CS 111 (W22): Homework 4

Due Wednesday, April 27th by 11:59 PM

NAME and PERM ID No.: Gaucho Olé, 1234567 (replace with yours)

UCSB EMAIL: GauchoOle@ucsb.edu (replace with yours)

Homework Buddy: (remove entire line if not applicable)

Additional Instructions: Make sure that the LaTeX command *\newpage* is at the end of each of your answers in your LaTeX source code, as they are at the end of each question here. This will put each question + answer on at least one page and make it easier to grade on Gradescope.

NOTE: In this homework, you'll be using a few of the functions introduced in lectures, like Lsolve, Usolve, Jsolve, the temperature setup, etc. These can all be found on our GauchoSpace website in our **Week 4 tab in the *Python Demos and Files* folder**. Look for a file called `CS111.py` - this will contain all the functions we create together in class (and I will update it as we go along in class).

You should be able to `import` that file within any program, which will enable you to make use the functions - just make sure that the .py file resides in the same directory as your program file or that wherever it resides is in the path of your Python IDE. See your lab TAs for help with this.

1. The temperature problem models our cabin in the woods in two dimensions, but most modern scientific simulations are done in three dimensions. Here you will create the matrix that corresponds to a 3-D version of the temperature problem. The "cabin" is now the unit cube. As before, we will discretize the interior by dividing it into k points in each dimension, but now there are k^3 points in all rather than k^2. The partial differential equation still leads to the approximation that the temperature at any given point is the average of the temperatures at the neighboring points, but now there are 6 neighbors, with 2 in each dimension.

   Using the routine `make_A(k)` from `Temperature.ipynb` as a model, write a routine `make_A_3D(k)` that returns the k^3-by-k^3 matrix A for the 3D version of the temperature problem. This matrix expresses the fact that, in a 3D k-by-k-by-k grid, each interior point has a temperature that is the average of its 6 neighbors (left, right, up, down, in, out). The diagonal elements of A are all equal to 6, and the off-diagonal elements are either 0 or -1. Most of the rows of A have 7 nonzeros.

   Here below, for debugging, is the correct matrix for $k = 2$. I converted it to a dense type for printing—you should also print it out as sparse, and indeed for $k > 2$ it's going to be too large to see what's going on in the dense matrix anyway.

```
[In:]
  k = 2
  A = make_A_3D(k)
  print('k:', k)
  print('dimensions:', A.shape)
  print('nonzeros:', A.size)
  #print('A as sparse matrix:'); print(A)
  print('A as dense matrix:'); print(A.todense())

[Out:]
  k: 2
  dimensions: (8, 8)
  nonzeros: 32
  A as dense matrix:
  [[ 6. -1. -1.  0. -1.  0.  0.  0.]
   [-1.  6.  0. -1.  0. -1.  0.  0.]
   [-1.  0.  6. -1.  0.  0. -1.  0.]
   [ 0. -1. -1.  6.  0.  0.  0. -1.]
   [-1.  0.  0.  0.  6. -1. -1.  0.]
   [ 0. -1.  0.  0. -1.  6.  0. -1.]
   [ 0.  0. -1.  0. -1.  0.  6. -1.]
   [ 0.  0.  0. -1.  0. -1. -1.  6.]]
```

Print out your matrix for $k = 2$ and $k = 3$ as a check that it's correct. Also use `plt.spy(A)` to make a spy plot of the nonzero structure for $k = 4$ or 5 (you may want to zoom in on the plot to see all the structure).

To complete a realistic simulation you would also write a routine `make_b_3D(k)` to compute the right-hand side b. For this problem, you don't have to do that; for the experiments in Problem 2 you can just use `np.random.rand()` to generate a random b.

What you will be submitting is a Python code file (must be called `make_A_3D.py`) of your `make_A_3D()` function on Gradescope under `h04_Q1`.

2. Now you will experiment with solving $At = b$ using various solvers from class and from `numpy`. For this problem, you should use the 3-D version of the temperature matrix from Problem 1. (You can get partial credit by using the 2-D temperature matrix from the class instead.) You can use a randomly chosen right-hand side vector b.

   Experiment with solving $At = b$ for the temperature t, for various values of k, using four different solvers:

   - The `Jsolve()` Jacobi solver, also from class. (Again you can vary `tol` and `max_iters`.)
   - The `scipy` sparse conjugate gradient solver `spla.cg()`. We may not discuss the conjugate gradient method before this homework is due, but try this function in any case and we'll be discussing the method soon after.
   - The `scipy` sparse LU solver `spla.spsolve()`.
   - The `LUsolve()` dense LU solver from class. (For this, you will have to use the dense form of A that you get from `A.todense()`. Warning! This will use too much memory if k gets very big at all.)

   Your answers to the following questions should be submitted as a PDF file on Gradescope under h04_Q2. You can create this from a blank original source (still, using LaTeX!)

   - For each solve, measure and report back the relative residual norm and the run time. This question should be in a format that looks like this:

     For Jsolve(): show residual norm for last iteration only
     As k gets larger...
     When k is ..., the relative residual norm is ..., and the run-time is ...
     When k is ..., the relative residual norm is ..., and the run-time is ...

     (reporting a couple of k values is enough here)

   - List solvers in order of accuracy. Then list them in order of speed. How do the answers to these questions change as you change k?

   Warning: Start with very small values of k, and be cautious as you increase k! The matrices get big in a hurry. Different solvers will fall over for different values of k; try to see how big a value of k each solver can handle with at most 30 seconds of compute time.