LAA Assignment

Himanshu, MDS202327

Driver Code

In [ ]:
```
from helper_functions import *
import pandas as pd
```

In [ ]:
```
tol = 10**(-10)
results = []
for i in range(2,11):
    A = hilbertMatrix(i)
    b = 20*np.random.random_sample(size=i)-20
    results.append([i, luSolveNorm(A,b, 'GEPP', tol,i), luSolveNorm(A,b, 'GERP', tol,i),
                     luSolveNorm(A,b, 'GECP', tol,i),scipyLUNorm(A,b),
                     luSolveNorm(A,b,'Cholesky',tol,i), scipyCholNorm(A,b)])
df = pd.DataFrame(results, columns = ['Order', 'GEPP', 'GERP', 'GECP', 'ScipyLU', 'Cholesky', 'ScipyChol'])
```

In [ ]:
```
# LU Norms Comparison
df
```

Out[ ]:

	Order	GEPP	GERP	GECP	ScipyLU	Cholesky	ScipyChol
0	2	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	3.061750	3.552714e-15
1	3	3.746768e-14	3.746768e-14	3.746768e-14	3.746768e-14	10.644229	1.344058e-13
2	4	3.330136e-13	3.330136e-13	3.330136e-13	8.608002e-13	6.765263	7.429356e-13
3	5	4.482813e-11	6.290807e-11	7.787435e-11	8.103213e-11	10.394742	5.818810e-11
4	6	6.692461e-10	1.009176e-09	5.754911e-10	8.898829e-10	17.967841	7.384851e-10
5	7	2.373236e-08	2.234370e-08	5.053934e-08	2.917992e-08	13.547332	3.123814e-08
6	8	1.622581e-06	1.019623e-06	2.446209e-06	1.375434e-06	27.780372	1.612921e-06
7	9	-1.000000e+00	-1.000000e+00	4.028373e-05	8.150863e-05	23.939432	6.917792e-05
8	10	-1.000000e+00	-1.000000e+00	-1.000000e+00	4.743223e-04	27.324559	4.252664e-04

In [ ]:
```
np.random.seed(21)
results = []
result2 = []
for i in [2,5,10,20,50,100,200,500,700,1000]:
    b=np.random.rand(i,1)*1000
    b_delta=np.random.rand(i,1)*(1e-12)
    A = hilbertMatrix(i)
    results.append([i,condNum(A,b,b_delta,'GEPP',i),condNum(A,b,b_delta,'GERP',i),
                    condNum(A,b,b_delta,'GECP',i)])
for j in range(1,11):
    b=np.random.rand(j,1)*1000
    b_delta=np.random.rand(j,1)*(1e-12)
    A = hilbertMatrix(j)
    result2.append(condNum(A,b,b_delta,'Cholesky',j))
df = pd.DataFrame(results, columns = ['Order', 'GEPP', 'GERP', 'GECP'])
df['Cholesky'] = result2
```

In [ ]:
```
# Bound on conditional number
df
```

Out[ ]:

	Order	GEPP	GERP	GECP	Cholesky
0	2	5.663295e-01	0.566330	0.566330	1.011436
1	5	1.347086e+00	1.163510	1.202622	1.012774
2	10	0.000000e+00	0.000000	0.547353	0.992819
3	20	1.654403e+03	7.474145	1.976511	2.753290
4	50	9.522863e+06	737.757323	5.622611	0.191970
5	100	2.582011e+08	2465.640401	2.263894	0.255180
6	200	1.581160e+10	78840.547729	3.224121	0.502048
7	500	5.000193e+11	51065.531522	7.157211	2.326155
8	700	1.196713e+12	90753.598613	11.012589	8.702383
9	1000	2.900402e+12	778796.452293	18.476805	1.566430

