Gaussian Elimination and LU Decomposition

Wanzhu Zheng

May 2023

1 Problem Description

This report demonstrates how to solve different systems of linear equations with Gaussian Elimination and LU Decomposition.

2 Results

Here we demonstrate the results of the two solving methods. To solve a system with Gaussian Elimination without pivoting, augment the matrix first so that it's in the form $[A|b]$ where A is some nxn matrix and b is some $nx1$ vector.

$$\begin{bmatrix} a & b & c & b_1 \\ d & e & f & b_2 \\ g & h & i & b_3 \end{bmatrix}$$

Then, find a factor $r = \frac{A_{j,i}}{A_{i,i}}$. We can reduce it to its upper-triangular form by performing row operations such as $(E_j - r \cdot E_i) \rightarrow (E_j)$ until it's in the form:

$$\begin{bmatrix} a' & b' & c' & b'_1 \\ 0 & e' & f' & b'_2 \\ 0 & 0 & i' & b'_3 \end{bmatrix}$$

For the LU Decomposition method, we solve for two matrices L, U such that $A = LU$, L is the lower-triangular matrix and U is the upper-triangular matrix. Then, use forward substitution to find the solutions using:

$$LY = b$$

$$UX = Y$$

to solve for X.

Based off of problem 3 in the second problem set using $b = [321]^T$, we get the following results:

- GE solutions: $[-0.255474450.138686130.59124088]$
- LUx solutions: $[-0.255474450.138686130.59124088]$

Notice how the two solutions are equal. This means that the matrix has unique solutions. The LU decomposition of matrix A.

L:

$$\begin{bmatrix} 1 & 0 & 0 \\ 3.5 & 1 & 0 \\ 4.5 & 0.875 & 1 \end{bmatrix}$$

U:

$$\begin{bmatrix} 2 & 4 & 5 \\ 0 & -8 & -12.5 \\ 0 & 0 & -8.5625 \end{bmatrix}$$

Now, we set a random matrix from the distribution $(5\sqrt{n})I + R$ 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. Then define b as a $(nx1)$ vector with random entries drawn from the same random distribution. The residuals for GE and LU for $n = 50, 100, 250, 500$ are:

n	GE	LU
50	3.1938e-15	8.2827e-16
100	3.7386e-15	1.9711e-15
250	1.2599e-14	4.4429e-15
500	2.3298e-14	9.3269e-15

We observe the runtime of Gaussian against LU Decomposition by timing how long it takes to solve the system of linear equations against the size of the matrix. We observe the results in a graph below:

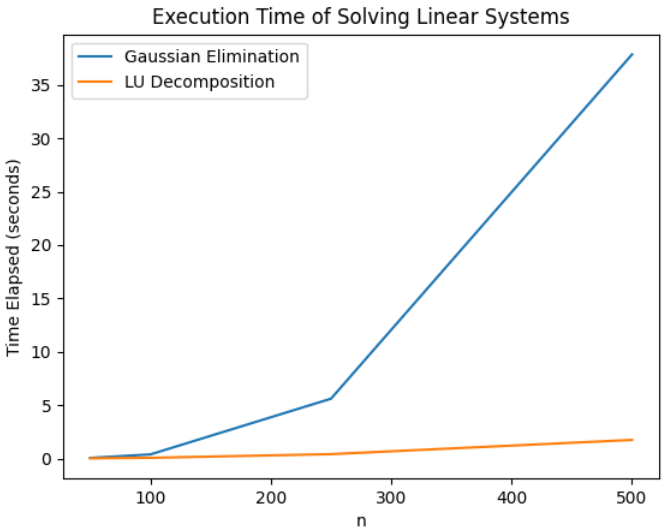

Figure 1. Plot of Time (seconds) against n

We see that the time it takes for Gaussian to solve the linear system is significantly longer compared to LU Decomposition. This makes sense because forward elimination is proportional to $O(\frac{n^3}{3})$, back substitution is proportional to $O(\frac{n^2}{2})$, and forward substitution is proportional to $\frac{n^2}{2}$ The computational time for the L, U matrices is $O(\frac{n^3}{3} + n(\frac{n^2}{2} + \frac{n^2}{2}) = \frac{4n^3}{3})$. Note that forward and backwards substitution needs to be done n times.

For Gaussian Elimination, the computational time is proportional to $\frac{n^3}{3} + n\frac{n^2}{2} = \frac{n^4}{3} + \frac{n^3}{2}$. Again, back substitution need to be done n times.

Thus, for a large n, for LU Decomposition, the computational time is proportional to $O(\frac{4n^3}{3})$, while for Gaussian Elimination, the computational time is proportional to $O(\frac{n^4}{3})$. So for large n, the ratio of the

computational time for Gaussian elimination to computational for LU Decomposition is $O(\frac{n^4}{3}/\frac{4n^3}{3} = \frac{n}{4})$. Thus, it will save more computational power to calculate using LU compared to GE.

