CS 111: Homework 7: Due by 6:00pm Friday, March 1

Homework must be submitted online as a PDF file to GradeScope. When you turn in your homework, tell GradeScope which page(s) contain each problem. Doing this correctly will be worth 2 points.

1. (Compare NCM problem 1.38.) The moral of this problem is that, with floating-point arithmetic, sometimes two algorithms look equivalent but one is better than the other at getting an accurate answer. Suppose you have a number $\hat{x}$ that is an approximation to another number x. Define the *relative error* in $\hat{x}$ as $|(\hat{x} - x)/x|$. (This only makes sense if $x \neq 0$.)

1a. The classic quadratic formula says that the two roots of the quadratic equation

$$ax^2 + bx + c = 0$$

are

$$x_0, x_1 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Use this formula in `numpy` (show your input and output) to compute both roots for

$$a = 1, \quad b = 10{,}000{,}000{,}000, \quad c = 1.$$

Also compute the roots two other ways: first with `numpy`'s `np.roots()`, and then by hand. To at least one significant digit, what is the relative error of the approximation computed using the quadratic formula to x_0? To x_1? What are the relative errors of the approximations computed using `np.roots()`?

1b. You should have found in (1a) that the classic formula is good for computing one root but not the other. Explain in a sentence why in this case one root isn't computed accurately. Hint: The answer involves IEEE floating-point arithmetic!

1c. Use the classic formula to compute one root accurately, and then use the fact that

$$x_0 x_1 = \frac{c}{a}$$

to compute the other. What are the relative errors now?

2. The standard form of a first-order ODE initial value problem is

$$\dot{y} = f(t, y), \quad y(t0) = y0,$$

where t is a scalar and y is a vector. Write each of the following ODEs as an equivalent first-order system of ODEs in standard form:

2a. Van der Pol equation:

$$\frac{d^2x}{dt^2} = (1 - x^2)\frac{dx}{dt} - x.$$

2b. Blasius equation:

$$\frac{d^3x}{dt^3} = -x\frac{dx}{dt}.$$

2c. Newton's second law of motion for a two-body problem in 2 dimensions (G and M are constants):

$$\frac{d^2x_0}{dt^2} = -GM\frac{x_0}{(x_0^2+x_1^2)^{3/2}}, \tag{1}$$

$$\frac{d^2x_1}{dt^2} = -GM\frac{x_1}{(x_0^2+x_1^2)^{3/2}}. \tag{2}$$

3. (Compare NCM problem 7.16.) This problem is partly about ODEs and partly about making nice plots with `matplotlib` (we import `matplotlib.pyplot` as `plt`).

Many modifications of the Lotka–Volterra predator-prey model that we saw in class on February 20 have been proposed to more accurately reflect what happens in nature. For example, the number of rabbits can be prevented from growing indefinitely by fixing a maximum number R and changing the equations to

$$\frac{dr}{dt} = 2\Big(1-\frac{r}{R}\Big) - \alpha r f, \tag{3}$$

$$\frac{df}{dt} = -f + \alpha r f, \tag{4}$$

where t is time, $r(t)$ is the number of rabbits, $f(t)$ is the number of foxes, and $\alpha > 0$ is a constant. This makes dr/dt negative whenever $r > R$, which guarantees that the number of rabbits can never grow to exceed R.

For $\alpha = 0.01$, compare the behavior of the original model with the behavior of this modified model with $R = 400$. Solve the equations (using `integrate.solve_ivp()` as we did in class) over 50 units of time, assuming that there are initially 300 rabbits and 150 foxes. Make four different plots to show the solutions and the phase space diagrams for both models as follows:

- number of foxes and rabbits (on the same plot) versus time for the original model,
- number of foxes and rabbits (on the same plot) versus time for the modified model,
- number of foxes versus number of rabbits (phase space) for the original model,
- number of foxes versus number of rabbits (phase space) for the modified model.

