

Vous : Pouvez vous programmer l'algorithme de jaccobi pour la recherche des valeurs propres et des vecteurs propres en python avec un exemple très concret

Oui — voici une implémentation **claire et complète** de l'algorithme de **Jacobi** (rotations de Jacobi) pour calculer **valeurs propres et vecteurs propres** d'une **matrice symétrique réelle** en Python, avec un exemple concret.

⚠️ L'algorithme de Jacobi fonctionne (dans cette forme classique) pour les matrices **symétriques**. Si votre matrice n'est pas symétrique, il faut une autre méthode (QR, Schur, etc.).

Implémentation Python (Jacobi)

```
import numpy as np

def jacobi_eigen(A, tol=1e-12, max_sweeps=100):
    """
    Algorithme de Jacobi (rotations) pour diagonaliser une matrice symétrique réelle A.
    Retourne:
        - w : valeurs propres (1D)
        - V : vecteurs propres (colonnes de V)
        - D : matrice presque diagonale finale (diagnostic)
        - sweeps : nombre de balayages effectués
    """
    A = np.array(A, dtype=float)
    n = A.shape[0]
    if A.shape[1] != n:
        raise ValueError("A doit être carrée.")
    if not np.allclose(A, A.T, atol=1e-14):
        raise ValueError("A doit être symétrique (A == A.T).")

    V = np.eye(n)

    def max_offdiag(A):
        # Trouve l'indice (p,q) du plus grand terme hors-diagonale en valeur absolue
        n = A.shape[0]
        p, q = 0, 1
        m = abs(A[p, q])
        for i in range(n):
            for j in range(i+1, n):
                if abs(A[i, j]) > m:
```

```

    m = abs(A[i, j])
    p, q = i, j
    return m, p, q

for sweep in range(1, max_sweeps + 1):
    m, p, q = max_offdiag(A)
    if m < tol:
        break

    app = A[p, p]
    aqq = A[q, q]
    apq = A[p, q]

    # Calcul de l'angle de rotation (version stable)
    if abs(apq) < tol:
        continue

    tau = (aqq - app) / (2.0 * apq)
    t = np.sign(tau) / (abs(tau) + np.sqrt(1.0 + tau*tau))
    c = 1.0 / np.sqrt(1.0 + t*t)
    s = t * c

    # Mise à jour des éléments de A (rotation sur p,q)
    A[p, p] = app - t * apq
    A[q, q] = aqq + t * apq
    A[p, q] = 0.0
    A[q, p] = 0.0

    for k in range(n):
        if k != p and k != q:
            akp = A[k, p]
            akq = A[k, q]
            A[k, p] = c * akp - s * akq
            A[p, k] = A[k, p]
            A[k, q] = s * akp + c * akq
            A[q, k] = A[k, q]

    # Mise à jour des vecteurs propres V
    for k in range(n):
        vkp = V[k, p]
        vkq = V[k, q]
        V[k, p] = c * vkp - s * vkq
        V[k, q] = s * vkp + c * vkq

w = np.diag(A).copy()
# Tri (optionnel) : valeurs propres croissantes et vecteurs propres correspondants

```

```
idx = np.argsort(w)
w = w[idx]
V = V[:, idx]
return w, V, A, sweep
```

Exemple très concret

Prenons une matrice symétrique (par exemple issue d'un problème de corrélation / inertie / rigidité simplifiée) :

```
A=(411130102)A=\begin{pmatrix} 4 & 1 & 1 \\ 1 & 3 & 0 \\ 1 & 0 & 2 \end{pmatrix}A=411130102
```

```
import numpy as np
```

```
A = np.array([[4, 1, 1],
              [1, 3, 0],
              [1, 0, 2]], dtype=float)
```

```
w, V, D, sweeps = jacobi_eigen(A, tol=1e-12, max_sweeps=100)
```

```
print("Balayages (sweeps) :", sweeps)
print("\nValeurs propres :")
print(w)
```

```
print("\nVecteurs propres (colonnes) :")
print(V)
```

```
# Vérification concrète : A v ≈ λ v pour chaque paire (λ, v)
print("\nVérification A @ V - V @ diag(w) (doit être proche de 0) :")
print(A @ V - V @ np.diag(w))
```

```
# Vérification orthonormalité : V^T V ≈ I
print("\nV^T V (doit être proche de I) :")
print(V.T @ V)
```

Ce que vous devez observer

- $A @ V - V @ \text{diag}(w)$ doit afficher une matrice **très proche de 0** (petites erreurs numériques).
 - $V.T @ V$ doit être **très proche de l'identité** : les vecteurs propres trouvés sont orthonormés.
 - w contient les **valeurs propres** et $V[:, i]$ le **vecteur propre** associé à $w[i]$.
-

Notes pratiques (importantes)

- **Complexité** : Jacobi est simple et robuste mais pas le plus rapide pour de grandes matrices.
 - **Cas d'usage** : très bon pour l'enseignement, petites/moyennes matrices, besoin de stabilité et d'orthogonalité.
 - Pour la production, `numpy.linalg.eigh(A)` est généralement préférable (très optimisé).
-

Si vous me donnez **votre matrice exacte** (taille, valeurs) ou votre contexte (mécanique, PCA, graphes...), je peux adapter l'exemple et ajouter l'interprétation physique/numérique des vecteurs propres.

Si votre matrice n'est pas symétrique, il faut une autre méthode (QR, Schur, etc.). Pouvez-vous donner un exemple très concret
Réflexion durant 11s

Oui — voici un exemple **très concret** de matrice **non symétrique** où on cherche une valeur propre / un vecteur propre avec une méthode de type **QR/Schur** (en pratique, `numpy.linalg.eig` et `scipy.linalg.schur` s'appuient sur des algos de la famille QR/Schur via LAPACK).

Exemple concret : chaîne de Markov (navigation entre pages)

Imaginez 3 pages web **A**, **B**, **C**. À chaque clic, l'utilisateur passe à une autre page avec certaines probabilités.

La matrice de transition PPP (lignes = “où je suis”, colonnes = “où je vais”) est :

$$P = \begin{pmatrix} 0.1 & 0.6 & 0.3 \\ 0.2 & 0.4 & 0.4 \\ 0.5 & 0.2 & 0.3 \end{pmatrix}$$

- Chaque ligne somme à 1 (matrice stochastique).
- Elle est **non symétrique** (par ex. $P_{12}=0.60 \neq P_{21}=0.40$ $P_{12}=0.60 \neq P_{21}=0.40$).

La **distribution stationnaire** π vérifie :

$$\pi P = \pi$$

Donc π est un **vecteur propre** associé à la **valeur propre 1** (à gauche).

1) Calcul via `numpy.linalg.eig` (méthodes QR/Schur en interne)

```

import numpy as np

P = np.array([[0.10, 0.60, 0.30],
              [0.40, 0.20, 0.40],
              [0.50, 0.20, 0.30]], dtype=float)

# Pour pi P = pi, on cherche un vecteur propre à gauche de P,
# ce qui revient à chercher un vecteur propre à droite de P.T : P.T v = v
w, V = np.linalg.eig(P.T)

# Trouver l'indice de la valeur propre la plus proche de 1
i = np.argmin(np.abs(w - 1))

pi = np.real(V[:, i])
pi = pi / pi.sum() # normaliser pour que la somme fasse 1

print("Valeurs propres de P :", w)
print("Distribution stationnaire pi :", pi)

# Vérification concrète: pi P ≈ pi
print("pi @ P :", pi @ P)
print("Différence (pi@P - pi) :", (pi @ P) - pi)

```

Interprétation concrète :

- π donne le pourcentage de temps passé sur A, B, C “à long terme”.
- La vérification $\pi @ P \approx \pi$ doit donner une différence proche de 0.

Remarque importante : pour une matrice non symétrique, les valeurs propres/vecteurs propres peuvent être **complexes**, et les vecteurs propres ne sont pas forcément orthogonaux.

2) (Optionnel) Voir la décomposition de Schur (si vous avez SciPy)

La **forme de Schur** est très utilisée numériquement :

$$P = QTQ^T \quad (\text{Schur réel, si possible}) \quad P = Q T Q^{\text{top}} \quad (\text{Schur complexe, si possible})$$

ou en complexe $P = QTQ^*P = Q T Q^*P = QTQ^*$, avec T quasi triangulaire/triangulaire.
Les valeurs propres sont sur la diagonale (ou dans des blocs 2×2 en Schur réel).

```

import numpy as np
from scipy.linalg import schur

P = np.array([[0.10, 0.60, 0.30],
              [0.40, 0.20, 0.40],
              [0.50, 0.20, 0.30]], dtype=float)

T, Q = schur(P, output='real') # Schur réel (T quasi-triangulaire)
print("T (Schur) =\n", T)

# Valeurs propres (approx) depuis T
eig_from_T = np.diag(T)
print("diag(T) (valeurs propres/approx) :", eig_from_T)

```

Pourquoi c'est “non symétrique” et pourquoi Jacobi n'est pas adapté ?

- Jacobi (rotations) exploite une diagonalisation orthogonale propre aux matrices symétriques.
 - Pour une matrice non symétrique, on passe plutôt par QR/Schur (robustes, standard en librairies).
-

Si vous me donnez un exemple de **votre** matrice non symétrique (taille 3×3 , 4×4 , etc.), je peux :

- montrer les valeurs propres/vecteurs propres,
- dire s'il y a des valeurs propres complexes,
- et interpréter le résultat selon votre contexte (contrôle, dynamique, graphes, etc.).

Voici alors ma question : Pouvez vous donner un programme en Python qui aurait pour rôle de calculer les valeurs propres et les vecteurs propres d'une matrice donnée. Mais ce qui distingue ce programme python des autres programmes est sa capacité à déterminer si c'est une matrice symétrique ou non puis de choisir l'algorithme qui sera le mieux adapté et également le plus performant afin de toujours trouver les vecteurs propres et les valeurs propres de n'importe quel matrice donnée par l'utilisateur. Pouvez vous accompagner cette superalgorithme d'un exemple très concret et très détaillé

Voici un “super-algorithme” Python qui :

1. **teste** si la matrice est **symétrique** (ou **hermitienne** en complexe)
2. choisit l'algo **le plus adapté et performant** :
 - **numpy.linalg.eigh** si symétrique/hermitienne (plus rapide + plus stable + valeurs propres réelles)

- `numpy.linalg.eig` sinon (cas général non symétrique : peut produire des valeurs/vecteurs propres complexes)

et 3) fournit des **vérifications** concrètes.

1) Le programme “auto-choix”

```
import numpy as np

def is_hermitian(A, rtol=1e-10, atol=1e-12):
    """
    True si A ≈ A* (A conjuguée transposée).
    Pour une matrice réelle, cela revient à tester la symétrie A ≈ A^T.
    """
    A = np.asarray(A)
    return np.allclose(A, A.conj().T, rtol=rtol, atol=atol)

def eig_auto(A, prefer_schur=False, rtol=1e-10, atol=1e-12):
    """
    Calcule valeurs propres et vecteurs propres en choisissant automatiquement
    la méthode la plus adaptée :
    - Hermitienne/symétrique -> eigh (rapide, stable, valeurs propres réelles)
    - Sinon -> eig (cas général)

    prefer_schur=True :
        - si SciPy est dispo, utilise Schur pour plus de robustesse numérique
          (surtout si on veut ensuite faire des choses avancées).
        - sinon retombe sur numpy.linalg.eig
    """
    A = np.asarray(A)
    if A.ndim != 2 or A.shape[0] != A.shape[1]:
        raise ValueError("La matrice A doit être carrée (n x n).")

    if is_hermitian(A, rtol=rtol, atol=atol):
        # Cas symétrique/hermitien : le meilleur choix en général
        w, V = np.linalg.eigh(A) # valeurs propres réelles (ou quasi) et V orthonormale
        method = "eigh (Hermitienne/Symétrique)"
    else:
        if prefer_schur:
            try:
                from scipy.linalg import schur, rsf2csf
                # Schur réel/complexe ; en non-symétrique, Schur est une base robuste
            except ImportError:
                print("Scipy not found, falling back to eig")
                w, V = np.linalg.eig(A)
                method = "eig (non-symétrique)"
```

```

T, Z = schur(A, output='real')
# Convertit en Schur complexe (triangulaire) si nécessaire
Tc, Zc = rsf2csf(T, Z)
w = np.diag(Tc)
# Remarque: Schur ne donne pas directement tous les vecteurs propres
# de façon "simple" comme eig; on peut les calculer mais c'est plus long.
# Donc ici, pour fournir vecteurs propres, on appelle eig ensuite.
w, V = np.linalg.eig(A)
method = "eig (Non symétrique) + Schur (diagnostic SciPy)"
except Exception:
    w, V = np.linalg.eig(A)
    method = "eig (Non symétrique) (SciPy Schur non disponible)"
else:
    w, V = np.linalg.eig(A)
    method = "eig (Non symétrique)"

return w, V, method

def check_eigendecomposition(A, w, V):
"""
Vérifie  $A V \approx V \text{diag}(w)$ .
Renvoie l'erreur relative (norme Frobenius).
"""
A = np.asarray(A)
w = np.asarray(w)
V = np.asarray(V)
lhs = A @ V
rhs = V @ np.diag(w)
num = np.linalg.norm(lhs - rhs, ord='fro')
den = np.linalg.norm(lhs, ord='fro') + 1e-30
return num / den

```

Pourquoi ce choix est “le plus performant” ?

- `eigh` exploite la structure **symétrique/hermitienne** : c'est **plus rapide** et **plus stable** numériquement (et les valeurs propres sont réelles).
- `eig` est le choix **général** pour n'importe quelle matrice carrée (y compris non symétrique), mais peut produire des résultats **complexes** et des vecteurs propres non orthogonaux.

2) Exemple très concret et très détaillé

On va tester sur **deux cas** typiques :

Cas A — Matrice symétrique (ex: “matrice de covariance” simplifiée)

Imaginez 3 variables (capteurs) x_1, x_2, x_3 , x_{-1}, x_{-2}, x_{-3} . Une matrice symétrique positive (type covariance) :

```
A_sym = np.array([
    [4.0, 1.0, 1.0],
    [1.0, 3.0, 0.0],
    [1.0, 0.0, 2.0]
])

w, V, method = eig_auto(A_sym)
err = check_eigendecomposition(A_sym, w, V)

print("==== CAS SYMÉTRIQUE ====")
print("Méthode choisie :", method)
print("Valeurs propres w :", w)
print("Vecteurs propres (colonnes de V) :\n", V)
print("Erreur relative ||AV - V diag(w)|| / ||AV|| :", err)

# Vérif orthonormalité (spécifique au cas eigh)
print("V^T V (doit ~ I) :\n", V.T @ V)
```

Interprétation concrète :

- Les **valeurs propres** indiquent la “quantité” de variance/inertie sur chaque axe principal.
- Les **vecteurs propres** (colonnes de V) sont les directions principales (combinaisons de variables) :
 - si une valeur propre est grande \rightarrow direction dominante.
- $V.T @ V \approx I$: les vecteurs propres sont **orthonormés** (propriété très pratique).

Cas B — Matrice non symétrique (ex: modèle de transitions / dynamique)

Exemple : 3 états d'un système (pages web, machines, modes). La matrice ci-dessous est **non symétrique** et pourtant on veut ses valeurs/vecteurs propres.

```
A_nonsym = np.array([
    [0.10, 0.60, 0.30],
    [0.40, 0.20, 0.40],
    [0.50, 0.20, 0.30]
```

```
])
```

```
w, V, method = eig_auto(A_nonsym)
err = check_eigendecomposition(A_nonsym, w, V)

print("\n==== CAS NON SYMÉTRIQUE ===")
print("Méthode choisie : ", method)
print("Valeurs propres w : ", w)
print("Vecteurs propres (colonnes de V) :\n", V)
print("Erreur relative ||AV - V diag(w)|| / ||AV|| : ", err)

# Remarque : les résultats peuvent être complexes ; c'est normal en non-symétrique.
```

Interprétation très concrète (chaîne de Markov) :

- Si c'est une matrice de transition (lignes qui somment à 1), la valeur propre **1** est centrale.
- Le vecteur propre associé donne un état “stationnaire” (à gauche pour Markov, mais on peut le récupérer via `A.T`).

Si vous voulez la **distribution stationnaire** (vecteur propre à gauche) :

```
# Distribution stationnaire pi telle que pi @ A = pi
wT, VT, _ = eig_auto(A_nonsym.T)
i = np.argmin(np.abs(wT - 1))
pi = np.real(VT[:, i])
pi = pi / pi.sum()

print("\nDistribution stationnaire pi : ", pi)
print("Vérif pi @ A : ", pi @ A_nonsym)
print("Différence (pi@A - pi) : ", (pi @ A_nonsym) - pi)
```

3) Notes importantes (pour “n’importe quelle matrice”)

- **Carrée obligatoire** : les valeurs propres/vecteurs propres sont définis pour $n \times nn$ \times $nn \times n$.
- **Non symétrique** :
 - valeurs propres/vecteurs propres peuvent être **complexes**
 - si la matrice est **défectueuse** (pas assez de vecteurs propres indépendants), `eig` renvoie quand même quelque chose, mais l’“ensemble complet” de vecteurs propres peut ne pas exister (c’est un fait mathématique, pas un bug).
- Si vous avez besoin d’une représentation *toujours possible* même en cas défectueux, on utilise plutôt **Schur** (ou Jordan, mais numériquement très instable). Le code ci-dessus peut déjà “préférer Schur” pour diagnostic, mais pour “vecteurs propres complets”, la matrice peut ne pas en avoir.

Si vous me dites :

- matrice **réelle ou complexe** ?
- taille typique (3×3 , 100×100 , 1000×1000) ?
- et si vous voulez **toutes** les valeurs propres ou seulement quelques-unes (les plus grandes) ?

... je peux adapter le “super-algorithme” pour être **beaucoup plus rapide** sur grandes matrices (méthodes itératives type Lanczos/Arnoldi via SciPy).

L'utilisateur devrait pouvoir saisir la dimension de la matrice lui-même et si vous voulez toutes les valeurs propres au grand complet ... je peux adapter le “super-algorithme” pour être beaucoup plus rapide sur grandes matrices (méthodes itératives type Lanczos/Arnoldi via SciPy) et sur toutes les matrices également

Voici une version **complète**, “**prête à lancer**”, où l'utilisateur :

- saisit **la dimension** nnn,
- saisit **tous les coefficients** de la matrice,
- le programme détecte **symétrique/hermitienne** vs **générale**,
- choisit automatiquement la méthode la plus **performante** pour **toutes** les valeurs propres :
 - **symétrique/hermitienne** → **eigh** (optimal)
 - **non symétrique** → **eig** (cas général)
- affiche aussi des **vérifications**.

Note importante : pour “toutes les valeurs propres” d'une matrice dense $n \times nn$ times $nn \times n$, le coût est en gros $O(n^3)$. Les méthodes Lanczos/Arnoldi sont surtout utiles quand on veut **quelques** valeurs propres ($k \ll n$), pas la totalité.

Programme complet (entrée utilisateur + choix automatique)

```
import numpy as np

def is_hermitian(A, rtol=1e-10, atol=1e-12):
    A = np.asarray(A)
    return np.allclose(A, A.conj().T, rtol=rtol, atol=atol)

def eig_auto_full(A, rtol=1e-10, atol=1e-12, sort=True):
```

```

"""
Calcule TOUTES les valeurs propres et vecteurs propres.
- Si A est symétrique/hermitienne -> eigh (plus performant + stable)
- Sinon -> eig (cas général)
"""

A = np.asarray(A)
if A.ndim != 2 or A.shape[0] != A.shape[1]:
    raise ValueError("A doit être carrée (n x n).")

if is_hermitian(A, rtol=rtol, atol=atol):
    w, V = np.linalg.eigh(A)
    method = "numpy.linalg.eigh (symétrique/hermitienne)"
    # w est réel (à l'erreur numérique près)
else:
    w, V = np.linalg.eig(A)
    method = "numpy.linalg.eig (cas général non symétrique)"
    # w peut être complexe

if sort:
    # Tri par ordre croissant de la partie réelle, puis imaginaire
    idx = np.lexsort((np.imag(w), np.real(w)))
    w = w[idx]
    V = V[:, idx]

return w, V, method

def check_eigendecomposition(A, w, V):
    A = np.asarray(A)
    w = np.asarray(w)
    V = np.asarray(V)
    res = A @ V - V @ np.diag(w)
    num = np.linalg.norm(res, ord='fro')
    den = np.linalg.norm(A @ V, ord='fro') + 1e-30
    return num / den

def read_matrix_from_user():
    print("== Saisie d'une matrice ==")
    n = int(input("Entrez la dimension n (matrice n x n) : ").strip())
    if n <= 0:
        raise ValueError("n doit être un entier positif.")

    print("\nEntrez les coefficients ligne par ligne.")
    print("Format: a11 a12 ... a1n")
    print("Vous pouvez entrer des nombres complexes en Python (ex: 1+2j).\n")

    A = np.zeros((n, n), dtype=complex) # accepte réel ou complexe

```

```

for i in range(n):
    while True:
        line = input(f'Ligne {i+1} : ").strip()
        parts = line.split()
        if len(parts) != n:
            print(f" -> Erreur: vous devez entrer exactement {n} valeurs.")
            continue
        try:
            row = [complex(p) for p in parts]
        except ValueError:
            print(" -> Erreur: une des valeurs n'est pas un nombre valide (ex: 3, -1.2, 1+2j).")
            continue
        A[i, :] = row
        break

# Si tout est (quasi) réel, on repasse en float pour être plus rapide
if np.allclose(A.imag, 0.0, atol=1e-14):
    A = A.real.astype(float)

return A

def main():
    A = read_matrix_from_user()

    print("\n==== Analyse ===")
    print("Matrice A =\n", A)
    print("Symétrique/Hermitienne ? ->", is_hermitian(A))

    w, V, method = eig_auto_full(A)

    print("\n==== Résultats ===")
    print("Méthode choisie :", method)
    print("\nValeurs propres (w) :")
    print(w)

    print("\nVecteurs propres (colonnes de V) :")
    print(V)

    err = check_eigendecomposition(A, w, V)
    print("\n==== Vérification ===")
    print("Erreur relative ||A V - V diag(w)|| / ||A V|| =", err)

    # Bonus : si hermitienne, vérif orthonormalité
    if is_hermitian(A):
        print("\nV* V (doit ~ I) :")
        print(V.conj().T @ V)

```

```
if __name__ == "__main__":
    main()
```

Exemple très concret et très détaillé (simulation d'une exécution)

Exemple 1 — Matrice symétrique (donc `eigh`)

Saisie :

- `n=3`
- `n=3`
- matrice :

`A=(411130102)A=\begin{pmatrix} 4 & 1 & 1 \\ 1 & 3 & 0 \\ 1 & 0 & 2 \end{pmatrix}A=411130102`

Entrée utilisateur (console) :

```
Entrez la dimension n (matrice n x n) : 3
Ligne 1 : 4 1 1
Ligne 2 : 1 3 0
Ligne 3 : 1 0 2
```

Ce que le programme fait :

1. teste `A == A.T` → vrai
2. choisit `numpy.linalg.eigh` (optimisé symétrique)
3. renvoie :
 - valeurs propres réelles triées
 - vecteurs propres orthonormés ($V^*V \approx I$)
4. affiche l'erreur de reconstruction $\|AV - V\text{diag}(w)\|$ (proche de 0)

Exemple 2 — Matrice non symétrique (donc `eig`)

Saisie :

- $n=3$
- matrice (transition / dynamique) :

$B=(0.100.600.300.400.200.400.500.200.30)$
 $B=\begin{pmatrix} 0.10 & 0.60 & 0.30 \\ 0.40 & 0.20 & 0.40 \\ 0.50 & 0.20 & 0.30 \end{pmatrix}$
 $B=0.100.400.50$
 $0.600.200.300.400.30$

Entrée :

```
Entrez la dimension n (matrice n x n) : 3
Ligne 1 : 0.10 0.60 0.30
Ligne 2 : 0.40 0.20 0.40
Ligne 3 : 0.50 0.20 0.30
```

Ce que le programme fait :

1. test symétrie → faux
2. choisit `numpy.linalg.eig`
3. renvoie valeurs propres/vecteurs propres (possiblement complexes)
4. vérifie $BV \approx V\Lambda V^{-1}$

Et Lanczos/Arnoldi “pour être beaucoup plus rapide” : quand et comment ?

Pour **toutes** les valeurs propres :

- sur matrice dense → on doit quasiment toujours faire un algo $O(n^3)$ (Schur/QR) : `eig` / `eigh` restent les meilleurs choix.

Pour être **beaucoup plus rapide**, il faut typiquement :

- **grande matrice** (ex: 10 000 x 10 000)
- **creuse (sparse)** ou produit matrice-vecteur rapide
- et on ne veut que **k** valeurs propres (ex: 6 plus grandes)

Dans ce cas, on utilise SciPy :

- `scipy.sparse.linalg.eigsh` (symétrique) → Lanczos
- `scipy.sparse.linalg.eigs` (non symétrique) → Arnoldi

Si vous voulez, je peux vous donner **une version “hybride”** qui :

- calcule **toutes** les valeurs propres si nnn est “petit” (ex: ≤ 800),
- bascule automatiquement vers Lanczos/Arnoldi si nnn est grand et l’utilisateur choisit un nombre kkk de valeurs propres (ce qui est le seul cas où on gagne vraiment).

Dites-moi juste : vous voulez fixer un seuil (ex: 500, 1000) ? et est-ce que vous voulez accepter des matrices **creuses** (saisie en liste de triplets i,j,val) ?

Je voudrais que le programme python soit capable de trouver les valeurs propres et les vecteurs propres de quasiment toutes les matrices sans aucune distinction
Réflexion durant 18s

Je comprends l’idée (“zéro distinction, ça marche pour tout”), mais il y a une limite **mathématique** incontournable :

- **Toutes les matrices carrées** ont des **valeurs propres** (en général **complexes**) et on peut **toujours** les calculer numériquement.
- En revanche, **toutes les matrices n’ont pas une base complète de vecteurs propres**.
Exemple classique (bloc de Jordan) :

$$A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Elle a bien la valeur propre 111 (multiplicité 2), mais **un seul vecteur propre indépendant** → impossible de retourner nnn vecteurs propres linéairement indépendants, parce qu’ils **n’existent pas**.

 Donc le meilleur “super-programme” réaliste est :

1. **toujours** retourner les **valeurs propres**
2. retourner des **vecteurs propres** quand ils existent en nombre suffisant et sont numériquement fiables
3. sinon, retourner une alternative **toujours définie** : la **décomposition de Schur** (très robuste), qui donne une base orthonormale d'**espaces invariants** (ce que les solveurs QR/Schur utilisent en interne).

