Coding Report

Joe Lenning

1 Problem Description

This report demonstrates the results of finding solutions to Ax = b. Specifically, we look at two similar methods Gauss Elimination(GE) with no partial pivoting and LU factorization. Gauss Elimination is intended to create an upper triangle matrix by solving for one variable in one equation and then using back substitution to solve for the rest of the variables. In some cases pivoting is used to ensure we do not get round-off errors, meaning if we do not pivot or swap rows certain entries might be very small or messy to work with, and when we pivot or swap we overcome this. Additionally, some requirements of our matrix A in order for GE to work is that it must have a nonzero determinant, or the matrix must be full rank. This ensures that the matrix has solutions. Furthermore, GE requires O(n^3/3) arithmetic operations to determine x.

While not all matrices have a LU decomposition, sometimes you have to permute rows, LU factorization decomposes our original matrix A into two matrices: L a lower triangular matrix, and U an upper triangular. Our L matrix comprises 1s on the diagonals and uses the identical factors we used in our U however the signs are just switched. To solve our Ax= b or LUx =b, we first solve Lb = y and then Uy = x. For Lb= y we'll use forward substitution and for Uy = x, we'll use backward substitution. In general LU requires O(n^2) operations to find x.

The approach for nxn matrices is the same however the procedure just takes a lot longer. In general, LU factorization is faster at lower n values however at higher n values such as n=50, 100, 250, etc LU takes slightly longer because we have to forward and back solve. Next, we use the same methods(GE and LU factorization) for a random matrix A, where A = (5√n)I + R where R is a matrix with random entries from the normal distribution centered at 0 with a standard deviation of 1 and I is the identity matrix.

2 Results

For our base matrix…

$$A = \begin{bmatrix} 2 & 4 & 5 \\ 7 & 6 & 6 \\ 9 & 11 & 3 \end{bmatrix}, B = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

We found that our reduced echelon form(GE) was:

$$ref(A) = \begin{bmatrix} 2 & 4 & 5 \\ 0 & -8 & -12.5 \\ 0 & 0 & -8.5625 \end{bmatrix}$$

Additionally, for our LU decomposition, we found that:

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 3.5 & 1 & 0 \\ 4.5 & .875 & 1 \end{bmatrix} \quad U = \begin{bmatrix} 2 & 4 & 5 \\ 0 & -8 & -12.5 \\ 0 & 0 & -8.5625 \end{bmatrix}$$

Note: We know that this is in fact correct because if we multiply L*U we get our original matrix A.

Now to Solve Ax= b or LUx = b, first, we will use substitution for Ly= B(solving for y) and then again use substitution to Ux= y(solving for x)

$$\begin{bmatrix} 1 & 0 & 0 \\ 3.5 & 1 & 0 \\ 4.5 & .875 & 1 \end{bmatrix} \times \begin{bmatrix} y1 \\ y2 \\ y3 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

And then,

$$\begin{bmatrix} 2 & 4 & 5 \\ 0 & -8 & -12.5 \\ 0 & 0 & -8.5625 \end{bmatrix} \times \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} = \begin{bmatrix} 3 \\ -17/2 \\ -81/16 \end{bmatrix}$$

To find the solution to Ax = b

$$X \approx \begin{bmatrix} -0.2554 \\ 0.1386 \\ 0.5912 \end{bmatrix}$$

For the matrix A, where A is an nxn matrix such that A = $(5\sqrt{n})I + R$ where R is a matrix with random entries from the normal distribution centered at 0 with a standard deviation of 1 and I is the identity matrix and n = 50, 100, 250, 500 we can observe the following results:

Methods vs Error Table

n	GE Error	LU Error

50	7.64e-16	4.07e+00
100	1.55e-15	7.09e+00
250	4.14e-15	1.19e+01
500	8.00e-15	1.64+01

Along with that, we have time vs matrix size. Note how the error for GE the error is much lower, however as previously mentioned the time to solve using GE is slightly longer.