When $A = R$, Gaussian seems to be less accurate compared to LU Decomposition because of rounding errors. This is due to the fact taht the matrix is no longer diagonally dominant.

3 Appendix

```
    import time
import numpy as np
import matplotlib.pyplot as plt


# gaussian elimination
def gaussian(A, b):
    n = len(b)
    B = np.copy(A)
    p = np.copy(b)
    for i in range(n-1):
        for j in range(i+1, n):
            r = B[j][i] / B[i][i]
            for k in range(i, n):
                B[j][k] -= r * B[i][k]
            p[j] -= r * p[i]
    return back_sub(B, p)


# backwards substitution
def back_sub(A, b):
    n = len(b)
    x = np.zeros(n)
    for i in range(n-1, -1, -1):
        x[i] = (b[i] - np.dot(A[i, i+1:], x[i+1:])) / A[i, i]
    return x


# LU decomposition
def lu(A):
    L = np.zeros_like(A)
    U = np.zeros_like(A)
    N = np.size(A, 0)

    for k in range(N):
        L[k, k] = 1
        U[k, k] = (A[k, k] - np.dot(L[k, :k], U[:k, k])) / L[k, k]
        for j in range(k + 1, N):
            U[k, j] = (A[k, j] - np.dot(L[k, :k], U[:k, j])) / L[k, k]
        for i in range(k + 1, N):
            L[i, k] = (A[i, k] - np.dot(L[i, :k], U[:k, k])) / U[k, k]
    return L, U


# forward substitution
def forward_sub(A, b):
```

```
    n = len(b)
    x = np.zeros(n)
    for i in range(n):
        x[i] = b[i] - np.dot(A[i, :i], x[:i]) / A[i][i]
    return x


# LUx = b
def lu_solver(A, b):
    L = lu(A)[0]
    U = lu(A)[1]
    y = forward_sub(L, b)
    x = back_sub(U, y)
    return x


# Q3
A = np.array([[2, 4, 5], [7, 6, 5], [9, 11, 3]], dtype=float)
b = np.array([3, 2, 1], dtype=float)

print(gaussian(A, b))
print(lu(A))
print(lu_solver(A, b))


def mat(n):
    A = 5 * np.sqrt(n) * np.eye(n) + np.random.normal(size=(n, n))
    return A


def sol(n):
    b = np.random.normal(size=(n, 1))
    return b


# GE
print(gaussian(mat(50), sol(50)))
print(gaussian(mat(100), sol(100)))
print(gaussian(mat(150), sol(250)))
print(gaussian(mat(200), sol(500)))

# LU
print(lu_solver(mat(50), sol(50)))
print(lu_solver(mat(100), sol(100)))
print(lu_solver(mat(150), sol(250)))
print(lu_solver(mat(200), sol(500)))


# residual errors
def ge_residual(n):
    A = mat(n)
    b = np.squeeze(sol(n))
    x = gaussian(A, b)
    P = np.dot(A, x)
```

```
    residual = np.linalg.norm(P - b, 2)
    return residual


def lu_residual(n):
    A = mat(n)
    b = np.squeeze(sol(n))
    x = lu_solver(A, b)
    P = np.dot(A, x)
    residual = np.linalg.norm(P - b, 2)
    return residual


print(ge_residual(50))
print(lu_residual(50))
print(ge_residual(100))
print(lu_residual(100))
print(ge_residual(250))
print(lu_residual(250))
print(ge_residual(500))
print(lu_residual(500))

# wall clock time
def clock(n, type=0):
    A = mat(n)
    b = sol(n)
    if type == 0:
        t0 = time.time()
        gaussian(A, b)
        t1 = time.time()
        total_time = t1 - t0
        return total_time
    if type == 1:
        t0 = time.time()
        lu_solver(A, b)
        t1 = time.time()
        total_time = t1 - t0
        return total_time


print(clock(50, 0))
print(clock(100, 0))
print(clock(50, 1))
print(clock(100, 0))
print(clock(100, 1))
print(clock(250, 0))
print(clock(250, 1))
print(clock(500, 0))
print(clock(500, 1))


# plotting
times_ge = np.empty((0, 0))
times_ge = np.append(times_ge, (clock(50, 0), clock(100, 0), clock(250, 0), clock(500, 0)))
```

```
times_lu = np.empty((0, 0))
times_lu = np.append(times_lu, (clock(50, 1), clock(100, 1), clock(250, 1), clock(500, 1)))

dims = [50, 100, 250, 500]
plt.plot(dims, times_ge)
plt.plot(dims, times_lu)
plt.xlabel("n")
plt.ylabel("Time Elapsed (seconds)")
plt.title("Execution Time of Solving Linear Systems")
plt.legend(["Gaussian Elimination", "LU Decomposition"], loc="upper left")
plt.show()
```