Programme Python robuste “quasi toutes matrices”

- Saisie de la dimension nnn et des coefficients par l’utilisateur

- Calcul via **Schur** (**SciPy**) si dispo (le plus robuste)
- Sinon fallback **NumPy eig**
- Vérifie le résidu $\|AV - V\Lambda\| \|AV - V\Lambda\| \|AV - V\Lambda\|$ et la “qualité” des vecteurs propres
- Si matrice défectueuse / instable : il l’annonce et fournit **Schur** (toujours possible)

```

import numpy as np

def read_matrix_from_user():
    print("== Saisie d'une matrice ==")
    n = int(input("Entrez la dimension n (matrice n x n) : ").strip())
    if n <= 0:
        raise ValueError("n doit être > 0")

    print("\nEnterz les coefficients ligne par ligne (séparés par espaces).")
    print("Réels: 3 -1.2 0")
    print("Complexes: 1+2j 3 -0.5j\n")

    A = np.zeros((n, n), dtype=complex)
    for i in range(n):
        while True:
            line = input(f"Ligne {i+1} : ").strip().split()
            if len(line) != n:
                print(f"  -> Il faut exactement {n} valeurs.")
                continue
            try:
                A[i, :] = [complex(x) for x in line]
                break
            except ValueError:
                print("  -> Valeur invalide. Exemple: 2, -1.5, 1+2j")
    # Si (quasi) réel, repasser en float (plus rapide)
    if np.allclose(A.imag, 0.0, atol=1e-14):
        return A.real.astype(float)
    return A

def residual_rel(A, w, V):
    """Erreur relative ||AV - V diag(w)||_F / ||AV||_F."""
    AV = A @ V
    R = AV - V @ np.diag(w)
    num = np.linalg.norm(R, ord='fro')
    den = np.linalg.norm(AV, ord='fro') + 1e-30
    return num / den

def robust_eigen(A, want_vectors=True, quality_tol=1e-8, cond_tol=1e12):
    """
    Retourne un dict avec:

```

```

- eigenvalues: toujours
- eigenvectors: si disponibles/fiables
- schur_T, schur_Z: (si SciPy dispo) toujours (robuste)
- status/message
"""

A = np.asarray(A)
if A.ndim != 2 or A.shape[0] != A.shape[1]:
    raise ValueError("La matrice doit être carrée.")

n = A.shape[0]
out = {"n": n}

# 1) Essayer la voie la plus robuste: Schur (SciPy)
try:
    from scipy.linalg import schur

    # Schur complexe: A = Z T Z^H, T triangulaire (toujours défini)
    T, Z = schur(A, output='complex')
    w = np.diag(T).copy()
    out["eigenvalues"] = w
    out["schur_T"] = T
    out["schur_Z"] = Z

    if not want_vectors:
        out["status"] = "ok"
        out["message"] = "Valeurs propres via Schur (vecteurs non demandés)."
    return out

# 2) Calculer des vecteurs propres à partir de T (triangulaire), puis remonter via Z
#     (On peut utiliser eig sur T, qui est numériquement plus stable qu'un eig direct parfois)
from scipy.linalg import eig as scipy_eig
w2, X = scipy_eig(T)           # T X = X diag(w2)
V = Z @ X                      # A (Z X) = (Z X) diag(w)

# 3) Contrôles de qualité
#     - résidu relatif
err = residual_rel(A, w2, V)
#     - conditionnement des vecteurs propres (si V est quasi singulière -> matrice défectueuse / proche)
try:
    cV = np.linalg.cond(V)
except np.linalg.LinAlgError:
    cV = np.inf

out["eigenvalues"] = w2
out["eigenvectors"] = V
out["residual_rel"] = float(err)

```

```

out["cond_vectors"] = float(cV)

if err < quality_tol and cV < cond_tol:
    out["status"] = "ok"
    out["message"] = "Vecteurs propres calculés et jugés numériquement fiables."
else:
    # Cas typique: matrice défectueuse ou presque-défectueuse
    # -> les valeurs propres sont ok, mais les vecteurs propres peuvent être très mal conditionnés
    out["status"] = "warning"
    out["message"] = (
        "Valeurs propres OK. Vecteurs propres potentiellement instables (matrice défectueuse ou proche). "
        "Utilisez plutôt la base de Schur (schur_Z) pour travailler avec des sous-espaces invariants."
    )
return out

except Exception:
    # 4) Fallback NumPy (pas SciPy)
    w, V = np.linalg.eig(A)
    out["eigenvalues"] = w
    if want_vectors:
        out["eigenvectors"] = V
        out["residual_rel"] = float(residual_rel(A, w, V))
        out["status"] = "ok"
        out["message"] = "Calcul via numpy.linalg.eig (SciPy/Schur non disponible)."
    else:
        out["status"] = "ok"
        out["message"] = "Valeurs propres via numpy.linalg.eig."
    return out

def main():
    A = read_matrix_from_user()

    print("\n==== Matrice A ===")
    print(A)

    result = robust_eigen(A, want_vectors=True)

    print("\n==== Résultat ===")
    print("Status :", result["status"])
    print("Message:", result["message"])

    print("\nValeurs propres :")
    print(result["eigenvalues"])

    if "eigenvectors" in result:
        print("\nVecteurs propres (colonnes) :")

```

```

print(result["eigenvectors"])
print("\nQualité:")
if "residual_rel" in result:
    print(" Résidu relatif : ", result["residual_rel"])
if "cond_vectors" in result:
    print(" cond(V) : ", result["cond_vectors"])

if "schur_T" in result and "schur_Z" in result:
    print("\n(Info robuste) Décomposition de Schur disponible:")
    print("T (triangulaire) =")
    print(result["schur_T"])
    # Z est souvent grande, on l'affiche uniquement si vous voulez:
    # print("Z (unitaire) =\n", result["schur_Z"])

if __name__ == "__main__":
    main()

```

Exemple très concret et détaillé : matrice “problème” (défectueuse)

Prenons **justement** la matrice :

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Saisie console

```

Entrez la dimension n (matrice n x n) : 2
Ligne 1 : 1 1
Ligne 2 : 0 1

```

Ce qui se passe

- Le programme retourne **les valeurs propres** : $[1,1][1, 1][1,1]$
- Il essaie de produire 2 vecteurs propres indépendants, mais :
 - mathématiquement, il n'y en a **qu'un seul**
 - numériquement, on voit typiquement :
 - soit `cond(V)` énorme
 - soit un avertissement “vecteurs propres instables”
- Et il propose la **décomposition de Schur**, qui elle est **toujours** bien définie et stable.

👉 Résultat : vous avez un programme qui “marche sur quasiment toutes les matrices” au sens opérationnel :

- **toujours** des valeurs propres
 - des vecteurs propres **quand ils existent et sont utilisables**
 - sinon une représentation robuste **Schur** (base orthonormale d'invariance), qui est ce qu'on utilise en calcul scientifique sérieux quand les vecteurs propres deviennent pathologiques.
-

Si vous me dites votre contexte (contrôle/dynamique, graphes, données/PCA, etc.), je peux aussi adapter la sortie pour **produire automatiquement** des *vecteurs propres généralisés* (chaînes de Jordan) dans certains cas — mais je préfère être transparent : la “forme de Jordan” est **très instable numériquement** et on évite souvent de la calculer en pratique.

Alors est ce que le programme que vous avez donné ci-dessus permet t il de corriger cette limite mathématique incontournable

Non — et personne ne peut “corriger” cette limite, parce que ce n'est pas une limite de programme, c'est une **propriété mathématique** : certaines matrices **n'ont pas** assez de vecteurs propres indépendants pour former une base.

Ce que le programme fait (et c'est le mieux qu'on puisse faire en pratique) :

- Toujours calculer des **valeurs propres** (souvent complexes).
 - Calculer des **vecteurs propres** quand ils existent en nombre suffisant et sont numériquement stables.
 - Déetecter quand les vecteurs propres deviennent **instables / quasi inexistant**s (matrice défectueuse ou presque).
 - Dans ce cas, fournir une alternative **robuste** qui existe toujours : la **décomposition de Schur**.
-

Pourquoi la limite est “incontournable” (exemple concret)

Prenez :

$$A = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

- Valeur propre : $\lambda=1$ ($\lambda=1$ (multiplicité 2))
- Vecteurs propres : solutions de $(A-I)v=0$ ($(A-I)v=0$)

$A - I = (0100) \Rightarrow v_2 = 0$, v_1 libre
 $A - I = \begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} \Rightarrow v_2 = 0, v_1$ libre

Donc **un seul** vecteur propre indépendant : $(10) \begin{pmatrix} 1 & 0 \end{pmatrix}$.

👉 Il est donc **impossible** de renvoyer 2 vecteurs propres indépendants, parce qu'ils **n'existent pas**.

Donc, que fait la “Schur” dans le programme ?

La Schur donne :

$$A = ZTZ^* = ZTZ^* A = ZTZ^*$$

- Z est **unitaire** (base orthonormale stable)
- T est **triangulaire**
- les valeurs propres sont sur la diagonale de T

Même si la matrice est défectueuse, **Schur marche** et reste stable numériquement.

Mais ce ne sont pas “des vecteurs propres complets”, c’est une base de **sous-espaces invariants**.

Si vous voulez vraiment “quelque chose à la place” des vecteurs propres

Deux options usuelles (chacune avec ses compromis) :

1. **Vecteurs propres généralisés** (chaînes de Jordan)
 - Ça “remplit” la dimension manquante
 - ⚠️ Très instable numériquement en général
2. **Schur / quasi-Jordan** (ce que font les solveurs pro)
 - ✅ Stable
 - Donne une structure exploitable pour calculer $AkA^{-1}kAk$, $eAt^{-1}\{At\}eAt$, résoudre des systèmes, etc.

Conclusion

- **Le programme ne supprime pas la limite**, impossible.
- **Il l'encadre** : il donne les valeurs propres + vecteurs propres quand c'est mathématiquement et numériquement sain, sinon il bascule vers une représentation (Schur) **qui existe toujours**.

Si vous voulez, je peux modifier le programme pour qu'il ait un mode :

- “**force_full_basis=True**” → il retourne *soit* une base de vecteurs propres (si possible), *soit* une base de Schur (toujours possible), avec un message clair “ce ne sont pas des vecteurs propres mais une base invariante robuste”.

Oui bien sûr et je voudrais que l'utilisateur puisse saisir la matrice avec ses coefficients dans une console. Pouvez vous produire un fichier pdf qui résumerait toutes les étapes importantes comme l'étape depuis la saisie de la matrice jusqu'à la proclamation des résultats

Réflexion durant 13s

Réflexion durant 34s

J'ai généré le PDF qui résume toutes les étapes (saisie console → calcul robuste → vérifications → proclamation des résultats, avec exemples et code complet).

Télécharger le PDF

Ok merci. Et si cela est possible, pouvez vous utiliser PyQt6 et QtDesigner pour une interface responsive et très élégante avec l'algorithme que vous m'avez donné ? S'il vous plaît ?

Oui. Voici une proposition **propre, responsive et élégante** avec **PyQt6 + Qt Designer**, qui réutilise exactement l'approche “robuste” (valeurs propres toujours + vecteurs propres si fiables + Schur si dispo).

Je te donne :

1. le workflow Qt Designer (fichier `.ui`)
2. un code Python **production-ready** qui charge le `.ui`, exécute le calcul, et affiche les résultats
3. un exemple d'utilisation concret

1) Qt Designer : interface (responsive)

A. Créer l'UI

1. Ouvre **Qt Designer**
2. Crée une fenêtre **Main Window**
3. Place un **centralWidget** avec un **QVBoxLayout** (important pour le responsive)
4. Ajoute ces éléments (dans l'ordre) :

Zone “Entrée”

- **QFormLayout**

- `QSpinBox (objectName: spinN) → Dimension n`
 - min: 1, max: 2000 (ou ce que tu veux)
- `QPushButton (objectName: btnCreate) → “Créer matrice”`
- `QComboBox (objectName: comboMode) → “Réels” / “Complexes”`
- `QDoubleSpinBox (objectName: spinTol) → Tolérance qualité (ex: 1e-8)`
- `QCheckBox (objectName: chkSciPy) → “Utiliser Schur si SciPy dispo” (coché par défaut)`

Tableau de matrice

- `QTableWidget (objectName: tableA)`
 - `sizePolicy: Expanding/Expanding`
 - `horizontalHeader: Stretch`
 - `verticalHeader: Stretch (optionnel)`

Boutons

- `QHBoxLayout`
 - `QPushButton btnExampleSym → “Exemple symétrique”`
 - `QPushButton btnExampleNonSym → “Exemple non symétrique”`
 - `QPushButton btnExampleDef → “Exemple défectueux”`
 - `QPushButton btnCompute → “Calculer”`

Sortie

- `QTabWidget`
 - Tab1: “Valeurs propres” → `QPlainTextEdit (objectName: txtEigenvalues, readOnly)`
 - Tab2: “Vecteurs propres” → `QPlainTextEdit (objectName: txtEigenvectors, readOnly)`

- o Tab3: “Diagnostic / Schur” → QPlainTextEdit (objectName: txtDiag, readOnly)

5. Enregistre le fichier sous eigen_gui.ui

Tu peux aussi ajouter un QLabel en haut pour un titre, et appliquer un style QSS (je le fais dans le code ci-dessous).

2) Code Python (PyQt6) — charge le .ui et calcule

Crée un fichier : eigen_gui.py

```
import sys
import numpy as np
from PyQt6 import QtWidgets, uic
from PyQt6.QtCore import Qt

# -----
# Coeur algo "robuste"
# -----
def residual_rel(A, w, V):
    AV = A @ V
    R = AV - V @ np.diag(w)
    num = np.linalg.norm(R, ord='fro')
    den = np.linalg.norm(AV, ord='fro') + 1e-30
    return num / den

def robust_eigen(A, want_vectors=True, quality_tol=1e-8, cond_tol=1e12, prefer_schur=True):
    A = np.asarray(A)
    if A.ndim != 2 or A.shape[0] != A.shape[1]:
        raise ValueError("La matrice doit être carrée.")

    out = {"n": A.shape[0]}

    if prefer_schur:
        try:
            from scipy.linalg import schur, eig as scipy_eig

            # Schur complexe : A = Z T Z*
            T, Z = schur(A, output='complex')
            out["schur_T"] = T
            out["schur_Z"] = Z
        except ImportError:
            print("Scipy not found, falling back to QR algorithm")
            out["eig"] = eig(A, want_vectors=want_vectors, quality_tol=quality_tol, cond_tol=cond_tol)
    else:
        out["eig"] = eig(A, want_vectors=want_vectors, quality_tol=quality_tol, cond_tol=cond_tol)

    return out
```

```

out["eigenvalues"] = np.diag(T).copy()

if not want_vectors:
    out["status"] = "ok"
    out["message"] = "Valeurs propres via Schur."
    return out

# Vecteurs propres via T puis remontée
w2, X = scipy_eig(T)
V = Z @ X

err = residual_rel(A, w2, V)
try:
    cV = np.linalg.cond(V)
except np.linalg.LinAlgError:
    cV = np.inf

out["eigenvalues"] = w2
out["eigenvectors"] = V
out["residual_rel"] = float(err)
out["cond_vectors"] = float(cV)

if err < quality_tol and cV < cond_tol:
    out["status"] = "ok"
    out["message"] = "Vecteurs propres calculés et jugés fiables."
else:
    out["status"] = "warning"
    out["message"] = (
        "Valeurs propres OK. Vecteurs propres potentiellement instables "
        "(matrice défectueuse ou proche). Utilisez Schur (Z, T)."
    )
return out

except Exception:
    # SciPy absent ou erreur -> fallback numpy
    pass

# Fallback NumPy (toujours dispo avec numpy)
w, V = np.linalg.eig(A)
out["eigenvalues"] = w
if want_vectors:
    out["eigenvectors"] = V
    out["residual_rel"] = float(residual_rel(A, w, V))
    out["cond_vectors"] = float(np.linalg.cond(V)) if A.shape[0] > 0 else 0.0
out["status"] = "ok"
out["message"] = "Calcul via numpy.linalg.eig."

```

```

    return out

# -----
# GUI PyQt6
# -----
APP_QSS = """
QMainWindow { background: #0b1220; }
QWidget { color: #e5e7eb; font-size: 12px; }
QLabel#lblTitle { font-size: 18px; font-weight: 700; padding: 6px; }
QGroupBox { border: 1px solid #24324a; border-radius: 10px; margin-top: 10px; }
QGroupBox::title { subcontrol-origin: margin; left: 12px; padding: 0 6px; color: #cbd5e1; }
QPushButton {
    background: #1f2a44; border: 1px solid #2b3a59; padding: 8px 12px;
    border-radius: 10px;
}
QPushButton:hover { background: #24324a; }
QPushButton:pressed { background: #2b3a59; }
QPlainTextEdit, QTableWidget {
    background: #0f172a; border: 1px solid #24324a; border-radius: 10px;
}
QHeaderView::section { background: #111c33; border: 1px solid #24324a; padding: 6px; }
QTabWidget::pane { border: 1px solid #24324a; border-radius: 10px; }
QTabBar::tab { background: #111c33; padding: 8px 10px; border-top-left-radius: 10px; border-top-right-radius: 10px; }
QTabBar::tab:selected { background: #1f2a44; }
"""

def parse_cell(text: str, mode: str):
    """
    mode: "real" ou "complex"
    - real: accepte float
    - complex: accepte complex (ex: 1+2j)
    """
    text = text.strip()
    if text == "":
        return 0.0 if mode == "real" else 0.0 + 0.0j
    if mode == "real":
        return float(text)
    return complex(text)

class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        # Charger le .ui crée via Qt Designer
        uic.loadUi("eigen_gui.ui", self)

```

```

self.setWindowTitle("Valeurs propres / Vecteurs propres - robuste")
self.setStyleSheet(APP_QSS)

# Connexions
self.btnCreate.clicked.connect(self.create_matrix)
self.btnCompute.clicked.connect(self.compute)

self.btnExampleSym.clicked.connect(self.load_example_sym)
self.btnExampleNonSym.clicked.connect(self.load_example_nonsym)
self.btnExampleDef.clicked.connect(self.load_example_defective)

# Init
self.spinN.setValue(3)
self.create_matrix()

def mode(self):
    # comboMode: "Réels" / "Complexes"
    txt = self.comboMode.currentText().lower()
    return "complex" if "complex" in txt else "real"

def create_matrix(self):
    n = int(self.spinN.value())
    self.tableA.setRowCount(n)
    self.tableA.setColumnCount(n)
    self.tableA.horizontalHeader().setSectionResizeMode(QtWidgets.QHeaderView.ResizeMode.Stretch)
    self.tableA.verticalHeader().setSectionResizeMode(QtWidgets.QHeaderView.ResizeMode.Stretch)

    # Pré-remplir à 0
    for i in range(n):
        for j in range(n):
            item = QtWidgets.QTableWidgetItem("0")
            item.setTextAlignment(Qt.AlignmentFlag.AlignCenter)
            self.tableA.setItem(i, j, item)

def read_matrix(self):
    n = self.tableA.rowCount()
    mode = self.mode()
    A = np.zeros((n, n), dtype=complex if mode == "complex" else float)

    for i in range(n):
        for j in range(n):
            item = self.tableA.item(i, j)
            txt = item.text() if item else "0"
            val = parse_cell(txt, mode)
            A[i, j] = val

```

```

# si tout réel, repasser en float
if np.iscomplexobj(A) and np.allclose(A.imag, 0.0, atol=1e-14):
    A = A.real.astype(float)
return A

def set_matrix(self, A):
    A = np.asarray(A)
    n = A.shape[0]
    self.spinN.setValue(n)
    self.create_matrix()

    # Adapter mode
    if np.iscomplexobj(A) and not np.allclose(A.imag, 0.0, atol=1e-14):
        self.comboMode.setCurrentText("Complexes")
    else:
        self.comboMode.setCurrentText("Réels")

    for i in range(n):
        for j in range(n):
            v = A[i, j]
            s = str(v)
            # Rendre plus lisible si float
            if isinstance(v, (float, np.floating)):
                s = f"{v:.6g}"
            item = QtWidgets.QTableWidgetItem(s)
            item.setTextAlignment(Qt.AlignmentFlag.AlignCenter)
            self.tableA.setItem(i, j, item)

def load_example_sym(self):
    A = np.array([[4.0, 1.0, 1.0],
                  [1.0, 3.0, 0.0],
                  [1.0, 0.0, 2.0]])
    self.set_matrix(A)

def load_example_nonsym(self):
    A = np.array([[0.10, 0.60, 0.30],
                  [0.40, 0.20, 0.40],
                  [0.50, 0.20, 0.30]])
    self.set_matrix(A)

def load_example_defective(self):
    A = np.array([[1.0, 1.0],
                  [0.0, 1.0]])
    self.set_matrix(A)

def compute(self):

```

```

try:
    A = self.read_matrix()
    tol = float(self.spinTol.value())
    prefer_schur = bool(self.chkSciPy.isChecked())

    result = robust_eigen(
        A,
        want_vectors=True,
        quality_tol=tol,
        cond_tol=1e12,
        prefer_schur=prefer_schur,
    )

    # Affichage valeurs propres
    w = result["eigenvalues"]
    self.txtEigenvalues.setPlainText(
        "\n".join([f"{k}: {w[k]}!r" for k in range(len(w))])
    )

    # Affichage vecteurs propres
    if "eigenvectors" in result:
        V = result["eigenvectors"]
        self.txtEigenvectors.setPlainText(np.array2string(V, precision=6, suppress_small=True))
    else:
        self.txtEigenvectors.setPlainText("(Non calculés)")

    # Diagnostic
    diag_lines = [
        f"Status: {result.get('status')}",
        f"Message: {result.get('message')}",
    ]
    if "residual_rel" in result:
        diag_lines.append(f"Résidu relatif ||AV - VΛ|| / ||AV|| = {result['residual_rel']:.3e}")
    if "cond_vectors" in result:
        diag_lines.append(f"cond(V) = {result['cond_vectors']:.3e}")

    if "schur_T" in result:
        diag_lines.append("\nSchur disponible (A = Z T Z*).")
        diag_lines.append("T =")
        diag_lines.append(np.array2string(result["schur_T"], precision=6, suppress_small=True))

    self.txtDiag.setPlainText("\n".join(diag_lines))

except Exception as e:
    QtWidgets.QMessageBox.critical(self, "Erreur", str(e))

```

```
def main():
    app = QtWidgets.QApplication(sys.argv)
    w = MainWindow()
    w.resize(1100, 700) # joli format
    w.show()
    sys.exit(app.exec())

if __name__ == "__main__":
    main()
```

3) Lancer l'application

Installation (si besoin)

```
pip install PyQt6 numpy
pip install scipy # optionnel mais recommandé (Schur robuste)
```

Exécution

```
python eigen_gui.py
```

4) Résultat attendu (très concret)

- Tu choisis $n=3$, tu cliques “Créer matrice”
- Tu remplis le tableau (ou tu cliques un bouton “Exemple ...”)
- Tu cliques **Calculer**
- L’onglet **Valeurs propres** affiche la liste
- L’onglet **Vecteurs propres** affiche la matrice des vecteurs propres (colonnes)
- L’onglet **Diagnostic / Schur** affiche :
 - Status ok si vecteurs propres fiables
 - warning si matrice défectueuse / proche défectueuse
 - et T (Schur) si SciPy est dispo

Si tu veux, je peux aussi :

- ajouter un bouton “**Exporter en PDF**” depuis l’interface (mêmes étapes que ton PDF),
- ajouter une option **coller une matrice** en texte (genre copier-coller depuis MATLAB/NumPy),
- améliorer encore le “look” (icônes, animations légères, thème clair/sombre).

Oui volontiers. Vous pouvez même ajouter une autre option comme "Exporter vers Excel" ou encore "Exporter vers JSON"

Parfait — on peut ajouter **Exporter PDF**, **Exporter Excel (.xlsx)** et **Exporter JSON** de façon très propre.

Je te donne ci-dessous :

- les **ajouts Qt Designer** (3 boutons)
- le **code PyQt6 complet** à ajouter (fonctions d’export + branchements)
- ce que chaque export contient (matrice A, résultats, diagnostics)

Prérequis exports :

```
pip install reportlab openpyxl
# (scipy optionnel mais recommandé pour Schur)
pip install scipy
```

1) Qt Designer : ajouter 3 boutons “Export”

Dans ton layout des boutons (là où tu as déjà “Calculer”, “Exemple …”), ajoute 3 `QPushButton` :

- `btnExportPDF` → texte “Exporter PDF”
- `btnExportExcel` → texte “Exporter Excel”
- `btnExportJSON` → texte “Exporter JSON”

(Place-les à droite du bouton “Calculer” ou sur une nouvelle ligne si tu préfères.)

2) Code : ajoute les exports dans `eigen_gui.py`

A) Ajouts d'imports (en haut du fichier)

Ajoute :

```
import json
from datetime import datetime
```

et pour les exports :

```
from reportlab.lib.pagesizes import A4
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Preformatted
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib import colors

from openpyxl import Workbook
from openpyxl.styles import Font, Alignment
```

B) Dans `MainWindow.__init__` : connecter les boutons

Ajoute après tes autres `.clicked.connect(...)` :

```
self.btnExitPDF.clicked.connect(self.export_pdf)
self.btnExitExcel.clicked.connect(self.export_excel)
self.btnExitJSON.clicked.connect(self.export_json)

self._last_result = None
self._last_A = None
```

C) Dans `compute()` : mémoriser les derniers résultats

À la fin de `compute()` (après `result = robust_eigen(...)` et avant d'afficher), ajoute :

```
self._last_result = result
self._last_A = A
```

D) Ajoute ces fonctions dans la classe `MainWindow`

Colle dans la classe MainWindow (au même niveau que compute, load_example_*, etc.):

```
def _ensure_result(self):
    if self._last_result is None or self._last_A is None:
        QtWidgets.QMessageBox.information(
            self, "Info",
            "Veuillez d'abord cliquer sur 'Calculer' avant d'exporter.")
    )
    return False
return True

def _result_to_serializable(self, A, result):
    # Convertir numpy -> python natif (y compris complexes)
    def conv(x):
        if isinstance(x, (np.floating, float, int)):
            return float(x)
        if isinstance(x, (np.integer,)):
            return int(x)
        if isinstance(x, complex) or isinstance(x, np.complexfloating):
            return {"re": float(np.real(x)), "im": float(np.imag(x))}

        if isinstance(x, np.ndarray):
            # convertir tableau (peut être complexe)
            if np.iscomplexobj(x):
                return [[conv(v) for v in row] for row in x]
            return x.tolist()
        return x

    payload = {
        "created_at": datetime.now().isoformat(timespec="seconds"),
        "n": int(A.shape[0]),
        "A": conv(np.asarray(A)),
        "status": result.get("status"),
        "message": result.get("message"),
        "eigenvalues": conv(np.asarray(result.get("eigenvalues"))),
        "eigenvectors": conv(np.asarray(result["eigenvectors"])) if "eigenvectors" in result else None,
        "residual_rel": result.get("residual_rel"),
        "cond_vectors": result.get("cond_vectors"),
        "schur_T": conv(np.asarray(result["schur_T"])) if "schur_T" in result else None,
        # schur_Z peut être très grand; on le met optionnellement (ici: non)
        "schur_Z_included": False,
    }
    return payload

def export_json(self):
    if not self._ensure_result():
```

```

    return

path, _ = QtWidgets.QFileDialog.getSaveFileName(
    self, "Exporter JSON", "resultats.json", "JSON (*.json)"
)
if not path:
    return

payload = self._result_to_serializable(self._last_A, self._last_result)
with open(path, "w", encoding="utf-8") as f:
    json.dump(payload, f, ensure_ascii=False, indent=2)

QtWidgets.QMessageBox.information(self, "Export JSON", f"Fichier créé :\n{path}")

def export_excel(self):
    if not self._ensure_result():
        return

path, _ = QtWidgets.QFileDialog.getSaveFileName(
    self, "Exporter Excel", "resultats.xlsx", "Excel (*.xlsx)"
)
if not path:
    return

A = np.asarray(self._last_A)
result = self._last_result
w = np.asarray(result["eigenvalues"])
V = np.asarray(result["eigenvectors"]) if "eigenvectors" in result else None

wb = Workbook()

# Feuille 1 : Matrice A
wsA = wb.active
wsA.title = "A"
wsA["A1"] = "Matrice A"
wsA["A1"].font = Font(bold=True, size=14)

# écrire A à partir de A3
start_row, start_col = 3, 1
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        val = A[i, j]
        # Excel ne gère pas les complexes directement -> écrire en texte si complexe
        if np.iscomplexobj(A) and abs(np.imag(val)) > 1e-14:
            cell_val = f"{val.real:.12g}+{val.imag:.12g}j"
        else:

```

```

    cell_val = float(np.real(val))
    wsA.cell(row=start_row+i, column=start_col+j, value=cell_val)

# Feuille 2 : Valeurs propres
wsW = wb.create_sheet("Valeurs_propres")
wsW["A1"] = "Valeurs propres"
wsW["A1"].font = Font(bold=True, size=14)
wsW["A2"] = "Index"
wsW["B2"] = "Valeur"
wsW["A2"].font = wsW["B2"].font = Font(bold=True)
for k in range(len(w)):
    wsW.cell(row=3+k, column=1, value=k)
    val = w[k]
    if np.iscomplexobj(w) and abs(np.imag(val)) > 1e-14:
        wsW.cell(row=3+k, column=2, value=f"{val.real:.12g}+{val.imag:.12g}j")
    else:
        wsW.cell(row=3+k, column=2, value=float(np.real(val)))

# Feuille 3 : Vecteurs propres (colonnes)
wsV = wb.create_sheet("Vecteurs_propres")
wsV["A1"] = "Vecteurs propres (colonnes)"
wsV["A1"].font = Font(bold=True, size=14)

if V is None:
    wsV["A3"] = "Non disponibles."
else:
    # écrire V à partir de A3
    for i in range(V.shape[0]):
        for j in range(V.shape[1]):
            val = V[i, j]
            if np.iscomplexobj(V) and abs(np.imag(val)) > 1e-14:
                cell_val = f"{val.real:.12g}+{val.imag:.12g}j"
            else:
                cell_val = float(np.real(val))
            wsV.cell(row=3+i, column=1+j, value=cell_val)

# Feuille 4 : Diagnostic
wsD = wb.create_sheet("Diagnostic")
wsD["A1"] = "Diagnostic"
wsD["A1"].font = Font(bold=True, size=14)
lines = [
    f"Status: {result.get('status')}",
    f"Message: {result.get('message')}",
]
if "residual_rel" in result:
    lines.append(f"Résidu relatif: {result['residual_rel']:.3e}")

```

```

if "cond_vectors" in result:
    lines.append(f"cond(V): {result['cond_vectors']:.3e}")
if "schur_T" in result:
    lines.append("Schur: disponible (T stocké dans l'export JSON/PDF; ici non imprimé en entier)")
for idx, line in enumerate(lines, start=3):
    wsD.cell(row=idx, column=1, value=line)

# un peu de mise en forme
for ws in [wsA, wsW, wsV, wsD]:
    for col in range(1, min(10, ws.max_column)+1):
        ws.column_dimensions[chr(64+col)].width = 18
    for row in ws.iter_rows(min_row=1, max_row=min(ws.max_row, 200), min_col=1, max_col=min(ws.max_column, 20)):
        for cell in row:
            cell.alignment = Alignment(vertical="center")

wb.save(path)
QtWidgets.QMessageBox.information(self, "Export Excel", f"Fichier créé :{path}")

def export_pdf(self):
    if not self._ensure_result():
        return

    path, _ = QtWidgets.QFileDialog.getSaveFileName(
        self, "Exporter PDF", "resultats.pdf", "PDF (*.pdf)"
    )
    if not path:
        return

    A = np.asarray(self._last_A)
    result = self._last_result
    w = np.asarray(result["eigenvalues"])
    V = np.asarray(result["eigenvectors"]) if "eigenvectors" in result else None

    styles = getSampleStyleSheet()
    story = []
    story.append(Paragraph("Rapport – Valeurs propres et vecteurs propres (mode robuste)", styles["Title"]))
    story.append(Spacer(1, 8))

    story.append(Paragraph(f"Date: {datetime.now().isoformat(timespec='seconds')}", styles["Normal"]))
    story.append(Paragraph(f"Dimension n = {A.shape[0]}", styles["Normal"]))
    story.append(Spacer(1, 10))

    story.append(Paragraph("1) Matrice A", styles["Heading2"]))
    story.append(Preformatted(np.array2string(A, precision=6, suppress_small=True), styles["Code"]))
    story.append(Spacer(1, 8))

```

```

story.append(Paragraph("2) Résultats", styles["Heading2"]))
story.append(Paragraph(f"Status: <b>{result.get('status')}</b>", styles["Normal"]))
story.append(Paragraph(f"Message: {result.get('message')}", styles["Normal"]))
if "residual_rel" in result:
    story.append(Paragraph(f"Résidu relatif ||AV - VΛ|| / ||AV|| = {result['residual_rel']:.3e}", styles["Normal"]))
if "cond_vectors" in result:
    story.append(Paragraph(f"cond(V) = {result['cond_vectors']:.3e}", styles["Normal"]))
story.append(Spacer(1, 8))

story.append(Paragraph("3) Valeurs propres", styles["Heading2"]))
story.append(Preformatted(np.array2string(w, precision=12, suppress_small=True), styles["Code"]))
story.append(Spacer(1, 8))

story.append(Paragraph("4) Vecteurs propres (colonnes)", styles["Heading2"]))
if V is None:
    story.append(Paragraph("Non disponibles.", styles["Normal"]))
else:
    story.append(Preformatted(np.array2string(V, precision=6, suppress_small=True), styles["Code"]))

if "schur_T" in result:
    story.append(Spacer(1, 10))
    story.append(Paragraph("5) Schur (information robuste)", styles["Heading2"]))
    story.append(Paragraph("A = Z T Z* ; les valeurs propres sont sur la diagonale de T.", styles["Normal"]))
    story.append(Preformatted(np.array2string(result["schur_T"]), precision=6, suppress_small=True), styles["Code"])

doc = SimpleDocTemplate(path, pagesize=A4, title="Rapport valeurs propres")
doc.build(story)

QtWidgets.QMessageBox.information(self, "Export PDF", f"Fichier créé :\n{path}")

```

3) Ce que tu obtiens (très concret)

Export JSON

- A
- eigenvalues
- eigenvectors (si dispo)
- status, message, residual_rel, cond_vectors
- schur_T si SciPy dispo

 Parfait pour réutiliser les résultats dans un autre programme.

Export Excel

- Feuille **A** (matrice saisie)
 - Feuille **Valeurs_propres**
 - Feuille **Vecteurs_propres**
 - Feuille **Diagnostic**
- Pratique pour rendre un devoir / faire des captures / comparer.

Export PDF

- Rapport lisible (matrice, valeurs propres, vecteurs propres, diagnostics, Schur si dispo)
- Prêt à imprimer / envoyer.

Bonus (si tu veux “encore plus élégant”)

Je peux te proposer :

- un bouton “**Copier résultats**” (clipboard)
- import “**Coller matrice depuis texte**” (format NumPy/MATLAB)
- progress dialog si nnn est grand
- export Excel avec mise en forme plus avancée (couleurs, entêtes, gel de volets)

Si tu me dis si tu veux une **fenêtre MainWindow** ou une **Dialog** (plus minimaliste), je te donne une version finale “clé en main” avec un `.ui` exemple prêt à utiliser.

Oui je voudrais tous ce que vous me proposez ci-dessus avec une fenêtre MainWindow. De plus pouvez vous ajouter un onglet nommé "API Valeurs Propres"

Oui — on peut faire une **MainWindow** très “pro” et *responsive*, avec :

- saisie matrice via **QTableWidget**
- coller/importer depuis texte (NumPy / MATLAB-like)
- copier résultats (clipboard)
- progress dialog quand nnn est grand

- exports : PDF, Excel, JSON
- onglet “API Valeurs Propres” (documentation + démarrage d'une petite API locale optionnelle)

Ci-dessous je te donne :

1. ce qu'il faut ajouter dans **Qt Designer**
 2. un **code Python complet** (fichier unique) qui fait tout
-

1) Qt Designer : structure MainWindow + onglets

Dans **Qt Designer**, ton `MainWindow` contient :

A) Header

- `QLabel` (`objectName: lblTitle`) texte : “Valeurs propres & vecteurs propres — Mode robuste”

B) Zone d'entrée (GroupBox “Entrée”)

- `QSpinBox spinN` (`min 1, max 5000`)
- `QPushButton btnCreate` : “Créer matrice”
- `QComboBox comboMode` : “Réels”, “Complexes”
- `QDoubleSpinBox spinTol` : tolérance (ex: `1e-8`)
 - `decimals: 12, value: 1e-8, minimum: 1e-15, maximum: 1e-1` (ou ce que tu veux)
- `QCheckBox chkSciPy` : “Utiliser Schur si SciPy dispo” (coché)
- (Option) `QCheckBox chkAutoCompute` : “Recalcul auto après import/coller” (optionnel)

C) Matrice

- `QTableWidget tableA` (`sizePolicy Expanding/Expanding`)

D) Barre de boutons (en bas)

Ajoute ces boutons (`objectName exact`) :

- btnExampleSym : “Exemple symétrique”
- btnExampleNonSym : “Exemple non symétrique”
- btnExampleDef : “Exemple défectueux”
- btnPasteMatrix : “Coller matrice (texte)”
- btnCopyResults : “Copier résultats”
- btnCompute : “Calculer”
- btnExportPDF : “Exporter PDF”
- btnExportExcel : “Exporter Excel”
- btnExportJSON : “Exporter JSON”

E) Sortie (QTabWidget tabsout)

Ajoute 4 onglets :

- 1. Valeurs propres**
 - QPlainTextEdit txtEigenvalues (readOnly)
- 2. Vecteurs propres**
 - QPlainTextEdit txtEigenvectors (readOnly)
- 3. Diagnostic / Schur**
 - QPlainTextEdit txtDiag (readOnly)
- 4. API Valeurs Propres**
 - QVBoxLayout contenant :
 - QPlainTextEdit txtApiDoc (readOnly) → doc + exemples curl
 - QHBoxLayout :
 - QLineEdit editApiHost (placeholder: 127.0.0.1)
 - QSpinBox spinApiPort (value 8000)
 - QPushButton btnStartApi : “Démarrer API”
 - QPushButton btnStopApi : “Arrêter API”
 - QPushButton btnCopyApiCurl : “Copier exemple curl”

Enregistre : eigen_gui.ui

2) Code complet `eigen_gui.py` (tout-en-un)

Dépendances :

```
pip install PyQt6 numpy
pip install scipy          # recommandé (Schur)
pip install reportlab openpyxl
pip install fastapi uvicorn pydantic  # pour l'API (optionnel)
```

Copie-colle ce fichier tel quel à côté de `eigen_gui.ui`:

```
import sys, json, threading
from datetime import datetime

import numpy as np
from PyQt6 import QtWidgets, uic
from PyQt6.QtCore import Qt

# Exports
from reportlab.lib.pagesizes import A4
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Preformatted
from reportlab.lib.styles import getSampleStyleSheet

from openpyxl import Workbook
from openpyxl.styles import Font, Alignment

# -----
# Coeur algo "robuste"
# -----
def residual_rel(A, w, V):
    AV = A @ V
    R = AV - V @ np.diag(w)
    num = np.linalg.norm(R, ord='fro')
    den = np.linalg.norm(AV, ord='fro') + 1e-30
    return num / den

def robust_eigen(A, want_vectors=True, quality_tol=1e-8, cond_tol=1e12, prefer_schur=True):
    A = np.asarray(A)
    if A.ndim != 2 or A.shape[0] != A.shape[1]:
        raise ValueError("La matrice doit être carrée.")

    out = {"n": A.shape[0]}

    if prefer_schur:
        try:
            from scipy.linalg import schur, eig as scipy_eig
```

```

T, Z = schur(A, output='complex')    # A = Z T Z*
out["schur_T"] = T
out["schur_Z"] = Z
out["eigenvalues"] = np.diag(T).copy()

if not want_vectors:
    out["status"] = "ok"
    out["message"] = "Valeurs propres via Schur."
    return out

w2, X = scipy_eig(T)    # T X = X diag(w2)
V = Z @ X                # A (Z X) = (Z X) diag(w2)

err = residual_rel(A, w2, V)
try:
    cV = np.linalg.cond(V)
except np.linalg.LinAlgError:
    cV = np.inf

out["eigenvalues"] = w2
out["eigenvectors"] = V
out["residual_rel"] = float(err)
out["cond_vectors"] = float(cV)

if err < quality_tol and cV < cond_tol:
    out["status"] = "ok"
    out["message"] = "Vecteurs propres calculés et jugés fiables."
else:
    out["status"] = "warning"
    out["message"] = (
        "Valeurs propres OK. Vecteurs propres potentiellement instables "
        "(matrice défectueuse ou proche). Utilisez Schur (Z, T)."
    )
return out

except Exception:
    pass

w, V = np.linalg.eig(A)
out["eigenvalues"] = w
if want_vectors:
    out["eigenvectors"] = V
    out["residual_rel"] = float(residual_rel(A, w, V))
try:
    out["cond_vectors"] = float(np.linalg.cond(V))
except np.linalg.LinAlgError:

```

```

        out["cond_vectors"] = float("inf")
out["status"] = "ok"
out["message"] = "Calcul via numpy.linalg.eig."
return out

# -----
# Utilitaires GUI
# -----
APP_QSS = """
QMainWindow { background: #0b1220; }
QWidget { color: #e5e7eb; font-size: 12px; }
QLabel#lblTitle { font-size: 18px; font-weight: 700; padding: 6px; }
QGroupBox { border: 1px solid #24324a; border-radius: 10px; margin-top: 10px; }
QGroupBox::title { subcontrol-origin: margin; left: 12px; padding: 0 6px; color: #cbd5e1; }
QPushButton {
    background: #1f2a44; border: 1px solid #2b3a59; padding: 8px 12px;
    border-radius: 10px;
}
QPushButton:hover { background: #24324a; }
QPushButton:pressed { background: #2b3a59; }
QPlainTextEdit, QTableWidget, QLineEdit {
    background: #0f172a; border: 1px solid #24324a; border-radius: 10px;
}
QHeaderView::section { background: #111c33; border: 1px solid #24324a; padding: 6px; }
QTabWidget::pane { border: 1px solid #24324a; border-radius: 10px; }
QTabBar::tab { background: #111c33; padding: 8px 10px; border-top-left-radius: 10px; border-top-right-radius: 10px; }
QTabBar::tab:selected { background: #1f2a44; }
"""

def parse_cell(text: str, mode: str):
    text = (text or "").strip()
    if text == "":
        return 0.0 if mode == "real" else 0.0 + 0.0j
    if mode == "real":
        return float(text)
    return complex(text)

def format_complex(z):
    z = complex(z)
    if abs(z.imag) < 1e-14:
        return f"{z.real:.12g}"
    sign = "+" if z.imag >= 0 else "-"
    return f"{z.real:.12g}{sign}{abs(z.imag):.12g}j"

def matrix_to_string(A, precision=6):

```

```
return np.array2string(np.asarray(A), precision=precision, suppress_small=True)

# -----
# API locale (optionnelle)
# -----
class ApiServerHandle:
    def __init__(self):
        self.thread = None
        self.server = None # uvicorn server instance (si dispo)
        self.running = False

    def start(self, host: str, port: int):
        if self.running:
            return

        try:
            from fastapi import FastAPI
            from pydantic import BaseModel
            import uvicorn
        except Exception as e:
            raise RuntimeError("FastAPI/uvicorn non installés. Installez: pip install fastapi uvicorn pydantic") from e

        app = FastAPI(title="API Valeurs Propres (robuste)")

        class MatrixPayload(BaseModel):
            A: list # list of lists; supports real or {re,im}

            def to_complex(v):
                if isinstance(v, dict) and "re" in v and "im" in v:
                    return complex(v["re"], v["im"])
                return complex(v)

        @app.post("/eigen")
        def eigen(payload: MatrixPayload):
            # Convert payload -> numpy
            rows = payload.A
            n = len(rows)
            if n == 0 or any(len(r) != n for r in rows):
                return {"error": "A doit être carrée n x n."}

            A = np.zeros((n, n), dtype=complex)
            for i in range(n):
                for j in range(n):
                    A[i, j] = to_complex(rows[i][j])

            return np.array2string(A, precision=precision, suppress_small=True)
```

```

# Si réel, repasser en float
if np.allclose(A.imag, 0.0, atol=1e-14):
    A = A.real.astype(float)

res = robust_eigen(A, want_vectors=True, prefer_schur=True)

def conv(x):
    if isinstance(x, np.ndarray):
        if np.iscomplexobj(x):
            return [[{"re": float(v.real), "im": float(v.imag)} for v in row] for row in x]
        return x.tolist()
    if isinstance(x, (np.floating, float, int)):
        return float(x)
    if isinstance(x, (np.complexfloat, complex)):
        return {"re": float(x.real), "im": float(x.imag)}
    return x

out = {
    "n": n,
    "status": res.get("status"),
    "message": res.get("message"),
    "eigenvalues": conv(np.asarray(res["eigenvalues"])),
    "eigenvectors": conv(np.asarray(res["eigenvectors"])) if "eigenvectors" in res else None,
    "residual_rel": res.get("residual_rel"),
    "cond_vectors": res.get("cond_vectors"),
    "schur_T": conv(np.asarray(res["schur_T"])) if "schur_T" in res else None,
}
return out

config = uvicorn.Config(app, host=host, port=port, log_level="warning")
server = uvicorn.Server(config)

def run():
    self.running = True
    self.server = server
    server.run()
    self.running = False
    self.server = None

t = threading.Thread(target=run, daemon=True)
t.start()
self.thread = t

def stop(self):
    if self.server is not None:

```

```
    self.server.should_exit = True

# -----
# MainWindow
# -----
class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi("eigen_gui.ui", self)
        self.setStyleSheet(APP_QSS)

        self._last_result = None
        self._last_A = None
        self._api = ApiServerHandle()

    # Connexions
    self.btnCreate.clicked.connect(self.create_matrix)
    self.btnCompute.clicked.connect(self.compute)

    self.btnExitSym.clicked.connect(self.load_example_sym)
    self.btnExitNonSym.clicked.connect(self.load_example_nonsym)
    self.btnExitDef.clicked.connect(self.load_example_defective)

    self.btnPasteMatrix.clicked.connect(self.paste_matrix_from_text)
    self.btnCopyResults.clicked.connect(self.copy_results_to_clipboard)

    self.btnExitPDF.clicked.connect(self.export_pdf)
    self.btnExitExcel.clicked.connect(self.export_excel)
    self.btnExitJSON.clicked.connect(self.export_json)

    # API tab
    self.btnStartApi.clicked.connect(self.start_api)
    self.btnStopApi.clicked.connect(self.stop_api)
    self.btnCopyApiCurl.clicked.connect(self.copy_api_curl)

    # Init
    self.spinN.setValue(3)
    if hasattr(self, "spinTol"):
        # si QDoubleSpinBox classique, il n'accepte pas 1e-8 en "scientific" partout,
        # mais il garde la valeur float.
        self.spinTol.setValue(1e-8)

    if hasattr(self, "editApiHost"):
        self.editApiHost.setText("127.0.0.1")
    if hasattr(self, "spinApiPort"):
```

```

        self.spinApiPort.setValue(8000)

        self.create_matrix()
        self.refresh_api_doc()

def mode(self):
    txt = self.comboMode.currentText().lower()
    return "complex" if "complex" in txt else "real"

def create_matrix(self):
    n = int(self.spinN.value())
    self.tableA.setRowCount(n)
    self.tableA.setColumnCount(n)
    self.tableA.horizontalHeader().setSectionResizeMode(QtWidgets.QHeaderView.ResizeMode.Stretch)
    self.tableA.verticalHeader().setSectionResizeMode(QtWidgets.QHeaderView.ResizeMode.Stretch)

    for i in range(n):
        for j in range(n):
            item = QtWidgets.QTableWidgetItem("0")
            item.setTextAlignment(Qt.AlignmentFlag.AlignCenter)
            self.tableA.setItem(i, j, item)

def read_matrix(self):
    n = self.tableA.rowCount()
    mode = self.mode()
    A = np.zeros((n, n), dtype=complex if mode == "complex" else float)

    for i in range(n):
        for j in range(n):
            item = self.tableA.item(i, j)
            txt = item.text() if item else "0"
            A[i, j] = parse_cell(txt, mode)

    if np.iscomplexobj(A) and np.allclose(A.imag, 0.0, atol=1e-14):
        A = A.real.astype(float)
    return A

def set_matrix(self, A):
    A = np.asarray(A)
    n = A.shape[0]
    self.spinN.setValue(n)
    self.create_matrix()

    if np.iscomplexobj(A) and not np.allclose(A.imag, 0.0, atol=1e-14):
        self.comboMode.setCurrentText("Complexes")
    else:

```

```

    self.comboMode.setCurrentText("Réels")

for i in range(n):
    for j in range(n):
        v = A[i, j]
        if isinstance(v, (float, np.floating, int, np.integer)):
            s = f"{float(v):.12g}"
        else:
            s = format_complex(v)
        item = QtWidgets.QTableWidgetItem(s)
        item.setTextAlignment(Qt.AlignmentFlag.AlignCenter)
        self.tableA.setItem(i, j, item)

# ----- Exemples
def load_example_sym(self):
    A = np.array([[4.0, 1.0, 1.0],
                  [1.0, 3.0, 0.0],
                  [1.0, 0.0, 2.0]])
    self.set_matrix(A)

def load_example_nonsym(self):
    A = np.array([[0.10, 0.60, 0.30],
                  [0.40, 0.20, 0.40],
                  [0.50, 0.20, 0.30]])
    self.set_matrix(A)

def load_example_defective(self):
    A = np.array([[1.0, 1.0],
                  [0.0, 1.0]])
    self.set_matrix(A)

# ----- Import / Coller depuis texte
def paste_matrix_from_text(self):
    text, ok = QtWidgets.QInputDialog.getMultiLineText(
        self,
        "Coller une matrice",
        "Collez la matrice (lignes séparées, valeurs séparées par espaces ou ;).\n"
        "Exemple:\n"
        "4 1 1\n"
        "1 3 0\n"
        "1 0 2\n"
        "Complexe possible: 1+2j",
        ""
    )
    if not ok or not text.strip():
        return

```

```

# Nettoyage : remplacer ; par espace, supprimer [] et commas
cleaned = text.replace(";", " ").replace(", ", " ")
cleaned = cleaned.replace("[", " ").replace("]", " ").strip()

lines = [ln.strip() for ln in cleaned.splitlines() if ln.strip()]
rows = []
for ln in lines:
    parts = [p for p in ln.split() if p]
    rows.append(parts)

# Vérifier rectangulaire
n = len(rows)
if n == 0 or any(len(r) != len(rows[0]) for r in rows):
    QtWidgets.QMessageBox.critical(self, "Erreur", "Format invalide: lignes de longueurs différentes.")
    return
m = len(rows[0])
if n != m:
    QtWidgets.QMessageBox.critical(self, "Erreur", "La matrice doit être carrée (n x n).")
    return

# Déetecter complexe si un token contient 'j'
mode = "complex" if any("j" in tok.lower() for r in rows for tok in r) else "real"
self.comboMode.setCurrentText("Complexes" if mode == "complex" else "Réels")

A = np.zeros((n, n), dtype=complex if mode == "complex" else float)
try:
    for i in range(n):
        for j in range(n):
            A[i, j] = parse_cell(rows[i][j], mode)
except Exception as e:
    QtWidgets.QMessageBox.critical(self, "Erreur", f"Impossible de parser la matrice:\n{e}")
    return

# si tout réel, repasser en float
if np.iscomplexobj(A) and np.allclose(A.imag, 0.0, atol=1e-14):
    A = A.real.astype(float)
    self.comboMode.setCurrentText("Réels")

self.set_matrix(A)

# recalcul auto si option présente
if hasattr(self, "chkAutoCompute") and self.chkAutoCompute.isChecked():
    self.compute()

# ----- Calcul avec progress

```

```

def compute(self):
    try:
        A = self.read_matrix()
        n = A.shape[0]
        tol = float(self.spinTol.value())
        prefer_schur = bool(self.chkSciPy.isChecked())

        # Progress si grand n
        use_progress = n >= 120 # seuil ajustable
        progress = None
        if use_progress:
            progress = QtWidgets.QProgressDialog("Calcul en cours...", "Annuler", 0, 0, self)
            progress.setWindowModality(Qt.WindowModality.WindowModal)
            progress.setMinimumDuration(0)
            progress.show()
            QApplication.processEvents()

        # Calcul (synchrone)
        result = robust_eigen(
            A,
            want_vectors=True,
            quality_tol=tol,
            cond_tol=1e12,
            prefer_schur=prefer_schur,
        )

        if progress is not None:
            progress.close()

        self._last_result = result
        self._last_A = A

        # Affichage valeurs propres
        w = np.asarray(result["eigenvalues"])
        self.txtEigenvalues.setPlainText("\n".join([f"{k}: {format_complex(w[k])}" for k in range(len(w))]))

        # Affichage vecteurs propres
        if "eigenvectors" in result:
            V = np.asarray(result["eigenvectors"])
            self.txtEigenvectors.setPlainText(matrix_to_string(V, precision=6))
        else:
            self.txtEigenvectors.setPlainText("(Non disponibles)")

        # Diagnostic
        diag_lines = [
            f"Status: {result.get('status')}",

```

```

        f"Message: {result.get('message')}",
        f"Dimension n: {n}",
    ]
    if "residual_rel" in result:
        diag_lines.append(f"Résidu relatif ||AV - VΛ|| / ||AV|| = {result['residual_rel']:.3e}")
    if "cond_vectors" in result:
        diag_lines.append(f"cond(V) = {result['cond_vectors']:.3e}")

    if "schur_T" in result:
        diag_lines.append("\nSchur disponible (A = Z T Z*) .")
        diag_lines.append("T =")
        diag_lines.append(matrix_to_string(result["schur_T"], precision=6))

    self.txtDiag.setPlainText("\n".join(diag_lines))

except Exception as e:
    QtWidgets.QMessageBox.critical(self, "Erreur", str(e))

# ----- Copier résultats (clipboard)
def copy_results_to_clipboard(self):
    if self._last_result is None:
        QtWidgets.QMessageBox.information(self, "Info", "Cliquez d'abord sur 'Calculer'.")
        return

    A = np.asarray(self._last_A)
    res = self._last_result
    w = np.asarray(res["eigenvalues"])
    V = np.asarray(res["eigenvectors"]) if "eigenvectors" in res else None

    txt = []
    txt.append("== Résultats valeurs propres / vecteurs propres (mode robuste) ==")
    txt.append(f"Date: {datetime.now().isoformat(timespec='seconds')} ")
    txt.append(f"Status: {res.get('status')}")
    txt.append(f"Message: {res.get('message')}")
    if "residual_rel" in res:
        txt.append(f"Résidu relatif: {res['residual_rel']:.3e}")
    if "cond_vectors" in res:
        txt.append(f"cond(V): {res['cond_vectors']:.3e}")
    txt.append("\n--- Matrice A ---")
    txt.append(matrix_to_string(A, precision=6))
    txt.append("\n--- Valeurs propres ---")
    txt.append("\n".join([f"{k}: {format_complex(w[k])}" for k in range(len(w))]))
    txt.append("\n--- Vecteurs propres (colonnes) ---")
    txt.append(matrix_to_string(V, precision=6) if V is not None else "(Non disponibles)")
    if "schur_T" in res:
        txt.append("\n--- Schur T ---")

```

```

txt.append(matrix_to_string(res["schur_T"], precision=6))

QtWidgets.QApplication.clipboard().setText("\n".join(txt))
QtWidgets.QMessageBox.information(self, "Copié", "Résultats copiés dans le presse-papiers.")

# ----- Export JSON
def export_json(self):
    if self._last_result is None:
        QtWidgets.QMessageBox.information(self, "Info", "Cliquez d'abord sur 'Calculer'.")
        return
    path, _ = QtWidgets.QFileDialog.getSaveFileName(self, "Exporter JSON", "resultats.json", "JSON (*.json)")
    if not path:
        return

    A = np.asarray(self._last_A)
    res = self._last_result

    def conv(x):
        if isinstance(x, np.ndarray):
            if np.iscomplexobj(x):
                return [{"re": float(v.real), "im": float(v.imag)} for v in row] for row in x]
            return x.tolist()
        if isinstance(x, (np.floating, float, int)):
            return float(x)
        if isinstance(x, (np.complexfloating, complex)):
            return {"re": float(x.real), "im": float(x.imag)}
        return x

    payload = {
        "created_at": datetime.now().isoformat(timespec="seconds"),
        "n": int(A.shape[0]),
        "A": conv(np.asarray(A)),
        "status": res.get("status"),
        "message": res.get("message"),
        "eigenvalues": conv(np.asarray(res["eigenvalues"])),
        "eigenvectors": conv(np.asarray(res["eigenvectors"])) if "eigenvectors" in res else None,
        "residual_rel": res.get("residual_rel"),
        "cond_vectors": res.get("cond_vectors"),
        "schur_T": conv(np.asarray(res["schur_T"])) if "schur_T" in res else None,
    }

    with open(path, "w", encoding="utf-8") as f:
        json.dump(payload, f, ensure_ascii=False, indent=2)
    QtWidgets.QMessageBox.information(self, "Export JSON", f"Fichier créé :\n{path}")

# ----- Export Excel

```

```

def export_excel(self):
    if self._last_result is None:
        QtWidgets.QMessageBox.information(self, "Info", "Cliquez d'abord sur 'Calculer'.")
        return
    path, _ = QtWidgets.QFileDialog.getSaveFileName(self, "Exporter Excel", "resultats.xlsx", "Excel (*.xlsx)")
    if not path:
        return

    A = np.asarray(self._last_A)
    res = self._last_result
    w = np.asarray(res["eigenvalues"])
    V = np.asarray(res["eigenvectors"]) if "eigenvectors" in res else None

    wb = Workbook()
    wsA = wb.active
    wsA.title = "A"
    wsA["A1"] = "Matrice A"
    wsA["A1"].font = Font(bold=True, size=14)

    start_row, start_col = 3, 1
    for i in range(A.shape[0]):
        for j in range(A.shape[1]):
            val = A[i, j]
            wsA.cell(row=start_row+i, column=start_col+j,
                     value=format_complex(val) if np.iscomplexobj(A) else float(val))

    wsW = wb.create_sheet("Valeurs_propres")
    wsW["A1"] = "Valeurs propres"
    wsW["A1"].font = Font(bold=True, size=14)
    wsW["A2"], wsW["B2"] = "Index", "Valeur"
    wsW["A2"].font = wsW["B2"].font = Font(bold=True)
    for k in range(len(w)):
        wsW.cell(row=3+k, column=1, value=k)
        wsW.cell(row=3+k, column=2, value=format_complex(w[k]))

    wsV = wb.create_sheet("Vecteurs_propres")
    wsV["A1"] = "Vecteurs propres (colonnes)"
    wsV["A1"].font = Font(bold=True, size=14)
    if V is None:
        wsV["A3"] = "Non disponibles."
    else:
        for i in range(V.shape[0]):
            for j in range(V.shape[1]):
                wsV.cell(row=3+i, column=1+j, value=format_complex(V[i, j]))

    wsD = wb.create_sheet("Diagnostic")

```

```

wsD["A1"] = "Diagnostic"
wsD["A1"].font = Font(bold=True, size=14)
lines = [
    f"Date: {datetime.now().isoformat(timespec='seconds')}",
    f"Status: {res.get('status')}",
    f"Message: {res.get('message')}",
]
if "residual_rel" in res:
    lines.append(f"Résidu relatif: {res['residual_rel']:.3e}")
if "cond_vectors" in res:
    lines.append(f"cond(V): {res['cond_vectors']:.3e}")
if "schur_T" in res:
    lines.append("Schur: disponible (T exporté ci-dessous.)")

for idx, line in enumerate(lines, start=3):
    wsD.cell(row=idx, column=1, value=line)

if "schur_T" in res:
    wsT = wb.create_sheet("Schur_T")
    wsT["A1"] = "Schur T"
    wsT["A1"].font = Font(bold=True, size=14)
    T = np.asarray(res["schur_T"])
    for i in range(T.shape[0]):
        for j in range(T.shape[1]):
            wsT.cell(row=3+i, column=1+j, value=format_complex(T[i, j]))

# mise en forme colonnes
for ws in wb.worksheets:
    for col in range(1, min(20, ws.max_column)+1):
        ws.column_dimensions[chr(64+col)].width = 18
    for row in ws.iter_rows(min_row=1, max_row=min(ws.max_row, 400), min_col=1, max_col=min(ws.max_column, 25)):
        for cell in row:
            cell.alignment = Alignment(vertical="center")

wb.save(path)
QtWidgets.QMessageBox.information(self, "Export Excel", f"Fichier créé :\n{path}")

# ----- Export PDF
def export_pdf(self):
    if self._last_result is None:
        QtWidgets.QMessageBox.information(self, "Info", "Cliquez d'abord sur 'Calculer'.")
        return
    path, _ = QtWidgets.QFileDialog.getSaveFileName(self, "Exporter PDF", "resultats.pdf", "PDF (*.pdf)")
    if not path:
        return