Helper functions

In [ ]:

```
import numpy as np
from scipy.linalg import lu_factor, lu_solve, cholesky


def hilbertMatrix(n):
    """
    Generates an n x n Hilbert matrix
    """
    H = np.zeros((n, n))
    for i in range(1, n + 1):
        for j in range(1, n + 1):
            H[i - 1, j - 1] = 1 / (i + j - 1)
    return H

def luGEPP(A,tol):
    """
    LU decomposition with partial pivoting
    """
    n = len(A)
    A = A.astype(float)
    P = np.identity(n, dtype=float)
    L = np.zeros((n,n),dtype=float)
    U = np.copy(A)
    for i in range(n):
        max_pivot = abs(U[i:,i])
        max_row = np.argmax(max_pivot) + i

        if abs(U[max_row, i]) < tol:
            P[0,0] = -1
            return (P,L,U)

        U[[i, max_row]] = U[[max_row, i]]
        P[[i, max_row]] = P[[max_row, i]]
        L[[i, max_row]] = L[[max_row, i]]

        for j in range(i+1, n):
            L[j,i] = U[j,i] / U[i,i]
            U[j,i:] -= L[j,i] * U[i,i:]

    np.fill_diagonal(L, 1.0)
    return P, L, U


def luGERP(A,tol):
    """
    LU decomposition with rook pivoting
    """
    n = len(A)
    A = A.astype(float)
    Pl = np.identity(n, dtype=float)
    Pr = np.identity(n, dtype=float)
    L = np.zeros((n,n),dtype=float)
    U = np.copy(A)
    for i in range(n):
        pivotColInd = np.argmax(abs(U[i:,i]))+i
        pivotRowInd = np.argmax(abs(U[i:,i]))+i

        if abs(U[pivotColInd, i]) < abs(U[i, pivotRowInd]):
```

```python
            pivotInd = pivotRowInd
            if abs(U[i, pivotRowInd]) < tol:
                Pl[0,0] = -1
                return (Pl,Pr,L,U)

            Pr[:, [pivotInd, i]] = Pr[:, [i, pivotInd]]
            U[:, [pivotInd,i]] = U[:, [i, pivotInd]]

        else:
            if abs(U[pivotColInd, i]) < tol:
                Pl[0,0] = -1
                return (Pl,Pr,L,U)
            pivotInd = pivotColInd

            Pl[[pivotInd, i]] = Pl[[i, pivotInd]]
            L[[pivotInd, i], :i] = L[[i, pivotInd], :i]
            U[[pivotInd, i]] = U[[i, pivotInd]]

        for j in range(i+1, n):
            L[j, i] = U[j, i]/U[i, i]
            U[j, i:] -= L[j, i] * U[i, i:]
    return (Pl,Pr,L,U)

def luGECP(A,tol):
    """
    LU decomposition with complete pivoting
    """
    n = len(A)
    A = A.astype(float)
    Pl = np.identity(n, dtype=float)
    Pr = np.identity(n, dtype=float)
    L = np.identity(n,dtype=float)
    U = np.copy(A)

    for i in range(n-1):

        max = np.unravel_index(np.argmax(abs(U[i:,i:])), U[i:,i:].shape)
        rowInd = max[0] + i
        colInd = max[1] + i
        if abs(U[rowInd,colInd]) < tol:
            Pl[0,0] = -1
            return (Pl,Pr,L,U)

        Pl[[rowInd, i]] = Pl[[i, rowInd]]
        Pr[:, [colInd, i]] = Pr[:, [i, colInd]]
        L[[rowInd, i], :i] = L[[i, rowInd], :i]
        U[:, [colInd, i]] = U[:, [i, colInd]]
        U[[rowInd, i]] = U[[i, rowInd]]

        for j in range(i+1, n):
            L[j, i] = U[j, i]/U[i, i]
            U[j, i:] -= L[j, i] * U[i, i:]
    return (Pl,Pr,L,U)


def cholesky(A, tol):
    """
    Cholesky decomposition
    """
    n = len(A)
    mid = A.copy().astype(np.float64)
    B = np.eye(n, dtype=np.float64)
    P = np.eye(n, dtype=np.float64)
    for i in range(n):
        pivot = np.argmax(np.diag(mid[i:,i:])) + i
        mid[[i,pivot],:] = mid[[pivot,i],:]
        mid[:,[i,pivot]] = mid[:,[pivot,i]]
        P[[i,pivot]] = P[[pivot,i]]
        B[[i,pivot],:i] = B[[pivot,i],:i]
        alpha = np.sqrt(mid[i,i])
        if abs(alpha)<tol:
            P[0,0] = -1
            return P,B
        wAlpha = mid[i,i+1:]/alpha
        B[i,i] = alpha
        B[i+1:,i] = wAlpha
        mid[i+1:,i+1:] -= [[i*j for i in wAlpha] for j in wAlpha ]
    return P,B

def forward_sub(P, L, b):
    """
    Forward substitution
```

```python
    """
    Pb = np.dot(P, b)
    n = len(L)
    y = np.zeros(n, dtype=float)
    for i in range(n):
        y[i] = Pb[i] - sum(L[i,:i] * y[:i])
    return y

def backward_sub(U, Q, y):
    """
    Backward substitution
    """

    n = len(y)
    x = np.zeros(n)
    x[n - 1] = y[n - 1] / U[n - 1, n - 1]
    for i in range(n - 2, -1, -1):
        x[i] = y[i]
        for j in range(i + 1, n):
            x[i] -= U[i, j] * x[j]
        x[i] /= U[i, i]
    return Q@x

def luSolve(Pl, Pr, L, U, b):
    """
    LU Solver using forward and backward substitution
    """
    y = forward_sub(Pl, L, b)
    x = backward_sub(U, Pr, y)
    return x


def luSolveNorm(A, b, solver, tol, i):
    """
    Norm Ax-b using various solvers
    """
    if solver == 'GEPP':
        P, L, U = luGEPP(A, tol)
        if P[0,0] == -1:
            return -1
        else:
            I = np.identity(i, dtype=float)
            x = luSolve(P, I, L, U, b)
            return np.linalg.norm(A@x-b)

    elif solver == 'Cholesky':
        P, L = cholesky(A, tol)
        if P[0,0] == -1:
            return -1
        else:
            x = luSolve(P, P.T, L, L.T, b)
            return np.linalg.norm(A@x-b)

    elif solver == 'GERP':
        Pl, Pr, L, U = luGERP(A, tol)
        if Pl[0,0] == -1:
            return -1
        else:
            x = luSolve(Pl, Pr, L, U, b)
            return np.linalg.norm(A@x-b)

    elif solver == 'GECP':
        Pl, Pr, L, U = luGECP(A, tol)
        if Pl[0,0] == -1:
            return -1
        else:
            x = luSolve(Pl, Pr, L, U, b)
            return np.linalg.norm(A@x-b)


def condNum(A, b, b_delta, solver, i):
    """
    Condition number using various solvers
    """
    if solver == 'GEPP':
        P, L, U = luGEPP(A, 0)
        Q = np.eye(i,dtype="float")

    elif solver == 'Cholesky':
        P, L = cholesky(A, 0)
        Q = P.T
        U = L.T
```

```python
    elif solver == 'GERP':
        P,Q,L,U = luGERP(A,0)

    elif solver == 'GECP':
        P,Q,L,U = luGECP(A,0)

    x = luSolve(P,Q,L,U,b)
    x_delta = luSolve(P,Q,L,U,b+b_delta)
    return np.linalg.norm(x_delta-x,ord=2)*np.linalg.norm(b,ord=2)/(np.linalg.norm(b_delta,ord=2)*(np.linalg.norm(x,ord=2



def scipyLUNorm(A,b):
    """
    Evaluates the norm of the residual of the LU factorization
    """
    lu, piv = lu_factor(A)
    exp_x = lu_solve((lu,piv),b)
    return np.linalg.norm(A@exp_x-b)

def scipyCholNorm(A, b):
    """
    Evaluates the norm of the residual of the Cholesky factorization
    """
    L = np.linalg.cholesky(A)
    y = np.linalg.solve(L, b)
    x = np.linalg.solve(L.T, y)
    return np.linalg.norm(A@x-b)
```