**CH2013**

**Computational Programming and Simulations Lab**

**Aug-Dec 2019**

**Problem Sheet #5(a)**

**Sep 18 2019 (Wednesday – Batch1)**

*This problem sheet is about linear algebra & curve fitting.*

1. Use the ‘rand’ and ‘ones’ commands to generate a matrix A (3000x3000) and right hand side vector b(3000x1).

Use the ‘tic’ and ‘toc’ commands to estimate the time taken to solve using three methods:

1. Use the \ operator . Let the time taken be designated t1 and the solution x1.
2. Find the inverse of A using *inv(A)* and get x from this. Time taken is t2 and solution x2.
3. Next use the *linsolve* in-built function – time taken is t3 and solution is x3.

Which method worked out better for you? Please note – displaying the matrix or results will take too long so you should suppress that. Also, you may want to re-run the code a few times before concluding anything (can you explain why?)

1. Develop a function “ThomasMethod” (that takes matrix A, vector b, and the size of the matrix A, n, as arguments, and provides the solution vector X as the output). The function should implement the steps in the Thomas Algorithm for a tridiagonal matrix problem. Use the same notations ei, fi, gi that is in the notes to denote the various matrix elements. Use the function in a script file to solve for **X** in the following equation:

I want to check the final values of f1, g1, e2, f2, g2, e3, f3, g3, e4, f4, r1,r2,r3,r4, in addition to the Xi. Use the timeit function to estimate the time taken for your function to run. Particularly note the warning in the command window (if any).

1. ; . Solve using LU decomposition. Use the **lu** in-built function in MATLAB to first decompose the matrix A. Examine **L** and **U**. Keep in mind that the permutation matrix “**P**” is involved and you have to be careful in how you do the steps after the decomposition. You can use the \ operator to do the forward substitution (to get the **d** vector) and back substitution steps (to get **x**). Estimate the time taken for the entire set of commands (**t\_lu**). Note that as this is a v small problem, it may not be very meaningful to use this time for comparisons of performance.
2. The following data is given

**Cdata=[0.24; 0.32; 0.70; 1.37; 1.58; 2.04; 2.26; 2.28; 2.39; 2.41]**

**rdata=[0.3839; 0.3935; 0.911; 1.4975; 1.5735; 1.749; 1.9355; 1.893; 1.970; 1.924]**

1. Perform a polynomial fit to this data. Use a quadratic equation. Report the coefficients as **P1**, **P2** and **P3** (in decreasing order). Find the model-predicted **r** values and report them in a vector “***rmodel1***” . The residual is defined as **rmodel1(i)-rdata(i)** for every data point. Find the residuals for the polynomial fit, and report it in a vector “***residual1***” . Find the total error or distance between the model prediction and the data – label it ***err1***. Note that you can just use a “norm” of the vector residual1 to get this.

1. Actually, a reasonable relationship between C and r can be expressed as

Use the non-linear least square fit function in MATLAB to find optimal A & B (as *Afinal* and *Bfinal*), the residual (*residual2)*, number of iterations (*iterations*), and the function count (*funccount*), starting with an idea that A & B might both have a value of 1.0. Find model predicted r values – in rmodel2. Report the total error between rmodel2 and rdata as ***err2***.

1. Create a plot with the original data in symbols and the model predictions from both (a) and (b) ***plot(Cdata,rdata,'r+',Cdata,rmodel1,'b-',Cdata,rmodel2,’g-‘).*** Examine the results and make your own conclusions about them.

**I HAVE REPORTED ALL AS COLUMN VECTORS IN THIS PROBLEM**