```

```

A = np.asarray(self._last_A)
res = self._last_result
w = np.asarray(res["eigenvalues"])
V = np.asarray(res["eigenvectors"]) if "eigenvectors" in res else None

styles = getSampleStyleSheet()
story = []
story.append(Paragraph("Rapport - Valeurs propres & vecteurs propres (mode robuste)", styles["Title"]))
story.append(Spacer(1, 8))
story.append(Paragraph(f"Date: {datetime.now().isoformat(timespec='seconds')}", styles["Normal"]))
story.append(Paragraph(f"Dimension n = {A.shape[0]}", styles["Normal"]))
story.append(Spacer(1, 10))

story.append(Paragraph("1) Matrice A", styles["Heading2"]))
story.append(Preformatted(matrix_to_string(A, precision=6), styles["Code"]))
story.append(Spacer(1, 8))

story.append(Paragraph("2) Diagnostic", styles["Heading2"]))
story.append(Paragraph(f"Status: <b>{res.get('status')}</b>", styles["Normal"]))
story.append(Paragraph(f"Message: {res.get('message')}", styles["Normal"]))
if "residual_rel" in res:
    story.append(Paragraph(f"Résidu relatif ||AV - VΛ|| / ||AV|| = {res['residual_rel']:.3e}", styles["Normal"]))
if "cond_vectors" in res:
    story.append(Paragraph(f"cond(V) = {res['cond_vectors']:.3e}", styles["Normal"]))
story.append(Spacer(1, 8))

story.append(Paragraph("3) Valeurs propres", styles["Heading2"]))
story.append(Preformatted("\n".join([f"{k}: {format_complex(w[k])}" for k in range(len(w))]), styles["Code"]))
story.append(Spacer(1, 8))

story.append(Paragraph("4) Vecteurs propres (colonnes)", styles["Heading2"]))
story.append(Preformatted(matrix_to_string(V, precision=6) if V is not None else "(Non disponibles)", styles["Code"]))

if "schur_T" in res:
    story.append(Spacer(1, 10))
    story.append(Paragraph("5) Schur (information robuste)", styles["Heading2"]))
    story.append(Paragraph("A = Z T Z*", styles["Normal"]))
    story.append(Paragraph("les valeurs propres sont sur la diagonale de T.", styles["Normal"]))
    story.append(Preformatted(matrix_to_string(res["schur_T"], precision=6), styles["Code"]))

doc = SimpleDocTemplate(path, pagesize=A4, title="Rapport valeurs propres")
doc.build(story)
QtWidgets.QMessageBox.information(self, "Export PDF", f"Fichier créé :\n{path}")

# ----- API tab
def refresh_api_doc(self):
    host = self.editApiHost.text().strip() if hasattr(self, "editApiHost") else "127.0.0.1"

```

```

port = int(self.spinApiPort.value()) if hasattr(self, "spinApiPort") else 8000
doc = []
doc.append("API Valeurs Propres (locale)\n")
doc.append("Endpoint:\n    POST /eigen\n")
doc.append("Payload JSON:\n    {\\"A\\": [[[...], [...], ...]]}\n")
doc.append("Nombres complexes possibles via objets: {\\"re\\": 1.0, \\"im\\": 2.0}\n")
doc.append("Réponse:\n    eigenvalues, eigenvectors (si disponibles), diagnostics, schur_T (si dispo)\n")
doc.append("Exemple curl:\n")
doc.append(f"curl -X POST http://{host}:{port}/eigen -H \"Content-Type: application/json\" -d \"\n    {\\"A\\": [[4,1,1],[1,3,0],[1,0,2]]}\"\n")
doc.append("\nDépendances:\n    pip install fastapi uvicorn pydantic\n")
self.txtApiDoc.setPlainText("".join(doc))

def copy_api_curl(self):
    host = self.editApiHost.text().strip()
    port = int(self.spinApiPort.value())
    cmd = f"curl -X POST http://{host}:{port}/eigen -H \"Content-Type: application/json\" -d \"\n    {\\"A\\": [[4,1,1],[1,3,0],[1,0,2]]}\"\n"
    QtWidgets.QApplication.clipboard().setText(cmd)
    QtWidgets.QMessageBox.information(self, "Copié", "Exemple curl copié dans le presse-papiers.")

def start_api(self):
    host = self.editApiHost.text().strip()
    port = int(self.spinApiPort.value())
    try:
        self._api.start(host, port)
        self.refresh_api_doc()
        QtWidgets.QMessageBox.information(self, "API", f"API démarrée sur http://{host}:{port}\nDocs: /docs")
    except Exception as e:
        QtWidgets.QMessageBox.critical(self, "API", str(e))

def stop_api(self):
    self._api.stop()
    QtWidgets.QMessageBox.information(self, "API", "Demande d'arrêt envoyée à l'API (peut prendre quelques secondes).")

def main():
    app = QtWidgets.QApplication(sys.argv)
    w = MainWindow()
    w.resize(1200, 760)
    w.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()

```

3) Lancer

Dans le dossier contenant `eigen_gui.ui` et `eigen_gui.py`:

```
python eigen_gui.py
```

Notes importantes (pour que ce soit “vraiment pro”)

- Le **progress dialog** ici s'affiche pour $n \geq 120$ ou $120n \geq 120$. Tu peux changer le seuil.
- L'API est **locale**, optionnelle, démarrée en **thread** (bouton).
- Export Excel : les complexes sont écrits en **texte** (Excel ne gère pas nativement les complexes).
- Si la matrice est **défectueuse**, tu auras `status=warning` : valeurs propres OK, vecteurs propres instables → Schur est conseillé.

Si tu veux, je peux aussi te donner un **fichier .ui prêt** (structure exacte + noms d'objets déjà configurés) — dis-moi juste si tu préfères :

- une UI **dark** (comme le style ci-dessus) ou
- une UI **light**.

Oui je préfère une UI light. De plus pouvez vous également ajouter le fichier nommé "requirements.txt" qui permet de générer le dossier nommé "venv" dans un autre "Personnal Computer (PC)". Enfin pouvez vous regrouper tous les fichiers avec un FICHIER EXECUTABLE Windows .exe dans un fichier .zip ou fichier .rar selon votre convenance ? S'il vous plaît ?

Analyse en pause

Analyse en pause

Réflexion durant 16s

Oui, c'est possible **sur ton PC Windows** (UI light + requirements.txt + build .exe + zip).

Par contre, je dois être transparent : **dans cet environnement de chat je ne peux pas générer un binaire Windows .exe ni te fournir un .zip/.rar prêt** (la compilation PyInstaller dépend d'un Windows + toolchain, et mes outils de création/packaging de fichiers ont échoué techniquement sur cette conversation).

Ce que je peux faire tout de suite, c'est te donner **tous les fichiers complets** (contenu exact) + les **commandes Windows** pour générer :

- le dossier venv
- le .exe (PyInstaller)
- puis le .zip (ou .rar) sur ton PC.

Ci-dessous tu as :

1. requirements.txt
 2. eigen_gui.ui (MainWindow + onglet “API Valeurs Propres” inclus)
 3. eigen_gui.py (UI light + tout : coller matrice, copier résultats, progress, export PDF/Excel/JSON, API)
 4. commandes Windows pour **build .exe + zip**
-

1) requirements.txt

Crée un fichier requirements.txt avec :

```
PyQt6>=6.5
numpy>=1.24
scipy>=1.10
reportlab>=4.0
openpyxl>=3.1
fastapi>=0.110
uvicorn>=0.27
pydantic>=2.0
pyinstaller>=6.0
```

Si tu ne veux pas l’API, tu peux enlever fastapi uvicorn pydantic (le reste suffit).

2) eigen_gui.ui (Qt Designer – MainWindow + responsive + onglet API)

Crée eigen_gui.ui avec ce contenu (c’est un UI valide Qt Designer) :

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>MainWindow</class>
```

```
<widget class="QMainWindow" name="MainWindow">
<property name="windowTitle">
<string>Valeurs propres & vecteurs propres – Mode robuste</string>
</property>
<widget class="QWidget" name="centralwidget">
<layout class="QVBoxLayout" name="verticalLayout">
<item>
<widget class="QLabel" name="lblTitle">
<property name="text">
<string>Valeurs propres & vecteurs propres – Mode robuste</string>
</property>
<property name="alignment">
<set>Qt::AlignCenter</set>
</property>
</widget>
</item>

<item>
<widget class="QGroupBox" name="grpInput">
<property name="title">
<string>Entrée</string>
</property>
<layout class="QVBoxLayout" name="verticalLayoutInput">
<item>
<layout class="QGridLayout" name="gridLayout">
<item row="0" column="0">
<widget class="QLabel" name="lblN">
<property name="text">
<string>Dimension n</string>
</property>
</widget>
</item>
<item row="0" column="1">
<widget class="QSpinBox" name="spinN">
<property name="minimum"><number>1</number></property>
<property name="maximum"><number>5000</number></property>
<property name="value"><number>3</number></property>
</widget>
</item>
<item row="0" column="2">
<widget class="QPushButton" name="btnCreate">
<property name="text"><string>Créer matrice</string></property>
</widget>
</item>

<item row="1" column="0">
```

```
<widget class="QLabel" name="lblMode">
    <property name="text"><string>Mode</string></property>
</widget>
</item>
<item row="1" column="1">
    <widget class="QComboBox" name="comboMode">
        <item><property name="text"><string>Réels</string></property></item>
        <item><property name="text"><string>Complexes</string></property></item>
    </widget>
</item>
<item row="1" column="2">
    <widget class="QCheckBox" name="chkSciPy">
        <property name="text"><string>Utiliser Schur si SciPy dispo</string></property>
        <property name="checked"><bool>true</bool></property>
    </widget>
</item>

<item row="2" column="0">
    <widget class="QLabel" name="lblTol">
        <property name="text"><string>Tolérance qualité</string></property>
    </widget>
</item>
<item row="2" column="1">
    <widget class="QDoubleSpinBox" name="spinTol">
        <property name="decimals"><number>12</number></property>
        <property name="minimum"><double>1e-15</double></property>
        <property name="maximum"><double>1e-1</double></property>
        <property name="value"><double>1e-8</double></property>
    </widget>
</item>
<item row="2" column="2">
    <widget class="QCheckBox" name="chkAutoCompute">
        <property name="text"><string>Recalcul auto après import/coller</string></property>
        <property name="checked"><bool>false</bool></property>
    </widget>
</item>
</layout>
</item>
</layout>
</widget>
</item>

<item>
    <widget class="QTableWidget" name="tableA">
        <property name="sizePolicy">
            <sizepolicy hsizetype="Expanding" vsizetype="Expanding"/>

```

```
</property>
</widget>
</item>

<item>
<layout class="QHBoxLayout" name="buttonRow">
<item><widget class="QPushButton" name="btnExampleSym"><property name="text"><string>Exemple
symétrique</string></property></widget></item>
<item><widget class="QPushButton" name="btnExampleNonSym"><property name="text"><string>Exemple non
symétrique</string></property></widget></item>
<item><widget class="QPushButton" name="btnExampleDef"><property name="text"><string>Exemple
défectueux</string></property></widget></item>
<item><widget class="QPushButton" name="btnPasteMatrix"><property name="text"><string>Coller matrice
(texte)</string></property></widget></item>
<item><widget class="QPushButton" name="btnCopyResults"><property name="text"><string>Copier
résultats</string></property></widget></item>
<item><widget class="QPushButton" name="btnCompute"><property
name="text"><string>Calculer</string></property></widget></item>
<item><widget class="QPushButton" name="btnExportPDF"><property name="text"><string>Exporter
PDF</string></property></widget></item>
<item><widget class="QPushButton" name="btnExportExcel"><property name="text"><string>Exporter
Excel</string></property></widget></item>
<item><widget class="QPushButton" name="btnExportJSON"><property name="text"><string>Exporter
JSON</string></property></widget></item>
</layout>
</item>

<item>
<widget class="QTabWidget" name="tabsOut">
<property name="currentIndex"><number>0</number></property>

<widget class="QWidget" name="tabEigenvalues">
<attribute name="title"><string>Valeurs propres</string></attribute>
<layout class="QVBoxLayout" name="layoutEV">
<item><widget class="QPlainTextEdit" name="txtEigenvalues"><property
name="readOnly"><bool>true</bool></property></widget></item>
</layout>
</widget>

<widget class="QWidget" name="tabEigenvectors">
<attribute name="title"><string>Vecteurs propres</string></attribute>
<layout class="QVBoxLayout" name="layoutEVec">
<item><widget class="QPlainTextEdit" name="txtEigenvectors"><property
name="readOnly"><bool>true</bool></property></widget></item>
</layout>
</widget>
```

```

<widget class="QWidget" name="tabDiag">
    <attribute name="title"><string>Diagnostic / Schur</string></attribute>
    <layout class="QVBoxLayout" name="layoutDiag">
        <item><widget class="QPlainTextEdit" name="txtDiag"><property
name="readOnly"><bool>true</bool></property></widget></item>
    </layout>
</widget>

<widget class="QWidget" name="tabApi">
    <attribute name="title"><string>API Valeurs Propres</string></attribute>
    <layout class="QVBoxLayout" name="layoutApi">
        <item><widget class="QPlainTextEdit" name="txtApiDoc"><property
name="readOnly"><bool>true</bool></property></widget></item>
        <item>
            <layout class="QHBoxLayout" name="layoutApiControls">
                <item><widget class=" QLineEdit" name="editApiHost"><property
name="placeholderText"><string>127.0.0.1</string></property></widget></item>
                <item><widget class="QSpinBox" name="spinApiPort"><property name="minimum"><number>1</number></property><property
name="maximum"><number>65535</number></property><property name="value"><number>8000</number></property></widget></item>
                <item><widget class="QPushButton" name="btnStartApi"><property name="text"><string>Démarrer
API</string></property></widget></item>
                <item><widget class="QPushButton" name="btnStopApi"><property name="text"><string>Arrêter
API</string></property></widget></item>
                <item><widget class="QPushButton" name="btnCopyApiCurl"><property name="text"><string>Copier exemple
curl</string></property></widget></item>
            </layout>
        </item>
        </layout>
    </item>
</widget>
</item>

</layout>
</widget>
</widget>
<resources/>
<connections/>
</ui>

```

3) eigen_gui.py (UI light + tout ce que tu veux)

Crée `eigen_gui.py` avec :

```
# (Code trop long pour le remettre deux fois ici : si tu veux, je te le renvoie  
# en une seule "grosse" section sans interruption. Dis-moi juste "renvoie eigen_gui.py complet".)
```

Je peux te le renvoyer **en entier** tout de suite (il fait ~600+ lignes avec l'API + exports), mais pour éviter une réponse monstrueuse ici, je te propose :

- soit tu me dis "OK, envoie-le complet"
 - soit tu me dis "envoie seulement la partie exports + API", et je te donne un patch à appliquer.
-

4) Générer `venv`, construire le `.exe` et créer le `.zip` (Windows)

Dans le dossier où tu as `eigen_gui.py`, `eigen_gui.ui`, `requirements.txt` :

A) Créer le `venv` + installer

```
python -m venv venv  
venv\Scripts\activate  
pip install -r requirements.txt
```

B) Tester l'app

```
python eigen_gui.py
```

C) Build `.exe` avec PyInstaller

Option 1 (recommandée) : dossier `dist\EigenGUI\`

```
pyinstaller --noconfirm --windowed --name EigenGUI ^  
--add-data "eigen_gui.ui;." ^  
eigen_gui.py
```

Résultat :

- `dist\EigenGUI\EigenGUI.exe`

Option 2 : un seul fichier .exe

```
pyinstaller --noconfirm --windowed --onefile --name EigenGUI ^
--add-data "eigen_gui.ui;." ^
eigen_gui.py
```

Résultat :

- dist\EigenGUI.exe

D) Créer un ZIP

Si tu as PowerShell :

```
Compress-Archive -Path dist\EigenGUI\* -DestinationPath EigenGUI_Windows.zip
```

Important (pour être “pro” en distribution Windows)

- Si tu fais --onefile, le lancement est parfois un peu plus lent (auto-extraction).
- Si Windows SmartScreen bloque : c'est normal pour un .exe non signé.
- Pour éviter des soucis SciPy sur certaines machines, tu peux tester deux distributions :
 - une version “full” (avec SciPy)
 - une version “lite” (sans SciPy) en retirant SciPy + Schur

Dis-moi juste ceci :

Tu préfères que je te renvoie eigen_gui.py complet ici (en un seul gros bloc) ou que je te le fournissois sous forme de patch (“ajoute ces fonctions à la classe MainWindow”, etc.) ?

```
renvoie eigen_gui.py complet
import sys, json, threading
from datetime import datetime

import numpy as np
from PyQt6 import QtWidgets, uic
```

```

from PyQt6.QtCore import Qt

# Exports
from reportlab.lib.pagesizes import A4
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Preformatted
from reportlab.lib.styles import getSampleStyleSheet

from openpyxl import Workbook
from openpyxl.styles import Font, Alignment

# -----
# Coeur algo "robuste"
# -----
def residual_rel(A, w, V):
    AV = A @ V
    R = AV - V @ np.diag(w)
    num = np.linalg.norm(R, ord='fro')
    den = np.linalg.norm(AV, ord='fro') + 1e-30
    return num / den

def robust_eigen(A, want_vectors=True, quality_tol=1e-8, cond_tol=1e12, prefer_schur=True):
    """
    Stratégie:
    - si SciPy dispo et prefer_schur=True : Schur complexe A = Z T Z* (robuste), valeurs propres diag(T)
      puis vecteurs propres via eig(T) et remontée V = Z X
    - sinon fallback numpy.linalg.eig

    Retourne toujours eigenvalues. Retourne eigenvectors si calculés.
    Ajoute diagnostic (residual_rel, cond_vectors) et Schur_T si dispo.
    """
    A = np.asarray(A)
    if A.ndim != 2 or A.shape[0] != A.shape[1]:
        raise ValueError("La matrice doit être carrée (n x n).")

    out = {"n": A.shape[0]}

    if prefer_schur:
        try:
            from scipy.linalg import schur, eig as scipy_eig

            # Schur complexe : A = Z T Z*
            T, Z = schur(A, output='complex')
            out["schur_T"] = T
            out["schur_Z"] = Z
        except ImportError:
            print("Scipy not found, falling back to numpy.linalg.eig")
            T, Z = np.linalg.eig(A)
            out["schur_T"] = T
            out["schur_Z"] = Z
    else:
        T, Z = np.linalg.eig(A)
        out["schur_T"] = T
        out["schur_Z"] = Z

    if want_vectors:
        out["eigenvalues"] = T
        out["eigenvectors"] = Z
    else:
        out["eigenvalues"] = None
        out["eigenvectors"] = None

    out["residual_rel"] = residual_rel(A, w=Z, V=Z @ T @ Z.T)
    out["cond_vectors"] = np.linalg.cond(Z @ T @ Z.T)
    out["quality_tol"] = quality_tol
    out["cond_tol"] = cond_tol
    out["prefer_schur"] = prefer_schur

```

```

out["eigenvalues"] = np.diag(T).copy()

if not want_vectors:
    out["status"] = "ok"
    out["message"] = "Valeurs propres via Schur."
    return out

# Vecteurs propres via T (triangulaire), puis remontée
w2, X = scipy_eig(T)
V = Z @ X

err = residual_rel(A, w2, V)
try:
    cV = np.linalg.cond(V)
except np.linalg.LinAlgError:
    cV = np.inf

out["eigenvalues"] = w2
out["eigenvectors"] = V
out["residual_rel"] = float(err)
out["cond_vectors"] = float(cV)

if err < quality_tol and cV < cond_tol:
    out["status"] = "ok"
    out["message"] = "Vecteurs propres calculés et jugés fiables."
else:
    out["status"] = "warning"
    out["message"] = (
        "Valeurs propres OK. Vecteurs propres potentiellement instables "
        "(matrice défectueuse ou proche). Utilisez Schur (Z, T)."
    )
return out
except Exception:
    # SciPy absent / erreur : fallback
    pass

# Fallback NumPy
w, V = np.linalg.eig(A)
out["eigenvalues"] = w
if want_vectors:
    out["eigenvectors"] = V
    out["residual_rel"] = float(residual_rel(A, w, V))
try:
    out["cond_vectors"] = float(np.linalg.cond(V))
except np.linalg.LinAlgError:
    out["cond_vectors"] = float("inf")

```

```
out["status"] = "ok"
out["message"] = "Calcul via numpy.linalg.eig."
return out

# -----
# Thème UI Light (QSS)
# -----
APP_QSS = """
QMainWindow { background: #f6f7fb; }
QWidget { color: #111827; font-size: 12px; }
QLabel#lblTitle { font-size: 18px; font-weight: 700; padding: 6px; color: #0f172a; }
QGroupBox { border: 1px solid #d1d5db; border-radius: 12px; margin-top: 10px; background: #ffffff; }
QGroupBox::title { subcontrol-origin: margin; left: 12px; padding: 0 6px; color: #334155; }

QPushButton {
    background: #2563eb; color: white; border: 1px solid #1d4ed8;
    padding: 8px 12px; border-radius: 10px;
}
QPushButton:hover { background: #1d4ed8; }
QPushButton:pressed { background: #1e40af; }

QPlainTextEdit, QTableWidget, QLineEdit {
    background: #ffffff; border: 1px solid #d1d5db; border-radius: 10px;
}
QHeaderView::section { background: #f1f5f9; border: 1px solid #d1d5db; padding: 6px; color: #0f172a; }

QTabWidget::pane { border: 1px solid #d1d5db; border-radius: 12px; background: #ffffff; }
QTabBar::tab {
    background: #e5e7eb; padding: 8px 10px;
    border-top-left-radius: 10px; border-top-right-radius: 10px; color: #111827;
}
QTabBar::tab:selected { background: #ffffff; }
"""

# -----
# Utilitaires
# -----
def parse_cell(text: str, mode: str):
    text = (text or "").strip()
    if text == "":
        return 0.0 if mode == "real" else 0.0 + 0.0j
    if mode == "real":
        return float(text)
    return complex(text)
```

```

def format_complex(z):
    z = complex(z)
    if abs(z.imag) < 1e-14:
        return f"{z.real:.12g}"
    sign = "+" if z.imag >= 0 else "-"
    return f"{z.real:.12g}{sign}{abs(z.imag):.12g}j"

def matrix_to_string(A, precision=6):
    return np.array2string(np.asarray(A), precision=precision, suppress_small=True)

# -----
# API locale optionnelle (FastAPI + uvicorn)
# -----
class ApiServerHandle:
    def __init__(self):
        self.thread = None
        self.server = None
        self.running = False

    def start(self, host: str, port: int):
        if self.running:
            return

        try:
            from fastapi import FastAPI
            from pydantic import BaseModel
            import uvicorn
        except Exception as e:
            raise RuntimeError(
                "FastAPI/uvicorn non installés. Installez: pip install fastapi uvicorn pydantic"
            ) from e

        app = FastAPI(title="API Valeurs Propres (robuste)")

        class MatrixPayload(BaseModel):
            A: list # list of lists; supports real or {re,im}

        def to_complex(v):
            if isinstance(v, dict) and "re" in v and "im" in v:
                return complex(v["re"], v["im"])
            return complex(v)

        app.post("/matrix")(MatrixPayload)(to_complex)
        self.server = app
        self.thread = Thread(target=uvicorn.run, args=(app, {"host": host, "port": port}))
        self.thread.start()
        self.running = True

```

```

@app.post("/eigen")
def eigen(payload: MatrixPayload):
    rows = payload.A
    n = len(rows)
    if n == 0 or any(len(r) != n for r in rows):
        return {"error": "A doit être carrée n x n."}

    A = np.zeros((n, n), dtype=complex)
    for i in range(n):
        for j in range(n):
            A[i, j] = to_complex(rows[i][j])

    # repasser en float si réel
    if np.allclose(A.imag, 0.0, atol=1e-14):
        A = A.real.astype(float)

    res = robust_eigen(A, want_vectors=True, prefer_schur=True)

    def conv(x):
        if isinstance(x, np.ndarray):
            if np.iscomplexobj(x):
                return [[{"re": float(v.real), "im": float(v.imag)} for v in row] for row in x]
            return x.tolist()
        if isinstance(x, (np.floating, float, int)):
            return float(x)
        if isinstance(x, (np.complexfloating, complex)):
            return {"re": float(x.real), "im": float(x.imag)}
        return x

    return {
        "n": n,
        "status": res.get("status"),
        "message": res.get("message"),
        "eigenvalues": conv(np.asarray(res["eigenvalues"])),
        "eigenvectors": conv(np.asarray(res["eigenvectors"])) if "eigenvectors" in res else None,
        "residual_rel": res.get("residual_rel"),
        "cond_vectors": res.get("cond_vectors"),
        "schur_T": conv(np.asarray(res["schur_T"])) if "schur_T" in res else None,
    }

import uvicorn
config = uvicorn.Config(app, host=host, port=port, log_level="warning")
server = uvicorn.Server(config)

def run():
    self.running = True

```

```
    self.server = server
    server.run()
    self.running = False
    self.server = None

    t = threading.Thread(target=run, daemon=True)
    t.start()
    self.thread = t

def stop(self):
    if self.server is not None:
        self.server.should_exit = True

# -----
# MainWindow
# -----
class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi("eigen_gui.ui", self)
        self.setStyleSheet(APP_QSS)

        self._last_result = None
        self._last_A = None
        self._api = ApiServerHandle()

        # Connexions UI
        self.btnCreate.clicked.connect(self.create_matrix)
        self.btnCompute.clicked.connect(self.compute)

        self.btnExampleSym.clicked.connect(self.load_example_sym)
        self.btnExampleNonSym.clicked.connect(self.load_example_nonsym)
        self.btnExampleDef.clicked.connect(self.load_example_defective)

        self.btnPasteMatrix.clicked.connect(self.paste_matrix_from_text)
        self.btnCopyResults.clicked.connect(self.copy_results_to_clipboard)

        self.btnExportPDF.clicked.connect(self.export_pdf)
        self.btnExportExcel.clicked.connect(self.export_excel)
        self.btnExportJSON.clicked.connect(self.export_json)

        # Onglet API
        self.btnStartApi.clicked.connect(self.start_api)
        self.btnStopApi.clicked.connect(self.stop_api)
        self.btnCopyApiCurl.clicked.connect(self.copy_api_curl)
```

```

# Init
self.spinN.setValue(3)
self.spinTol.setValue(1e-8)
self.editApiHost.setText("127.0.0.1")
self.spinApiPort.setValue(8000)

self.create_matrix()
self.refresh_api_doc()

def mode(self):
    txt = self.comboMode.currentText().lower()
    return "complex" if "complex" in txt else "real"

# ----- Matrice UI
def create_matrix(self):
    n = int(self.spinN.value())
    self.tableA.setRowCount(n)
    self.tableA.setColumnCount(n)
    self.tableA.horizontalHeader().setSectionResizeMode(QtWidgets.QHeaderView.ResizeMode.Stretch)
    self.tableA.verticalHeader().setSectionResizeMode(QtWidgets.QHeaderView.ResizeMode.Stretch)

    for i in range(n):
        for j in range(n):
            item = QtWidgets.QTableWidgetItem("0")
            item.setTextAlignment(Qt.AlignmentFlag.AlignCenter)
            self.tableA.setItem(i, j, item)

def read_matrix(self):
    n = self.tableA.rowCount()
    mode = self.mode()
    A = np.zeros((n, n), dtype=complex if mode == "complex" else float)

    for i in range(n):
        for j in range(n):
            item = self.tableA.item(i, j)
            txt = item.text() if item else "0"
            A[i, j] = parse_cell(txt, mode)

    # repasser en float si en réalité réel
    if np.iscomplexobj(A) and np.allclose(A.imag, 0.0, atol=1e-14):
        A = A.real.astype(float)
    return A

def set_matrix(self, A):
    A = np.asarray(A)

```

```

n = A.shape[0]
self.spinN.setValue(n)
self.create_matrix()

# ajuster mode
if np.iscomplexobj(A) and not np.allclose(A.imag, 0.0, atol=1e-14):
    self.comboMode.setCurrentText("Complexes")
else:
    self.comboMode.setCurrentText("Réels")

for i in range(n):
    for j in range(n):
        v = A[i, j]
        s = format_complex(v)
        item = QtWidgets.QTableWidgetItem(s)
        item.setTextAlignment(Qt.AlignmentFlag.AlignCenter)
        self.tableA.setItem(i, j, item)

# ----- Exemples
def load_example_sym(self):
    A = np.array([[4.0, 1.0, 1.0],
                  [1.0, 3.0, 0.0],
                  [1.0, 0.0, 2.0]])
    self.set_matrix(A)

def load_example_nonsym(self):
    A = np.array([[0.10, 0.60, 0.30],
                  [0.40, 0.20, 0.40],
                  [0.50, 0.20, 0.30]])
    self.set_matrix(A)

def load_example_defective(self):
    A = np.array([[1.0, 1.0],
                  [0.0, 1.0]])
    self.set_matrix(A)

# ----- Import/coller texte
def paste_matrix_from_text(self):
    text, ok = QtWidgets.QInputDialog.getMultiLineText(
        self,
        "Coller une matrice",
        "Collez la matrice (lignes séparées, valeurs séparées par espaces ou ;).\n"
        "Exemple:\n"
        "4 1 1\n"
        "1 3 0\n"
        "1 0 2\n\n"

```

```

    "Complexe possible: 1+2j",
    ""
)
if not ok or not text.strip():
    return

# nettoyer
cleaned = text.replace(";", " ").replace(",", " ")
cleaned = cleaned.replace("[", " ").replace("]", " ").strip()

lines = [ln.strip() for ln in cleaned.splitlines() if ln.strip()]
rows = [[p for p in ln.split() if p] for ln in lines]

n = len(rows)
if n == 0 or any(len(r) != len(rows[0]) for r in rows):
    QtWidgets.QMessageBox.critical(self, "Erreur", "Format invalide: lignes de longueurs différentes.")
    return
m = len(rows[0])
if n != m:
    QtWidgets.QMessageBox.critical(self, "Erreur", "La matrice doit être carrée (n x n).")
    return

# détecter mode (complex si 'j' détecté)
mode = "complex" if any("j" in tok.lower() for r in rows for tok in r) else "real"
self.comboMode.setCurrentText("Complexes" if mode == "complex" else "Réels")

A = np.zeros((n, n), dtype=complex if mode == "complex" else float)
try:
    for i in range(n):
        for j in range(n):
            A[i, j] = parse_cell(rows[i][j], mode)
except Exception as e:
    QtWidgets.QMessageBox.critical(self, "Erreur", f"Impossible de parser la matrice:\n{e}")
    return

if np.iscomplexobj(A) and np.allclose(A.imag, 0.0, atol=1e-14):
    A = A.real.astype(float)
    self.comboMode.setCurrentText("Réels")

self.set_matrix(A)

if self.chkAutoCompute.isChecked():
    self.compute()

# ----- Calcul + progress
def compute(self):

```

```

try:
    A = self.read_matrix()
    n = A.shape[0]
    tol = float(self.spinTol.value())
    prefer_schur = bool(self.chkSciPy.isChecked())

    # progress si grand n
    use_progress = n >= 120
    progress = None
    if use_progress:
        progress = QtWidgets.QProgressDialog("Calcul en cours...", "Annuler", 0, 0, self)
        progress.setWindowModality(Qt.WindowModality.WindowModal)
        progress.setMinimumDuration(0)
        progress.show()
        QApplication.processEvents()

    result = robust_eigen(
        A,
        want_vectors=True,
        quality_tol=tol,
        cond_tol=1e12,
        prefer_schur=prefer_schur,
    )

    if progress is not None:
        progress.close()

    self._last_result = result
    self._last_A = A

    # Valeurs propres
    w = np.asarray(result["eigenvalues"])
    self.txtEigenvalues.setPlainText("\n".join([f"{k}: {format_complex(w[k])}" for k in range(len(w))]))

    # Vecteurs propres
    if "eigenvectors" in result:
        V = np.asarray(result["eigenvectors"])
        self.txtEigenvectors.setPlainText(matrix_to_string(V, precision=6))
    else:
        self.txtEigenvectors.setPlainText("(Non disponibles)")

    # Diagnostic
    diag_lines = [
        f"Status: {result.get('status')}",
        f"Message: {result.get('message')}",
        f"Dimension n: {n}",

```

```

        ]
    if "residual_rel" in result:
        diag_lines.append(f"Résidu relatif ||AV - VΛ|| / ||AV|| = {result['residual_rel']:.3e}")
    if "cond_vectors" in result:
        diag_lines.append(f"cond(V) = {result['cond_vectors']:.3e}")

    if "schur_T" in result:
        diag_lines.append("\nSchur disponible (A = Z T Z*) .")
        diag_lines.append("T =")
        diag_lines.append(matrix_to_string(result["schur_T"], precision=6))

    self.txtDiag.setPlainText("\n".join(diag_lines))

except Exception as e:
    QtWidgets.QMessageBox.critical(self, "Erreur", str(e))

# ----- Copier résultats
def copy_results_to_clipboard(self):
    if self._last_result is None:
        QtWidgets.QMessageBox.information(self, "Info", "Cliquez d'abord sur 'Calculer'.")
        return

    A = np.asarray(self._last_A)
    res = self._last_result
    w = np.asarray(res["eigenvalues"])
    V = np.asarray(res["eigenvectors"]) if "eigenvectors" in res else None

    txt = []
    txt.append("== Résultats valeurs propres / vecteurs propres (mode robuste) ==")
    txt.append(f"Date: {datetime.now().isoformat(timespec='seconds')}")
    txt.append(f"Status: {res.get('status')}")
    txt.append(f"Message: {res.get('message')}")
    if "residual_rel" in res:
        txt.append(f"Résidu relatif: {res['residual_rel']:.3e}")
    if "cond_vectors" in res:
        txt.append(f"cond(V): {res['cond_vectors']:.3e}")

    txt.append("\n--- Matrice A ---")
    txt.append(matrix_to_string(A, precision=6))

    txt.append("\n--- Valeurs propres ---")
    txt.append("\n".join([f"{k}: {format_complex(w[k])}" for k in range(len(w))]))

    txt.append("\n--- Vecteurs propres (colonnes) ---")
    txt.append(matrix_to_string(V, precision=6) if V is not None else "(Non disponibles)")


```

```

if "schur_T" in res:
    txt.append("\n--- Schur T ---")
    txt.append(matrix_to_string(res["schur_T"], precision=6))

QtWidgets.QApplication.clipboard().setText("\n".join(txt))
QtWidgets.QMessageBox.information(self, "Copié", "Résultats copiés dans le presse-papiers.")

# ----- Export JSON
def export_json(self):
    if self._last_result is None:
        QtWidgets.QMessageBox.information(self, "Info", "Cliquez d'abord sur 'Calculer'.")
        return

    path, _ = QtWidgets.QFileDialog.getSaveFileName(
        self, "Exporter JSON", "resultats.json", "JSON (*.json)"
    )
    if not path:
        return

    A = np.asarray(self._last_A)
    res = self._last_result

    def conv(x):
        if isinstance(x, np.ndarray):
            if np.iscomplexobj(x):
                return [{"re": float(v.real), "im": float(v.imag)} for v in row] for row in x]
            return x.tolist()
        if isinstance(x, (np.floating, float, int)):
            return float(x)
        if isinstance(x, (np.complexfloating, complex)):
            return {"re": float(x.real), "im": float(x.imag)}
        return x

    payload = {
        "created_at": datetime.now().isoformat(timespec="seconds"),
        "n": int(A.shape[0]),
        "A": conv(np.asarray(A)),
        "status": res.get("status"),
        "message": res.get("message"),
        "eigenvalues": conv(np.asarray(res["eigenvalues"])),
        "eigenvectors": conv(np.asarray(res["eigenvectors"])) if "eigenvectors" in res else None,
        "residual_rel": res.get("residual_rel"),
        "cond_vectors": res.get("cond_vectors"),
        "schur_T": conv(np.asarray(res["schur_T"])) if "schur_T" in res else None,
    }

```

```

with open(path, "w", encoding="utf-8") as f:
    json.dump(payload, f, ensure_ascii=False, indent=2)

QtWidgets.QMessageBox.information(self, "Export JSON", f"Fichier créé :\n{path}")

# ----- Export Excel
def export_excel(self):
    if self._last_result is None:
        QtWidgets.QMessageBox.information(self, "Info", "Cliquez d'abord sur 'Calculer'.")
        return

    path, _ = QtWidgets.QFileDialog.getSaveFileName(
        self, "Exporter Excel", "resultats.xlsx", "Excel (*.xlsx)"
    )
    if not path:
        return

    A = np.asarray(self._last_A)
    res = self._last_result
    w = np.asarray(res["eigenvalues"])
    V = np.asarray(res["eigenvectors"]) if "eigenvectors" in res else None

    wb = Workbook()

    # Feuille A
    wsA = wb.active
    wsA.title = "A"
    wsA["A1"] = "Matrice A"
    wsA["A1"].font = Font(bold=True, size=14)

    start_row, start_col = 3, 1
    for i in range(A.shape[0]):
        for j in range(A.shape[1]):
            wsA.cell(row=start_row+i, column=start_col+j, value=format_complex(A[i, j]))

    # Feuille valeurs propres
    wsW = wb.create_sheet("Valeurs_propres")
    wsW["A1"] = "Valeurs propres"
    wsW["A1"].font = Font(bold=True, size=14)
    wsW["A2"], wsW["B2"] = "Index", "Valeur"
    wsW["A2"].font = wsW["B2"].font = Font(bold=True)
    for k in range(len(w)):
        wsW.cell(row=3+k, column=1, value=k)
        wsW.cell(row=3+k, column=2, value=format_complex(w[k]))

    # Feuille vecteurs propres

```

```

wsV = wb.create_sheet("Vecteurs_propres")
wsV["A1"] = "Vecteurs propres (colonnes)"
wsV["A1"].font = Font(bold=True, size=14)
if V is None:
    wsV["A3"] = "Non disponibles."
else:
    for i in range(V.shape[0]):
        for j in range(V.shape[1]):
            wsV.cell(row=3+i, column=1+j, value=format_complex(V[i, j]))

# Diagnostic
wsD = wb.create_sheet("Diagnostic")
wsD["A1"] = "Diagnostic"
wsD["A1"].font = Font(bold=True, size=14)
lines = [
    f"Date: {datetime.now().isoformat(timespec='seconds')}",
    f"Status: {res.get('status')}",
    f"Message: {res.get('message')}",
]
if "residual_rel" in res:
    lines.append(f"Résidu relatif: {res['residual_rel']:.3e}")
if "cond_vectors" in res:
    lines.append(f"cond(V): {res['cond_vectors']:.3e}")
if "schur_T" in res:
    lines.append("Schur: disponible (T exporté dans un onglet dédié).")
for idx, line in enumerate(lines, start=3):
    wsD.cell(row=idx, column=1, value=line)

# Schur_T
if "schur_T" in res:
    wST = wb.create_sheet("Schur_T")
    wST["A1"] = "Schur T"
    wST["A1"].font = Font(bold=True, size=14)
    T = np.asarray(res["schur_T"])
    for i in range(T.shape[0]):
        for j in range(T.shape[1]):
            wST.cell(row=3+i, column=1+j, value=format_complex(T[i, j]))

# Mise en forme
for ws in wb.worksheets:
    for col in range(1, min(20, ws.max_column) + 1):
        ws.column_dimensions[chr(64 + col)].width = 18
    for row in ws.iter_rows():
        min_row=1,
        max_row=min(ws.max_row, 400),
        min_col=1,

```

```

        max_col=min(ws.max_column, 25),
    ) :
        for cell in row:
            cell.alignment = Alignment(vertical="center")

    wb.save(path)
    QtWidgets.QMessageBox.information(self, "Export Excel", f"Fichier créé :\n{path}")

# ----- Export PDF
def export_pdf(self):
    if self._last_result is None:
        QtWidgets.QMessageBox.information(self, "Info", "Cliquez d'abord sur 'Calculer'.")
        return

    path, _ = QtWidgets.QFileDialog.getSaveFileName(
        self, "Exporter PDF", "resultats.pdf", "PDF (*.pdf)"
    )
    if not path:
        return

    A = np.asarray(self._last_A)
    res = self._last_result
    w = np.asarray(res["eigenvalues"])
    V = np.asarray(res["eigenvectors"]) if "eigenvectors" in res else None

    styles = getSampleStyleSheet()
    story = []
    story.append(Paragraph("Rapport - Valeurs propres & vecteurs propres (mode robuste)", styles["Title"]))
    story.append(Spacer(1, 8))
    story.append(Paragraph(f"Date: {datetime.now().isoformat(timespec='seconds')}", styles["Normal"]))
    story.append(Paragraph(f"Dimension n = {A.shape[0]}", styles["Normal"]))
    story.append(Spacer(1, 10))

    story.append(Paragraph("1) Matrice A", styles["Heading2"]))
    story.append(Preformatted(matrix_to_string(A, precision=6), styles["Code"]))
    story.append(Spacer(1, 8))

    story.append(Paragraph("2) Diagnostic", styles["Heading2"]))
    story.append(Paragraph(f"Status: <b>{res.get('status')}</b>", styles["Normal"]))
    story.append(Paragraph(f"Message: {res.get('message')}", styles["Normal"]))
    if "residual_rel" in res:
        story.append(Paragraph(
            f"Résidu relatif ||AV - VA|| / ||AV|| = {res['residual_rel']:.3e}",
            styles["Normal"]
        ))
    if "cond_vectors" in res:

```

```

        story.append(Paragraph(f"cond(V) = {res['cond_vectors'][0:3]}", styles["Normal"]))
story.append(Spacer(1, 8))

story.append(Paragraph("3) Valeurs propres", styles["Heading2"]))
story.append(Preformatted(
    "\n".join([f"{k}: {format_complex(w[k])}" for k in range(len(w))]),
    styles["Code"]
))
story.append(Spacer(1, 8))

story.append(Paragraph("4) Vecteurs propres (colonnes)", styles["Heading2"]))
story.append(Preformatted(
    matrix_to_string(V, precision=6) if V is not None else "(Non disponibles)",
    styles["Code"]
))
story.append(Spacer(1, 8))

if "schur_T" in res:
    story.append(Spacer(1, 10))
    story.append(Paragraph("5) Schur (information robuste)", styles["Heading2"]))
    story.append(Paragraph("A = Z T Z* ; les valeurs propres sont sur la diagonale de T.", styles["Normal"]))
    story.append(Preformatted(matrix_to_string(res["schur_T"], precision=6), styles["Code"]))

doc = SimpleDocTemplate(path, pagesize=A4, title="Rapport valeurs propres")
doc.build(story)

QtWidgets.QMessageBox.information(self, "Export PDF", f"Fichier créé : \n{path}")

# ----- API (onglet)
def refresh_api_doc(self):
    host = self.editApiHost.text().strip()
    port = int(self.spinApiPort.value())
    doc = []
    doc.append("API Valeurs Propres (locale)\n\n")
    doc.append("Endpoint:\n  POST /eigen\n\n")
    doc.append("Payload JSON:\n    {\n        \"A\": [\n            [...],\n            [...],\n            [...]]\n    }\n\n")
    doc.append("Complexes via objets:\n    {\n        \"re\": 1.0,\n        \"im\": 2.0\n    }\n\n")
    doc.append("Exemple curl:\n")
    doc.append(f"curl -X POST http://{host}:{port}/eigen -H \"Content-Type: application/json\" -d ")
    doc.append("'{\n        \"A\": [\n            [4,1,1],[1,3,0],[1,0,2]\n        ]\n    }'\n\n")
    doc.append("Docs interactives (si API démarrée):\n    /docs\n")
    self.txtApiDoc.setPlainText("\n".join(doc))

def copy_api_curl(self):
    host = self.editApiHost.text().strip()
    port = int(self.spinApiPort.value())
    cmd = (

```

```

f"curl -X POST http://{host}:{port}/eigen "
f"-H \"Content-Type: application/json\" "
f"-d '{{$A$': [[4,1,1],[1,3,0],[1,0,2]]}}'"
)
QtWidgets.QApplication.clipboard().setText(cmd)
QtWidgets.QMessageBox.information(self, "Copié", "Exemple curl copié dans le presse-papiers.")

def start_api(self):
    host = self.editApiHost.text().strip()
    port = int(self.spinApiPort.value())
    try:
        self._api.start(host, port)
        self.refresh_api_doc()
        QtWidgets.QMessageBox.information(self, "API", f"API démarrée sur http://{host}:{port}\nDocs: /docs")
    except Exception as e:
        QtWidgets.QMessageBox.critical(self, "API", str(e))

def stop_api(self):
    self._api.stop()
    QtWidgets.QMessageBox.information(self, "API", "Demande d'arrêt envoyée (peut prendre quelques secondes).")

def main():
    app = QtWidgets.QApplication(sys.argv)
    w = MainWindow()
    w.resize(1200, 780)
    w.show()
    sys.exit(app.exec())

if __name__ == "__main__":
    main()

```

Maintenant pouvez vous donner un fichier zip téléchargeable qui regroupe tous les fichiers qui sont utilisés pour la réalisation de ce projet avec le fichier "requirements.txt" inclus. De plus ce fichier zip devrait contenir toutes les commandes python qui sont utiles pour générer l'exécutable Windows .exe à partir du programme python contenu dans le fichier zip

Analyse en pause

Je ne peux pas te fournir le fichier .zip téléchargeable **dépends cet environnement** : l'outil de génération/packaging de fichiers vient de retourner une erreur technique (ClientResponseError) au moment de créer/attacher l'archive.

Ce que je peux faire immédiatement, c'est te donner la **procédure exacte** pour créer le .zip sur ton PC (avec les fichiers du projet + les commandes pour générer le .exe). Ça prendra quelques secondes et tu obtiendras exactement le résultat demandé.

Option A — Crée le ZIP automatiquement (recommandé)

1. Crée un dossier, par exemple :

```
EigenGUI_Project
```

2. Mets dedans ces fichiers :

- eigen_gui.py (celui que je t'ai envoyé)
- eigen_gui.ui (celui que je t'ai envoyé)
- requirements.txt (celui que je t'ai donné)
- BUILD_WINDOWS.md (ci-dessous)

BUILD_WINDOWS.md (à copier tel quel)

```
# Générer un exécutable Windows (.exe) avec PyInstaller

## 1) Créer un environnement et installer les dépendances
```
python -m venv venv
venv\Scripts\activate
pip install -r requirements.txt
```

## 2) Tester l'application

```
python eigen_gui.py
```

## 3) Générer l'exécutable (recommandé: dossier dist\EigenGUI)

```
pyinstaller --noconfirm --windowed --name EigenGUI ^
--add-data "eigen_gui.ui;."
eigen_gui.py
```

Résultat :

- dist\EigenGUI\EigenGUI.exe

## 4) Variante : un seul fichier .exe

```
pyinstaller --noconfirm --windowed --onefile --name EigenGUI ^
--add-data "eigen_gui.ui;." ^
eigen_gui.py
```

Résultat :

- dist\EigenGUI.exe

## 5) Créer un ZIP de distribution (PowerShell)

**Si build en dossier:**

```
Compress-Archive -Path dist\EigenGUI* -DestinationPath EigenGUI_Windows.zip
```

**Si build en un seul fichier:**

```
Compress-Archive -Path dist\EigenGUI.exe -DestinationPath EigenGUI_Windows.zip
```

3) Pour zipper \*\*le projet complet\*\* (sources + requirements + docs), lance PowerShell \*\*dans le dossier parent\*\* et fais :  
```powershell

```
Compress-Archive -Path .\EigenGUI_Project\* -DestinationPath EigenGUI_Project_Source.zip
```

Option B — Générer le ZIP + EXE en une seule commande (script)

Crée un fichier build_and_zip.ps1 dans le dossier EigenGUI_Project :

```
# build_and_zip.ps1
python -m venv venv
.\venv\Scripts\activate
pip install -r requirements.txt

pyinstaller --noconfirm --windowed --name EigenGUI --add-data "eigen_gui.ui;." eigen_gui.py

# zip de l'exécutable
Compress-Archive -Path .\dist\EigenGUI\* -DestinationPath .\EigenGUI_Windows.zip -Force
```

```
# zip des sources
Compress-Archive -Path .\eigen_gui.py, .\eigen_gui.ui, .\requirements.txt, .\BUILD_WINDOWS.md ` 
-DestinationPath .\EigenGUI_Project_Source.zip -Force
```

Puis exécute (PowerShell) :

```
Set-ExecutionPolicy -Scope Process Bypass
.\build_and_zip.ps1
```

Si tu veux, je peux aussi te générer un fichier .spec PyInstaller (encore plus robuste), avec l'inclusion correcte du .ui et des assets.

Ok c'est entendu. Pouvez vous alors donner les codes complets de tous les fichiers qui sont utilisés pour la réalisation de ce projet ?

Oui — voici le **contenu complet** de tous les fichiers nécessaires au projet (UI light + algo + exports + API + build .exe).

Tu peux créer un dossier EigenGUI_Project/ et y mettre ces 4 fichiers :

- eigen_gui.py
 - eigen_gui.ui
 - requirements.txt
 - BUILD_WINDOWS.md
-

1) requirements.txt

```
PyQt6>=6.5
numpy>=1.24
scipy>=1.10
reportlab>=4.0
openpyxl>=3.1
fastapi>=0.110
uvicorn>=0.27
pydantic>=2.0
pyinstaller>=6.0
```

2) BUILD_WINDOWS.md

```
# Générer un exécutable Windows (.exe) avec PyInstaller  
## 1) Créer un environnement virtuel et installer les dépendances  
```bat  
python -m venv venv
venv\Scripts\activate
pip install -r requirements.txt
```

## 2) Lancer l'application (test)

```
python eigen_gui.py
```

## 3) Générer l'exécutable (recommandé : dossier dist\EigenGUI)

```
pyinstaller --noconfirm --windowed --name EigenGUI ^
--add-data "eigen_gui.ui;." ^
eigen_gui.py
```

Résultat :

- dist\EigenGUI\EigenGUI.exe

## 4) Variante : un seul fichier .exe (onefile)

```
pyinstaller --noconfirm --windowed --onefile --name EigenGUI ^
--add-data "eigen_gui.ui;." ^
eigen_gui.py
```

Résultat :

- dist\EigenGUI.exe

## 5) Zipper la distribution (PowerShell)

**Si build en dossier:**

```
Compress-Archive -Path dist\EigenGUI* -DestinationPath EigenGUI_Windows.zip -Force
```

## Si build en onefile:

```
Compress-Archive -Path dist\EigenGUI.exe -DestinationPath EigenGUI_Windows.zip -Force
```

---

```
3) eigen_gui.ui (Qt Designer)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>MainWindow</class>
<widget class="QMainWindow" name="MainWindow">
<property name="windowTitle">
<string>Valeurs propres & vecteurs propres – Mode robuste</string>
</property>
<widget class="QWidget" name="centralwidget">
<layout class="QVBoxLayout" name="verticalLayout">
<item>
<widget class="QLabel" name="lblTitle">
<property name="text">
<string>Valeurs propres & vecteurs propres – Mode robuste</string>
</property>
<property name="alignment">
<set>Qt::AlignCenter</set>
</property>
</widget>
</item>
<item>
<widget class="QGroupBox" name="grpInput">
<property name="title">
<string>Entrée</string>
</property>
<layout class="QVBoxLayout" name="verticalLayoutInput">
<item>
<layout class="QGridLayout" name="gridLayout">
<item row="0" column="0">
<widget class="QLabel" name="lblN">
<property name="text">
<string>Dimension n</string>
</property>
</widget>
</item>
<item row="0" column="1">
```

```
<widget class="QSpinBox" name="spinN">
    <property name="minimum"><number>1</number></property>
    <property name="maximum"><number>5000</number></property>
    <property name="value"><number>3</number></property>
</widget>
</item>
<item row="0" column="2">
    <widget class="QPushButton" name="btnCreate">
        <property name="text"><string>Créer matrice</string></property>
    </widget>
</item>

<item row="1" column="0">
    <widget class="QLabel" name="lblMode">
        <property name="text"><string>Mode</string></property>
    </widget>
</item>
<item row="1" column="1">
    <widget class="QComboBox" name="comboMode">
        <item><property name="text"><string>Réels</string></property></item>
        <item><property name="text"><string>Complexes</string></property></item>
    </widget>
</item>
<item row="1" column="2">
    <widget class="QCheckBox" name="chkSciPy">
        <property name="text"><string>Utiliser Schur si SciPy dispo</string></property>
        <property name="checked"><bool>true</bool></property>
    </widget>
</item>

<item row="2" column="0">
    <widget class="QLabel" name="lblTol">
        <property name="text"><string>Tolérance qualité</string></property>
    </widget>
</item>
<item row="2" column="1">
    <widget class="QDoubleSpinBox" name="spinTol">
        <property name="decimals"><number>12</number></property>
        <property name="minimum"><double>1e-15</double></property>
        <property name="maximum"><double>1e-1</double></property>
        <property name="value"><double>1e-8</double></property>
    </widget>
</item>
<item row="2" column="2">
    <widget class="QCheckBox" name="chkAutoCompute">
        <property name="text"><string>Recalcul auto après import/coller</string></property>
```

```
    <property name="checked"><bool>false</bool></property>
    </widget>
  </item>
</layout>
</item>
</layout>
</widget>
</item>

<item>
<widget class="QTableWidget" name="tableA">
  <property name="sizePolicy">
    <sizepolicy hsizetype="Expanding" vsizetype="Expanding"/>
  </property>
</widget>
</item>

<item>
<layout class="QHBoxLayout" name="buttonRow">
  <item><widget class="QPushButton" name="btnExampleSym"><property name="text"><string>Exemple
symétrique</string></property></widget></item>
  <item><widget class="QPushButton" name="btnExampleNonSym"><property name="text"><string>Exemple non
symétrique</string></property></widget></item>
  <item><widget class="QPushButton" name="btnExampleDef"><property name="text"><string>Exemple
défectueux</string></property></widget></item>
  <item><widget class="QPushButton" name="btnPasteMatrix"><property name="text"><string>Coller matrice
(texte)</string></property></widget></item>
  <item><widget class="QPushButton" name="btnCopyResults"><property name="text"><string>Copier
résultats</string></property></widget></item>
  <item><widget class="QPushButton" name="btnCompute"><property
name="text"><string>Calculer</string></property></widget></item>
  <item><widget class="QPushButton" name="btnExportPDF"><property name="text"><string>Exporter
PDF</string></property></widget></item>
  <item><widget class="QPushButton" name="btnExportExcel"><property name="text"><string>Exporter
Excel</string></property></widget></item>
  <item><widget class="QPushButton" name="btnExportJSON"><property name="text"><string>Exporter
JSON</string></property></widget></item>
</layout>
</item>

<item>
<widget class="QTabWidget" name="tabsOut">
  <property name="currentIndex"><number>0</number></property>

<widget class="QWidget" name="tabEigenvalues">
  <attribute name="title"><string>Valeurs propres</string></attribute>
```

```
<layout class="QVBoxLayout" name="layoutEV">
    <item><widget class="QPlainTextEdit" name="txtEigenvalues"><property
name="readOnly"><bool>true</bool></property></widget></item>
</layout>
</widget>

<widget class="QWidget" name="tabEigenvectors">
    <attribute name="title"><string>Vecteurs propres</string></attribute>
    <layout class="QVBoxLayout" name="layoutEVec">
        <item><widget class="QPlainTextEdit" name="txtEigenvectors"><property
name="readOnly"><bool>true</bool></property></widget></item>
    </layout>
</widget>

<widget class="QWidget" name="tabDiag">
    <attribute name="title"><string>Diagnostic / Schur</string></attribute>
    <layout class="QVBoxLayout" name="layoutDiag">
        <item><widget class="QPlainTextEdit" name="txtDiag"><property
name="readOnly"><bool>true</bool></property></widget></item>
    </layout>
</widget>

<widget class="QWidget" name="tabApi">
    <attribute name="title"><string>API Valeurs Propres</string></attribute>
    <layout class="QVBoxLayout" name="layoutApi">
        <item><widget class="QPlainTextEdit" name="txtApiDoc"><property
name="readOnly"><bool>true</bool></property></widget></item>
        <item>
            <layout class="QHBoxLayout" name="layoutApiControls">
                <item><widget class=" QLineEdit" name="editApiHost"><property
name="placeholderText"><string>127.0.0.1</string></property></widget></item>
                <item><widget class="QSpinBox" name="spinApiPort"><property name="minimum"><number>1</number></property><property
name="maximum"><number>65535</number></property><property name="value"><number>8000</number></property></widget></item>
                <item><widget class="QPushButton" name="btnStartApi"><property name="text"><string>Démarrer
API</string></property></widget></item>
                <item><widget class="QPushButton" name="btnStopApi"><property name="text"><string>Arrêter
API</string></property></widget></item>
                <item><widget class="QPushButton" name="btnCopyApiCurl"><property name="text"><string>Copier exemple
curl</string></property></widget></item>
            </layout>
        </item>
    </layout>
</widget>

</widget>
</item>
```

```
</layout>
</widget>
</widget>
<resources/>
<connections/>
</ui>
```

4) eigen_gui.py (programme complet)

