Compte Rendu TP1

October 22, 2023

Realised by : El ghalbzouri Akram

1 Importation du module numpy

[ ]:
```
import numpy as np
```

2 ex1 : solsup

2.1 Implementation de la fonction

2.1.1 solsup.py

[ ]:
```
import numpy as np

eps = np.finfo(float).eps


def solsup(U: np.ndarray, b: np.ndarray) -> np.ndarray:
    """Cette focntion resolu l'equation Ax = b

    Args:
    ------
        U: Matrice triangulair sup
        b: vecteur

    Return:
    -------
        x : vecteur
    """
    n = len(U)
    x = np.zeros(n, dtype=int)
    for i in range(n):
        if abs(U[i][i]) < eps:
            return 0
    x[n - 1] = (b[n - 1] / U[n-1][n-1])
    for i in range(n-1, -1, -1):
        S = 0
```

```
        for j in range(i+1, n):
            S += U[i][j] * x[j]
        x[i] = ((b[i] - S) / U[i][i])
    return x
```

2.2 Test du fonction

[ ]:
```
from solsup import solsup
#les tests de solsup
A = np.array([[1, 2, 3], [0, 4, 8], [0, 0, 5]])
b = np.array([6, 16, 15])
x = solsup(A, b)
print("x = " , x)
```

```
x =  [ 1 -2  3]
```

3 ex2 : trisup

3.1 Implementation

3.1.1 trisup.py

[ ]:
```
import numpy as np

eps = np.finfo(float).eps


def trisup(A: np.ndarray, b: np.ndarray) -> np.ndarray:
    """Retourne une matrice triangulair sup

    Args:
    -----
        A (np.ndarray): matrice
        b (np.ndarray): vecteur

    Returns:
    --------
        U: Une matrice triangulair sup
        e: le vecteur b modifier
    """
    n = len(A)
    for k in range(0, n - 1):
        if abs(A[k][k]) < eps:
            print("Error 1")
            return 1
        else:
            c = 0
```

```
            for i in range(k + 1, n):
                c = A[i][k] / A[k][k]
                b[i] = b[i] - c * b[k]
                A[i][k] = 0
                for j in range(k + 1, n):
                    A[i][j] = A[i][j] - c * A[k][j]
    if abs(A[n - 1][n - 1]) < eps:
        print("Error de A[n][n]")
        return 1

    return A, b
```

3.2 Test du fonction

[ ]:
```
from trisup import trisup
#les tests de solsup
A = np.array([[3, 1, 2], [3, 2, 6], [6, 1, -1]])
b = np.array([2, 1, 4])
U, e = trisup(A, b)
print("U = \n" , U)
print("e = " , e)
```

```
U =
 [[ 3  1  2]
 [ 0  1  4]
 [ 0  0 -1]]
e =  [ 2 -1 -1]
```

4 ex3 : resolG

4.1 Implementation

4.1.1 resolG.py

[ ]:
```
from trisup import trisup
from solsup import solsup


def resolG(A, b):
    """resolu l'equation Ax = b par la methode gaussien

    Args:
    -----
        A (np.ndarray): matrice
        b (np.ndarray): vecteur

    Returns:
```

```
        x (np.ndarray): vecteur solution du l'equation
    """
    if trisup(A, b) != 1:
        U, e = trisup(A, b)
        x = solsup(U, e)
        return x
    else:
        return 0
```

4.2 Test du fonction

[ ]:
```
import numpy as np
from resolG import resolG

#test 1
A = np.array([[1, 2, 3], [5, 2, 1], [3, -1, 1]])
b = np.array([5, 5, 6])
#test 2
A2 = np.array([[2, 1, 5], [1, 2, 4], [3, 4, 10]])

x2 = resolG(A, b)
print("x2 = ", x2)

x3 = resolG(A2, b)
print("x3 = ", x3)
```

```
x2 =  [ 1 -1  2]
Error de A[n][n]
x3 =  0
```

