Homework-4

[] RREF = $\frac{4n^3}{3}$

In LU decomposition we know we could say that for each column from 1 to n-1 we perform:

1] We perform (n-k) divisions to compute the multipliers

2] We then update the remaining (n-k)x(n-k) submatrix, which requires $(n-k)^2$ multiplications and $(n-k)^2$ additions

The total number of operations can be expressed as:

$$\sum_{k=1}^{n-1} [(n-k) + 2(n-k)^2]$$

Let $i = n-k$

$$\sum_{k=1}^{n-1} [n + 2n^2]$$

We can approximate this sum using integrals

$$\int_0^n (x + 2x^2)dx = \int_0^n n\,dx + 2\int_0^n x^2\,dx$$

$$= \left[\frac{x^2}{2}\right]_0^n + 2\left[\frac{x^3}{3}\right]_0^n$$

$$= \frac{n^2}{2} + \frac{2n^3}{3}$$

For large n, the dominant term is $\frac{2n^3}{3}$

Solving linear systems cost

1] Forward substitutions: $\frac{n^2}{2}$ ops

2] Back substition: $\frac{n^2}{2}$ ops

Total cost for solving three linear systems

$$3 \times \left[\frac{n^2}{2} + \frac{n^2}{2}\right] = 3n^2 \quad = n^3$$

Total cost for large n is actually $\frac{2n^3}{3}$

2] 1000 problems $Ux = c$ for an 500×500 matrices per second

$\frac{n^2}{2}$ operations $= \frac{(500)^2}{2} = \frac{250000}{2}$

$= 125000$ ops/s

~~1000~~

Number of operations per second $= 1000 \times 125000$

$= 125,000,000$

$= 1.25 \times 10^8$

Gaussian Elimination number of operations for 5000

$\frac{2n^3}{3} = \frac{2(5000)^3}{3} = \frac{2(125000000000)}{3}$

$= \frac{250,000,000,000}{3}$

$= 8.333 \times 10^{10}$ operations

Time for doing gaussian elimination $= \frac{2}{3} \frac{25 \times 10^{10}}{3 \times 1.25 \times 10^8}$

$= \frac{25}{375} \frac{\times 10^{10}}{\times 10^8}$

$= \frac{25 \times 10^4}{375}$

$= \frac{10^4}{15}$

$= \frac{2000}{3} = 666.667$s

$\approx$ 11.1 minutes

Question 3:

```python
import numpy as np
from numpy.linalg import solve, norm, cond
from scipy import linalg

def hilbert_analysis(n_max):
    relative_error=0
    n=2
    while(relative_error<100):
        H = linalg.hilbert(n)
        x = np.ones(n)
        b = H @ x
        x_hat = solve(H, b)
        r = b - H @ x_hat
        delta_x = x - x_hat
        r_norm = norm(r, np.inf)
        error_norm = norm(delta_x, np.inf)
        relative_error = error_norm / norm(x, np.inf)
        cond_num = cond(H, np.inf)
        print(f"n = {n}")
        print(f"Residual norm: {r_norm:.2e}")
        print(f"Error norm: {error_norm:.2e}")
        print(f"Relative error: {relative_error}")
        print(f"Condition number: {cond_num:.2e}")
        print()

        if relative_error >= 1:
            print(f"Relative error reached 100% at n = {n}")
            break
        n+=1
hilbert_analysis(20)
```

Output:

```
PS C:\Users\Lenovo\CMU\21671-A> python q3.py
n = 2
Residual norm: 0.00e+00
Error norm: 6.66e-16
Relative error: 6.661338147750939e-16
Condition number: 2.70e+01

n = 3
Residual norm: 0.00e+00
Error norm: 9.99e-15
Relative error: 9.992007221626409e-15
Condition number: 7.48e+02

n = 4
Residual norm: 0.00e+00
Error norm: 1.95e-13
Relative error: 1.949551631241775e-13
Condition number: 2.84e+04

n = 5
Residual norm: 2.22e-16
Error norm: 3.10e-11
Relative error: 3.096567446903009e-11
Condition number: 9.44e+05
```

```
n = 6
Residual norm: 4.44e-16
Error norm: 4.15e-10
Relative error: 4.153275501295184e-10
Condition number: 2.91e+07

n = 7
Residual norm: 4.44e-16
Error norm: 1.90e-08
Relative error: 1.902250212904022e-08
Condition number: 9.85e+08

n = 8
Residual norm: 4.44e-16
Error norm: 2.35e-07
Relative error: 2.3511776769957748e-07
Condition number: 3.39e+10

n = 9
Residual norm: 2.22e-16
Error norm: 1.60e-05
Relative error: 1.5975441239879373e-05
Condition number: 1.10e+12

n = 10
Residual norm: 2.22e-16
Error norm: 8.15e-04
Relative error: 0.0008145607536313992
Condition number: 3.54e+13
```

```
n = 11
Residual norm: 4.44e-16
Error norm: 1.41e-02
Relative error: 0.01406078583249859
Condition number: 1.23e+15

n = 12
Residual norm: 2.22e-16
Error norm: 3.85e-01
Relative error: 0.38474542578766135
Condition number: 3.92e+16

n = 13
Residual norm: 8.88e-16
Error norm: 8.29e-01
Relative error: 0.8285697266507566
Condition number: 8.53e+17

n = 14
Residual norm: 8.88e-16
Error norm: 7.56e+00
Relative error: 7.560665531874491
Condition number: 1.69e+18

Relative error reached 100% at n = 14
```

