisy_message.wav` use the code `[V,fr] = wavread('noisy_message.wav');`, where `V` is the audio data and `fr` is the frame rate. Throughout this exercise, you can use `sound(V,fr)` to listen to an audio file, but make sure to comment out any sounds before you submit to Scorelater.

a) Listen to `noisy_message.wav`. Since the signal has white noise uniformly distributed across all frequencies, the hope is filtering out the frequencies of the signal that are lower magnitudewill help to clean it up. Take the fast Fourier transform (`fft`) of `V` to convert the audio data to its frequency spectrum.
Save the first 1000 values of the **magnitude** (absolute value) of the frequency spectrum as a column vector in `A18.dat`.

b) Plot the magnitude of the frequencies and note the thick band across the bottom of the plot. This is largely noise. Filter out any frequency whose magnitude is less than 50. When we say "filter out" we mean to set those frequencies to zero.
Save the first 1000 values of the absolute value of the filtered frequency spectrum as A19.dat
Use the inverse FFT (`ifft`) to convert this back to audio. Listening to the audio, it should start to sound more familiar, but there is still a lot of noise and a some degradation to the original signal.
Save the first 1000 values of the filtered audio signal as a column vector in `A20.dat`

c) We will try another method. Break the original noisy audio data into 8 equal lengths. Take the FFT of each of these lengths; filter out frequencies whose magnitude is below $3.2\times$(average magnitude in the section); take the inverse FFT of each section, and combine the filtered audio sections into a vector of audio data equal to the original in length. By adapting the filter to each region, we will hopefully do a better job of filtering. If you listen to the audio, it seems we have removed more noise, but in the process has created many artifacts.
Save the first 1000 values of this filtered audio signal as a column vector in `A21.dat`. Which version sounds the best?