

Tarea4

Métodos Numéricos

Benjamin Rivera

27 de septiembre de 2020

Índice

1. Tarea 3	2
1.0.1. Como ejecutar	2
1.1. Ejercicio 1	3
1.2. Ejercicio 2	7
1.3. Segundo	9

1. Tarea 3

Tarea 4 de Benjamín Rivera para el curso de **Métodos Numéricos** impartido por Joaquín Peña Acevedo. Fecha límite de entrega **27 de Septiembre de 2020**.

1.0.1. Como ejecutar

Requerimientos Este programa se ejecuto en mi computadora con la version de **Python 3.8.2** y con estos [requerimientos](#)

Jupyter En caso de tener acceso a un *servidor jupyter*, con los requerimientos antes mencionados, unicamente basta con ejecutar todas las celdas de este *notebook*. Probablemente no todas las celdas de *markdown* produzcan el mismo resultado por las *Nbextensions*.

Consola Habrá archivos e instrucciones para poder ejecutar cada uno de los ejercicios desde la consola.

Si todo sale mal En caso de que todo salga mal, tratare de dejar una copia disponible en [GoogleColab](#)

```
[ ]: """
Usage:
  Tarea4.py ejercicio1 <matA> <vecB> [-p]
  Tarea4.py ejercicio2
  Tarea4.py -h | --help

Options:
  -h --help          Show this screen.
  -v --version        Show version.
  --path=<path>      Directorio para buscar archivos [default: data/].
"""

import sys
import scipy
import numpy as np
import matplotlib.pyplot as plt
from copy import deepcopy
from scipy.linalg import solve_triangular

if __name__ == "__main__":
    from docopt import docopt
    args = docopt(usage, version='Tarea4, prb')

    if args['ejercicio1']:
        Ejercicio1(args['<matA>'], args['<vecB>'], args['--path'])
    elif args['ejercicio2']:
        Ejercicio2(args['<matA>'], args['<vecB>'], args['--path'])
```

1.1. Ejercicio 1

Programar el algoritmo de factorización LU con pivoteo parcial y probarlo resolviendo un sistema de ecuaciones lineales.

[3]: # Parte 1

```
def Algoritmo1 (A, n, t,/,dtype=np.float64):
    """ Algoritmo1 de notas de la tarea

    Python dificulta el pase de variables por referencia,
    por lo que regresaremos las matrices L, U y p mediante
    return.
    """
    return 0, L, U, p

def factLU(A, n, t,/,dtype=np.float64):
    """ Generar factorizacion A=LU con pivoteo parcial

    Funcion para hacer la factorizacion LU con pivoteo
    parcial.

    Python dificulta el pase de variables por referencia,
    por lo que regresaremos las matrices L, U y p mediante
    return.

    Input:
        A := apuntador a la matriz a factorizar
        n := tamaño n de la matriz
        t := tolerancia de cercanía con el 0

        dtype := [opcional] Tipo de dato a usar

    Output: ret, L, U, p
        ret := Variable de estado que sera
            0 si se pudo factorizar
            1 si hubo algun problema
        L := Matriz triangular inferior obtenida de
            la descomposicion de A, tamaño nxn
        U := Matriz triangular superior con tamaño
            nxn de A = LU
        p := vector de permutacion de pivoteo parcial
    """
    return ret
```

[4]: # Parte 2

```
# Para mejor rendimiento usare la implementacion de scipy
#solo hay que tener cuidado porque funciona con numpy.float64
backwardSubstitution = lambda U,b: solve_triangular(U, b, lower=False)
forwardSubstitution = lambda L,b: solve_triangular(L, b, lower=True)
```

```
def genSolLU(L, U, n, b, p, t,/, dtype=np.float64):
    """ Generar solucion de  $LUX = Pb$ 

    Funcion que trata de resolver el sistema  $LUX = Pb$ ,
    donde  $L$  es una matriz triangular inferior y  $U$  es una
    matriz triangular superior.

    Debe crear un arreglo  $\hat{b} = (\hat{b}_1, \dots, \hat{b}_n)^T$  con los elementos  $b = (b_1, \dots, b_n)^T$  reordenados de acuerdo al vec  $p$  [ $\hat{b}_1$ 
    =  $b_{\{p_i\}}$ ]

    Input:
        L := Apuntador a matriz L
        U := Apuntador a matriz U
        n := tamaño de la matriz
        b := el vector b
        p := el apuntador a un arreglo de enteros de
            longitud n
        t := tolerancia de cercanía con 0

    Output:
        ret := apuntador a arreglo de soluciones x.
        En caso de que no se encuentre solución,
        se devuelve NULL
    """
```

