AMATH 352: Problem set 2

Niall Mangan

October 16, 2016

1 Matlab Portion

1.1 Gaussian Elimination

I have provided you with Gaussian elimination (called GE_NMM.m) code that steps through the Gaussian Elimination algorithm. This code is written as a function, so you can call this function as long as your current MATLAB folder is where you saved this function. This code does not check whether the pivot is zero or not. Write a script that

(a) Here is a test matrix that has non-zero pivots: $\mathbf{a} = \begin{bmatrix} 1 & 6 & 2 & 0 \\ 1 & 4 & 1 & 1 \\ 2 & 1 & 3 & 0 \\ 3 & 2 & 4 & -1 \end{bmatrix}$ Remove the ';', `fac=a(ii,ii)/a(jj,ii);` and `a(jj,:) = fac*a(jj,:)-a(ii,:);`. Define the matrix a in Matlab and call the function from the command line using `[a_out] = GE_NMM(a,0)` and examine the output. There will be questions on the Quiz about what these lines of code are doing. Suppress the output by adding the ';' back on to the end of the line so as to not mess up scorelator.

(b) Here is a matrix that has a pivots that are zero. $\mathbf{a2} = \begin{bmatrix} 1 & 6 & 2 & 0 \\ 1 & 6 & 1 & 1 \\ 1 & 6 & 3 & 0 \\ 3 & 2 & 4 & -1 \end{bmatrix}$

Add code that does the following (you will have to modify the function):

- check if the current pivot is zero,
- find the first non-zero entry in column below the pivot,
- swap the corresponding row with the current pivot row

Do not swap any other rows.

Uncomment the `% if ii == 2 % save(filename, 'a', '-ascii') % end` code by removing the %. Define `filename = 'A11b1.dat';` and then call the modified function `[a2_out] = GE_NMM(a2,filename);` in

your script. This will save the matrix after the code has zeroed out below the second pivot. Also save `a2_out` in `'A11b2.dat'` in your script.

(c) The code I gave you creates a upper triangular matrix. If we are interested in solving $\mathbf{E}x = b$, the algorithm does not currently perform Gaussian elimination on the augmented matrix $[\mathbf{E}|b]$. Alter the algorithm to perform Gaussian elimination on the augmented matrix and save it as a new function called `GE_aug.m` that takes in a vector b as well. $b = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 1 \end{bmatrix}$ Call the function for `a2` and `b`. Save the augmented matrix after being zeroed out below the 2nd pivot as `'A11c1.dat'` and the final augmented matrix a2_aug out of the function as `'A11c2.dat'`.

(d) The final step is to take this upper triangular matrix and back substitute to solve for x. Add a for loop to the algorithm to do this and have the function output both the final augmented matrix and the solution vector x. (save this as a separate function `GE_solve.m`). You will need to index carefully. Note that you can make a vector in decreasing order using k = 10:-1:1;.

For each x_i in the solution vector this can be divided into 3 steps, and you will need to save the `lhs` as defined in the second step below. If you have the equation $2x_1 + 3x_2 - x_3 = 9$ and $x_2 = 1$ and $x_3 = 2$.

- Select the part of the matrix containing $3x_2 - x_3$, "plug in" $x_2 = 1$ and $x_3 = 2$ using matrix multiplication to get $known = [3 \quad -1] * [1 \quad 2]^T = 1$, (so the equation in your mind should be $2x_1 + known = 9$)
- Combine with the 9 to get a left hand side $lhs = 9 - known = 8$
- Divide the lhs by the coefficient in front of x_1 to get $x_1 = lhs/2 = 4$

You can check your answer by using the backslash operator in MATLAB. (syntax x = E\b). This calls MATLAB's algorithm for solving $\mathbf{E}x = b$. In the for loop you write to do back substitution add the code:

```
if ii == 2
save([filename(1:5), 'lhs.dat'], 'lhs', '-ascii')
end
```

to save the left hand side for the 2nd variable. Now when you run your script it should also produce a `'A11c2lhs.dat'` file. Save the solution x in `'A11d.dat'`

Summary of output for this section:

- `'A11b1.dat'`

- `'A11b2.dat'`
- `'A11c1.dat'`
- `'A11c2.dat'`
- `'A11c2lhs.dat'`
- `'A11d.dat'`

1.2 Test your GE algorithm: Solve some systems of linear equations

Use your algorithm to solve the following system of equations. First you will have to organize them into a matrix and vector b for the right hand side.

