Name : _____

Homework Project #2

Math 365, Spring 2015

Due February 25th, 2015

Please following the guidelines below when asking for help from others, especially those in the class.

- If you are asking for help from a classmate, you may show them your code.
- If you are helping another classmate, you may *not* show them your solutions.
- Under no circumstances should you share a copy (electronic or any other easily accessible form) of your code with any other students in the class.
- Keep in mind the spirit of this class. While I encourage collaboration, everyone is responsible for learning the material.
- And most important, start early and have fun!

Learning objectives for this assignment

- Setting up linear systems that arise in common engineering settings,
- Solving the resulting linear systems using Matlab's "backslash" operator ('/').
- Learning about the performance of a typical PC and complexity of the LU decomposition algorithm.

For this homework, you will turn in two separate m-files, one for problems 1 and 2, and a separate file for problem 3.

1. (**Electric circuits**) Find the currents $\mathbf{y}$ along each edge and the voltages $\mathbf{x}$ at each node of the circuit shown in Figure 1. To do this, you will need to set up an "adjacency matrix" A, a diagonal "capacitance matrix" C, and any voltage (i.e. battery) sources $\mathbf{b}$.

   To find the currents and voltages, we use Ohm's Law and Kirchhoff's Current Law to obtain the following **block** linear system for the node voltages and edge currents.

$$\begin{bmatrix} C^{-1} & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ 0 \end{bmatrix}$$

   Using "block" Gaussian elimination, this system reduces to

$$\begin{aligned} A^T C A \mathbf{x} &= A^T C \mathbf{b} \\ \mathbf{y} &= C(\mathbf{b} - A\mathbf{x}) \end{aligned}$$

   for the currents $\mathbf{y}$ and the voltages $\mathbf{x}$. Solve the first equation for $\mathbf{x}$ (the nodes voltages). Then, use the second equation to compute $\mathbf{y}$ (the edge currents) using these currents. Store your voltage values $\mathbf{x}$ in the file `voltages.out` and your current values $\mathbf{y}$ in the file `currents.out`.

   Check your solution : The voltage at node 1 is 9.771 V and the current along edge 7 is 14.393 mA (1 mA = 0.001 A).

   More details on this problem can be found in the linear algebra text *Linear Algebra and its Applications*, by Gilbert Strang. The relevant chapter is posted on the course website.

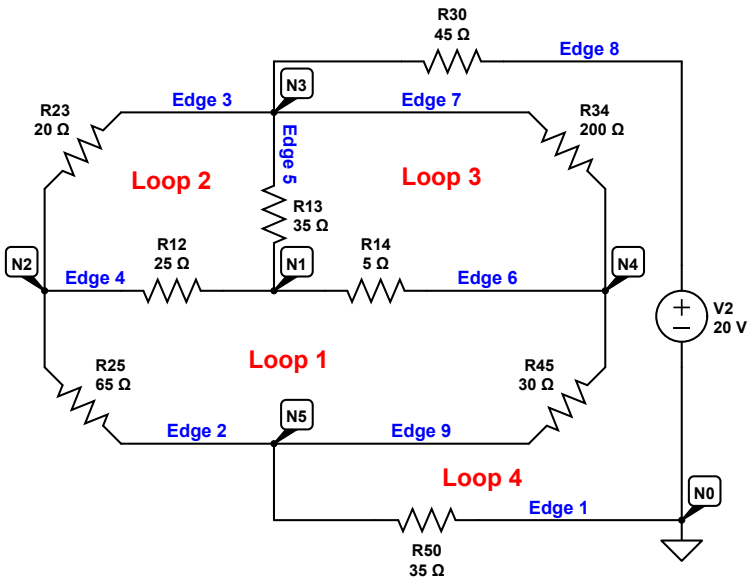

Figure 1: Circuit for Problem 1

2. (**Temperature conductivity**) This problem illustrates a common situation in which you will need to solve very large linear systems. Imagine that we have a metal rod of length L (m) whose temperature at each end is held fixed at $20C^o$ and which is being heated by a flame held at the midpoint. We can approximate the steady state temperature at equally spaced points along the rod using the following model. We divide the rod into N intervals of equal length. This partitioning of the rod gives us $N+1$ equally spaced points

$$x_j = hj, \qquad h = \frac{L}{N}, \qquad j = 0, 1, 2, \ldots N$$

The steady state temperature T_j (K^o) at interior points x_j in the rod can be modeled as

$$T_j = \frac{T_{j+1} + T_{j-1}}{2} + \frac{h^2}{2} f(x_j), \qquad j = 1, 2, \ldots, N-1$$

