Sebastiaan_Bye_Julius_Watenaar_homework2

November 14, 2021

1 Homework Set 2

2 Exercise 0

Julius Wantenaar 11339055

Sebastiaan Bye 14084880

[3]:
```
import numpy as np
import scipy.linalg as la
import matplotlib.pyplot as plt
import random
```

3 Exercise 1

The goal of this problem is to show that apparently harmless looking systems of linear equations may be very difficult to solve. Some functions that may be useful are `numpy.triu`, `numpy.tril`, `numpy.eye`, `random.randrange`. ## (a) Generate an $n \times n$ matrix B with random integer elements in the range $b_{ij} \in [-10, 10]$. Choose for instance $n = 20$.

[4]:
```
n = 20
B = np.rint(np.random.rand(n,n)*20 - 10)
```

The cell above generates a $20 * 20$ matrix with integer elements in the range $b_{ij} \in [-10, 10]$

3.1 (b)

Remove the diagonal of B, save the upper triangular part in U and the lower triangular part in L, and put ones on the diagonals $l_{ii} = u_{ii} = 1$.

[5]:
```
L = np.zeros((n,n))
U = np.zeros((n,n))
for row in range(n):
    for column in range(n):
        if row == column:
            # diagonal
            B[row][column] = 0

for row in range(n):
```

```
    for column in range(n):
        if row < column:
            # lower
            L[row][column] = B[row][column]
        elif row > column:
            #upper
            U[row][column] = B[row][column]

np.fill_diagonal(L, 1)
np.fill_diagonal(U, 1)
```

Assuming that removing the diagonal of B means setting the elements to 0, the first loop removes the diagonal of B. The second loop saves the elements of the triangular matrices and the final command sets the diagonals to 1.

3.2 (c)

Compute $A = L \cdot U$. What is the value of $\det(A)$ and why? Compute the determinant using the appropriate python command and confirm your prediction. In case that you have doubts about the result, compute separately $\det(L)$ and $\det(U)$.

[6]:
```
A = np.matmul(L,U)
np.linalg.det(A)
```

[6]: 804835905.521148

The above cell computes the determinant of A which is in this case 804835905.521148. This appears to be wrong. The determinant should be 1 as the determinants of L and U are one and $\det(A) = \det(L) * \det(U)$. This happens as finding the determinant is prone to error.

[7]:
```
print(np.linalg.det(L),np.linalg.det(U))
```

1.0 0.9999986129845544

The cell above computes the determinates of L and U seperately. They are respectivly 1.0 and 0.9999986129845544. This result is in line with the theory as the determinant of a triangular matrix is the product the diagonal. In this case the elements of the diagonal are all 1 hence, this result is correct.

[8]:
```
np.linalg.slogdet(A)
```

[8]: (1.0, 20.506148970530703)

The above cell uses the natural log to compute the determinant of A. Now the determinant is 1.

3.3 (d)

Choose now an exact solution, for instance $x_e =$ `numpy.ones(n)`, and compute the corresponding right hand side $b = Ax_e$.

[9]:
```python
b = np.matmul(A, np.ones(n))

b
```

[9]:
```
array([ 636.,  331.,  320.,  326.,  669., -406.,  325.,  -86.,  548.,
        156., -251., -236., -422., -349.,  -86.,   62., -189., -297.,
       -136.,  -27.])
```

The above cell generates a vector with all elements one and calcaultes what b is. The resulting vector b is

$$\begin{bmatrix} 636. \\ 331. \\ 320. \\ 326. \\ 669. \\ -406. \\ 325. \\ -86. \\ 548. \\ 156. \\ -251. \\ -236. \\ -422. \\ -349. \\ -86. \\ 62. \\ -189. \\ -297. \\ -136. \\ -27. \end{bmatrix}$$

3.4 (e)

Solve $Ax = b$ using `scipy.linalg.lu_factor` and `scipy.linalg.lu_solve` and compare the solution with the exact x_e.

[10]:
```python
lu, piv = la.lu_factor(A)
```

[11]:
```python
la.lu_solve((lu, piv), b)
```

[11]:
```
array([ 1.        ,  1.        ,  1.        ,  1.        ,  1.        ,
        0.99999998,  0.99999983,  0.99999869,  0.99999922,  0.99999149,
        1.00003146,  1.00018874,  0.99983995,  0.99855031,  0.99709845,
        1.02101795,  1.10214885,  1.86904292, -1.00410593,  6.41994475])
```

Using the suggested libraries, the solution of x_e is

$$\begin{bmatrix} 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 0.99999998 \\ 0.99999983 \\ 0.99999869 \\ 0.99999922 \\ 0.99999149 \\ 1.00003146 \\ 1.00018874 \\ 0.99983995 \\ 0.99855031 \\ 0.99709845 \\ 1.02101795 \\ 1.10214885 \\ 1.86904292 \\ -1.00410593 \\ 6.41994475 \end{bmatrix}$$

3.5 (f)

Explain the bad results by computing the condition number of A.

[12]:
```
np.linalg.cond(A)
```

[12]: `1.2260241100456064e+18`

The condition number is very high. It is $1.2260241100456064e + 18$. This means that the matrix is very sensitive to input errors and that can be seen in the last 5 entries of the vector in exercise (e).