[5]: # Parte 3

```
def solFactLU( A, b,/, t=np.finfo(np.float64).eps, dtype=np.float64):
    """ Resuelve el sistema  $Ax=b$ 

    Esta función trata de resolver el sistema de ecuaciones
     $Ax=b$  usando la factorización LU. El ejercicio pide
    crear las matrices LU y el arreglo p pero eso se hace
    en `factLU`.

    Input:
        A := Matriz para resolver y factorizar
        b := vector de respuestas

    Output:
        ret := se regresa el vector x respuesta, o None
        en caso de que no haya habido respuestas.
    """
```

[6]: # Parte 4

```
def Ejercicio1(matA, vecb,/, path='datosLU/np', dtype=np.float64):

    A = readFile(matA, path=path, dtype=dtype)
```

```

b = readFile(vecb, path=path, dtype=dtype).transpose()

print(f' Size\n A := {A.shape}, b := {b.shape}')

x = solFactLU(A, b)

if not isinstance(x, np.ndarray):
    print('La matriz es singular')
else:
    print('x =>')
    show1D(x, len(x))
    error = np.linalg.norm(A*x-b.transpose()) #Norma de numpy
    print(f'Error =\n      {error}')
```

[7]: # Parte 5

```

if NOTEBOOK:
    for sz in [5, 50, 500]:
        Ejercicio1('matrizA'+str(sz), 'vecb'+str(sz))
    print('\n')
```

```

Size
A := (5, 5), b := (5, 1)
x =>
[-1.59276006]
[-0.6615294 ]
[ 0.07245867]
[ 1.53885238]
[-0.55274998]
Error =
79.18429857423777
```

```

Size
A := (50, 50), b := (50, 1)
x =>
[-4.61269450e+16]
[-4.01251736e+15]
[-1.23061213e+10]
[-8.39451558e+09], ... , [-2805.59934209]
[ 1700.57368067]
[-1983.03283935]
[-8070.952726 ]
Error =
11277.171179812098
```

```

Size
A := (500, 500), b := (500, 1)
x =>
[-6.77521047e+22]
[ 1.17903375e+20]
```

```
[ 2.09143243e+19]
[ 1.29143263e+20], ... , [-5.69485138e+11]
[ 6.32088421e+11]
[ 1.94677309e+09]
[-6.40471784e+11]
Error =
    220943697.4180571
```

```
(MN-env) bench@bench-dell:~/Documents/Academico/UG/LicenciaturaDEMAT/GitHubRepo/
MN/Tareas/T4$ python3 Tarea4.py ejercicio1 matrizA5 vecb5 --path='datosLU/npv/'
Size
A := (5, 5), b := (5, 1)
x =>
[-1.59276006]
[-0.6615294 ]
[ 0.07245867]
[ 1.53885238]
[-0.55274998]
Error =
    79.18429857423786
(MN-env) bench@bench-dell:~/Documents/Academico/UG/LicenciaturaDEMAT/GitHubRepo/
```

Como ejecutar

1.2. Ejercicio 2

Programar el algoritmo de **factorización de Cholesky** y resuelva un sistema de ecuaciones lineales.

```
[8]: # Parte 1

def factChol(A, n, t):
    """ Factorizacion de Cholesky

    Funcion que busca calcular la matriz  $L$  de la
    factorizacion de Cholesky.

    Input:
        A := Apuntador a matriz A
        n := tamanio de la matriz cuadarada n
        t := Tolerancia con cercania a cero
    Output:
        L := Matriz L, None si algo salio mal
    """
```

```
[9]: # Parte 2

def transpose(M, n):
    """ Calculo de la matriz transpuesta """
    ret = np.copy(M)
    for i in range(n):
        for j in range(n):
            ret[j,i] = M[i,j]
    return ret
```

```
[10]: # Parte 3

def solChol( A, n, b,/, t=np.finfo(np.float64).eps, dtype=np.
    float64):
    """ Funcion para resolver con Cholesky

    Esta funcion recibira una funcion A simetrica y
    positiva. Luego con ella se usara alguna implemen_
    tacion de forwardSubstitution para resolver el
    sistema  $Ax=b \Rightarrow LL^T x=b$ 

    Input:
        A := Apuntador a matriz A del sistema
        n := tamanio n de matriz A
        b := apuntador a vector b del sistema
        t := tolerancia de similaridad a cero

    Output:
        ret := None si algo salio mal, en otro
        caso se regresa el apuntador al
        vector de respuestas
```

```
"""
```

```
[11]: # Parte 4
def Ejercicio2(matA, vecB,/, path='datosChol/np', dtype=np.
    float64):
    A = readFile(matA, path=path, dtype=dtype)
    b = readFile(vecB, path=path, dtype=dtype).transpose()

    print(f' Size\n A := {A.shape}, b := {b.shape}')

    x = solChol(A, len(A), b)

    if not isinstance(x, np.ndarray):
        print('La matriz es singular')
    else:
        print('x =>')
        show1D(x, len(x))
        error = np.linalg.norm(A*x-b.transpose()) #Norma de numpy
        print(f'Error =\n      {error}')
```

```
[12]: # Parte 5

if NOTEBOOK:
    for sz in [5, 50, 500]:
        Ejercicio2('matSim'+str(sz), 'vecb'+str(sz))
        print('\n')
```

```
Size
A := (5, 5), b := (5, 1)
x =>
[ 3081.0484097 ]
[-6499.54355634]
[-5898.43734767]
[-2215.32263164]
[-2092.459337  ]
Error =
    9.813041367486225
```

```
Size
A := (50, 50), b := (50, 1)
x =>
[ 9.66990267e+15]
[-2.65458392e+15]
[-2.04242294e+15]
[ 3.42632604e+15], ... , [ 1.28518615e+07]
[-1.11195375e+05]
[ 1.56146299e+05]
[-3.49228255e+03]
Error =
    717.0017444741419
```