The flame $f(x)$ is given by

$$f(x) = \frac{S}{\rho c_p \beta} \exp\left(- \left(\frac{x - L/2}{\varepsilon} \right)^2 \right)$$

where

- S is the heating rate for the flame (W/m^3)
- β is the thermal diffusivity of the metal (m^2/s)
- c_p is the heat capacity of the metal $(J/(kg \cdot K^o))$
- ρ is the density of the metal (kg/m^3)
- ε is a scaling factor specifying the flame width (m)

Set up a linear system to solve for the steady state temperature distribution in the rod of length $L = 1$. Solve the system for a thermal conductivity of $\beta = 10^{-3}$, heat rate $S = 1 \times 10^4$, heat capacity of $c_p = 200$, density of $\rho = 10$ and a scaling factor $\varepsilon = 5 \times 10^{-2}$.

For this problem, use the `spdiags` command to construct the matrix that you need to solve for T.

- Compute the result at equally spaced points $x_1 = 0.01$, $x_2 = 0.02$, ..., $x_{N-1} = 0.99$ on the rod. Plot the results and be sure to add labels and a legend to your graph.
- In steady state, the flux of heat in or out of the endpoints of the rod should sum to zero. To show that our solution satisfies this property, compute the flux of heat at each end point of the rod. You can use the approximations

$$\text{Flux}_\ell \approx \frac{T_1 - T_0}{h}$$
$$\text{Flux}_r \approx \frac{T_N - T_{N-1}}{h}$$

  Show that the sum of these fluxes is exactly zero. Use `fprintf` to print your fluxes to 6 decimal places and show that the sum of the fluxes is zero (to within machine precision).
- Use `write_file.m` to write the solution to your linear system to the file `temp.out`.
- Answer the following question : Where is the rod too hot to touch? Indicate this value on your graph, using either a text label, or some other means to highlight where the rod is likely to burn your skin.

3. (**Timing a linear solve**)
In this problem, you will investigate the cost of solving linear systems and compare your results to the theoretical computational cost we have discussed in class.

Note : You will write a separate script for this problem and generate your own PDF file on the computer you are testing. Please turn in your published PDF in your Dropbox folder, along with a m-file for this problem.

Before starting this problem, you will need the following information about the laptop or PC you are working on.

- The amount of RAM in your computer. Typical values are betweem 4 and 12 GB (gigabytes).
- The clock speed of your PC or laptop. A typical value is 2.5 GHz (gigahertz).

(a) **Experimental setup.** Determine a matrix size that you can comfortably fit into your available RAM. For example, if you have a 8 GB machine, you should be able to comfortably store a matrix that occupies about 200MB. Store this value in a variable "`Mb`". Use the following information to compute a maximum matrix dimension N that you can store in `Mb` megabytes of memory.

- A megabyte has 1024 kilobytes
- A kilobyte is 1024 bytes
- A floating point number is 8 bytes.
- An $N \times N$ matrix contains N^2 floating point numbers.

Call the N you compute '`nmax`' (N_{max}). Use `fprintf` to print out the the size of your matrix (in megabytes) and the `Nmax` you computed.

(b) **Calibrate the timing experiment.** We need to "calibrate" our experimental set up by timing an operation whose flop count is easy to compute. Create two random matrices A and B each of size $N_{max} \times N_{max}$. Using the Matlab functions `tic` and `toc`, determine how much time (seconds) it takes to compute the product AB. Determine the number of floating point operations (additions and multiplications) it takes to compute the $N_{max} \times N_{max}$ matrix-matrix product. Use this number to estimate the number of floating point operations per second ('flops') your computer can carry out. Call this flop rate '`flops`'.

Compare this number to the theoretical number obtained using your computer's clock speed. The following information might be useful.

- A gigahertz is 10^9 hertz.

- A hertz is equal to one clock cycle.
- A typical microprocessor (CPU) can compute 4 flops per clock cycle.

If you have a dual or quad core machine, you may also want to investigate whether Matlab is making use of multiple cores.