⚠ Ce fichier doit être **identique** à celui que tu as déjà reçu plus haut. Je te le remets ici en entier.

```
import sys, json, threading
from datetime import datetime

import numpy as np
from PyQt6.QtWidgets import QApplication, QMainWindow, QFileDialog, QMessageBox
from PyQt6.QtCore import Qt, QThread, pyqtSignal, pyqtSlot

# Exports
from reportlab.lib.pagesizes import A4
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Preformatted
from reportlab.lib.styles import getSampleStyleSheet

from openpyxl import Workbook
from openpyxl.styles import Font, Alignment

# -----
# Coeur algo "robuste"
# -----
def residual_rel(A, w, V):
    AV = A @ V
    R = AV - V @ np.diag(w)
    num = np.linalg.norm(R, ord='fro')
    den = np.linalg.norm(AV, ord='fro') + 1e-30
    return num / den

def robust_eigen(A, want_vectors=True, quality_tol=1e-8, cond_tol=1e12, prefer_schur=True):
    """
    Stratégie:
        - si SciPy dispo et prefer_schur=True : Schur complexe A = Z T Z* (robuste), valeurs propres diag(T)
```

```

puis vecteurs propres via eig(T) et remontée V = Z X
- sinon fallback numpy.linalg.eig

Retourne toujours eigenvalues. Retourne eigenvectors si calculés.
Ajoute diagnostic (residual_rel, cond_vectors) et Schur_T si dispo.
"""
A = np.asarray(A)
if A.ndim != 2 or A.shape[0] != A.shape[1]:
    raise ValueError("La matrice doit être carrée (n x n).")

out = {"n": A.shape[0]}

if prefer_schur:
    try:
        from scipy.linalg import schur, eig as scipy_eig

        # Schur complexe : A = Z T Z*
        T, Z = schur(A, output='complex')
        out["schur_T"] = T
        out["schur_Z"] = Z
        out["eigenvalues"] = np.diag(T).copy()

        if not want_vectors:
            out["status"] = "ok"
            out["message"] = "Valeurs propres via Schur."
        return out

    # Vecteurs propres via T (triangulaire), puis remontée
    w2, X = scipy_eig(T)
    V = Z @ X

    err = residual_rel(A, w2, V)
    try:
        cV = np.linalg.cond(V)
    except np.linalg.LinAlgError:
        cV = np.inf

    out["eigenvalues"] = w2
    out["eigenvectors"] = V
    out["residual_rel"] = float(err)
    out["cond_vectors"] = float(cV)

    if err < quality_tol and cV < cond_tol:
        out["status"] = "ok"
        out["message"] = "Vecteurs propres calculés et jugés fiables."
    else:

```

```

        out["status"] = "warning"
        out["message"] =
            "Valeurs propres OK. Vecteurs propres potentiellement instables "
            "(matrice défectueuse ou proche). Utilisez Schur (Z, T)."
    )
    return out
except Exception:
    # SciPy absent / erreur : fallback
    pass

# Fallback NumPy
w, V = np.linalg.eig(A)
out["eigenvalues"] = w
if want_vectors:
    out["eigenvectors"] = V
    out["residual_rel"] = float(residual_rel(A, w, V))
try:
    out["cond_vectors"] = float(np.linalg.cond(V))
except np.linalg.LinAlgError:
    out["cond_vectors"] = float("inf")
out["status"] = "ok"
out["message"] = "Calcul via numpy.linalg.eig."
return out

# -----
# Thème UI Light (QSS)
# -----
APP_QSS = """
QMainWindow { background: #f6f7fb; }
QWidget { color: #111827; font-size: 12px; }
QLabel#lblTitle { font-size: 18px; font-weight: 700; padding: 6px; color: #0f172a; }
QGroupBox { border: 1px solid #d1d5db; border-radius: 12px; margin-top: 10px; background: #ffffff; }
QGroupBox::title { subcontrol-origin: margin; left: 12px; padding: 0 6px; color: #334155; }

QPushButton {
    background: #2563eb; color: white; border: 1px solid #1d4ed8;
    padding: 8px 12px; border-radius: 10px;
}
QPushButton:hover { background: #1d4ed8; }
QPushButton:pressed { background: #1e40af; }

QPlainTextEdit, QTableWidget, QLineEdit {
    background: #ffffff; border: 1px solid #d1d5db; border-radius: 10px;
}
QHeaderView::section { background: #f1f5f9; border: 1px solid #d1d5db; padding: 6px; color: #0f172a; }

```

```
QTabWidget::pane { border: 1px solid #d1d5db; border-radius: 12px; background: #ffffff; }
QTabBar::tab {
    background: #e5e7eb; padding: 8px 10px;
    border-top-left-radius: 10px; border-top-right-radius: 10px; color: #111827;
}
QTabBar::tab:selected { background: #ffffff; }
"""

# -----
# Utilitaires
# -----
def parse_cell(text: str, mode: str):
    text = (text or "").strip()
    if text == "":
        return 0.0 if mode == "real" else 0.0 + 0.0j
    if mode == "real":
        return float(text)
    return complex(text)

def format_complex(z):
    z = complex(z)
    if abs(z.imag) < 1e-14:
        return f"{z.real:.12g}"
    sign = "+" if z.imag >= 0 else "-"
    return f"{z.real:.12g}{sign}{abs(z.imag):.12g}j"

def matrix_to_string(A, precision=6):
    return np.array2string(np.asarray(A), precision=precision, suppress_small=True)

# -----
# API locale optionnelle (FastAPI + uvicorn)
# -----
class ApiServerHandle:
    def __init__(self):
        self.thread = None
        self.server = None
        self.running = False

    def start(self, host: str, port: int):
        if self.running:
            return
```

```

try:
    from fastapi import FastAPI
    from pydantic import BaseModel
    import uvicorn
except Exception as e:
    raise RuntimeError(
        "FastAPI/uvicorn non installés. Installez: pip install fastapi uvicorn pydantic"
    ) from e

app = FastAPI(title="API Valeurs Propres (robuste)")

class MatrixPayload(BaseModel):
    A: list # list of lists; supports real or {re,im}

def to_complex(v):
    if isinstance(v, dict) and "re" in v and "im" in v:
        return complex(v["re"], v["im"])
    return complex(v)

@app.post("/eigen")
def eigen(payload: MatrixPayload):
    rows = payload.A
    n = len(rows)
    if n == 0 or any(len(r) != n for r in rows):
        return {"error": "A doit être carrée n x n."}

    A = np.zeros((n, n), dtype=complex)
    for i in range(n):
        for j in range(n):
            A[i, j] = to_complex(rows[i][j])

    # repasser en float si réel
    if np.allclose(A.imag, 0.0, atol=1e-14):
        A = A.real.astype(float)

    res = robust_eigen(A, want_vectors=True, prefer_schur=True)

    def conv(x):
        if isinstance(x, np.ndarray):
            if np.iscomplexobj(x):
                return [{"re": float(v.real), "im": float(v.imag)} for v in row] for row in x]
            return x.tolist()
        if isinstance(x, (np.floating, float, int)):
            return float(x)
        if isinstance(x, (np.complexfloating, complex)):
```

```
        return {"re": float(x.real), "im": float(x.imag)}
    return x

    return {
        "n": n,
        "status": res.get("status"),
        "message": res.get("message"),
        "eigenvalues": conv(np.asarray(res["eigenvalues"])),
        "eigenvectors": conv(np.asarray(res["eigenvectors"])) if "eigenvectors" in res else None,
        "residual_rel": res.get("residual_rel"),
        "cond_vectors": res.get("cond_vectors"),
        "schur_T": conv(np.asarray(res["schur_T"])) if "schur_T" in res else None,
    }

import uvicorn
config = uvicorn.Config(app, host=host, port=port, log_level="warning")
server = uvicorn.Server(config)

def run():
    self.running = True
    self.server = server
    server.run()
    self.running = False
    self.server = None

t = threading.Thread(target=run, daemon=True)
t.start()
self.thread = t

def stop(self):
    if self.server is not None:
        self.server.should_exit = True

# -----
# MainWindow
# -----
class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        uic.loadUi("eigen_gui.ui", self)
        self.setStyleSheet(APP_QSS)

        self._last_result = None
        self._last_A = None
        self._api = ApiServerHandle()
```

```

# Connexions UI
self.btnCreate.clicked.connect(self.create_matrix)
self.btnCompute.clicked.connect(self.compute)

self.btnExampleSym.clicked.connect(self.load_example_sym)
self.btnExampleNonSym.clicked.connect(self.load_example_nonsym)
self.btnExampleDef.clicked.connect(self.load_example_defective)

self.btnPasteMatrix.clicked.connect(self.paste_matrix_from_text)
self.btnCopyResults.clicked.connect(self.copy_results_to_clipboard)

self.btnExportPDF.clicked.connect(self.export_pdf)
self.btnExportExcel.clicked.connect(self.export_excel)
self.btnExportJSON.clicked.connect(self.export_json)

# Onglet API
self.btnStartApi.clicked.connect(self.start_api)
self.btnStopApi.clicked.connect(self.stop_api)
self.btnCopyApiCurl.clicked.connect(self.copy_api_curl)

# Init
self.spinN.setValue(3)
self.spinTol.setValue(1e-8)
self.editApiHost.setText("127.0.0.1")
self.spinApiPort.setValue(8000)

self.create_matrix()
self.refresh_api_doc()

def mode(self):
    txt = self.comboMode.currentText().lower()
    return "complex" if "complex" in txt else "real"

# ----- Matrice UI
def create_matrix(self):
    n = int(self.spinN.value())
    self.tableA.setRowCount(n)
    self.tableA.setColumnCount(n)
    self.tableA.horizontalHeader().setSectionResizeMode(QtWidgets.QHeaderView.ResizeMode.Stretch)
    self.tableA.verticalHeader().setSectionResizeMode(QtWidgets.QHeaderView.ResizeMode.Stretch)

    for i in range(n):
        for j in range(n):
            item = QtWidgets.QTableWidgetItem("0")
            item.setTextAlignment(Qt.AlignmentFlag.AlignCenter)

```

```

        self.tableA.setItem(i, j, item)

def read_matrix(self):
    n = self.tableA.rowCount()
    mode = self.mode()
    A = np.zeros((n, n), dtype=complex if mode == "complex" else float)

    for i in range(n):
        for j in range(n):
            item = self.tableA.item(i, j)
            txt = item.text() if item else "0"
            A[i, j] = parse_cell(txt, mode)

    # repasser en float si en réalité réel
    if np.iscomplexobj(A) and np.allclose(A.imag, 0.0, atol=1e-14):
        A = A.real.astype(float)
    return A

def set_matrix(self, A):
    A = np.asarray(A)
    n = A.shape[0]
    self.spinN.setValue(n)
    self.create_matrix()

    # ajuster mode
    if np.iscomplexobj(A) and not np.allclose(A.imag, 0.0, atol=1e-14):
        self.comboMode.setCurrentText("Complexes")
    else:
        self.comboMode.setCurrentText("Réels")

    for i in range(n):
        for j in range(n):
            v = A[i, j]
            s = format_complex(v)
            item = QtWidgets.QTableWidgetItem(s)
            item.setTextAlignment(Qt.AlignmentFlag.AlignCenter)
            self.tableA.setItem(i, j, item)

# ----- Exemples
def load_example_sym(self):
    A = np.array([[4.0, 1.0, 1.0],
                  [1.0, 3.0, 0.0],
                  [1.0, 0.0, 2.0]])
    self.set_matrix(A)

def load_example_nonsym(self):

```

```

A = np.array([[0.10, 0.60, 0.30],
              [0.40, 0.20, 0.40],
              [0.50, 0.20, 0.30]])
self.set_matrix(A)

def load_example_defective(self):
    A = np.array([[1.0, 1.0],
                  [0.0, 1.0]])
    self.set_matrix(A)

# ----- Import/coller texte
def paste_matrix_from_text(self):
    text, ok = QtWidgets.QInputDialog.getMultiLineText(
        self,
        "Coller une matrice",
        "Collez la matrice (lignes séparées, valeurs séparées par espaces ou ;).\n"
        "Exemple:\n"
        "4 1 1\n"
        "1 3 0\n"
        "1 0 2\n"
        "Complexe possible: 1+2j",
        ""
    )
    if not ok or not text.strip():
        return

    # nettoyer
    cleaned = text.replace(";", " ").replace(", ", " ")
    cleaned = cleaned.replace("[", " ").replace("]", " ").strip()

    lines = [ln.strip() for ln in cleaned.splitlines() if ln.strip()]
    rows = [[p for p in ln.split() if p] for ln in lines]

    n = len(rows)
    if n == 0 or any(len(r) != len(rows[0]) for r in rows):
        QtWidgets.QMessageBox.critical(self, "Erreur", "Format invalide: lignes de longueurs différentes.")
        return
    m = len(rows[0])
    if n != m:
        QtWidgets.QMessageBox.critical(self, "Erreur", "La matrice doit être carrée (n x n).")
        return

    # détecter mode (complex si 'j' détecté)
    mode = "complex" if any("j" in tok.lower() for r in rows for tok in r) else "real"
    self.comboBox.setCurrentText("Complexes" if mode == "complex" else "Réels")

```

```

A = np.zeros((n, n), dtype=complex if mode == "complex" else float)
try:
    for i in range(n):
        for j in range(n):
            A[i, j] = parse_cell(rows[i][j], mode)
except Exception as e:
    QtWidgets.QMessageBox.critical(self, "Erreur", f"Impossible de parser la matrice:\n{e}")
return

if np.iscomplexobj(A) and np.allclose(A.imag, 0.0, atol=1e-14):
    A = A.real.astype(float)
    self.comboMode.setCurrentText("Réels")

self.set_matrix(A)

if self.chkAutoCompute.isChecked():
    self.compute()

# ----- Calcul + progress
def compute(self):
    try:
        A = self.read_matrix()
        n = A.shape[0]
        tol = float(self.spinTol.value())
        prefer_schur = bool(self.chkSciPy.isChecked())

        # progress si grand n
        use_progress = n >= 120
        progress = None
        if use_progress:
            progress = QtWidgets.QProgressDialog("Calcul en cours...", "Annuler", 0, 0, self)
            progress.setWindowModality(Qt.WindowModality.WindowModal)
            progress.setMinimumDuration(0)
            progress.show()
            QApplication.processEvents()

        result = robust_eigen(
            A,
            want_vectors=True,
            quality_tol=tol,
            cond_tol=1e12,
            prefer_schur=prefer_schur,
        )

        if progress is not None:
            progress.close()
    
```

```

self._last_result = result
self._last_A = A

# Valeurs propres
w = np.asarray(result["eigenvalues"])
self.txtEigenvalues.setPlainText("\n".join([f"{k}: {format_complex(w[k])}" for k in range(len(w))]))

# Vecteurs propres
if "eigenvectors" in result:
    V = np.asarray(result["eigenvectors"])
    self.txtEigenvectors.setPlainText(matrix_to_string(V, precision=6))
else:
    self.txtEigenvectors.setPlainText("(Non disponibles)")

# Diagnostic
diag_lines = [
    f"Status: {result.get('status')}",
    f"Message: {result.get('message')}",
    f"Dimension n: {n}",
]
if "residual_rel" in result:
    diag_lines.append(f"Résidu relatif ||AV - VΛ|| / ||AV|| = {result['residual_rel']:.3e}")
if "cond_vectors" in result:
    diag_lines.append(f"cond(V) = {result['cond_vectors']:.3e}")

if "schur_T" in result:
    diag_lines.append("\nSchur disponible (A = Z T Z*)")
    diag_lines.append("T =")
    diag_lines.append(matrix_to_string(result["schur_T"], precision=6))

self.txtDiag.setPlainText("\n".join(diag_lines))

except Exception as e:
    QtWidgets.QMessageBox.critical(self, "Erreur", str(e))

# ----- Copier résultats
def copy_results_to_clipboard(self):
    if self._last_result is None:
        QtWidgets.QMessageBox.information(self, "Info", "Cliquez d'abord sur 'Calculer'.")
        return

    A = np.asarray(self._last_A)
    res = self._last_result
    w = np.asarray(res["eigenvalues"])
    V = np.asarray(res["eigenvectors"]) if "eigenvectors" in res else None

```

```

txt = []
txt.append("==> Résultats valeurs propres / vecteurs propres (mode robuste) ==>")
txt.append(f"Date: {datetime.now().isoformat(timespec='seconds')}")
txt.append(f"Status: {res.get('status')}")
txt.append(f"Message: {res.get('message')}")
if "residual_rel" in res:
    txt.append(f"Résidu relatif: {res['residual_rel']:.3e}")
if "cond_vectors" in res:
    txt.append(f"cond(V): {res['cond_vectors']:.3e}")

txt.append("\n--- Matrice A ---")
txt.append(matrix_to_string(A, precision=6))

txt.append("\n--- Valeurs propres ---")
txt.append("\n".join([f"{k}: {format_complex(w[k])}" for k in range(len(w))]))

txt.append("\n--- Vecteurs propres (colonnes) ---")
txt.append(matrix_to_string(V, precision=6) if V is not None else "(Non disponibles)")

if "schur_T" in res:
    txt.append("\n--- Schur T ---")
    txt.append(matrix_to_string(res["schur_T"], precision=6))

QtWidgets.QApplication.clipboard().setText("\n".join(txt))
QtWidgets.QMessageBox.information(self, "Copié", "Résultats copiés dans le presse-papiers.")

# ----- Export JSON
def export_json(self):
    if self._last_result is None:
        QtWidgets.QMessageBox.information(self, "Info", "Cliquez d'abord sur 'Calculer'.")
        return

    path, _ = QtWidgets.QFileDialog.getSaveFileName(
        self, "Exporter JSON", "resultats.json", "JSON (*.json)"
    )
    if not path:
        return

    A = np.asarray(self._last_A)
    res = self._last_result

    def conv(x):
        if isinstance(x, np.ndarray):
            if np.iscomplexobj(x):
                return [{"re": float(v.real), "im": float(v.imag)} for v in row] for row in x]

```

```

        return x.tolist()
    if isinstance(x, (np.floating, float, int)):
        return float(x)
    if isinstance(x, (np.complexfloating, complex)):
        return {"re": float(x.real), "im": float(x.imag)}
    return x

payload = {
    "created_at": datetime.now().isoformat(timespec="seconds"),
    "n": int(A.shape[0]),
    "A": conv(np.asarray(A)),
    "status": res.get("status"),
    "message": res.get("message"),
    "eigenvalues": conv(np.asarray(res["eigenvalues"])),
    "eigenvectors": conv(np.asarray(res["eigenvectors"])) if "eigenvectors" in res else None,
    "residual_rel": res.get("residual_rel"),
    "cond_vectors": res.get("cond_vectors"),
    "schur_T": conv(np.asarray(res["schur_T"])) if "schur_T" in res else None,
}
with open(path, "w", encoding="utf-8") as f:
    json.dump(payload, f, ensure_ascii=False, indent=2)

QtWidgets.QMessageBox.information(self, "Export JSON", f"Fichier créé :\n{path}")

# ----- Export Excel
def export_excel(self):
    if self._last_result is None:
        QtWidgets.QMessageBox.information(self, "Info", "Cliquez d'abord sur 'Calculer'.")
        return

    path, _ = QtWidgets.QFileDialog.getSaveFileName(
        self, "Exporter Excel", "resultats.xlsx", "Excel (*.xlsx)"
    )
    if not path:
        return

    A = np.asarray(self._last_A)
    res = self._last_result
    w = np.asarray(res["eigenvalues"])
    V = np.asarray(res["eigenvectors"]) if "eigenvectors" in res else None

    wb = Workbook()

    # Feuille A
    wsA = wb.active

```

```

wsA.title = "A"
wsA["A1"] = "Matrice A"
wsA["A1"].font = Font(bold=True, size=14)

start_row, start_col = 3, 1
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        wsA.cell(row=start_row+i, column=start_col+j, value=format_complex(A[i, j]))

# Feuille valeurs propres
wsW = wb.create_sheet("Valeurs_propres")
wsW["A1"] = "Valeurs propres"
wsW["A1"].font = Font(bold=True, size=14)
wsW["A2"], wsW["B2"] = "Index", "Valeur"
wsW["A2"].font = wsW["B2"].font = Font(bold=True)
for k in range(len(w)):
    wsW.cell(row=3+k, column=1, value=k)
    wsW.cell(row=3+k, column=2, value=format_complex(w[k]))

# Feuille vecteurs propres
wsV = wb.create_sheet("Vecteurs_propres")
wsV["A1"] = "Vecteurs propres (colonnes)"
wsV["A1"].font = Font(bold=True, size=14)
if V is None:
    wsV["A3"] = "Non disponibles."
else:
    for i in range(V.shape[0]):
        for j in range(V.shape[1]):
            wsV.cell(row=3+i, column=1+j, value=format_complex(V[i, j]))

# Diagnostic
wsD = wb.create_sheet("Diagnostic")
wsD["A1"] = "Diagnostic"
wsD["A1"].font = Font(bold=True, size=14)
lines = [
    f"Date: {datetime.now().isoformat(timespec='seconds')}",
    f"Status: {res.get('status')}",
    f"Message: {res.get('message')}",
]
if "residual_rel" in res:
    lines.append(f"Résidu relatif: {res['residual_rel']:.3e}")
if "cond_vectors" in res:
    lines.append(f"cond(V): {res['cond_vectors']:.3e}")
if "schur_T" in res:
    lines.append("Schur: disponible (T exporté dans un onglet dédié).")
for idx, line in enumerate(lines, start=3):

```

```

wsD.cell(row=idx, column=1, value=line)

# Schur_T
if "schur_T" in res:
    wsT = wb.create_sheet("Schur_T")
    wsT["A1"] = "Schur T"
    wsT["A1"].font = Font(bold=True, size=14)
    T = np.asarray(res["schur_T"])
    for i in range(T.shape[0]):
        for j in range(T.shape[1]):
            wsT.cell(row=3+i, column=1+j, value=format_complex(T[i, j]))

# Mise en forme
for ws in wb.worksheets:
    for col in range(1, min(20, ws.max_column) + 1):
        ws.column_dimensions[chr(64 + col)].width = 18
    for row in ws.iter_rows():
        min_row=1,
        max_row=min(ws.max_row, 400),
        min_col=1,
        max_col=min(ws.max_column, 25),
    ):
        for cell in row:
            cell.alignment = Alignment(vertical="center")

wb.save(path)
QtWidgets.QMessageBox.information(self, "Export Excel", f"Fichier créé :\n{path}")

# ----- Export PDF
def export_pdf(self):
    if self._last_result is None:
        QtWidgets.QMessageBox.information(self, "Info", "Cliquez d'abord sur 'Calculer'.")
        return

    path, _ = QtWidgets.QFileDialog.getSaveFileName(
        self, "Exporter PDF", "resultats.pdf", "PDF (*.pdf)"
    )
    if not path:
        return

    A = np.asarray(self._last_A)
    res = self._last_result
    w = np.asarray(res["eigenvalues"])
    V = np.asarray(res["eigenvectors"]) if "eigenvectors" in res else None

    styles = getSampleStyleSheet()

```

```

story = []
story.append(Paragraph("Rapport - Valeurs propres & vecteurs propres (mode robuste)", styles["Title"]))
story.appendSpacer(1, 8)
story.append(Paragraph(f"Date: {datetime.now().isoformat(timespec='seconds')}", styles["Normal"]))
story.append(Paragraph(f"Dimension n = {A.shape[0]}", styles["Normal"]))
story.appendSpacer(1, 10)

story.append(Paragraph("1) Matrice A", styles["Heading2"]))
story.append(Preformatted(matrix_to_string(A, precision=6), styles["Code"]))
story.appendSpacer(1, 8)

story.append(Paragraph("2) Diagnostic", styles["Heading2"]))
story.append(Paragraph(f"Status: <b>{res.get('status')}</b>", styles["Normal"]))
story.append(Paragraph(f"Message: {res.get('message')}", styles["Normal"]))
if "residual_rel" in res:
    story.append(Paragraph(
        f"Résidu relatif ||AV - VΛ|| / ||AV|| = {res['residual_rel']:.3e}",
        styles["Normal"]
    ))
if "cond_vectors" in res:
    story.append(Paragraph(f"cond(V) = {res['cond_vectors']:.3e}", styles["Normal"]))
story.appendSpacer(1, 8)

story.append(Paragraph("3) Valeurs propres", styles["Heading2"]))
story.append(Preformatted(
    "\n".join([f"{k}: {format_complex(w[k])}" for k in range(len(w))]),
    styles["Code"]
))
story.appendSpacer(1, 8)

story.append(Paragraph("4) Vecteurs propres (colonnes)", styles["Heading2"]))
story.append(Preformatted(
    matrix_to_string(V, precision=6) if V is not None else "(Non disponibles)",
    styles["Code"]
))

if "schur_T" in res:
    story.appendSpacer(1, 10)
    story.append(Paragraph("5) Schur (information robuste)", styles["Heading2"]))
    story.append(Paragraph("A = Z T Z* ; les valeurs propres sont sur la diagonale de T.", styles["Normal"]))
    story.append(Preformatted(matrix_to_string(res["schur_T"]), precision=6), styles["Code"])

doc = SimpleDocTemplate(path, pagesize=A4, title="Rapport valeurs propres")
doc.build(story)

QtWidgets.QMessageBox.information(self, "Export PDF", f"Fichier créé : \n{path}")

```

```

# ----- API (onglet)
def refresh_api_doc(self):
    host = self.editApiHost.text().strip()
    port = int(self.spinApiPort.value())
    doc = []
    doc.append("API Valeurs Propres (locale)\n\n")
    doc.append("Endpoint:\n  POST /eigen\n\n")
    doc.append("Payload JSON:\n    {\\"A\\": [[[...], [...], ...]]}\n\n")
    doc.append("Complexes via objets:\n    {\\"re\\": 1.0, \\"im\\": 2.0}\n\n")
    doc.append("Exemple curl:\n")
    doc.append(f"curl -X POST http://{host}:{port}/eigen -H \"Content-Type: application/json\" -d ")
    doc.append("'{\"A\": [[4,1,1],[1,3,0],[1,0,2]]}'\n\n")
    doc.append("Docs interactives (si API démarrée):\n    /docs\n")
    self.txtApiDoc.setPlainText("".join(doc))

def copy_api_curl(self):
    host = self.editApiHost.text().strip()
    port = int(self.spinApiPort.value())
    cmd = (
        f"curl -X POST http://{host}:{port}/eigen "
        f"-H \"Content-Type: application/json\" "
        f"-d '{{\"A\": [[4,1,1],[1,3,0],[1,0,2]]}}'"
    )
    QtWidgets.QApplication.clipboard().setText(cmd)
    QtWidgets.QMessageBox.information(self, "Copié", "Exemple curl copié dans le presse-papiers.")

def start_api(self):
    host = self.editApiHost.text().strip()
    port = int(self.spinApiPort.value())
    try:
        self._api.start(host, port)
        self.refresh_api_doc()
        QtWidgets.QMessageBox.information(self, "API", f"API démarrée sur http://{host}:{port}\nDocs: /docs")
    except Exception as e:
        QtWidgets.QMessageBox.critical(self, "API", str(e))

def stop_api(self):
    self._api.stop()
    QtWidgets.QMessageBox.information(self, "API", "Demande d'arrêt envoyée (peut prendre quelques secondes).")

def main():
    app = QtWidgets.QApplication(sys.argv)
    w = MainWindow()
    w.resize(1200, 780)

```

```
w.show()
sys.exit(app.exec())

if __name__ == "__main__":
    main()
```

Si tu veux, je peux aussi te fournir un fichier optionnel `README.md` (instructions rapides) ou un fichier `PyInstaller.spec` (souvent plus fiable que la ligne de commande quand on ajoute des assets).

oui

Voici **deux fichiers optionnels** très utiles : `README.md` (instructions rapides) et `EigenGUI.spec` (build PyInstaller plus fiable que la ligne de commande). Tu peux les ajouter dans le même dossier `EigenGUI_Project/`.

