MATH299M/CMSC389W

Fall 2019 – Dan Zou, Devan Tamot, Vlad Dobrin

Model H5: Gaussian Row Reduction (or any algorithm)

Assigned: Tuesday September 24th, 2019

Due: Monday October 7th, 2019 11:59PM

This week we're working with CS-style scripting, practicing using Mathematica's tools that are familiar to you from Java, C, etc. such as If, For, While, etc. Mathematica has an edge on other languages though in that you can do math in-line seamlessly. For algorithms like MergeSort, this isn't all that helpful, and while Mathematica does have awesome list functionality etc., you don't really get an advantage writing it Mathematica vs. Java or C or something. However, for algorithms that involve a lot of math, Mathematica is awesome. If you're doing anything that involves a derivative, matrix or vector algebra, or any number of mathematical structures, Mathematica will make the task quicker, faster, and so much prettier. Mathematica has a RowReduce function, but I thought writing it ourselves would be a good exercise to showcase Mathematica's awesomeness. Below I'm going to explain what Gaussian Row Reduction is if you don't know, but you don't have to do this algorithm for this assignment. **You can choose whatever algorithm you can think of**, preferably one that uses math, and definitely one that needs some loops and conditionals. I just want you to get some practice using Mathematica for normal coding things. When making math or physics models it's satisfying to be able to do it completely functionally, so that the whole code is really just one giant nested function to evaluate, but sometimes you need to get into the grimy comp sci details to do things properly, so knowing how to use these tools is important.

Gaussian Row Reduction is an algorithm to solve systems of linear equations. Let's start at the beginning:

Say we want to solve the system

$$\{x + y + z = 5\ 2x + y - z = -2\ x - y = 1$$

Well then really we're working with the matrix equation

$$[1\ 1\ 1\ 2\ 1\ -1\ 1\ -1\ 0\ ]\,[x\ y\ z\ ] = [5\ -2\ 1\ ]$$

Okay, if you haven't taken linear algebra or vector calc, at this point you're probably pretty lost. You have to be able to do matrix multiplication for this to make sense. Again, you can do a different algorithm than Gaussian Row Reduction for this assignment, maybe the Newton-Raphson method for locating zeroes of a function, BubbleSort for sorting a list, or calculating square roots by hand, for example (it's okay if Mathematica already has a function to do it, we're writing our own as an exercise). From here forward I'm going to assume you're familiar with matrix algebra.

We can write this system as:

$$A\vec{x} = \vec{b}$$

where **A** is a matrix, $\vec{x}$ is our vector of unknowns, and $\vec{b}$ is our vector of constants.

The idea now is that we hit both sides with simple matrices that will gradually transform the matrix on the left into the identity matrix. When that happens our vector on the right will be have been transformed into a different vector $\vec{\beta}$, and then the equation will read:

$$I\vec{x} = \vec{\beta}$$

where **I** is the identity matrix. From that if we convert back to a system of linear equations, it's just:

$$\{x + 0 + 0 = \beta_1 \; 0 + y + 0 = \beta_2 \; 0 + 0 + z = \beta_3$$

where β_i are the components of $\vec{\beta}$, so we see that we have explicitly found x, y and z.

As far as computation goes, we don't need to concern ourselves with matrix transformations (although you could implement this that way). These simple matrix transformations do three things – scaling rows, switching rows, and adding rows together. Let's work through the example above.

$$[1\ 1\ 1\ 2\ 1\ -1\ 1\ -1\ 0\ \ldots 5\ -2\ 1\ ]$$

This is what we start with, it's called the *augmented matrix*; we've stuck $\vec{b}$ on to the end purely as a visual tool. Now, whenever we scale a row, switch two rows, must or add two rows, we must also do that to $\vec{b}$, as recall that behind the scenes we're multiplying both by the same matrix that achieves these things. Okay, let's get started. First, let's subtract two times the first row from the second row, and then subtract the first row from the third row:

$$[1\ 1\ 1\ 0\ -1\ -3\ 0\ -2\ -1\ \ldots 5\ -12\ -4\ ]$$

When I said we can "add two rows together", I really meant we can add a multiple of a row to another, including negative multiples (so, subtraction). Okay, so now we have that the left column is all zeroes except for the top row, awesome. Let's now make the leading term of the second row positive by multiplying the row by -1:

$$[1\ 1\ 1\ 0\ 1\ 3\ 0\ -2\ -1\ \ldots 5\ 12\ -4\ ]$$

And now let's add two times the second row to the third row to kill the leading term of the third row (notice a pattern?):

$$[1\ 1\ 1\ 0\ 1\ 3\ 0\ 0\ 5\ \ldots 5\ 12\ 20\ ]$$

Awesome, now let's divide the last row by 5:

$$[1\ 1\ 1\ 0\ 1\ 3\ 0\ 0\ 1\ \ldots 5\ 12\ 4\ ]$$

Now the system should be easily solvable, if we convert it back to a system of equations we get this:

$$\{x + y + z = 5\ 0 + y + z = 12\ 0 + 0 + z = 4$$

However, if we further row reduce this matrix all the way down to a diagonal matrix, then we can just read off the answers without doing any algebra afterwards at all. So, let's three times the third row from the second row, and subtract the third row from the first row:

$$[1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ \ldots 1\ 0\ 4\ ]$$

And finally, let's subtract the second row from the first row:

$$[1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ \ldots 1\ 0\ 4\ ]$$

And we read off that x is 1, y is 0, and z is 4.

Gaussian Row Reduction is a ton of work, and coding requires debugging which takes time. Definitely feel free to do a different algorithm, or if you do this one then you can just do 3x3. I'll be super impressed at anyone that does this for a general n by n matrix – it's certainly feasible, just a decent amount of headache. When implementing this, don't bother checking anything, like don't bother checking to see if the leading term of a row is already 0. An example of your first step would be: make a dummy list equal to the first row, multiply everything in it by the leading term of the second row over the leading term of the first row, multiply by -1, and then add it to the second row. Also edge cases are a problem, since some systems aren't uniquely solvable – you can just assume the inputted matrix is invertible (as in, it works).