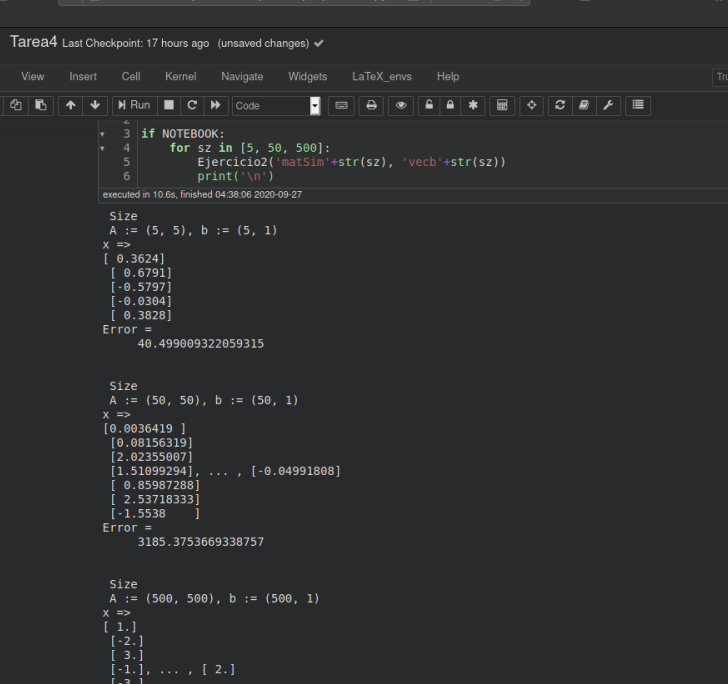


```
Size
A := (500, 500), b := (500, 1)

<ipython-input-8-4d933d506b9a>:20: RuntimeWarning: invalid value
  encountered in
sqrt
  L[j,j] = np.sqrt(A[j,j] - sum([L[j,k]**2 for k in range(j)]))
Err: array must not contain infs or NaNs
La matriz es singular
```

```
(MN-env) bench@bench-dell:~/Documents/Academico/UG/LicenciaturaDEMAT/GitHubRepo/
MN/Tareas/T4$ python3 Tarea4.py ejercicio2 matSim5 vecb5 --path='datosChol/npv/'
Size
A := (5, 5), b := (5, 1)
x =>
[ 3081.0484097 ]
[ -6499.54355634]
[ -5898.43734767]
[ -2215.32263164]
[ -2092.459337 ]
Error =
9.813041367487159
(MN-env) bench@bench-dell:~/Documents/Academico/UG/LicenciaturaDEMAT/GitHubRepo/
MN/Tareas/T4$
```

1.3. Segunda ejecución con nuevos datos



```
3 c
4 if NOTEBOOK:
5     for sz in [5, 50, 500]:
6         Ejercicio2('matSim'+str(sz), 'vecb'+str(sz))
7         print('\n')
8
9 executed in 10.6s, finished 04:38:06 2020-09-27
10
11 Size
12 A := (5, 5), b := (5, 1)
13 x =>
14 [ 0.3624]
15 [ 0.6791]
16 [-0.5797]
17 [-0.0304]
18 [ 0.3828]
19 Error =
20 40.499009322059315
21
22 Size
23 A := (50, 50), b := (50, 1)
24 x =>
25 [0.0036419 ]
26 [0.00156319]
27 [2.02355007]
28 [1.51099294], ... , [-0.04991808]
29 [ 0.85987288]
30 [ 2.53718333]
31 [-1.5538 ]
32 Error =
33 3185.3753669338757
34
35 Size
36 A := (500, 500), b := (500, 1)
37 x =>
38 [ 1.]
39 [-2.]
40 [ 3.]
41 [-1.], ... , [ 2.]
42 [-3.]
43 [ 1.]
44 [-2.]
45 Error =
46 457091.5347991374
```