Gaussian_Elimination-student

September 11, 2024

1 Gaussian Elimination

Gaussian elimination is an algorithm for solving a linear system by matrix operations. You will write code that performs Gaussian elimination on a solvable system and returns the solution. First, we work through an instructive example.

$$\begin{aligned} x + 2y - z &= 1 \\ 2x - y + 3z &= 5 \\ 3x + y + 3z &= 10 \end{aligned}$$

Now, we'll solve this system using reduced row echelon form (RREF).

1.0.1 Step 1: Write the Augmented Matrix

The augmented matrix for this system is:

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 1 \\ 2 & -1 & 3 & 5 \\ 3 & 1 & 3 & 10 \end{array}\right)$$

Step 2: Apply Row Operations to Achieve RREF

We'll perform row operations to transform the matrix into reduced row echelon form (RREF).

1. **Make the pivot in the first row (top-left corner) a 1.**
   This is already 1, so no changes are needed.
2. **Make the first column below the pivot zeros.**
   - Add -2 times the first row to the second row: $R_2 \to R_2 - 2R_1$

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 1 \\ 0 & -5 & 5 & 3 \\ 3 & 1 & 3 & 10 \end{array}\right)$$

   - Add -3 times the first row to the third row: $R_3 \to R_3 - 3R_1$

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 1 \\ 0 & -5 & 5 & 3 \\ 0 & -5 & 6 & 7 \end{array}\right)$$

3. **Make the pivot in the second row a 1.**

- Divide the second row by -5: $R_2 \to \frac{1}{-5} R_2$

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 1 \\ 0 & 1 & -1 & -\frac{3}{5} \\ 0 & -5 & 6 & 7 \end{array}\right)$$

4. **Make the second column below the pivot zeros.**
   - Add 5 times the second row to the third row: $R_3 \to R_3 + 5R_2$

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 1 \\ 0 & 1 & -1 & -\frac{3}{5} \\ 0 & 0 & 1 & 4 \end{array}\right)$$

5. **Make the third column above the pivot zeros.**
   - Add 1 times the third row to the second row: $R_2 \to R_2 + 1R_3$

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 1 \\ 0 & 1 & 0 & \frac{17}{5} \\ 0 & 0 & 1 & 4 \end{array}\right)$$

   - Add 1 times the third row to the first row: $R_1 \to R_1 + 1R_3$

$$\left(\begin{array}{ccc|c} 1 & 2 & 0 & 5 \\ 0 & 1 & 0 & \frac{17}{5} \\ 0 & 0 & 1 & 4 \end{array}\right)$$

6. **Make the second column above the pivot zeros.**
   - Add -2 times the second row to the first row: $R_1 \to R_1 - 2R_2$

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & -\frac{9}{5} \\ 0 & 1 & 0 & \frac{17}{5} \\ 0 & 0 & 1 & 4 \end{array}\right)$$

1.0.2 Step 3: Extract the solution

The system has a unique solution, given by the final column.

$$x = -\frac{9}{5}, \quad y = -\frac{17}{5}, \quad z = 4$$

1.1 Coding

Write a method `system_solve(A,b)` that solves the linear system $\mathbf{Ax} = \mathbf{b}$ using Gaussian elimination. You may assume the solution exists and is unique. $\mathbf{A}$ should be an $n \times n$ coefficient matrix and $\mathbf{b}$ is an $n \times 1$ column vector. Some example input is given for you.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 300 ## for high-dpi displays. edit as needed
```

[ ]:
```
A = np.array([[1,2,-1],[2,-1,3],[3,1,3]], np.float64)
b = np.array([[1,3,10]], np.float64).T
```