On remarque que la matrice A2 donne un erreur

5 ex 4 : LU

5.1 Implementation

5.2 LU.py

[ ]:
```
import numpy as np

eps = np.finfo(float).eps


def LU(A: np.ndarray) -> np.ndarray:
    """retourn la decomposition LU du matrice A

    Args:
    -----
```

```
      A (np.ndarray): Matrice

    Returns:
    --------
      L : matrice tirangulair sup
      U : matrice tirangulair inf
    """
    n = len(A)
    L = np.identity(n)
    U = A.copy()
    for k in range(0, n - 1):
        if abs(U[k][k]) < eps:
            print("Error 1")
            return 1
        else:
            c = 0
            for i in range(k + 1, n):
                c = U[i][k] / U[k][k]
                U[i][k] = 0
                L[i][k] = c
                for j in range(k + 1, n):
                    U[i][j] = U[i][j] - c * U[k][j]
    if abs(U[n - 1][n - 1]) < eps:
        # print(A[n - 1][n - 1])
        print("Error de A[n][n]")
        return 1
    return L, U
```

5.3 Test du fonction

[ ]:
```
import numpy as np
from LU import LU

A = np.array([[3, 1, 2], [3, 2, 6], [6, 1, -1]])
L, U = LU(A)
print("L = \n", L)
print("U = \n", U)
```

```
L =
 [[ 1.  0.  0.]
 [ 1.  1.  0.]
 [ 2. -1.  1.]]
U =
 [[ 3  1  2]
 [ 0  1  4]
 [ 0  0 -1]]
```

6 ex5 : solinf

6.1 Implementation

6.1.1 solinf.py

[ ]:
```
import numpy as np

eps = np.finfo(float).eps


def solinf(U: np.ndarray, b: np.ndarray) -> np.ndarray[float]:
    """Cette focntion resolu l'equation Ax = b

    Args:
    ------
        U: Matrice triangulair inf
        b: vecteur

    Return:
    -------
        x : vecteur
    """
    n = len(U)
    x = np.zeros(n)
    for i in range(n):
        if abs(U[i][i]) < eps:
            return 0

    x[0] = (b[0] / U[0][0])
    for i in range(1, n):
        S = 0
        for j in range(0, i):
            S += U[i][j] * x[j]
        x[i] = ((b[i] - S) / U[i][i])
    return x
```

6.2 Test du fonction

[ ]:
```
import numpy as np
from solinf import solinf

A = np.array([[1, 0, 0], [2, 3, 0], [1, 4, -1]])
b = np.array([1, 8, 10])
x = solinf(A, b)
print("x = ", x)
```

```
x =  [ 1.  2. -1.]
```

7 ex6 : resolLU

7.1 Implementation

7.2 resolLU.py

[ ]:
```
import numpy as np
from LU import LU
from solinf import solinf
from solsup import solsup

eps = np.finfo(float).eps


def resolLU(A: np.ndarray, b: np.ndarray) -> np.ndarray:
    """Resolu l'equation Ax = b par la methode LU

    Args:
    -----
      A (np.ndarray): matrice
      b (np.ndarray): vecteur

    Returns:
    -------
      x (np.ndarray): vecteur solution de l'equation
    """
    L, U = LU(A)
    y = solinf(L, b)
    x = solsup(U, y)
    return x
```

7.3 Test du fonction

[ ]:
```
import numpy as np
from resolLU import resolLU

A = np.array([[1, 2, 3], [5, 2, 1], [3, -1, 1]])
b = np.array([5, 5, 6])
x = resolLU(A, b)
print("x = ", x)
```

```
x =  [2 0 1]
```

8 ex7 : inverse

8.1 Implementation

8.1.1 inverse.py

[ ]:
```
import numpy as np
from ex1 import resolLU


def inverse(A: np.ndarray) -> np.ndarray:
    """retourn l'inverse du matrice A

    Args:
        A (np.ndarray): matrice

    Returns:
        B (np.ndarray): matrice inverse de A
    """
    n = len(A)
    I = np.identity(n)
    b = np.zeros(n, dtype=int)
    B = np.zeros(A.shape)
    U , L = LU(A, B)
    for i in range(n):
        y = solinf(L, I[:, i])
        x = solsup(U, y)
        B[i] = x
    return B.T
```

8.2 Test du fonction

[ ]:
```
import numpy as np
from inverse import inverse

A = np.array([[1, 2, 3], [5, 2, 1], [3, -1, 1]])
B = inverse(A)
print("B = \n", B)
```

```
B =
 [[-0.078125   0.140625   0.125     ]
 [ 0.0234375  0.2578125 -0.4375    ]
 [ 0.34375   -0.21875    0.25      ]]
```

9 ex8 : cholesky

9.1 Implementation

9.1.1 cholesky.py

[ ]:
```
import numpy as np
import math


def cholesky(A: np.ndarray) -> np.ndarray:
    """ Fair la decomposition de A on de matrice B et Bt

    Parameters :
    ----------
      A: Matrice

    Retour :
      B : Matrice triangulair inf
      Bt : Le transpose du matrice B
    """
    n = len(A)
    B = np.zero(A.shape)
    for j in range(n):
        S = 0
        for k in range(1, j-1):
            S += B[j, k] ** 2
        B[j, j] = (1 / B[j, j]) * (A[j, j] - S)
        for i in range(j + 1, n):
            S = 0
            for k in range(1, j - 1):
                S += B[i, k] * B[j, k]
            B[i, j] = (1 / B[j, j]) * (A[i, j] - S)
    S = 0
    for k in range(1, n - 1):
        S += B[n - 1, k] ** 2
    B[n - 1, n - 1] = math.sqrt(A[n - 1, n - 1] - S)

    return B
```

9.2 Test du fonction

[ ]:
```
import numpy as np
from cholesky import cholesky

A = np.array([[15, 10, 18, 12], [10, 15, 7, 13], [18, 7, 27, 7], [12, 13, 7,␣
 ↪22]])
B = cholesky(A)
```

```
print("B = \n", B)

# On multiple B par le transpose de B et on obtient A :
print("\nVerification du fonction : \n")
print("B * Bt = \n", B @ B.T)
```

```
B =
 [[ 3.87298335  0.          0.          0.        ]
 [ 2.5819889   2.88675135  0.          0.        ]
 [ 4.64758002 -1.73205081  1.54919334  0.        ]
 [ 3.09838668  1.73205081 -2.84018779  1.15470054]]

Verification du fonction :

B * Bt =
 [[15. 10. 18. 12.]
 [10. 15.  7. 13.]
 [18.  7. 27.  7.]
 [12. 13.  7. 22.]]
```

9.2.1 resolchol.py

[ ]:
```
import numpy as np
from cholesky import cholesky
from solinf import solinf
from solsup import solsup


def resolchol(A: np.ndarray, b: np.ndarray) -> np.ndarray:
    """Resolution du system Ax = b par la methode de cholesky

    Args:
    -----
        A (np.ndarray): Matrice symetrique definie positive
        b (np.ndarray) : Vecteur

    Returns:
    --------
        x (np.ndarray) : Vecteur
    """

    B = cholesky(A)
    y = solinf(B, b)
    #B.t est le transpose de B
    x = solsup(B.T, y)
    return x
```

9.2.2 Test :

[ ]:
```python
import numpy as np
from resolchol import resolchol

A = np.array([[15, 10, 18, 12], [10, 15, 7, 13], [18, 7, 27, 7], [12, 13, 7,
 ↪22]])
b = np.array([53, 72, 26, 97])
x = resolchol(A, b)
print("x = ", x)
```

```
x =  [ 1.  2. -1.  3.]
```