Ay190 – Worksheet 9
Daniel DeFelippis
Date: February 11, 2014

Solving Large Systems of Linear Equations

1

Writing a script to load these data files as matrix objects is fairly easy using np.loadtxt as shown below.

```
LSE1_m = np.loadtxt('./LSE1_m.dat')
```

Typing "LSE1_m.shape" gives the dimensions of the matrix LSE1_m. Doing this, we see that the matrices LSEi_m with $i = 1, ..., 5$ are all square matrices with number of rows (=number of columns) being 10, 100, 200, 1000, and 2000 respectively. Using the function "slogdet" located in NumPy's linalg module, we can calculate the natural log of the determinant to make sure that the determinant isn't 0 so the LSE is solvable. This is indeed true.

2

I wrote my own code that implements the Gauss algorithm. I defined two functions, one which gets the LSE into the desired triangular form, and the other which backsubstitutes to find all of the x_i in the LSE equation $A\mathbf{x} = \mathbf{b}$. I ran the algorithm on the contrived example

$$\begin{pmatrix} 2 & 5 & -3 \\ 1 & -7 & 4 \\ 6 & -2 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ -1 \\ 4 \end{pmatrix}$$

with the known solution of

$$\mathbf{x} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

to see if the algorithm returned the correct answer. It did!

So, I could then use it on the much larger LSE matrices. I timed how long running the algorithm on each LSE took using the "time" function in the time module to store the starting and stopping time and then print the difference between the two (the times are measured in seconds). The timing results are given for a particular run of the algorithm on the five matrices.

LSE	Size	Time (seconds)
1	(10, 10)	0.00103902816772
2	(100, 100)	0.173527002335
3	(200, 200)	0.333551883698
4	(1000, 1000)	10.1985230446
5	(2000, 2000)	50.3575429916

The actual values did sometimes vary, probably just due to how many other random processes happened to be running on my laptop at that time.

3

Next, I try doing solving the same systems with NumPy's "solve" function in its linalg module. I get faster results compared to Gaussian Elimination as shown in the table below.

LSE	Size	Time (Gauss)	Time (NumPy)	Ratio of Times
1	(10, 10)	0.00103902816772	0.000102043151855	10.182242990655809
2	(100, 100)	0.173527002335	0.00051212310791	338.8384543770586
3	(200, 200)	0.333551883698	0.0143749713898	23.203655482380803
4	(1000, 1000)	10.1985230446	0.521645069122	19.55069384968118
5	(2000, 2000)	50.3575429916	3.08289694786	16.334487932383144

NumPy's solver is clearly much better. For all sizes, it is at least an order of magnitude faster.

SciPy also has a bunch of solvers in its "sparse.linalg" module. However, they are all iterative, and the ones I tried (spsolve and cg) were nowhere near NumPy's solver's speed. I also tried SciPy's "solve" function located in its own linalg module, and it performed almost identically well as (but no better than) NumPy's solver.