APPM 4600 Lab 14

5 December 2024

The code for this lab can be seen at the end of this document, or on github here.

1 Solving Square Systems

The time taken for the two solution techniques with different size matrices is given in the table below.

N	`scipy.linalg.solve` [ms]	LU factorization [ms]	LU Solve [ms]
100	2.557516098022461	0.1647472381591797	0.0388622839355469
500	13.90385627746582	2.142190933227539	0.10395050048828125
1000	19.914865493774414	10.042428970336914	0.3666877746582031
2000	80.60383796691895	43.37739944458008	2.206087112426758
4000	489.3050193786621	353.6355495452881	6.650209426879883
5000	789.3681526184082	638.4170055389404	9.52768325805664

We find that LU factorization and solving is always faster than using the built in solve function, which is unexpected. This may be an artifact from how timing was measured.

```
#! /usr/bin/env python3

import matplotlib.pyplot as plt
import numpy as np
import numpy.linalg as la
import scipy.linalg as scila
import time


def question3_2_2(N):

    ''' create  matrix for testing different ways of solving a square
    linear system '''

    '''' N = size of system '''
    print(str(N) + " & ", end="")
    ''' Right hand side '''
    b = np.random.rand(N,1)
    A = np.random.rand(N,N)


    t = time.time()
    x1 = scila.solve(A,b)
    e = time.time()
    print(str((e-t)*1000) + " & ", end="")

    t = time.time()
    lu, piv = scila.lu_factor(A)
    e = time.time()
    print(str((e-t)*1000) + " & ", end="")

    t = time.time()
    x2 = scila.lu_solve((lu, piv), b)
    e = time.time()
    print(str((e-t)*1000) + " \\\\")

# test many right hand side solves
def question3_2_3(N):
```

```
    A = np.random.rand(N, N)

    print(str(N) + " & ", end="")

    t = time.time()
    for i in range(100):
        b = np.random.rand(N, 1)
        x1 = scila.solve(A, b)
    e = time.time()
    print(str(e-t) + " & ", end="")

    t = time.time()
    lu, piv = scila.lu_factor(A)
    for i in range(100):
        b = np.random.rand(N, 1)
        x2 = scila.lu_solve((lu, piv), b)
    e = time.time()
    print(str(e-t) + " \\\\")

def question3_4_1():
    ''' Create an ill-conditioned rectangular matrix '''
    N = 10
    M = 5
    A = create_rect(N,M)
    b = np.random.rand(N,1)



def create_rect(N,M):
    ''' this subroutine creates an ill-conditioned rectangular matrix'''
    a = np.linspace(1,10,M)
    d = 10**(-a)

    D2 = np.zeros((N,M))
    for j in range(0,M):
       D2[j,j] = d[j]

    '''' create matrices needed to manufacture the low rank matrix'''
    A = np.random.rand(N,N)
    Q1, R = la.qr(A)
    test = np.matmul(Q1,R)
    A =    np.random.rand(M,M)
    Q2,R = la.qr(A)
    test = np.matmul(Q2,R)

    B = np.matmul(Q1,D2)
    B = np.matmul(B,Q2)
    return B


if __name__ == '__main__':
    # run the drivers only if this is called from the command line
    for N in [100, 500, 1000, 2000, 4000, 5000]:
        question3_2_3(N)
```