Question 4:

```python
import numpy as np
from time import time
from numpy import eye, zeros_like
from numpy.random import rand
import matplotlib.pyplot as plt
def lu_factorization(A):
    n = A.shape[0]
    L = eye(n)
    U = A.copy()
    for k in range(n-1):
        for i in range(k+1, n):
            L[i, k] = U[i, k] / U[k, k]
            U[i, k:] -= L[i, k] * U[k, k:]
    return L, U
def forward_substitution(L, b):
    n = L.shape[0]
    y = zeros_like(b, dtype=float)
    for i in range(n):
        y[i] = b[i] - np.dot(L[i, :i], y[:i])
    return y
def backward_substitution(U, y):
    n = U.shape[0]
    x = zeros_like(y, dtype=float)
    for i in range(n-1, -1, -1):
        x[i] = (y[i] - np.dot(U[i, i+1:], x[i+1:])) / U[i, i]
    return x
def solve_linear_system(A, b):
    L, U = lu_factorization(A)
    y = forward_substitution(L, b)
    x = backward_substitution(U, y)
    return x
```

```
def time_lu_solve(n):
    A = rand(n, n)
    b = rand(n)
    start_time = time()
    L, U = lu_factorization(A)
    lu_time = time() - start_time
    start_time = time()
    y = forward_substitution(L, b)
    x = backward_substitution(U, y)
    solve_time = time() - start_time
    return lu_time, solve_time
A_test = np.array([[1, 1, 0], [2, 1, -1], [3, -1, -1]], dtype=float)
b_test = np.array([2, 1, 1], dtype=float)
x_test = solve_linear_system(A_test, b_test)
print("Test solution:", x_test)
print("Verification:", np.allclose(A_test @ x_test, b_test))
n_values = np.logspace(2, 4.39794000897, 10, dtype=int)
lu_times = []
solve_times = []
for n in n_values:
    lu_time, solve_time = time_lu_solve(n)
    lu_times.append(lu_time)
    solve_times.append(solve_time)
    print(f"n = {n}: LU factorization time = {lu_time:.6f}s, Solve time = {solve_time:.6f}s")
plt.figure(figsize=(10, 6))
plt.loglog(n_values, lu_times, 'bo-', label='LU Factorization')
plt.loglog(n_values, solve_times, 'ro-', label='Triangular Solve')
plt.xlabel('Matrix Size (n)')
plt.ylabel('Time (seconds)')
plt.title('LU Factorization and Triangular Solve Times')
plt.legend()
plt.grid(True)
plt.savefig('lu_solve_times.png')
plt.show()
```

Output:

```
PS C:\Users\Lenovo\CMU\21671-A> python q4.py
Test solution: [1.33333333 0.66666667 2.33333333]
Verification: True
n = 100: LU factorization time = 0.007000s, Solve time = 0.000000s
n = 184: LU factorization time = 0.025740s, Solve time = 0.000000s
n = 341: LU factorization time = 0.091369s, Solve time = 0.000000s
n = 629: LU factorization time = 0.349710s, Solve time = 0.002000s
n = 1163: LU factorization time = 1.319506s, Solve time = 0.003018s
n = 2148: LU factorization time = 5.773919s, Solve time = 0.005374s
n = 3968: LU factorization time = 28.925834s, Solve time = 0.013260s
n = 7329: LU factorization time = 156.265233s, Solve time = 0.141809s
n = 13536: LU factorization time = 1270.698444s, Solve time = 0.466932s
n = 25000: LU factorization time = 12516.412250s, Solve time = 10.301320s
```