Use `fprintf` to print out the flop rate ('`flops`') you compute, your computer's theoretical clock speed (in gigahertz), and the theoretical flop rate (based on your computer's clock speed). **Note :** The floprate you get from the matrix-matrix multiply above could be much larger than what you might predict, based on your hardware. The reason for this is that Matlab has a highly optimized matrix-matrix multiply that takes advantage of all the cores on your machine, among other things.

(c) **Time a dense linear solve.** Create an integer sequence of values `Nvec` between, say, $N = 100$ and the N_{max} you found above. To generate this sequence, it is a good idea to use the `logspace` command.

Using a `for` loop, loop over the entries N_i of the sequence. In each pass through the loop, create a random matrix A of size $N_i \times N_i$, and a random right hand side vector $\mathbf{b}$ of size $N_i \times 1$. Then, using `tic` and `toc`, time how long it takes to solve the system $A\mathbf{x} = \mathbf{b}$ using the backslash operator. Store this time as the i^{th} entry in a vector '`lutimes`'.

(d) On a **log-log** plot, plot the time values you found above verses the N_i in your sequence of N values. On the same set of axis, plot the curve of the theoretical time estimated by the operation count we discussed in class. Be sure to use the flop rate '`flops`' you computed above to get a time (in seconds) from an operation count. The two plots should be very close. Be sure to label your plots. Use the Matlab `legend` command to identify each curve that you get. Your plot should look something like the figure shown in Figure 2.

(e) (**Extra Credit - 10 points**) Visit the website `http://www.top500.org` and answer the following questions.

- Using the value from the 'Linpack Benchmark', what is the performance of the world's fastest supercomputer, measured in GFlops ($= 10^9$ flops)?
- How much faster is the world's fastest computer than your computer or laptop?
- How recently would your computer have made it on the Top 500 list of the world's fastest computers?
- When can we expect to see an exaflop machine?

The 'Linpack Benchmark' solves a dense linear system of equations, very much like what you have done for this problem.

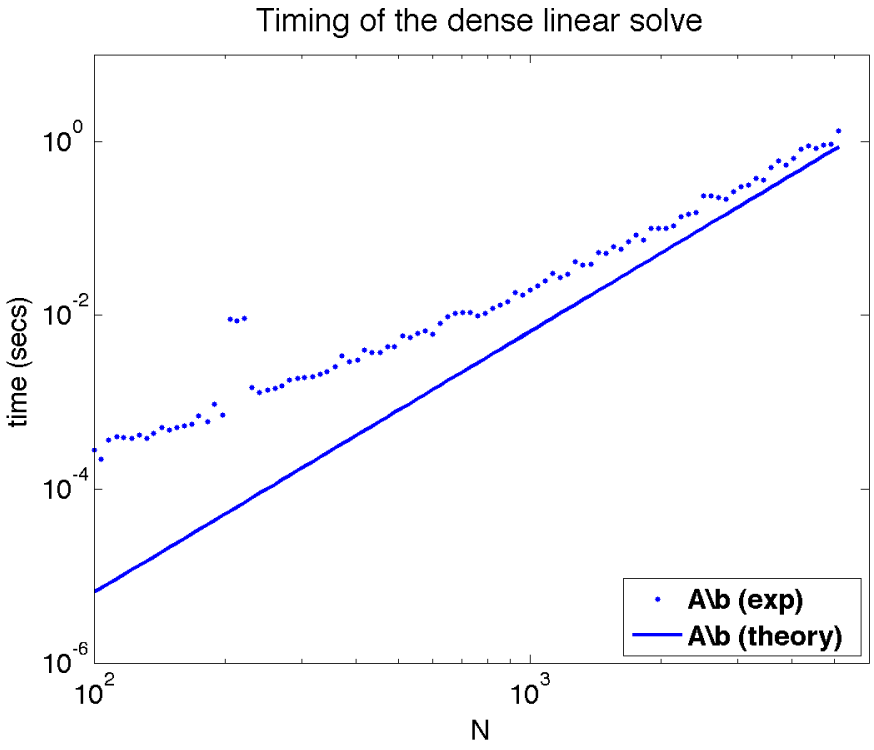

Figure 2: Timing plot for problem 3. The x-axis is the matrix size N ($N \times N$ matrix), and the y-axis is the time (in seconds) as measured by `tic` and `toc` for solving a linear system of size N.