Assignment 3

Arifur Rahman

400300356

Feb 11, 2023

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

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

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