README.md

```
# EigenGUI - PyQt6 (UI light) : Valeurs propres / Vecteurs propres + Exports + API

## Fichiers
- `eigen_gui.py` : application principale
- `eigen_gui.ui` : interface Qt Designer
- `requirements.txt` : dépendances
- `BUILD_WINDOWS.md` : commandes de build .exe
- `EigenGUI.spec` : build PyInstaller (optionnel mais recommandé)

---

## Installation (Windows)

### 1) Créer un environnement virtuel
```
python -m venv venv
venv\Scripts\activate
```

## 2) Installer les dépendances

```
pip install -r requirements.txt
```

### 3) Lancer l'application

```
python eigen_gui.py
```

---

## Utilisation rapide

### Saisie de la matrice

- Choisis n, clique **Créer matrice**
- Remplis le tableau
- Clique **Calculer**

### Import / Coller une matrice

- Clique **Coller matrice (texte)**  
Exemples acceptés :

```
4 1 1
1 3 0
1 0 2
```

Complexes :

```
1+2j 0
0 1-2j
```

### Exports

- **Exporter JSON** : idéal pour réutiliser dans un autre script
- **Exporter Excel** : crée des onglets (A, valeurs propres, vecteurs propres, diagnostic)
- **Exporter PDF** : rapport lisible

### API Valeurs Propres (onglet)

- Renseigne host/port
- Clique **Démarrer API**

- Test via curl (bouton copier)
  - Docs interactives (si API en marche) : <http://host:port/docs>
- 

## Notes

- Certaines matrices sont **défectueuses** : elles n'ont pas assez de vecteurs propres indépendants. Dans ce cas le programme retourne les valeurs propres et signale `warning`. Si SciPy est installé, la décomposition de **Schur** est affichée dans l'onglet Diagnostic.
- Pour distribuer sous Windows : utilise PyInstaller (voir `BUILD_WINDOWS.md` ou `EigenGUI.spec`).

---

```
EigenGUI.spec (PyInstaller)

> Avantage : il inclut proprement `eigen_gui.ui` sans se battre avec `--add-data`.
> Build : `pyinstaller --noconfirm EigenGUI.spec`

```python
# -*- mode: python ; coding: utf-8 -*-

import os
from PyInstaller.utils.hooks import collect_submodules

block_cipher = None

project_dir = os.path.abspath(os.path.dirname(__file__))
ui_path = os.path.join(project_dir, "eigen_gui.ui")

# FastAPI/uvicorn peuvent être optionnels : PyInstaller peut rater des imports dynamiques.
# On collecte les sous-modules pour être plus robuste.
hiddenimports = []
hiddenimports += collect_submodules("fastapi")
hiddenimports += collect_submodules("uvicorn")
hiddenimports += collect_submodules("pydantic")

a = Analysis(
    ["eigen_gui.py"],
    pathex=[project_dir],
    binaries=[],
```

```
datas=[(ui_path, ".")],
hiddenimports=hiddenimports,
hookspath=[],
hooksconfig={},
runtime_hooks=[],
excludes=[],
win_no_prefer_redirects=False,
win_private_assemblies=False,
cipher=block_cipher,
noarchive=False,
)

pyz = PYZ(a.pure, a.zipped_data, cipher=block_cipher)

exe = EXE(
    pyz,
    a.scripts,
    a.binaries,
    a.zipfiles,
    a.datas,
    [],
    name="EigenGUI",
    debug=False,
    bootloader_ignore_signals=False,
    strip=False,
    upx=True,
    upx_exclude=[],
    runtime_tmpdir=None,
    console=False, # fenêtre GUI (pas de console)
    disable_windowed_traceback=False,
    argv_emulation=False,
    target_arch=None,
    codesign_identity=None,
    entitlements_file=None,
)

coll = COLLECT(
    exe,
    a.binaries,
    a.zipfiles,
    a.datas,
    strip=False,
    upx=True,
    upx_exclude=[],
    name="EigenGUI",
)
```

Commande de build avec le `.spec`

Dans ton venv activé :

```
pyinstaller --noconfirm EigenGUI.spec
```

Résultat :

- `dist\EigenGUI\EigenGUI.exe`
-

Si tu veux, je peux aussi te fournir un fichier optionnel `build_and_zip.ps1` (script PowerShell) qui :

1. crée le venv
2. installe
3. build le .exe
4. génère automatiquement `EigenGUI_Windows.zip` et `EigenGUI_Project_Source.zip`

oui

Voici le fichier optionnel `build_and_zip.ps1` (script PowerShell) qui fait tout automatiquement :

1. crée venv
2. installe les dépendances
3. lance un test rapide
4. génère l'exécutable Windows via PyInstaller (avec `EigenGUI.spec` si présent, sinon via ligne de commande)
5. crée **deux ZIP** :
 - `EigenGUI_Windows.zip` (distribution exécutable)
 - `EigenGUI_Project_Source.zip` (sources + docs + requirements)

Crée un fichier `build_and_zip.ps1` à la racine de `EigenGUI_Project/` avec ce contenu :

```

# build_and_zip.ps1
# Script PowerShell pour build + zip (Windows)

$ErrorActionPreference = "Stop"

Write-Host "==== EigenGUI : build + zip ===" -ForegroundColor Cyan

# Aller dans le dossier du script
Set-Location -Path $PSScriptRoot

# 1) Créer venv si absent
if (-Not (Test-Path ".\venv")) {
    Write-Host "Création de l'environnement virtuel (venv)..." -ForegroundColor Yellow
    python -m venv venv
}

# 2) Activer venv
Write-Host "Activation de venv..." -ForegroundColor Yellow
& ".\venv\Scripts\Activate.ps1"

# 3) Installer dépendances
Write-Host "Installation des dépendances..." -ForegroundColor Yellow
pip install --upgrade pip
pip install -r requirements.txt

# 4) Test rapide (optionnel)
Write-Host "Test (import)..." -ForegroundColor Yellow
python -c "import numpy; import PyQt6; print('OK imports')"

# 5) Build EXE
if (Test-Path ".\EigenGUI.spec") {
    Write-Host "Build via PyInstaller (.spec)..." -ForegroundColor Yellow
    pyinstaller --noconfirm .\EigenGUI.spec
} else {
    Write-Host "Build via PyInstaller (ligne de commande)..." -ForegroundColor Yellow
    pyinstaller --noconfirm --windowed --name EigenGUI --add-data "eigen_gui.ui;." eigen_gui.py
}

# 6) Crée ZIP exécutable
Write-Host "Création du ZIP exécutable..." -ForegroundColor Yellow
if (Test-Path ".\dist\EigenGUI\EigenGUI.exe") {
    # Build en dossier
    Compress-Archive -Path ".\dist\EigenGUI\*" -DestinationPath ".\EigenGUI_Windows.zip" -Force
    Write-Host "-> EigenGUI_Windows.zip créé (dossier dist\EigenGUI\*)" -ForegroundColor Green
} elseif (Test-Path ".\dist\EigenGUI.exe") {
    # Build onefile
}

```

```

Compress-Archive -Path ".\dist\EigenGUI.exe" -DestinationPath ".\EigenGUI_Windows.zip" -Force
Write-Host "-> EigenGUI_Windows.zip créé (dist\EigenGUI.exe)" -ForegroundColor Green
} else {
    Write-Host "ERREUR: exécutable introuvable dans dist\.." -ForegroundColor Red
    exit 1
}

# 7) ZIP des sources
Write-Host "Création du ZIP sources..." -ForegroundColor Yellow

$sourceFiles = @(
    ".\eigen_gui.py",
    ".\eigen_gui.ui",
    ".\requirements.txt",
    ".\BUILD_WINDOWS.md"
)

if (Test-Path ".\README.md") { $sourceFiles += ".\README.md" }
if (Test-Path ".\EigenGUI.spec") { $sourceFiles += ".\EigenGUI.spec" }
if (Test-Path ".\build_and_zip.ps1") { $sourceFiles += ".\build_and_zip.ps1" }

Compress-Archive -Path $sourceFiles -DestinationPath ".\EigenGUI_Project_Source.zip" -Force
Write-Host "-> EigenGUI_Project_Source.zip créé" -ForegroundColor Green

Write-Host "==== Terminé ===" -ForegroundColor Cyan
Write-Host "Fichiers générés :" -ForegroundColor Cyan
Write-Host " - EigenGUI_Windows.zip" -ForegroundColor Cyan
Write-Host " - EigenGUI_Project_Source.zip" -ForegroundColor Cyan

```

Exécution (PowerShell)

Dans EigenGUI_Project/ :

```

Set-ExecutionPolicy -Scope Process Bypass
.\build_and_zip.ps1

```

Si tu veux, je peux aussi te donner une version .bat (invite de commande classique) en plus du .ps1.

oui

Voici une version **Invite de commandes Windows** (fichier .bat) qui fait le même travail que le .ps1 :

- crée venv
- installe requirements.txt
- build l'exécutable avec PyInstaller (utilise EigenGUI.spec si présent, sinon ligne de commande)
- crée deux ZIP :
 - EigenGUI_Windows.zip
 - EigenGUI_Project_Source.zip

Crée un fichier build_and_zip.bat dans le dossier EigenGUI_Project\ avec ce contenu :

```
@echo off
setlocal enabledelayedexpansion

echo === EigenGUI : build + zip (BAT) ===

cd /d "%~dp0"

REM 1) Créer venv si absent
if not exist "venv\" (
  echo [1/6] Creation de venv...
  python -m venv venv
)

REM 2) Activer venv
echo [2/6] Activation de venv...
call "venv\Scripts\activate.bat"

REM 3) Installer dependances
echo [3/6] Installation des dependances...
python -m pip install --upgrade pip
pip install -r requirements.txt

REM 4) Test import rapide
echo [4/6] Test imports...
python -c "import numpy; import PyQt6; print('OK imports')"
if errorlevel 1 (
  echo ERREUR: imports impossible. Verifiez Python/venv.
  exit /b 1
)

REM 5) Build exe
echo [5/6] Build exe...
if exist "EigenGUI.spec" (
  pyinstaller --noconfirm "EigenGUI.spec"
) else (

```

```

pyinstaller --noconfirm --windowed --name EigenGUI --add-data "eigen_gui.ui;." eigen_gui.py
)

REM 6) ZIPs (utilise PowerShell pour zipper)
echo [6/6] Creation des ZIP...

if exist "dist\EigenGUI\EigenGUI.exe" (
    powershell -NoProfile -Command "Compress-Archive -Path 'dist\EigenGUI\*' -DestinationPath 'EigenGUI_Windows.zip' -Force"
    echo -> EigenGUI_Windows.zip cree (dist\EigenGUI\*)
) else if exist "dist\EigenGUI.exe" (
    powershell -NoProfile -Command "Compress-Archive -Path 'dist\EigenGUI.exe' -DestinationPath 'EigenGUI_Windows.zip' -Force"
    echo -> EigenGUI_Windows.zip cree (dist\EigenGUI.exe)
) else (
    echo ERREUR: exécutable introuvable dans dist\
    exit /b 1
)

REM ZIP sources
set SOURCES=eigen_gui.py eigen_gui.ui requirements.txt BUILD_WINDOWS.md
if exist "README.md" set SOURCES=!SOURCES! README.md
if exist "EigenGUI.spec" set SOURCES=!SOURCES! EigenGUI.spec
if exist "build_and_zip.ps1" set SOURCES=!SOURCES! build_and_zip.ps1
if exist "build_and_zip.bat" set SOURCES=!SOURCES! build_and_zip.bat

powershell -NoProfile -Command "Compress-Archive -Path %SOURCES% -DestinationPath 'EigenGUI_Project_Source.zip' -Force"
echo -> EigenGUI_Project_Source.zip cree

echo === Termine ===
echo Fichiers generes :
echo - EigenGUI_Windows.zip
echo - EigenGUI_Project_Source.zip

endlocal
pause

```

Utilisation

Double-clique build_and_zip.bat
ou lance-le depuis cmd :

```

cd EigenGUI_Project
build_and_zip.bat

```

Si tu veux, je peux aussi ajouter dans le .bat une option interactive :

- choisir --onefile ou dist\ (dossier)
- inclure/exclure l'API (FastAPI) pour alléger l'exécutable

oui

Voici une version **interactive** de build_and_zip.bat qui te demande :

1. **mode de build :**
 - o 1 = dossier dist\EigenGUI\ (recommandé)
 - o 2 = --onefile (un seul .exe)
2. **inclure l'API :**
 - o O = oui (garde FastAPI/uvicorn/pydantic)
 - o N = non (installe une version "lite" sans API)

→ Elle génère ensuite :

- EigenGUI_Windows.zip
- EigenGUI_Project_Source.zip

Crée/écrase build_and_zip.bat dans EigenGUI_Project\ avec ceci :

```
@echo off
setlocal enabledelayedexpansion

echo =====
echo EigenGUI : build + zip (BAT interactif)
echo =====

cd /d "%~dp0"

REM --- Choix 1: mode build
echo.
echo Choisissez le mode de build :
echo [1] Dossier dist\EigenGUI\ (recommandé)
echo [2] Onefile (un seul .exe)
set /p BUILD_MODE="Votre choix (1/2) : "
if "%BUILD_MODE%"=="1" set BUILD_MODE=1
```

```
REM --- Choix 2: inclure l'API ?
echo.
echo Inclure l'API (FastAPI/uvicorn/pydantic) ?
echo [O] Oui
echo [N] Non (version lite)
set /p INCLUDE_API="Votre choix (O/N) : "
if "%INCLUDE_API%""==" set INCLUDE_API=0

REM Normaliser
set INCLUDE_API=%INCLUDE_API:~0,1%
if /I not "%INCLUDE_API%"=="O" if /I not "%INCLUDE_API%"=="N" set INCLUDE_API=0

echo.
echo --- Resume ---
if "%BUILD_MODE%"=="2" (
    echo Mode build : Onefile
) else (
    echo Mode build : Dossier dist\EigenGUI\
)
if /I "%INCLUDE_API%"=="O" (
    echo API : Incluse
) else (
    echo API : Exclue (lite)
)
echo.

REM 1) Créer venv si absent
if not exist "venv\" (
    echo [1/7] Creation de venv...
    python -m venv venv
)

REM 2) Activer venv
echo [2/7] Activation de venv...
call "venv\Scripts\activate.bat"

REM 3) Installer dependances
echo [3/7] Installation des dependances...
python -m pip install --upgrade pip

REM Si API exclue: installer requirements sans fastapi/uvicorn/pydantic
if /I "%INCLUDE_API%"=="N" (
    echo -> Installation mode LITE (sans API)...
    python -m pip install PyQt6 numpy scipy reportlab openpyxl pyinstaller
) else (
    echo -> Installation mode FULL (avec API)...
```

```
    pip install -r requirements.txt
)

REM 4) Test import rapide
echo [4/7] Test imports...
python -c "import numpy; import PyQt6; print('OK imports')"
if errorlevel 1 (
    echo ERREUR: imports impossible. Verifiez Python/venv.
    exit /b 1
)

REM 5) Nettoyage anciens builds (optionnel)
echo [5/7] Nettoyage ancien build (dist/build)...
if exist "dist\" rmdir /s /q dist
if exist "build\" rmdir /s /q build
if exist "EigenGUI.spec" (
    REM On garde le .spec, pas touche
)

REM 6) Build exe
echo [6/7] Build exe...
set PYI_CMD=

REM Si spec existe et API incluse, on peut l'utiliser
REM Si API exclue, le spec peut tirer des hiddenimports fastapi -> on préfère la ligne de commande
if exist "EigenGUI.spec" (
    if /I "%INCLUDE_API%"=="O" (
        set PYI_CMD=pyinstaller --noconfirm "EigenGUI.spec"
    ) else (
        set PYI_CMD=
    )
)

if "%PYI_CMD%"=="" (
    REM build via ligne de commande
    if "%BUILD_MODE%"=="2" (
        pyinstaller --noconfirm --windowed --onefile --name EigenGUI --add-data "eigen_gui.ui;." eigen_gui.py
    ) else (
        pyinstaller --noconfirm --windowed --name EigenGUI --add-data "eigen_gui.ui;." eigen_gui.py
    )
) else (
    %PYI_CMD%
)

REM 7) ZIPs
echo [7/7] Creation des ZIP...
```

```

REM ZIP exe
if exist "dist\EigenGUI\EigenGUI.exe" (
    powershell -NoProfile -Command "Compress-Archive -Path 'dist\EigenGUI\*' -DestinationPath 'EigenGUI_Windows.zip' -Force"
    echo -> EigenGUI_Windows.zip cree (dist\EigenGUI\*)
) else if exist "dist\EigenGUI.exe" (
    powershell -NoProfile -Command "Compress-Archive -Path 'dist\EigenGUI.exe' -DestinationPath 'EigenGUI_Windows.zip' -Force"
    echo -> EigenGUI_Windows.zip cree (dist\EigenGUI.exe)
) else (
    echo ERREUR: exécutable introuvable dans dist\
    exit /b 1
)

REM ZIP sources
set SOURCES=eigen_gui.py eigen_gui.ui requirements.txt BUILD_WINDOWS.md
if exist "README.md" set SOURCES!=SOURCES! README.md
if exist "EigenGUI.spec" set SOURCES!=SOURCES! EigenGUI.spec
if exist "build_and_zip.ps1" set SOURCES!=SOURCES! build_and_zip.ps1
if exist "build_and_zip.bat" set SOURCES!=SOURCES! build_and_zip.bat

powershell -NoProfile -Command "Compress-Archive -Path %SOURCES% -DestinationPath 'EigenGUI_Project_Source.zip' -Force"
echo -> EigenGUI_Project_Source.zip cree

echo.
echo =====
echo    Termine !
echo    - EigenGUI_Windows.zip
echo    - EigenGUI_Project_Source.zip
echo =====
echo.

pause
endlocal

```

Comment l'utiliser

- Double-clique build_and_zip.bat
- Réponds aux 2 questions (mode build + API)
- À la fin, tu auras les deux ZIP

Petite recommandation

- Pour distribuer à d'autres PCs, le mode **dossier** (`dist\EigenGUI\`) est souvent plus fiable que `--onefile`.
- Si tu n'as pas besoin d'API, choisir **API = N** réduit la taille et les dépendances.

Si tu veux, je peux aussi te proposer une version "ultra-lite" qui n'inclut même pas SciPy (donc pas Schur) pour réduire encore la taille de l'exe.

En effet je préfère utiliser l'Invite de Commande de Windows

Parfait — dans ce cas, tu peux tout faire **uniquement avec l'Invite de commandes (cmd.exe)**, sans PowerShell, *y compris la création des ZIP*. Je te donne donc :

1. un `build_and_zip_cmd.bat` **100% cmd** (utilise seulement les outils Windows standards : `tar` est inclus sur Windows 10/11)
 2. les commandes manuelles équivalentes si tu préfères
-

1) `build_and_zip_cmd.bat` (100% Invite de commandes)

Crée un fichier `build_and_zip_cmd.bat` dans ton dossier `EigenGUI_Project\` :

```
@echo off
setlocal enabledelayedexpansion

echo =====
echo EigenGUI : build + zip (CMD only)
echo =====

cd /d "%~dp0"

REM --- Choix 1: mode build
echo.
echo Choisissez le mode de build :
echo [1] Dossier dist\EigenGUI\ (recommandé)
echo [2] Onefile (un seul .exe)
set /p BUILD_MODE="Votre choix (1/2) : "
if "%BUILD_MODE%"=="1" set BUILD_MODE=1

REM --- Choix 2: inclure l'API ?
echo.
echo Inclure l'API (FastAPI/uvicorn/pydantic) ?
echo [O] Oui
echo [N] Non (version lite)
```

```
set /p INCLUDE_API="Votre choix (O/N) : "
if "%INCLUDE_API%"==""
set INCLUDE_API=0
set INCLUDE_API=%INCLUDE_API:~0,1%  
  
echo.  
echo --- Resume ---  
if "%BUILD_MODE%"=="2" (
    echo Mode build : Onefile
) else (
    echo Mode build : Dossier dist\EigenGUI\
)  
if /I "%INCLUDE_API%"=="O" (
    echo API : Incluse
) else (
    echo API : Exclue (lite)
)  
echo.  
  
REM 1) venv  
if not exist "venv\" (
    echo [1/8] Creation de venv...
    python -m venv venv
)  
  
REM 2) activer venv  
echo [2/8] Activation de venv...
call "venv\Scripts\activate.bat"  
  
REM 3) install  
echo [3/8] Installation dependances...
python -m pip install --upgrade pip  
  
if /I "%INCLUDE_API%"=="N" (
    echo -> Mode LITE (sans API)...
    python -m pip install PyQt6 numpy scipy reportlab openpyxl pyinstaller
) else (
    echo -> Mode FULL (avec API)...
    pip install -r requirements.txt
)  
  
REM 4) test imports  
echo [4/8] Test imports...
python -c "import numpy; import PyQt6; print('OK imports')"
if errorlevel 1 (
    echo ERREUR: imports impossible.
    exit /b 1
```

```

)

REM 5) clean old build
echo [5/8] Nettoyage dist/build...
if exist "dist\" rmdir /s /q dist
if exist "build\" rmdir /s /q build

REM 6) build exe
echo [6/8] Build exe...
if exist "EigenGUI.spec" (
    REM Si API exclue, le spec peut embarquer fastapi -> on prefere ligne de commande
    if /I "%INCLUDE_API%"=="0" (
        pyinstaller --noconfirm "EigenGUI.spec"
    ) else (
        goto :BUILD_CMD
    )
) else (
    goto :BUILD_CMD
)

goto :AFTER_BUILD

:BUILD_CMD
if "%BUILD_MODE%"=="2" (
    pyinstaller --noconfirm --windowed --onefile --name EigenGUI --add-data "eigen_gui.ui;." eigen_gui.py
) else (
    pyinstaller --noconfirm --windowed --name EigenGUI --add-data "eigen_gui.ui;." eigen_gui.py
)

:AFTER_BUILD

REM 7) zip exe (cmd only) - utilise tar (Windows 10/11)
echo [7/8] ZIP exe...
if exist "dist\EigenGUI\EigenGUI.exe" (
    REM zip du dossier dist\EigenGUI\
    if exist "EigenGUI_Windows.zip" del /q "EigenGUI_Windows.zip"
    tar -a -c -f "EigenGUI_Windows.zip" -C "dist\EigenGUI" .
    echo -> EigenGUI_Windows.zip cree
) else if exist "dist\EigenGUI.exe" (
    if exist "EigenGUI_Windows.zip" del /q "EigenGUI_Windows.zip"
    tar -a -c -f "EigenGUI_Windows.zip" "dist\EigenGUI.exe"
    echo -> EigenGUI_Windows.zip cree
) else (
    echo ERREUR: exécutable introuvable dans dist\
    exit /b 1
)

```

```

REM 8) zip sources
echo [8/8] ZIP sources...
if exist "EigenGUI_Project_Source.zip" del /q "EigenGUI_Project_Source.zip"

set SOURCES=eigen_gui.py eigen_gui.ui requirements.txt BUILD_WINDOWS.md
if exist "README.md" set SOURCES!=SOURCES! README.md
if exist "EigenGUI.spec" set SOURCES!=SOURCES! EigenGUI.spec
if exist "build_and_zip_cmd.bat" set SOURCES!=SOURCES! build_and_zip_cmd.bat

tar -a -c -f "EigenGUI_Project_Source.zip" %SOURCES%
echo -> EigenGUI_Project_Source.zip cree

echo.
echo =====
echo    Termine !
echo    - EigenGUI_Windows.zip
echo    - EigenGUI_Project_Source.zip
echo =====
echo.
pause
endlocal

```

Exécution (cmd.exe)

Ouvre **Invite de commandes**, va dans le dossier :

```

cd EigenGUI_Project
build_and_zip_cmd.bat

```

2) Version “commandes manuelles” (cmd.exe)

Si tu préfères ne pas utiliser le .bat, voici la séquence :

```

cd EigenGUI_Project
python -m venv venv
venv\Scripts\activate
pip install -r requirements.txt
pyinstaller --noconfirm --windowed --name EigenGUI --add-data "eigen_gui.ui;." eigen_gui.py

REM zip exe (Windows 10/11)

```

```
tar -a -c -f EigenGUI_Windows.zip -C dist\EigenGUI .
REM zip sources
tar -a -c -f EigenGUI_Project_Source.zip eigen_gui.py eigen_gui.ui requirements.txt BUILD_WINDOWS.md
```

Remarque importante

Le .bat utilise tar -a pour créer des .zip.

- Sur Windows 10/11, tar est généralement disponible (bsdtar).
- Si ton Windows n'a pas tar, dis-le moi et je te donne une variante **sans tar** (avec un zip via VBScript pur cmd, plus long mais 100% natif).

Si tu veux, je peux aussi adapter le script pour **forcer onefile** par défaut et **désactiver l'API** par défaut, selon ton usage.

Si ton Windows n'a pas tar, dis-le moi et je te donne une variante sans tar (avec un zip via VBScript pur cmd, plus long mais 100% natif). Puis Si tu veux, je peux aussi adapter le script pour forcer onefile par défaut et désactiver l'API par défaut, selon ton usage.

Voici une variante **100% Invite de commandes (cmd.exe)** qui :

- **n'utilise pas tar**
- **crée des ZIP via VBScript** (présent sur Windows)
- **force --onefile par défaut**
- **désactive l'API par défaut** (mode "lite")
- garde quand même un menu pour changer si tu veux

Tu n'as besoin que de python installé.

build_and_zip_cmd_no_tar.bat (ZIP sans tar + onefile par défaut + API OFF par défaut)

Crée un fichier build_and_zip_cmd_no_tar.bat dans EigenGUI_Project\ :

```
@echo off
setlocal enabledelayedexpansion
echo =====
```

```
echo EigenGUI : build + zip (CMD only, no tar)
echo - ZIP via VBScript (built-in)
echo - onefile par defaut
echo - API OFF (lite) par defaut
echo =====

cd /d "%~dp0"

REM --- Defaults
set BUILD_MODE=2
set INCLUDE_API=N

echo.
echo Choisir (ENTREE = valeurs par defaut) :
echo Mode build : [1]=dossier dist\EigenGUI\  [2]=onefile  (defaut=2)
set /p BUILD_MODE_IN="Votre choix (1/2) : "
if not "%BUILD_MODE_IN%"==" " set BUILD_MODE=%BUILD_MODE_IN%

echo.
echo API : [O]=inclure  [N]=exclure (lite)  (defaut=N)
set /p INCLUDE_API_IN="Votre choix (O/N) : "
if not "%INCLUDE_API_IN%"==" " set INCLUDE_API=%INCLUDE_API_IN%
set INCLUDE_API=%INCLUDE_API:~0,1%

if /I not "%INCLUDE_API%"=="O" if /I not "%INCLUDE_API%"=="N" set INCLUDE_API=N
if not "%BUILD_MODE%"=="1" if not "%BUILD_MODE%"=="2" set BUILD_MODE=2

echo.
echo --- Resume ---
if "%BUILD_MODE%"=="2" (echo Mode build : onefile) else (echo Mode build : dossier dist\EigenGUI\)
if /I "%INCLUDE_API%"=="O" (echo API : incluse) else (echo API : exclue (lite))
echo.

REM 1) venv
if not exist "venv\" (
    echo [1/9] Creation de venv...
    python -m venv venv
)

REM 2) activate venv
echo [2/9] Activation de venv...
call "venv\Scripts\activate.bat"

REM 3) install deps
echo [3/9] Installation dependances...
python -m pip install --upgrade pip
```

```
if /I "%INCLUDE_API%"=="N" (
    echo -> Mode LITE (sans API)...
    python -m pip install PyQt6 numpy scipy reportlab openpyxl pyinstaller
) else (
    echo -> Mode FULL (avec API)...
    pip install -r requirements.txt
)

REM 4) quick import test
echo [4/9] Test imports...
python -c "import numpy; import PyQt6; print('OK imports')"
if errorlevel 1 (
    echo ERREUR: imports impossible.
    exit /b 1
)

REM 5) clean old
echo [5/9] Nettoyage dist/build...
if exist "dist\" rmdir /s /q dist
if exist "build\" rmdir /s /q build

REM 6) build exe
echo [6/9] Build exe...
if "%BUILD_MODE%"=="2" (
    pyinstaller --noconfirm --windowed --onefile --name EigenGUI --add-data "eigen_gui.ui;." eigen_gui.py
) else (
    pyinstaller --noconfirm --windowed --name EigenGUI --add-data "eigen_gui.ui;." eigen_gui.py
)

REM 7) prepare VBScript zipper
echo [7/9] Preparation du zipper (VBScript)...
set VBSZIP=%TEMP%\zip_folder.vbs
(
    echo ' zip_folder.vbs
    echo Option Explicit
    echo Dim src, zipf
    echo src = WScript.Arguments(0)
    echo zipf = WScript.Arguments(1)
    echo Dim fso: Set fso = CreateObject("Scripting.FileSystemObject")
    echo If fso.FileExists(zipf) Then fso.DeleteFile zipf, True
    echo ' Create empty zip file header
    echo Dim ts: Set ts = fso.CreateTextFile(zipf, True)
    echo ts.Write "PK" ^& Chr(5) ^& Chr(6) ^& String(18, Chr(0))
    echo ts.Close
    echo Dim sh: Set sh = CreateObject("Shell.Application")
```

```

echo Dim z: Set z = sh.NameSpace(zipf)
echo If z Is Nothing Then WScript.Quit(2)
echo Dim s: Set s = sh.NameSpace(src)
echo If s Is Nothing Then WScript.Quit(3)
echo z.CopyHere s.Items, 16
echo ' wait copy finishes (simple loop)
echo Dim i
echo For i = 1 To 200
echo   WScript.Sleep 100
echo Next
) > "%VBSZIP%"

REM 8) zip exe
echo [8/9] Creation ZIP exe...
if exist "EigenGUI_Windows.zip" del /q "EigenGUI_Windows.zip"

if exist "dist\EigenGUI.exe" (
  REM onefile
  mkdir "_zip_tmp" >nul 2>&1
  copy /y "dist\EigenGUI.exe" "_zip_tmp\EigenGUI.exe" >nul
  cscript //nologo "%VBSZIP%" "%~dp0_zip_tmp" "%~dp0EigenGUI_Windows.zip"
  rmdir /s /q "_zip_tmp"
  echo -> EigenGUI_Windows.zip cree (onefile)
) else if exist "dist\EigenGUI\EigenGUI.exe" (
  REM folder build
  cscript //nologo "%VBSZIP%" "%~dp0dist\EigenGUI" "%~dp0EigenGUI_Windows.zip"
  echo -> EigenGUI_Windows.zip cree (folder dist\EigenGUI)
) else (
  echo ERREUR: exécutable introuvable dans dist\
  exit /b 1
)

REM 9) zip sources
echo [9/9] Creation ZIP sources...
if exist "EigenGUI_Project_Source.zip" del /q "EigenGUI_Project_Source.zip"

mkdir "_src_tmp" >nul 2>&1
copy /y "eigen_gui.py" "_src_tmp\" >nul
copy /y "eigen_gui.ui" "_src_tmp\" >nul
copy /y "requirements.txt" "_src_tmp\" >nul
if exist "BUILD_WINDOWS.md" copy /y "BUILD_WINDOWS.md" "_src_tmp\" >nul
if exist "README.md" copy /y "README.md" "_src_tmp\" >nul
if exist "EigenGUI.spec" copy /y "EigenGUI.spec" "_src_tmp\" >nul
copy /y "%~nx0" "_src_tmp\" >nul

cscript //nologo "%VBSZIP%" "%~dp0_src_tmp" "%~dp0EigenGUI_Project_Source.zip"

```

```
rmdir /s /q "_src_tmp"

echo.
echo =====
echo  Termine !
echo  - EigenGUI_Windows.zip
echo  - EigenGUI_Project_Source.zip
echo =====
echo.
pause
endlocal
```

Utilisation (cmd.exe)

```
cd EigenGUI_Project
build_and_zip_cmd_no_tar.bat
```

Notes (importantes)

- Le ZIP via VBScript utilise `Shell.Application` et peut être un peu lent selon la taille.
- Si tu veux un ZIP plus rapide sans `tar`, la meilleure option reste PowerShell, mais tu as demandé **cmd only**, donc on reste natif.
- Ce script met l'EXE dans un zip en copiant dans un dossier temporaire (propre).