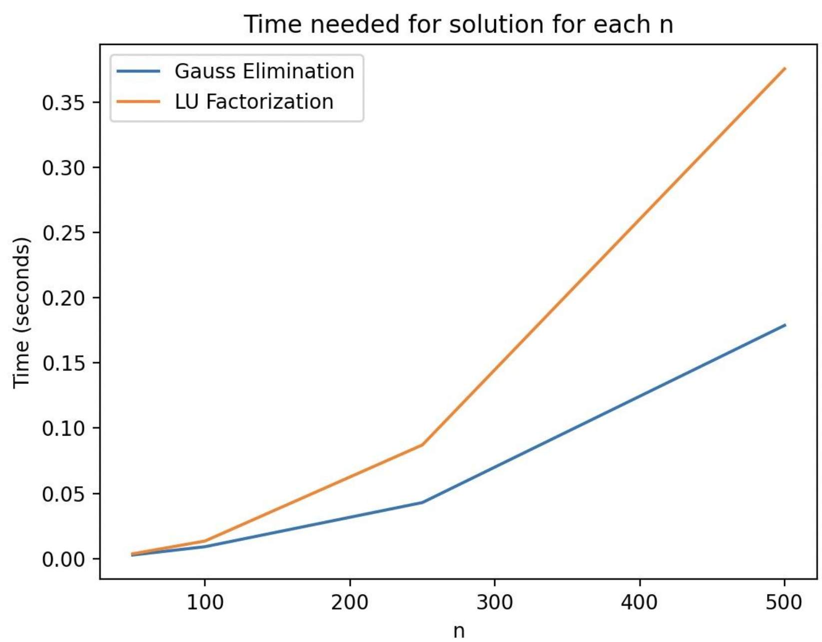

Figure 1: As expected our LU factorization is slightly slower than the GE method.

With respect to,

$\hat{A} = R$, the diagonally dominant I would think is much faster than the original. Additionally, the row swapping that must be done for diagonally dominant matrices makes solving the system much faster. The swapping of rows ensures that round off errors are pretty minimal.

3 Collaboration:

None

4 Academic Integrity

On my personal integrity as a student and member of the UCD community, I have not given nor received any unauthorized assistance on this assignment.