For all plots, label all curves (with `plt.legend()`) and all axes, and put a title on each plot that identifies it clearly. For the phase space plots, set the aspect ratio so that equal increments on the x- and y-axes are equal in size. (You may find the `matplotlib` tutorial linked under the "help" menu in Jupyter useful.)

4. This problem is about another connection between graphs, matrices, and eigenvalues. This time the graph in question is undirected, with no arrows on its edges. Let G be an undirected graph with n vertices, which we take to be the integers 0 through $n-1$. An edge of G is an unordered pair of integers (i, j). We assume that G has no multiple edges (that is, edge (i, j) only occurs once) and no loops (that is, no edges (i, i)).

The *Laplacian matrix* of G is the n-by-n matrix L whose diagonal element $L[i, i]$ is the number of neighbors of vertex i (also called the *degree* of vertex i), and whose off-diagonal element $L[i, j]$ is -1 if (i, j) is an edge of G and is 0 otherwise. For example, the Laplacian matrix of the graph consisting of 4 vertices connected in a square (also called a 4-cycle) is

$$L = \begin{pmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{pmatrix}.$$

The Laplacian matrix is symmetric because the graph is undirected. It's a theorem that all the eigenvalues of any symmetric matrix are real numbers, and it's also a theorem that all the eigenvalues of the Laplacian matrix of a graph are greater than or equal to zero.

4a. Use `linalg.eigh()` (note the h!) to find the four eigenvalues of the Laplacian matrix of the 4-cycle as above. You should find that they are all nonnegative real numbers, and that one of them is equal to zero.

4b. In fact, zero is an eigenvalue of the Laplacian matrix of every graph. Prove this by exhibiting an n-vector $v^{(0)}$ that is an eigenvector for the eigenvalue zero for *every* n-vertex graph, and explaining why $Lv^{(0)} = 0v^{(0)}$.

4c. Given an n-vertex graph G and any n-vector v, we can think of the n elements of the vector v as labels for the n vertices of the graph; $v[0]$ labels vertex 0, $v[1]$ labels vertex 1, and so forth. Used as labels in this way, the eigenvectors of the Laplacian matrix of a graph are in some ways analogous to the fundamental modes of vibration of a physical object.

For example, let P_n be the graph with n vertices joined in a single path, so that P_n has the $n-1$ edges $\{(0,1),(1,2),\ldots,(n-2,n-1)\}$. Write a python function `path(n)` that computes the n-by-n Laplacian matrix L_n of the graph P_n as a `numpy` array. Show your function, and show its output for $n = 5$ as an example. Also show the output of `linalg.eigh()` on L_5.

Now we'll see how the Laplacian eigenvectors of the graph P_n correspond to "modes of vibration." The idea is to think of the path P_n as a violin string. Use your function `path()` to compute the Laplacian matrix L_{100} of the graph P_{100}. Then use `linalg.eigh()` to compute the eigenvalues $d_0,\ldots,d_{99}$ and eigenvectors $v^{(0)},\ldots,v^{(99)}$ of the matrix L_{100}. Check to see whether the eigenvalues come out of `linalg.eigh()` in increasing order of size; if not, reorder both the eigenvalues and eigenvectors so that $d_0 \le d_1 \le \cdots \le d_{99}$. Don't print out L_{100} or the lists of eigenvalues and eigenvectors, but make and turn in the following plots (all nicely labeled, of course):

- Plot the 100 elements $v_0^{(0)}, v_1^{(0)}, \ldots, v_{99}^{(0)}$ of the eigenvector $v^{(0)}$ (this is a pretty simple picture).
- Make one plot that has 6 lines on it, plotting the elements of each of the eigenvectors $v^{(1)}$ through $v^{(6)}$.
- Plot the 100 eigenvalues d_i versus i.

4d. How close is the temperature matrix to being a Laplacian matrix? Specifically, which (and how many) of the nonzero elements of the 2D temperature matrix with $k = 100$ would you need to change to make it into the Laplacian matrix of some graph? Can you describe in words what that graph is?