```
## The following line creates the augmented matrix $(A | b)$. You should use this in your method
AA= np.hstack([A,b])
```

[ ]:
```
## Your code goes here. Insert cells as needed.
```

The following cell will check your code on 10 random 5x5 matrices

[ ]:
```
for i in range(10):
    A = np.random.rand(5,5)*10
    x = np.random.rand(5,1)*10
    b = A@x
    assert((np.abs(system_solve(A,b)-x)<1e-10).all())
```

1.2 Measuring Error

The Gaussian Elimination algorithm has an error that increases with the size of the input matrix. In this section you will approximate the rate at which that error grows. Assume the error can be modeled by a polynomial

$$err(N) \sim N^k$$

where N is the number of unknowns in the linear system and k is a constant to be determined. It should be noted that in general the error term depends on the relative sizes of the elements in the coefficient matrix. We are choosing them to be random from $[0, 1]$ so they will usually behave reasonably the same and so the problem is simplified in our case.

1.2.1 Approach

You will calculate the error in your linear system solver on several systems up to size $N = 1000$. For each size N you will solve 10 random systems and average the error $e(N)$ over the 25 runs. You will then find a polynomial fit for the dataset N vs. $e(N)$.

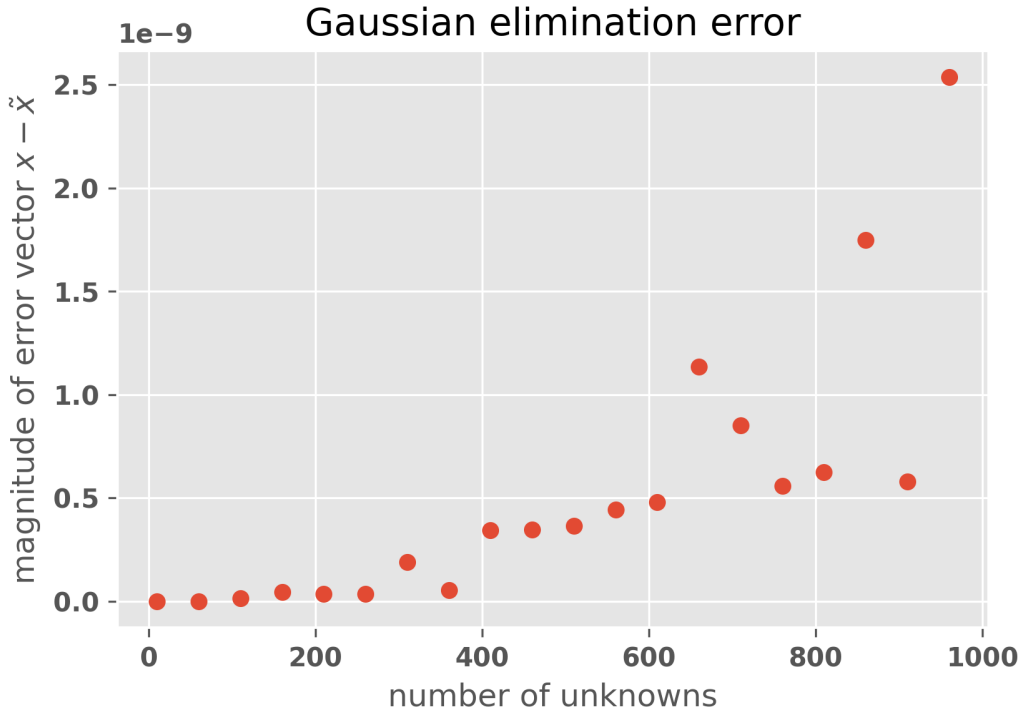
Write a method that takes as parameters the matrix size N and the number of repetitions to compute. Create two random uniform matrices: A and x (uniformly random over $[0, 1]$, by using `np.random.rand`. Compute $b = Ax$. Use your linear solver to find $\tilde{x}$ given A, b and determine the length of the error vector $x - \tilde{x}$. Do this for each repetition and return the *median* error (length of the error vector). We are using the **median** instead of the mean because the mean is too sensitive to outliers and this investigation is rife with outliers!

[ ]:
```
## Your code goes here. Insert cells as needed.
```

Now collect data on various values of N up to 1000. Be judicious: this problem takes a while to solve for large matrices. You should end up with a vector X that contains matrix sizes (N) up to 1000 and Y that contains the average error $e(N)$.

[ ]:
```
## Your code goes here. Insert cells as needed.
```

Now create a scatter plot of N vs $e(N)$. An example plot is shown here



.

```
[ ]: ## Your code goes here. Insert cells as needed.
```

1.3 Regression

You have written a regression routine before, but we will use some built into numpy. Here's sample code for doing a quadratic regression

```
x = # x data as numpy array or python list
y = # y data as numpy array or python list
coefficients = np.polyfit(x, y, 2)  # 2 indicates quadratic
a2, a1, a0 = coefficients # in decreasing order of powers
quadratic_model = np.poly1d(coefficients) # make a2x^2 + a1x + a0
y_fit = quadratic_model(x) # now y_fit is a vector
```

First you should compute a quadratic regression and superimpose the resulting parabola on a scatterplot of the data

```
[ ]: ## Your code goes here. Insert cells as needed.
```

1.3.1 A regression problem and a solution

Depending on your data the parabola you've plotted may or may not look like a decent fit. But it has one glaring problem. By finding a quadratic fit we are assuming $O(N^2)$ growth of our error term. But it could be $O(n^3)$ or $O(n^{2.3})$. What we want to find is the best **exponent** $O(n^k)$ for polynomial growth. This is the perfect time to use a log-log plot. If you transform your data $x_{log} = \log(x)$ and $y_{log} = \log(y)$ and perform a linear fit, the slope of the best fit line tells you the

order of growth k (derivation of this is discussed in class.)

You should find the best log-log plot slope. Then make a scatter plot of the values `x_log` and `y_log` along with the best fit line. Above the graph print the coefficients of the plot and state your best estimate of the growth rate $err(N) \sim N^k$

[ ]:
```
## Your code goes here. Insert cells as needed.
```

Finally, find the correlation coefficient for `x_log` and `y_log`. What amount of the variance is explained by your linear model? (There are several built in methods than can find this value)

[ ]:
```
## Your code goes here. Insert cells as needed.
```