5 Appendix

```
import numpy as np
def gauss_elimination(A, b):
   n = len(b)
   # Elimination
   for k in range(n - 1):
       if abs(A[k, k]) < 1e-10:
           raise ValueError('Matrix is Singular')
       for i in range(k + 1, n):
           factor = A[i, k] / A[k, k]
           A[i, k:n] -= factor * A[k, k:n]
           b[i] -= factor * b[k]

   # Back substitution
   x = np.zeros(n)
   x[n - 1] = b[n - 1] / A[n - 1, n - 1]
   for i in range(n - 2, -1, -1):
       x[i] = (b[i] - np.dot(A[i, i + 1:n], x[i + 1:n])) / A[i, i]

   return A, x
# Back Substitution
def back_substitution(A, b):
   n = len(b)
   x = np.zeros(n)
   #Go over rows in reverse order:
   for i in range(n-1, -1, -1):
       a1 = b[i]
       a2 = np.dot(A[i, i + 1:], np.array(x[i + 1:]))
       a3 = int(a1 - a2)  # cast to integer
       x[i] = a3 / A[i, i]
       # Compute x_0 separately
   x[0] = (b[0] - np.dot(A[0, 1:], x[1:])) / A[0, 0]
   return x
#def back_substitution
def LUF(a_matrix):
   n = len(a_matrix)
   L = np.identity(n)
   U = np.zeros((n, n))

   for k in range(0, n):
       for i in range(k + 1, n):
           if a_matrix[k, k] == 0.0:
               raise ValueError('Matrix is singular')
           left_augmented_matrix = a_matrix[i, k] / a_matrix[k, k]
           L[i, k] = left_augmented_matrix
```

```
            a_matrix[i, k + 1:n] = a_matrix[i, k + 1:n] - left_augmented_matrix
* a_matrix[k, k + 1:n]
            U[i, k:] = a_matrix[i, k:] - left_augmented_matrix * a_matrix[k,
k:]
        U[k, k:] = a_matrix[k, k:]
    return L, U

def lu_solve(L, U, b):
    y = back_substitution(L, b)
    x = back_substitution(U, y)
    return x

# defining matrices
A = np.array([[2.0, 4.0, 5.0], [7.0, 6.0, 5.0], [9.0, 11.0, 3.0]])
a_matrix = np.array([[2.0, 4.0, 5.0], [7.0, 6.0, 5.0], [9.0, 11.0, 3.0]])
b = np.array([3.0, 2.0, 1.0])
#Gauss elimination w/ no pivots
A1, x = gauss_elimination(A, b)
print(f'This is ref matrix{A1}')
#back sub
x1 = back_substitution(A, b)
# Lu Factorization
L, U = LUF(a_matrix)
print(f'L factor:{L}')
print(f'U factor{U}')
x2 = lu_solve(L, U, b)
print(f'Solution to Ax = b without pivoting:{x}')

n_values = [50, 100, 250, 500]
#Generate matrix A and vector b
for n in n_values:
    An = (5 * np.sqrt(n)) * np.eye(n) + np.random.normal(0, 1, size=(n, n))
    bn = np.random.normal(0, 1, size=(n,))
    #Solving gauss elimination


    Fork = gauss_elimination(An, bn)
    print(Fork)
    Spoon = back_substitution(An, bn)
    print(Spoon)
    L, U = LUF(An)
    print(L)
    print(U)
    x3 = lu_solve(L, U, bn)

import time

# initialize table
table = [['n', 'GE Error', 'LU Error']]
```

```
# loop over n values
for n in n_values:
    # generate matrix A and vector b
    An = (5 * np.sqrt(n)) * np.eye(n) + np.random.normal(0, 1, size=(n, n))
    bn = np.random.normal(0, 1, size=(n,))

    # Gauss elimination
    start = time.time()
    A1, x = gauss_elimination(An, bn)
    ge_error = np.linalg.norm(np.dot(An, x) - bn)
    ge_time = time.time() - start

    # LU factorization
    start = time.time()
    L, U = LUF(An)
    x2 = lu_solve(L, U, bn)
    lu_error = np.linalg.norm(np.dot(An, x2) - bn)
    lu_time = time.time() - start

    # add row to table
    table.append([n, f'{ge_error:.2e}', f'{lu_error:.2e}'])

    # print results
    print(f'n = {n}')
    print(f'GE Error: {ge_error:.2e}, Time: {ge_time:.4f} seconds')
    print(f'LU Error: {lu_error:.2e}, Time: {lu_time:.4f} seconds')
    print('')

# print table
for row in table:
    print('{:<10}{:<20}{:<20}'.format(*row))

import matplotlib.pyplot as plt

# initialize lists for GE and LU times
ge_times = []
lu_times = []

# loop over n values
for n in n_values:
    # generate matrix A and vector b
    An = (5 * np.sqrt(n)) * np.eye(n) + np.random.normal(0, 1, size=(n, n))
    bn = np.random.normal(0, 1, size=(n,))

    # Gauss elimination
    start = time.time()
    A1, x = gauss_elimination(An, bn)
    ge_time = time.time() - start
```

```
    # LU factorization
    start = time.time()
    L, U = LUF(An)
    x2 = lu_solve(L, U, bn)
    lu_time = time.time() - start

    # add times to lists
    ge_times.append(ge_time)
    lu_times.append(lu_time)

# plot times
plt.plot(n_values, ge_times, label='Gauss Elimination')
plt.plot(n_values, lu_times, label='LU Factorization')
plt.xlabel('n')
plt.ylabel('Time (seconds)')
plt.title('Time needed for solution for each n')
plt.legend()
plt.show()
```