4 Exercise 2

4.1 (a)

Verify that

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I_p & 0 \\ CA^{-1} & I_q \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & D - CA^{-1}B \end{bmatrix} \begin{bmatrix} I_p & A^{-1}B \\ 0 & I_q \end{bmatrix}$$

The first step:

$$\begin{bmatrix} I_p & 0 \\ CA^{-1} & I_q \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & D - CA^{-1}B \end{bmatrix} = \begin{bmatrix} (I_P * A) + (0 * 0) & (I_p * 0) + (0 * D - CA^{-1}B) \\ (CA^{-1} * A) + (I_p * 0) & (CA^{-1} * 0) + (I_p * D - CA^{-1}B) \end{bmatrix} = \begin{bmatrix} A & 0 \\ C & D - CA^{-1}B \end{bmatrix}$$

The second step:

$$\begin{bmatrix} A & 0 \\ C & D - CA^{-1}B \end{bmatrix} \begin{bmatrix} I_p & A^{-1}B \\ 0 & I_q \end{bmatrix} = \begin{bmatrix} (A * I_p) + 0 * 0 & (A * A^{-1}B) + (0 * I_p) \\ (C * I_p) + (0 * (D - CA^{-1}B)) & (C * A^{-1}B) + (D - CA^{-1}B * I_p) \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

Thus

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I_p & 0 \\ CA^{-1} & I_q \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & D - CA^{-1}B \end{bmatrix} \begin{bmatrix} I_p & A^{-1}B \\ 0 & I_q \end{bmatrix}$$

.

4.2 (b)

Describe how a system $Mx = b$, with x and b in $\mathbb{R}^{p+q}$, can be solved by applying matrix-vector products with C and B and solves with A and $(D - CA^{-1}B)$.

$$Mx = b$$

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} x = b$$

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

From the question it is known that A is invertible. Hence, x_1 can be eliminated by

$$Ax_1 + Bx_2 = b_1$$

$$x_1 = A^{-1}(b_1 - Bx_2).$$

For the second equation

$$Cx_1 + Dx_2 = b_2$$

$$C(A^{-1}(b_1 - Bx_2)) + Dx_2 = b_2$$

$$(D - CA^{-1})x_2 = b_2 - CA^{-1}b_1.$$

Now let $S = D - CA^{-1}B$, then

$$x_2 = S^{-1}(b_2 - CA^{-1}b_1).$$

Substituting this expression into the expression for x_1 gives

$$x_1 = A^{-1}(b_1 - B(S^{-1}(b_2 - CA^{-1}b_1)))$$

$$x_1 = (A^{-1} + A^{-1}BS^{-1}CA^{-1})b_1 - A^{-1}BS^{-1}b_2$$

.

Now $Mx = b$ can be solved with x, b by

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} A^{-1} + A^{-1}BS^{-1}CA^{-1} & -A^{-1}BS^{-1} \\ -S^{-1}CA^{-1} & S^{-1} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

4.3 (c)

What is the cost, to highest order, of LU-factorizing A and of computing and LU-factorizing $D - CA^{-1}B$?

The cost to the highest order of LU-factorizing A is as follows:

Eliminiating the first column requires p additions and p multiplications on $p-1$ rows. Thus $2p(p-1)$ operations.

For the second column there are $p-1$ additions and $p-1$ multiplications on $p-2$ rows. Thus $2(p-1)(p-2)$ operations.

Thus $\sum_{i=1}^{p} 2(p-1)(p-i+1)$ operations per column. Now let $j = p - i + 1$.

$2\sum_{j=0}^{p-1} j(j+1)$

$2\sum_{j=0}^{p-1} (j^2 + j)$

$2(\frac{p^3}{3} - \frac{p}{3})$

$\frac{2p^3}{3} - \frac{2p}{3}$.

So in the worst case the cost of LU-factorizing A is $\frac{2p^3}{3} - \frac{2p}{3}$. So the cost to the highest order of LU-factorizing A is $\frac{2p^3}{3}$.

Computing $D - CA^{-1}B$. Inverting a matrix has a time-complexity of $O(n^3)$, in this case $O(p^3)$.

First, compute CA^{-1}. C is $q * p$ and A^{-1} is $p * p$, which computing the matrix multiplication is of order $O(pqp) = O(p^2q)$. This is because adding and multiplying p terms for each pq element. The resulting matrix is of size $q * p$.

Second, compute CA^{-1} times B which is $q * p$ times $p * q$. This has a time complexity of $O(qpq) = O(q^2p)$. This is because adding and multiplying q terms for each qp element. The resulting matrix is of size $q * q$.

The final operation is subtracting a $q*q$ matrix from a $q*q$ matrix which is a total of q^2 operations.

The total time complexity is $O(p^3 + p^2q + q^2p + q^2)$.

The LU-factorizing $D - CA^{-1}B$ cost follows the same logical as the LU-factorizing cost of A

Thus $\sum_{i=1}^{q} 2(q-1)(q-i+1)$ operations per column. Now let $j = q - i + 1$.

$2\sum_{j=0}^{q-1} j(j+1)$

$2\sum_{j=0}^{q-1}(j^2+j)$

$2(\frac{q^3}{3}-\frac{q}{3})$

$\frac{2q^3}{3}-\frac{2q}{3}$.

So in the worst case the cost of LU-factorizing $D-CA^{-1}B$ is $\frac{2q^3}{3}-\frac{2q}{3}$. So the cost to the highest order of LU-factorizing $D-CA^{-1}B$ is $\frac{2q^3}{3}$.

[ ]: