Série 8 (Corrigé)

Parcourez les chapitres des notebooks Jupyter 4.1 et resolvez les exercices qui y sont proposés (ce sont les mêmes qu'ici).

Exercice 1

On considère le système linéaire $A\mathbf{x} = \mathbf{b}$ où :

$$A = \begin{pmatrix} 3 & 6 & 7 \\ 1 & 1 & 4 \\ 2 & 4 & 8 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}.$$

a) Calculez la factorisation LU de la matrice A avec Python à l'aide du code ci-dessous.

b) Résolvez le système linéaire $A\mathbf{x} = \mathbf{b}$ en utilisant la factorisation trouvée au point précédent (Ne plus utiliser Python.)

c) Calculez le déterminant de la matrice A en utilisant sa factorisation LU.

```
# importing libraries used in this book
import numpy as np
import scipy.linalg as linalg
import pprint

A = np.array([[3, 6, 7],
              [1, 1, 4],
              [2, 4, 8] ])

# LU factorisation with pivoting
P, L, U = linalg.lu(A)

print("A = P L U")
pprint.pprint(P.dot(L.dot(U)) )

print("P:")
pprint.pprint(P)

print ("L:")
pprint.pprint(L)

print ("U:")
pprint.pprint(U)
```

Sol. :

Si P est l'identité, $A = LU$.

1. *résoudre pour* $\mathbf{y}$ *tel que* $L\mathbf{y} = \mathbf{b}$ *par substitution progressive*
2. *résoudre pour* $\mathbf{x}$ *tel que* $U\mathbf{x} = \mathbf{y}$ *par substitution retrograde*

Si P n'est pas l'identité, $A = PLU$.

Attention, une matrice de permutation est une matrice orthogonale car les colonnes sont orthonormée. Donc $P^{-1} = P^T$. Du coup : $P^T A = LU$ et

$$A\mathbf{x} = \mathbf{b} \Leftrightarrow P^T A\mathbf{x} = P^T\mathbf{b} \Leftrightarrow LU\mathbf{x} = P^T\mathbf{b}.$$

Alors il faut modifier les calculs précedants comme suit :

1. *résoudre pour* $\mathbf{y}$ *tel que* $L\mathbf{y} = P^T\mathbf{b}$ *par substitution progressive*
2. *résoudre pour* $\mathbf{x}$ *tel que* $U\mathbf{x} = \mathbf{y}$ *par substitution retrograde*

```
def subst_progressive(L, b):
    """
    substitution progressive: resout y, L*y = b
    Input:
    - L: matrice carree nxn, triangulaire inferieure
    - b: vector de dimension n
    Output:
    - y: vector de dimension n
    """

    # Initialisation de la solution
    y = np.zeros(L.shape[1])

    # La premiere ligne de Ly = b est L_{11} y_1 = b_1
    # Ensuite pour la ligne k on connait y_1, ..., y_{k-1} et elle s'ecrit
    # L_{kk} y_k = b_k - ( L_{k1} y_1 + ... L_{kk-1} y_{k-1} )
    for k in range(L.shape[0]):
        # sum_k est ( L_{k1} y_1 + ... L_{kk-1} y_{k-1} )
        sum_k = 0
        for j in range(k):
            sum_k += L[k,j]*y[j]
        y[k] = 1/L[k,k]*(b[k]-sum_k)
    return y

def subst_retrograde(U, y):
    """
    substitution retrograde: resout pour x, U*x = y
    Input:
    - U: matrice carree nxn, triangulaire superieure
    - y: vector de dimension n
    Output:
```

```
    - x: vector de dimension n
    """

    # Initialisation de la solution
    x = np.zeros(U.shape[1])

    # La derniere ligne de Yx = y est U_{nn} x_n = y_n
    # Ensuite pour la ligne k on connait x_n, ..., x_{k+1} et elle s'ecrit
    # U_{kk} x_k = y_k - ( U_{kk+1} x_{k+1} + ... U_{kn} y_{n} )
    for k in reversed(range(U.shape[0])):
        # sum_k est ( U_{kk+1} x_{k+1} + ... U_{kn} y_{n} )
        sum_k = 0
        for j in range(k+1, U.shape[0]):
            sum_k += U[k,j]*x[j]
        x[k] = 1/U[k,k]*(y[k]-sum_k)
    return x
```

```
b = np.array([4, 5, 6])
Ptb = P.T.dot(b)

y = subst_progressive(L, Ptb)
print("y =", y)

x = subst_retrograde(U, y)
print("x =", x)

# check the residual of the equation
print("residual =",b - A.dot(x))
```

$$\mathbf{x} = \begin{pmatrix} 3 \\ -2 \\ 1 \end{pmatrix}$$

$$\det(A) = \det(P) \cdot \det(L) \cdot \det(U) = 1 \cdot 1 \cdot (3 \cdot (-1) \cdot 3.3333) = -10$$

Exercice 2

Les mineurs principaux d'une matrice $A \in \mathbb{R}^{n \times n}$ sont les déterminants des matrices $A_p = (a_{i,j})_{1 \leq i,j \leq p}$, $p = 1, ..., n$.

Critère de Sylvester : une matrice symétrique $A \in \mathbb{R}^{n \times n}$ est définie positive si et seulement si les mineurs principaux de A sont tous positifs.

On considère le système linéaire $A\mathbf{x} = \mathbf{b}$ où

$$A = \begin{pmatrix} \varepsilon & 1 & 2 \\ 1 & 3 & 1 \\ 2 & 1 & 3 \end{pmatrix}.$$

1. Déterminez pour quelles valeurs du paramètre réel $\varepsilon \in \mathbb{R}$, la matrice A est symétrique définie positive.
2. Soit maintenant $\varepsilon = 0$. On veut résoudre le système $A\mathbf{x} = \mathbf{b}$ par une méthode directe ; quelle factorisation de la matrice A envisageriez-vous ? Justifiez votre réponse.
3. En considérant $\varepsilon = 2$, vérifier que dans ce cas la matrice A est définie positive et calculer sa factorisation de Cholesky $A = LL^T$.
4. En supposant que $\mathbf{b} = (1, 1, 1)^T$, résolvez le système linéaire $A\mathbf{x} = \mathbf{b}$ en utilisant la factorisation de Cholesky calculée au point c).

Référence Python pour la factorisation de Cholesky `scipy.linalg.cholesky` : `https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.linalg.cholesky.html`

Sol. :

1. *En appliquant le critère de Sylvester, il suffit d'imposer*

$$\begin{cases} \varepsilon > 0, \\ det \begin{pmatrix} \varepsilon & 1 \\ 1 & 3 \end{pmatrix} = 3\varepsilon - 1 > 0, \\ det\, A = 8\varepsilon - 11 > 0, \end{cases} \qquad \Rightarrow \quad \varepsilon > \frac{11}{8}.$$

2. *Si $\varepsilon = 0$ la matrice A est symétrique, mais elle n'est pas définie positive ; donc on ne peut pas calculer la factorisation de Cholesky. On utilisera la méthode d'élimination de Gauss avec changement de pivot, puisque $a_{11} = 0$; par exemple, on peut considérer la matrice de permutation P par lignes :*

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

On peut facilement voir que $A = PLU$ avec

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & -5 & 1 \end{pmatrix} \qquad et \qquad U = \begin{pmatrix} 1 & 3 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 11 \end{pmatrix}.$$

3. *Si $\varepsilon = 2$, la matrice A est symétrique définie positive. Ici on va utiliser $A = LL^T$. Les éléments de la matrice L de la factorisation de Cholesky de A sont :*

$$\begin{aligned} l_{11} &= \sqrt{a_{11}} = \sqrt{2} \\ l_{21} &= \frac{1}{l_{11}} \cdot a_{21} = \frac{1}{\sqrt{2}} \\ l_{22} &= \sqrt{a_{22} - l_{21}^2} = \sqrt{\frac{5}{2}} \\ l_{31} &= \frac{1}{l_{11}} \cdot a_{31} = \sqrt{2} \\ l_{32} &= \frac{1}{l_{22}} \cdot (a_{32} - l_{31} l_{21}) = 0 \\ l_{33} &= \sqrt{a_{33} - (l_{31}^2 + l_{32}^2)} = 1 \end{aligned}$$

c'est-à-dire :

$$L = \begin{pmatrix} \sqrt{2} & 0 & 0 \\ \frac{1}{\sqrt{2}} & \sqrt{\frac{5}{2}} & 0 \\ \sqrt{2} & 0 & 1 \end{pmatrix}.$$

```
# scipy.linalg.eigh : eigenvalues for symmetric matric
from scipy.linalg import eigh
# scipy.linalg.eig : eigenvalues of a matrix
from scipy.linalg import eig


epsilon = 2
A = np.array([[epsilon, 1, 2],
              [1, 3, 1],
              [2, 1, 3] ])

# Is A symmetric ?
print(f'max(abs(A-AT)) = {np.max(np.abs(A-A.T))} ')

# What are the eigenvlues of $A$ ? (usually, use eig, but here A is
    symmetric, we can use eigh )
lk, v = eigh(A)
print(f'The eigenvalues of A are {lk}')

# Cholesky factorisation: lower : return lower-triangular matrix, A = L
    L^T
L = cholesky(A, lower=True)
print(f"\n A = L^T L = {L.dot(L.T)}\n")

print (f"L = {L}")
```

4. *On résout le système linéaire de la façon suivante :*

$$L\mathbf{y} = \mathbf{b} \qquad et \qquad L^T\mathbf{x} = \mathbf{y}.$$

On applique l'algorithme de substitution progressive pour résoudre le système $L\mathbf{y} = \mathbf{b}$ *et on obtient* $\mathbf{y} = \left(1/\sqrt{2}, \sqrt{2}/(2\sqrt{5}), 0\right)^T$. *Puis on calcule la solution du système* $L^T\mathbf{x} = \mathbf{y}$ *par la méthode de substitution rétrograde et on trouve* $\mathbf{x} = (2/5, 1/5, 0)^T$.

```
 b = np.array([1,1, 1])

y = np.linalg.solve(L,b)
x = np.linalg.solve(L.T,y)

print(x)

# check the residual of the equation
```

```
print(b - A.dot(x))
```

Exercice 3

Problèmes de précision

Les erreurs d'arrondis peuvent causer des différences importantes entre la solution calculée par la méthode d'élimination de Gauss (MEG) et la solution exacte. Cela arrive si le *conditionnement de la matrice du système est très grand.*

La matrice de Hilbert de taille $n \times n$ est une matrice symétrique définie par

$$a_{ij} = \frac{1}{i+j-1}, \qquad i,j = 1,\dots,n$$

On peut construire une matrice de Hilbert de taille n quelconque en utilisant la commande 'A = scipy.linalg.hilbert(n)'. Par exemple, pour $n = 4$, on a :

$$A = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

On considère les systèmes linéaires $A_n \mathbf{x}_n = \mathbf{b}_n$ ou A_n est la matrice de Hilbert de taille n avec $n = 4, 6, 8, 10, 12, 14, \dots, 20$ tandis que $\mathbf{b}_n$ est choisi de sorte que la solution exacte soit $\mathbf{x}_n = (1, 1, \cdots, 1)^T$.

1. Pour chaque n, calculez le conditionnement de la matrice
2. Résolvez le système linéaire par la factorisation LU et notez $\mathbf{x}_n^{LU}$ la solution calculée.
3. Dessinez le graphique avec le conditionnement obtenu ainsi que l'erreur rélative $\|\mathbf{x}_n - \mathbf{x}_n^{LU}\|/\|\mathbf{x}_n\|$ (où $\|\cdot\|$ est la norme euclidienne d'un vecteur, $\|\mathbf{x}\| = \sqrt{\mathbf{x}^T \cdot \mathbf{x}}$). Utilisez une échelle logarithmique pour l'axe y.
4. Sur le même graphique, reportez le conditionnement de la matrice A, 'np.linalg.cond(A)'

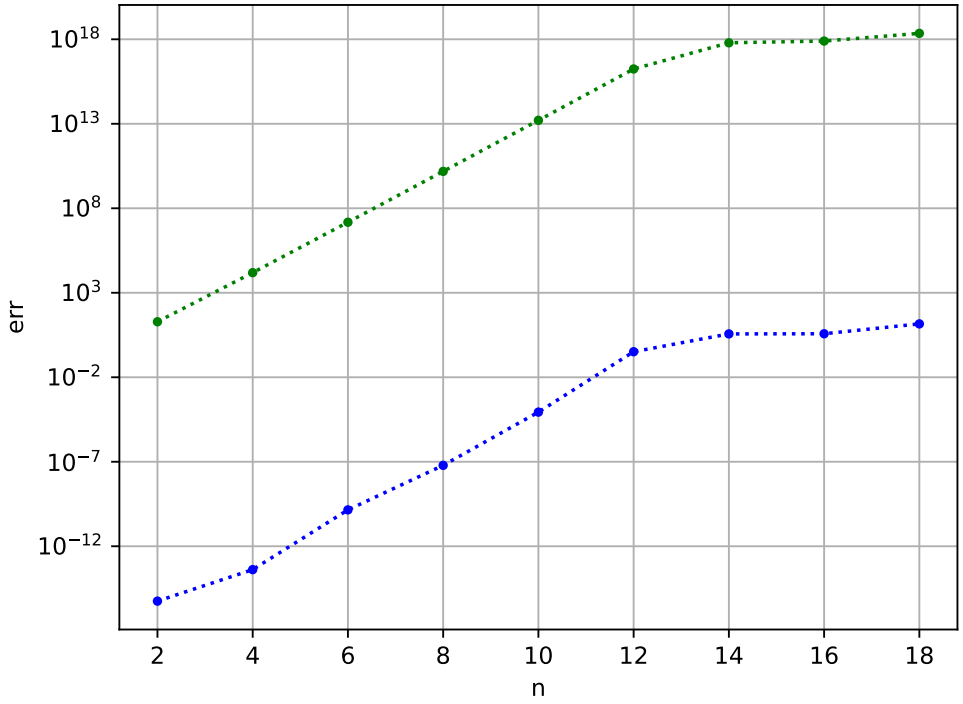
Répétez la même chose avec la factorisation de Cholesky 'L = cholesky(A, lower=True)' pour $n = 4, 6, 8, 10, 12$. Que se passe-t-il si $n = 14$? ***Sol. :***

```
from scipy.linalg import hilbert
```

```
Nrange = range(2,20,2)
err = []
cond = []

for n in Nrange :
    A = hilbert(n)
    P, L, U = lu(A)

    x = np.ones([n,1])

    b = A.dot(x)
```

```
    y = np.linalg.solve(L,P.T.dot(b))
    xLU = np.linalg.solve(U,y)

    err.append( np.linalg.norm(x-xLU) / np.linalg.norm(x) )
    cond.append( np.linalg.cond(A) )
```

```
plt.plot(Nrange, err, 'b:.',Nrange, cond, 'g:.')

plt.xlabel('n'); plt.ylabel('err');
# plt.xscale('log')
plt.yscale('log')
plt.grid(True)
plt.show()
```




```
Nrange = range(2,13,2)
err = []
cond = []

for n in Nrange :
    A = hilbert(n)
    L = cholesky(A, lower=True)

    x = np.ones([n,1])

    b = A.dot(x)
```

```
    y = np.linalg.solve(L,b)
    xCho = np.linalg.solve(L.T,y)

    err.append( np.linalg.norm(x-xCho) / np.linalg.norm(x) )
    cond.append( np.linalg.cond(A) )
```

```
plt.plot(Nrange, err, 'b:.',Nrange, cond, 'g:.')

plt.xlabel('n'); plt.ylabel('err');
# plt.xscale('log')
plt.yscale('log')
plt.grid(True)
plt.show()
```