Math 350 Final Exam

Instructions: This is a take home final exam. You may use your notes and anything on our class website to assist you, **but you may not use anything else on the internet**. You may also use the help files available in Matlab. **You may work in groups of two.** If you have questions, you may ask me (I may not be able to answer fully, but it is better to ask me than not to know).

For each problem below, upload the script or function file you created, together with a screen shot of the output. Alternatively, you may "publish" your work and upload that- In either event, I want to see your code **and** what your code produced.

Due: The deadline will be at 11:59PM on Monday, May 15th.

No late solutions will be accepted, so start early!

Problem 1: Linear System

We want to analyze the matrix equation $A\mathbf{x} = \mathbf{b}$, where A and $\mathbf{b}$ are provided in the file `FinalExamData01.mat`. In this case, A is a "wide" matrix, so we run into the problem of not only having an exact solution, but since we have free variables, any least squares solution we get should have an infinite number of possibilities.

You may answer the questions in your Matlab script file. Turn in the script file and give a screen shot of the output (or publish the script).

1. Using the SVD, compute a least squares solution, $\hat{x}$ by creating the pseudoinverse of A. You're going to need to decide on the rank of the matrix.
2. Show (using the SVD) that $\hat{x}$ is in the row space of A. Similarly, show that $A\hat{x}$ is in the column space of A.
3. We said that there were an infinite number of possible solutions- Use the SVD to come up with two more (distinct, linearly independent) solutions, and show that they're correct by computing $A\hat{x}$ for each of your answers.

Problem 2: Optimization

Consider the function

$$F(x_1, x_2) = x_1^4 + x_1 x_2 + (1 + x_2)^2$$

We wish to approximate the minimum, and we are currently at the point $(3/4, -5/4)$.

1. Use 10 iterations of gradient descent with an appropriate step-size.
2. Compute 4 iterations of Newton's method (starting over again).
3. Make an observation about which gives a better approximation.

(Links for problems like these from class are found mainly on Friday of Week 9, but there is information in Week 10 as well).

Turn in your script file and a screen shot of the computer output (or publish the script file).

Problem 3: A k-NN classifier on mushroom data

The data will be measurements taken from mushrooms. We have 22 measurements taken from 4062 mushrooms, and we want to classify them as edible or poisonous.

The dataset `mushrooms.mat` is on the class website, and when you load the data, you'll see three sets: X is 22×4062 (so each "point" is a column, and there are 4062 of them). Matrix T is 2×4062. The column is either $[1, 0]^T$ or $[0, 1]^T$ for poisonous/edible (it doesn't matter which is which until we actually have to taste one).

We will create our own classifier using $k-$nearest neighbors. Remember that we have template files you can use from Monday of Week 12, and in particular, the file `knnapp2.m`. So build the classifier using $5-$fold cross validation to determine the best number of neighbors to use. Keep the training/test split percentages at 70-30, as it was already.

You might find Problem 3's template file useful to build the confusion matrix, since the mushroom data outputs a vector in $\mathbb{R}^2$ rather than a scalar value for the class.

What to turn in: Your modified `knnapp2.m` and a screenshot of the final confusion matrix.

Problem 4: A Feedforward Neural Network

We'll be using `main.m` from the Week 14 links as a template. In that template, we constructed a neural net from scratch to classify the iris data.

For this problem, we'll have data from 768 patients, and from each patient, we'll have 8 measurements. We want to predict if the patient has diabetes or not.

If you download `diabetes1.mat` and load it into Matlab, you will see two matrices: Matrix P will be 768×8 and matrix T will be 768×2.

Most of the neural network will be constructed using the parameters that are already set in `main.m`. In particular,

- Leave the split for training and testing at 70-30.
- You might drop `NumEpochs` to something like 10 until you're sure your code runs, and then crank it up to around 50 or 100. (If you use Octave-online, be prepared to click the "extra time" button several times).
- You might choose to change α, depending on what happens in training (for example, if your weights and biases blow up).
- Lastly, there are two important changes we want to make:
  1. Add an extra hidden layer, so that the network is $8 - 15 - 15 - 2$.
  2. Change the activation function to the ReLu function rather than the sigmoidal.
- Be sure to turn in your modified `main.m` file as well as a screenshot of the training (showing your final confusion matrices).