Assignment 3

Arifur Rahman
400300356
Feb 11, 2023

```python
from tabulate import tabulate
import numpy as np

def generateHb(n):
    H = []
    for i in range(n):
        row = []
        for j in range(n):
            row.append(1/(i + j + 1))
        H.append(row)

    x = [1 for i in range(n)]
    b = [0 for i in range(n)]
    for i in range(n):
        for j in range(n):
            b[i] += H[i][j] * x[j]

    return (H, b)

def forward_substitution(L, b):
    n = len(b)
    x = [[0.0] * n for i in range(n)]
    for i in range(n):
        if L[i][i] == 0.0:
            return None
        sum = 0
        for j in range(i):
            sum += L[i][j] * x[j]
        x[i] = (b[i] - sum) / L[i][i]
    return x

def backward_substitution(U, b):
    n = len(b)
    x = [[0.0] * n for i in range(n)]
    for i in range(n - 1, -1, -1):
        if U[i][i] == 0.0:
            return None
        sum = 0.0
        for j in range(i + 1, n):
            sum += U[i][j] * x[j]
        x[i] = (b[i] - sum) / U[i][i]
    return x
```

```python
def upper_triangular_matrix_U(k, n, L, U, A):
    # Get the upper triangular matrix U
    for j in range(k, n):
        sum = 0.0
        for p in range(k):
            sum += L[k][p] * U[p][j]
        U[k][j] = A[k][j] - sum
    # print(f"upper triangular matrix step {k+1}: {U}")

def lower_triangular_matrix_L(k, n, L, U, A):
    # Get the lower triangular matrix L
    for i in range(k + 1, n):
        sum = 0.0
        for p in range(k):
            sum += L[i][p] * U[p][k]
        L[i][k] = (A[i][k] - sum) / U[k][k]
    # print(f"Lower triangular matrix step {k+1}: {L}\n")

def LU_decomposition(A):
    n = len(A)
    L = [[0.0] * n for i in range(n)]
    U = [[0.0] * n for i in range(n)]

    for k in range(n):
        if A[k][k] == 0:
            return None

        L[k][k] = 1
        upper_triangular_matrix_U(k, n, L, U, A)
        lower_triangular_matrix_L(k, n, L, U, A)

    return L, U

def GaussElimination(A, b):
    n = len(A)
    for i in range(n):
        for j in range(i + 1, n):
            m = A[j][i] / A[i][i]
            b[j] = b[j] - m * b[i]
            for k in range(n):
                A[j][k] = A[j][k] - m * A[i][k]
    return A, b


def gauss_elimination(A, b):
    L, U = LU_decomposition(A)
    y = forward_substitution(L, b)
    x = backward_substitution(U, y)
    return x
```

```python
def getAllCalculatedValues(H, b, x_hat, x_true):
    residual = b - np.dot(H, x_hat)
    x2_norm = np.linalg.norm(x_hat, ord=2)
    deltaX = np.subtract(x_hat, x_true)
    residual_norm = np.linalg.norm(residual, np.inf)
    deltaX_2Norm = np.linalg.norm(deltaX, ord=2)
    xRE_norm = np.divide(deltaX_2Norm, x2_norm)
    xRE_percentage = np.divide(deltaX_2Norm, x2_norm) * 100
    return residual, deltaX, x2_norm, xRE_norm, xRE_percentage, residual_norm,
deltaX_2Norm

def condition_number(H):
    return np.linalg.cond(H, np.inf)

def toPrecisionString(val, pre):
    return np.format_float_positional(np.float32(val), unique=False,
precision=pre)


if __name__ == '__main__':
    # Generate Hilbert Matrix
    print("")
    print("========================================= Generate Hilbert Matrix
========================================================================|")
    print(f"|N            |Error%          |Residual/||r||_inf         |Cond(H)
  |")
    print("======================================================================
====================================================================|")
    error_Percentage = []
    for n in range(2, 20):
        H, b = generateHb(n)
        x = np.ones(n)
        x_hat = gauss_elimination(H, b)
        res, deltaX, x2_norm, xRE_norm, xRE_percentage, res_norm, deltaX_2Norm =
getAllCalculatedValues(H, b, x_hat, x)
        cond = condition_number(H)
        error_Percentage.append(xRE_percentage)
        if xRE_percentage > 100:
           break;
        else:
            print(f"n = {n}     | error% = {xRE_percentage}           | residual
= {res_norm}                    | Cond(H) = {toPrecisionString(cond, 2)}")
    print("")
```

Output:

N	Error%	Residual/||r||_inf	Cond(H)
n = 2	5.6610488670036755e-14%	0	27.00
n = 3	1.0181417058872435e-12%	2.220446049250313e-16	748.00
n = 4	3.777949315939031e-11%	2.220446049250313e-16	28375.00
n = 5	1.553022741749636e-10%	2.220446049250313e-16	943656.00
n = 6	2.5167853651689557e-8%	2.220446049250313e-16	29070280.00
n = 7	8.425196637989275e-7%	2.220446049250313e-16	985194880.00
n = 8	0.000025057255617154207%	1.1102230246251565e-16	33872791552.00
n = 9	0.0008608677939788814%	2.220446049250313e-16	1099652005888.00
n = 10	0.022377310679996314%	4.440892098500626e-16	35356846063616.00
n = 11	0.46409471707846783%	2.220446049250313e-16	1234532783095808.00
n = 12	9.545766814455332%	4.440892098500626e-16	38865448085194300.00
n = 13	306.54677556167655%	2.220446049250313e-16	741653484333594900.00