For each you should define the matrix and vector in your script and save 4 .dat files. For example for part a you will have

- `'A12a1.dat'` the matrix with the values under the 2nd pivot zeroed out. (saved in function)
- `'A12a2.dat'` the final augmented matrix.
- `'A12a1lhs.dat'` the left hand side for the 2nd variable as defined above. (saved in function)
- `'A12ax.dat'` the solution x

(a)

$$2x_1 + 4x_2 - 6x_3 = -4 \tag{1}$$
$$x_1 + 5x_2 + 3x_3 = 10 \tag{2}$$
$$x_1 + 3x_2 + 2x_3 = 5 \tag{3}$$

(b)

$$x_1 + x_2 - 6x_3 = 7 \tag{4}$$
$$x_1 + 2x_2 + 9x_3 = 2 \tag{5}$$
$$x_1 + 2x_2 + 3x_3 = 10 \tag{6}$$

(c)

$$4x_1 + 8x_2 + 4x_3 = 8 \tag{7}$$
$$x_1 + 5x_2 + 4x_3 - 3x_4 = -4 \tag{8}$$
$$x_1 + 4x_2 + 7x_3 + 2x_4 = 10 \tag{9}$$
$$x_1 + 3x_2 - 2x_4 = 4 \tag{10}$$

Remember you can check your work by using the backslash operator (x = A\b). You should be able to solve all of these using the same algorithm from problem 1.1d, GE_solve.m. If you cannot, fix your algorithm.

1.3 Computational Complexity

It is important to be able to determine how many operations an algorithm requires. Here you will count the number of operations the Gaussian elimination algorithm takes for a variety of matrices. To prevent errors in scorelator use the original algorithm I provided (`GE_NMM.m`) to start this problem.

(a) Starting with my GE_NMM.m code, add a counter that keeps track of the number of times the algorithm performs a divide or multiply step (scaling operation) or an addition/subtraction step *on each element*. Note that some of the operations are vector-wise, so multiple operations happen in a single line of code. Have the new algorithm output the counter value and save as 'GE_count.m' `[a_out, count] = GE_count(a1, filename)`.

For the matrices in problems 1.2a-c, save the operation count as `'A13a.dat'`, `'A13b.dat'`, and `'A13c.dat'`. (Do not save the matrix – the save code should be commented out in the function.)

(b) The next section of the written portion will require changing this code and producing plots. Only upload the code up to this point to scorelator.

2 Written portion

While this section requires coding, do not upload the scripts to scorelator. Save the plots for parts a and b in your .pdf file and include a short sentence or two describing what you see. You can save the plots as pdfs and add your commentary by hand or use LaTeX

2.1 Computational Complexity

(a) Now we are going to see how the computational complexity scales with the size of our $N \times N$ matrix. Write a loop from N = 1 to 100 that:

- creates matrix of size N with random entries,
- calls `GE_count` for each matrix,
- saves the number of operations in a vector `countvector` (one entry for each matrix)
- define a vector, `Nvector` from 1 to 100

You can generate random matrices of size $N \times N$ by using the MATLAB command: rand(N).

(b) Make a plot of the size of the matrix (x-axis) by the number of scaling operations and number of subtraction/replacement operations (y-axis). Use the commands

```
figure(1) %generates a new figure
plot(Nvector, countvector) %Nvector and countvector need to be the same length
xlabel('Size of Matrix, N')
ylabel('Number of operations')
title('Computational complexity of GE')
```

(c) Do the operations both grow like $\mathcal{O}(N^3)$? Plot on a log–log scale to check. You can also plot (N vs N^3) to compare the slope. The command for plotting on log–log is `loglog(x,y)`.

(d) Find the number of necessary floating point operations required to compute the following operations (using the big-oh notation introduced in class). Explain your reasoning in each case.

   (a) Compute the sum $A + B$ for $A, B \in R^{m \times n}$.

   (b) Compute the product Ax for $x \in R^n$ and $A \in R^{n \times n}$ where A is upper triangular.