Si tu me dis ton OS exact (Windows 10 ou 11) et ta version Python, je peux aussi te proposer une version qui **déetecte automatiquement** la présence de `tar` et choisit la méthode la plus rapide.

```
import numpy as np
def read_matrix_from_user() -> np.ndarray:
    """Gère la saisie d'une matrice carrée par l'utilisateur via la console."""
    print("== Saisie d'une matrice ==")
    while True:
        try:
            n = int(input("Entrez la dimension n (matrice n x n) : ").strip())
            if n <= 0:
                print(" -> Erreur : n doit être un entier strictement positif.")
                continue
            break
        except ValueError:
            print(" -> Erreur : Veuillez entrer un nombre entier valide.")
            print("\nEntrez les coefficients ligne par ligne (séparés par des espaces).")
            print("Réels : 3 -1.2 0 4.5")
            print("Complexes : 1+2j 3 -0.5j 2\n")
            A = np.zeros((n, n), dtype=complex)
            for i in range(n):
                while True:
                    parts = input(f'Ligne {i+1} : ').strip().split()
                    if len(parts) != n:
                        print(f' -> Il faut exactement {n} valeurs.')
                        continue
                    try:
                        A[i] = [complex(x) for x in parts]
                    except ValueError:
                        print(" -> Valeur invalide. Exemple : 2, -1.5, 1+2j")
                        # Si la matrice est purement réelle, on la convertit en float pour optimiser les calculs
                        if np.allclose(A.imag, 0.0, atol=1e-14):
                            A = A.real.astype(float)
                    break
            break
    return A

def residual_rel(A: np.ndarray, w: np.ndarray, V: np.ndarray) -> float:
    """Calcule le résidu relatif ||AV - V diag(w)|| / ||AV||."""
    AV = A @ V
    R = AV - V @ np.diag(w)
    num = np.linalg.norm(R, ord='fro')
    den = np.linalg.norm(AV, ord='fro') + 1e-30
    return float(num / den)

def robust_eigen(A: np.ndarray, quality_tol: float = 1e-8, cond_tol: float = 1e12) -> dict:
    """Calcule les valeurs et vecteurs propres d'une matrice. Fournit la décomposition de Schur si les vecteurs propres sont instables."""
    A = np.asarray(A)
```

```

if A.ndim != 2 or A.shape[0] != A.shape[1]: raise ValueError("La matrice doit être carrée.") out = {"n": A.shape[0]} # 1. Calcul direct des valeurs et vecteurs propres
try: from scipy.linalg import eig as scipy_eig, schur w, V = scipy_eig(A) scipy_available = True except ImportError: # Fallback si SciPy n'est pas installé
w, V = np.linalg.eig(A) scipy_available = False out["eigenvalues"] = w out["eigenvectors"] = V # 2. Évaluation de la qualité (Résidu et Conditionnement)
err = residual_rel(A, w, V) out["residual_rel"] = err try: CV = np.linalg.cond(V) except np.linalg.LinAlgError: CV = np.inf
out["cond_vectors"] = float(CV) # 3. Stratégie de décision if err < quality_tol and CV < cond_tol: out["status"] = "ok" out["message"] = "Vecteurs propres calculés et jugés fiables." else: out["status"] = "warning" out["message"] = ("Valeurs propres OK. Vecteurs propres potentiellement instables" "(matrice défectueuse ou proche).") # 4. Calcul de Schur en cas de matrice défectueuse (si SciPy est là) if scipy_available: T, Z = schur(A, output='complex')
out["schur_T"] = T out["schur_Z"] = Z out["message"] += "Utilisez la décomposition de Schur (T, Z) fournie." else: out["message"] += "Installez SciPy pour obtenir la décomposition de Schur." return out def main(): A = read_matrix_from_user() print("\n==== Matrice A ====") with np.printoptions(precision=4, suppress=True): # Formatage propre pour l'affichage
print(A) result = robust_eigen(A) print("\n==== Résultats ====") print(f"Status : {result['status'].upper()}") print(f"Message : {result['message']}") with np.printoptions(precision=4, suppress=True): print("\nValeurs propres :") print(result["eigenvalues"]) if "eigenvectors" in result: print("\nVecteurs propres (en colonnes) :") print(result["eigenvectors"])
print(f"\nErreur relative : {result.get('residual_rel', 'N/A'):.2e}") print(f"Conditionnement cond(V) : {result.get('cond_vectors', 'N/A'):.2e}") if "schur_T" in result: print("\n--- (Info robuste) Décomposition de Schur A = Z T Z* ---") print("Matrice triangulaire supérieure T (Valeurs propres sur la diagonale) :") print(result["schur_T"])
if __name__ == "__main__": main()

```

Ton script est déjà solide 🤘. Je te propose ci-dessous une **revue + corrections/optimisations** (sans changer l'esprit), avec une version améliorée prête à copier-coller.

Points importants à corriger/améliorer

1. SciPy import

Tu fais `from scipy.linalg import eig as scipy_eig, schur` dans un `try`, c'est bien.

Mais tu recalculs ensuite `schur` seulement si `scipy_available`. OK.

2. Résidu relatif

Ton résidu utilise $\|Av\|$ au dénominateur. C'est correct, mais pour une mesure plus standard on préfère souvent $\|A\| \cdot \|v\|$ (évite des cas où Av est petit). Je te laisse ton choix; je propose une variante plus robuste.

3. Conditionnement

`np.linalg.cond(V)` peut exploser si V est presque singulière (ce qui est précisément l'indicateur). C'est très bien. Je conseille juste de forcer `float(CV)` et d'afficher proprement.

4. Matrices réelles

Tu convertis en `float` si imaginaire nul: bien. Mais ensuite `eig/schur` peuvent retourner complexe quand même (normal). L'affichage avec `suppress=True` peut masquer des petites valeurs; ok pour lisibilité.

5. Cas "défectueux"

Ton programme "corrige" correctement le problème : il **n'invente pas** des vecteurs propres, il **prévient** et propose Schur. C'est exactement la bonne approche.

Version améliorée (même comportement, un peu plus robuste)

```
import numpy as np

def read_matrix_from_user() -> np.ndarray:
    """Saisie d'une matrice carrée (réelle ou complexe) via console."""
    print("== Saisie d'une matrice ==")
    while True:
        try:
            n = int(input("Entrez la dimension n (matrice n x n) : ").strip())
            if n <= 0:
                print(" -> Erreur : n doit être un entier strictement positif.")
                continue
            break
        except ValueError:
            print(" -> Erreur : Veuillez entrer un nombre entier valide.")

    print("\nEntrez les coefficients ligne par ligne (séparés par des espaces).")
    print("Réels : 3 -1.2 0 4.5")
    print("Complexes : 1+2j 3 -0.5j 2\n")

    A = np.zeros((n, n), dtype=complex)

    for i in range(n):
        while True:
            parts = input(f"Ligne {i+1} : ").strip().split()
            if len(parts) != n:
                print(f" -> Il faut exactement {n} valeurs.")
                continue
            try:
                A[i, :] = [complex(x) for x in parts]
                break
            except ValueError:
                print(" -> Valeur invalide. Exemple : 2, -1.5, 1+2j")

    # Si la matrice est (quasi) réelle, convertir en float
    if np.allclose(A.imag, 0.0, atol=1e-14):
        return A.real.astype(float)
    return A

def residual_rel(A: np.ndarray, w: np.ndarray, V: np.ndarray) -> float:
```

```

"""
Résidu relatif plus standard :
    ||A V - V diag(w)||_F / (||A||_F * ||V||_F)
"""

A = np.asarray(A)
w = np.asarray(w)
V = np.asarray(V)

R = A @ V - V @ np.diag(w)
num = np.linalg.norm(R, ord="fro")
den = (np.linalg.norm(A, ord="fro") * np.linalg.norm(V, ord="fro")) + 1e-30
return float(num / den)

def robust_eigen(A: np.ndarray, quality_tol: float = 1e-8, cond_tol: float = 1e12) -> dict:
    """
    Calcule valeurs/vecteurs propres. Si instable, fournit Schur (si SciPy dispo).
    """
    A = np.asarray(A)
    if A.ndim != 2 or A.shape[0] != A.shape[1]:
        raise ValueError("La matrice doit être carrée.")

    out = {"n": int(A.shape[0])}

    # 1) eig (SciPy si dispo, sinon NumPy)
    scipy_available = False
    try:
        from scipy.linalg import eig as scipy_eig, schur
        w, V = scipy_eig(A)
        scipy_available = True
    except Exception:
        w, V = np.linalg.eig(A)

    out["eigenvalues"] = w
    out["eigenvectors"] = V

    # 2) Qualité : résidu + conditionnement
    err = residual_rel(A, w, V)
    out["residual_rel"] = float(err)

    try:
        CV = np.linalg.cond(V)
    except np.linalg.LinAlgError:
        CV = np.inf
    out["cond_vectors"] = float(CV)

```

```

# 3) Décision
if err < quality_tol and CV < cond_tol:
    out["status"] = "ok"
    out["message"] = "Vecteurs propres calculés et jugés fiables."
    return out

out["status"] = "warning"
out["message"] =
    "Valeurs propres OK. Vecteurs propres potentiellement instables "
    "(matrice défectueuse ou proche). "
)

# 4) Schur si possible
if scipy_available:
    T, Z = schur(A, output="complex")      # A = Z T Z*
    out["schur_T"] = T
    out["schur_Z"] = Z
    out["message"] += "Utilisez la décomposition de Schur (T, Z) fournie."
else:
    out["message"] += "Installez SciPy pour obtenir la décomposition de Schur."

return out

def main():
    A = read_matrix_from_user()
    print("\n==== Matrice A ====")
    with np.printoptions(precision=4, suppress=True):
        print(A)

    result = robust_eigen(A)

    print("\n==== Résultats ====")
    print(f"Status : {result['status'].upper()}")
    print(f"Message : {result['message']}")

    with np.printoptions(precision=4, suppress=True):
        print("\nValeurs propres :")
        print(result["eigenvalues"])

        print("\nVecteurs propres (en colonnes) :")
        print(result["eigenvectors"])

        print(f"\nErreur relative : {result['residual_rel']:.2e}")
        print(f"Conditionnement cond(V) : {result['cond_vectors']:.2e}")

```

```
if "schur_T" in result:  
    print("\n--- (Info robuste) Décomposition de Schur A = Z T Z* ---")  
    print("Matrice triangulaire supérieure T :")  
    print(result["schur_T"])  
  
if __name__ == "__main__":  
    main()
```

Mini test concret à essayer

1. Matrice “normale” :

```
n=3  
4 1 1  
1 3 0  
1 0 2
```

→ status OK

2. Matrice défectueuse :

```
n=2  
1 1  
0 1
```

→ souvent WARNING + Schur si SciPy est installé

Si tu veux, je peux aussi ajouter :

- une option --no-scipy en ligne de commande,
- un tri des valeurs propres,
- ou un affichage “joli” (formatage complexe $a+bi$).

```
PS D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\ PYTHON> python -m venv env PS D:\PYTHON DJANGO HOPES\VALEURS PROPRES  
python\ PYTHON> env\Scripts\activate (env) PS D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\ PYTHON> pip install -r requirements.txt
```

Collecting numpy (from -r requirements.txt (line 1)) Downloading numpy-2.4.2-cp311-cp311-win_amd64.whl.metadata (6.6 kB) Collecting scipy (from -r requirements.txt (line 2)) Downloading scipy-1.17.1-cp311-cp311-win_amd64.whl.metadata (60 kB) _____

61.0/61.0 kB 190.6 kB/s eta 0:00:00 Collecting matplotlib (from -r requirements.txt (line 3)) Downloading matplotlib-3.10.8-cp311-cp311-win_amd64.whl.metadata (52 kB) _____

52.8/52.8 kB 304.2 kB/s eta 0:00:00 Collecting contourpy>=1.0.1 (from matplotlib->-r requirements.txt (line 3)) Using cached contourpy-1.3.3-cp311-cp311-win_amd64.whl.metadata (5.5 kB) Collecting cycler>=0.10 (from matplotlib->-r requirements.txt (line 3)) Using cached cycler-0.12.1-py3-none-any.whl.metadata (3.8 kB) Collecting fonttools>=4.22.0 (from matplotlib->-r requirements.txt (line 3)) Downloading fonttools-4.61.1-cp311-win_amd64.whl.metadata (116 kB) _____ 116.4/116.4 kB 378.0 kB/s eta 0:00:00 Collecting kiwisolver>=1.3.1 (from matplotlib->-r requirements.txt (line 3)) Using cached kiwisolver-1.4.9-cp311-cp311-win_amd64.whl.metadata (6.4 kB) Collecting packaging>=20.0 (from matplotlib->-r requirements.txt (line 3)) Using cached packaging-26.0-py3-none-any.whl.metadata (3.3 kB) Collecting pillow>=8 (from matplotlib->-r requirements.txt (line 3)) Downloading pillow-12.1.1-cp311-cp311-win_amd64.whl.metadata (9.0 kB) Collecting pyparsing>=3 (from matplotlib->-r requirements.txt (line 3)) Downloading pyparsing-3.3.2-py3-none-any.whl.metadata (5.8 kB) Collecting python-dateutil>=2.7 (from matplotlib->-r requirements.txt (line 3)) Using cached python_dateutil-2.9.0.post0-py2.py3-none-any.whl.metadata (8.4 kB) Collecting six>=1.5 (from python-dateutil>=2.7->matplotlib->-r requirements.txt (line 3)) Using cached six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB) Downloading numpy-2.4.2-cp311-cp311-win_amd64.whl (12.6 MB) _____

12.6/12.6 MB 242.6 kB/s eta 0:00:00 Downloading scipy-1.17.1-cp311-cp311-win_amd64.whl (36.6 MB) _____

36.6/36.6 MB 212.6 kB/s eta 0:00:00 Downloading matplotlib-3.10.8-cp311-cp311-win_amd64.whl (8.1 MB) _____ 3.4/8.1 MB

77.4 kB/s eta 0:01:02 ERROR: Exception: Traceback (most recent call last): File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-packages\pip_vendor\urllib3\response.py", line 438, in _error_catcher yield File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-packages\pip_vendor\urllib3\response.py", line 561, in read data = self._fp_read(amt) if not fp_closed else b"" ^^^^^^^^^^^^^^^^^^ File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-packages\pip_vendor\urllib3\response.py", line 527, in _fp_read return self._fp.read(amt) if amt is not None else self._fp.read() ^^^^^^^^^^^^^^^^^^ File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-packages\pip_vendor\cachecontrol\filewrapper.py", line 98, in read data: bytes = self._fp.read(amt) ^^^^^^^^^^^^^^^^^^ File "C:\Program Files\Python311\Lib\http\client.py", line 473, in read s = self.fp.read(amt) ^^^^^^^^^^^^^^^^^^ File "C:\Program Files\Python311\Lib\socket.py", line 706, in readinto return self._sock.recv_into(b) ^^^^^^^^^^^^^^^^^^ File "C:\Program Files\Python311\Lib\ssl.py", line 1314, in recv_into return self.read(nbytes, buffer) ^^^^^^^^^^^^^^^^^^ File "C:\Program Files\Python311\Lib\ssl.py", line 1166, in read return self._sslobj.read(len, buffer) ^^^^^^^^^^^^^^^^^^ TimeoutError: The read operation timed out During handling of the above exception, another exception occurred: Traceback (most recent call last): File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-packages\pip_internal\cli\base_command.py", line 180, in exc_logging_wrapper status = run_func(*args) ^^^^^^^^^^^^^^ File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-packages\pip_internal\cli\req_command.py", line 245, in wrapper return func(self, options, args) ^^^^^^^^^^^^^^ File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-packages\pip_internal\commands\install.py", line 377, in run requirement_set = resolver.resolve(^^^^^^^^^^^^^^ File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-packages\pip_internal\resolution\resolvelib\resolver.py", line 179, in resolve self.factory.preparer.prepare_linked_requirements_more(reqs) File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-packages\pip_internal\operations\prepare.py", line 552, in prepare_linked_requirements_more self._complete_partial_requirements(File "D:\PYTHON

```
DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-packages\pip\_internal\operations\prepare.py", line 467, in
complete_partial_requirements for link, (filepath, _) in batch_download: File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES
python\PYTHON\env\Lib\site-packages\pip\_internal\network\download.py", line 183, in __call__ for chunk in chunks: File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-packages\pip\_internal\cli\progress_bars.py", line 53, in _rich_progress_bar for chunk in
iterable: File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-packages\pip\_internal\network\utils.py", line 63, in
response_chunks for chunk in response.raw.stream( File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-
packages\pip\_vendor\urllib3\response.py", line 622, in stream data = self.read(amt=amt, decode_content=decode_content)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-
packages\pip\_vendor\urllib3\response.py", line 560, in read with self._error_catcher(): File "C:\Program Files\Python311\Lib\contextlib.py", line 158, in
__exit__ self.gen.throw(typ, value, traceback) File "D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-
packages\pip\_vendor\urllib3\response.py", line 443, in _error_catcher raise ReadTimeoutError(self._pool, None, "Read timed out.")
pip._vendor.urllib3.exceptions.ReadTimeoutError: HTTPSConnectionPool(host='files.pythonhosted.org', port=443): Read timed out. (env) PS D:\PYTHON
DJANGO HOPES\VALEURS PROPRES python\PYTHON>
```

Tu es tombé sur un **timeout réseau** pendant le téléchargement (ici `matplotlib`, mais ça peut arriver avec n'importe quel gros wheel). Ce n'est pas un problème de code : c'est la connexion ou le temps limite de pip.

Voici les solutions **les plus efficaces** (tu peux les essayer dans l'ordre).

1) Reprendre avec plus de temps + moins de bruit

Dans ton environnement `env` :

```
pip install -r requirements.txt --default-timeout=300 --retries 10
```

Si tu veux éviter que pip “abandonne” sur un paquet, tu peux aussi faire :

```
pip install matplotlib --default-timeout=300 --retries 10
```

2) Utiliser un miroir (souvent beaucoup plus rapide)

Exemple avec le miroir officiel “simple” (parfois ça aide selon les routes réseau) :

```
pip install -r requirements.txt -i https://pypi.org/simple --default-timeout=300 --retries 10
```

Ou tester un miroir proche (si tu en as un dans ton pays/entreprise).

3) Installer en plusieurs étapes (recommandé)

Comme numpy/scipy/matplotlib sont gros, fais-les un par un :

```
pip install --upgrade pip
pip install numpy --default-timeout=300 --retries 10
pip install scipy --default-timeout=300 --retries 10
pip install matplotlib --default-timeout=300 --retries 10
pip install -r requirements.txt --default-timeout=300 --retries 10
```

4) Si tu n'as pas besoin de matplotlib : enlève-le

Dans ton projet GUI, **matplotlib n'est pas obligatoire** (on ne l'utilise pas dans le code PyQt6 que je t'ai fourni).

Donc tu peux simplement **retirer la ligne matplotlib** de requirements.txt (ou commenter) et relancer :

```
pip install -r requirements.txt --default-timeout=300 --retries 10
```

Ça évite ~8–10 MB de download + dépendances.

5) Installer depuis un “wheelhouse” local (solution offline / réseau lent)

Si tu as un autre PC avec une bonne connexion :

Sur le PC “rapide” :

```
mkdir wheels
pip download -r requirements.txt -d wheels
```

Copie le dossier wheels/ sur ton PC lent puis :

```
pip install --no-index --find-links=wheels -r requirements.txt
```

6) Si tu es derrière un proxy / inspection SSL

(Fréquent en entreprise/université) : configure le proxy pip :

```
pip install -r requirements.txt --proxy http://user:pass@proxy:port --default-timeout=300 --retries 10
```

Ou mets-le dans pip.ini.

Ce que je te conseille pour ton cas précis

Vu ton log : le timeout arrive sur matplotlib. Donc le plus simple :

1. Supprimer matplotlib de requirements.txt (si tu ne fais pas de graphiques)
2. relancer avec timeout/retries.

Si tu me colles ton requirements.txt exact, je te le réécrit en version **minimum** (PyQt6 + numpy + scipy + reportlab + openpyxl + pyinstaller, avec ou sans API) pour réduire au maximum les téléchargements.

Import "matplotlib.pyplot" could not be resolved from sourcePylancereportMissingModuleSource

Ce message vient de **Pylance (VS Code)** : il ne trouve pas `matplotlib` dans l'interpréteur Python que VS Code utilise, ou bien `matplotlib` n'est pas installé (ton install a d'ailleurs timeout).

Voici la solution rapide.

1) Vérifie que tu es bien dans le bon environnement

Dans ton terminal (celui où tu codes), fais :

```
where python
python -m pip --version
python -c "import sys; print(sys.executable)"
```

Tu dois voir un chemin qui pointe vers ton ... \env\Scripts\python.exe.

2) Dans VS Code : sélectionne l'interpréteur `env`

1. Ctrl+Shift+P
2. **Python: Select Interpreter**
3. Choisis celui qui ressemble à :
... \VALEURS PROPRES python\PYTHON\env\Scripts\python.exe

Ensuite :

- Ctrl+Shift+P → **Developer: Reload Window** (ou redémarre VS Code)
-

3) Installe matplotlib (si tu en as vraiment besoin)

Comme ton téléchargement a timeout, fais :

```
pip install matplotlib --default-timeout=300 --retries 10
```

Puis teste :

```
python -c "import matplotlib.pyplot as plt; print('ok')"
```

4) Si tu n'en as pas besoin : enlève l'import

Dans les fichiers que je t'ai fournis pour l'interface PyQt6, **matplotlib n'est pas utilisé**.
Donc tu peux simplement :

- supprimer `import matplotlib.pyplot as plt`
 - et retirer `matplotlib` de `requirements.txt`
-

5) Si Pylance continue malgré l'installation

Parfois VS Code garde un cache :

- `Ctrl+Shift+P → Python: Restart Language Server`
 - ou redémarre VS Code
-

Si tu veux, colle ici :

1. ton `requirements.txt`
2. la sortie de `where python` et `python -c "import sys; print(sys.executable)"`

et je te dis exactement si VS Code pointe vers le mauvais venv ou si c'est juste matplotlib non installé.

`numpy` `scipy` `matplotlib`

OK — avec ce `requirements.txt` minimal :

```
numpy
scipy
matplotlib
```

le message Pylance “**Import matplotlib.pyplot could not be resolved**” arrive dans 2 cas :

1. `matplotlib` n'est pas installé (ton téléchargement a timeout)
2. VS Code utilise un autre Python que ton `env`

Voici la résolution la plus directe.

Étape A — Vérifier que VS Code pointe vers ton venv

Dans VS Code :

1. Ctrl+Shift+P
2. **Python: Select Interpreter**
3. Choisis : ...\\env\\Scripts\\python.exe

Ensuite, rouvre un terminal intégré dans VS Code et tape :

```
python -c "import sys; print(sys.executable)"
```

Le chemin doit contenir ...\\env\\Scripts\\python.exe.

Étape B — Installer matplotlib (avec timeout plus grand)

Toujours dans le venv activé ((env) visible) :

```
python -m pip install --upgrade pip
pip install matplotlib --default-timeout=300 --retries 10
```

Puis teste :

```
python -c "import matplotlib.pyplot as plt; print('matplotlib OK')"
```

Si ça timeout encore (solution très efficace)

Installe en séparant, avec plus de patience :

```
pip install numpy --default-timeout=300 --retries 10
pip install scipy --default-timeout=300 --retries 10
pip install matplotlib --default-timeout=300 --retries 10
```

Option “je veux juste que Pylance arrête l’erreur”

Si ton projet n'a pas besoin de graphiques, supprime `matplotlib` et l'import :

- enlève `matplotlib` de `requirements.txt`
 - supprime `import matplotlib.pyplot as plt`
-

Petit check final

Dis-moi ce que retourne :

```
where python
python -m pip show matplotlib
```

- Si `pip show matplotlib` ne renvoie rien → `matplotlib` pas installé.
- Si `where python` ne pointe pas vers ...\\env\\Scripts\\python.exe → mauvais interpréteur (cause #1 chez VS Code).

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. Name: Stix fonts Files: `matplotlib/mpl-data/fonts/ttf/STIX*.ttf` Description: STIX fonts License: TERMS AND CONDITIONS 1. Permission is hereby granted, free of charge, to any person obtaining a copy of the STIX Fonts-TM set accompanying this license (collectively, the "Fonts") and the associated documentation files (collectively with the Fonts, the "Font Software"), to reproduce and distribute the Font Software, including the rights to use, copy, merge and publish copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so same, subject to the following terms and conditions (the "License"). 2. The following copyright and trademark notice and these Terms and Conditions shall be included in all copies of one or more of the Font typefaces and any derivative work created as permitted under this License: Copyright (c) 2001-2005 by the STI Pub Companies, consisting of the American Institute of Physics, the American Chemical Society, the American Mathematical Society, the American Physical Society, Elsevier, Inc., and The Institute of Electrical and Electronic Engineers, Inc. Portions copyright (c) 1998-2003 by MicroPress, Inc. Portions copyright (c) 1990 by Elsevier, Inc. All rights reserved. STIX Fonts-TM is a trademark of The Institute of Electrical and Electronics Engineers, Inc. 3. You may (a) convert the Fonts from one format to another (e.g., from TrueType to PostScript), in which case the normal and reasonable distortion that occurs during such conversion shall be permitted and (b) embed or include a subset of the Fonts in a document for the purposes of allowing users to read text in the document that utilizes the Fonts. In each case, you may use the STIX Fonts-TM mark to designate the resulting Fonts or subset of the Fonts. 4. You may also (a) add glyphs or characters to the Fonts, or modify the shape of existing glyphs, so long as the base set of glyphs is not removed and (b) delete glyphs or characters from the Fonts, provided that the resulting font set is distributed with the following disclaimer: "This [name] font does not include all the Unicode points covered in the STIX Fonts-TM set but may include others." In each case, the name used to denote the resulting font set shall not include the term "STIX" or any similar term. 5. You may charge a fee in connection with the distribution of the Font Software, provided that no copy of one or more of the individual Font typefaces that

form the STIX Fonts-TM set may be sold by itself.

6. THE FONT SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK OR OTHER RIGHT. IN NO EVENT SHALL MICROPRESS OR ANY OF THE STI PUB COMPANIES BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING, BUT NOT LIMITED TO, ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM OR OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

7. Except as contained in the notice set forth in Section 2, the names MicroPress Inc. and STI Pub Companies, as well as the names of the companies/organizations that compose the STI Pub Companies, shall not be used in advertising or otherwise to promote the sale, use or other dealings in the Font Software without the prior written consent of the respective company or organization.

8. This License shall become null and void in the event of any material breach of the Terms and Conditions herein by licensee.

9. A substantial portion of the STIX Fonts set was developed by MicroPress Inc. for the STI Pub Companies. To obtain additional mathematical fonts, please contact MicroPress, Inc., 68-30 Harrow Street, Forest Hills, NY 11375, USA - Phone: (718) 575-1816.

Name: Yorick Colormaps
Files: lib/matplotlib/_cm.py
Description: Gist/Yorick colormaps
License: BSD-style license for gist/yorick colormaps.
Copyright: Copyright (c) 1996. The Regents of the University of California. All rights reserved. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. This work was produced at the University of California, Lawrence Livermore National Laboratory under contract no. W-7405-ENG-48 between the U.S. Department of Energy and The Regents of the University of California for the operation of UC LLNL.

DISCLAIMER This software was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately-owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

AUTHOR David H. Munro wrote Yorick and Gist. Berkeley Yacc (byacc) generated the Yorick parser. The routines in Math are from LAPACK and FFTPACK; MathC contains C translations by David H. Munro. The algorithms for Yorick's random number generator and several special functions in Yorick/include were taken from Numerical Recipes by Press, et. al., although the Yorick implementations are unrelated to those in Numerical Recipes. A small amount of code in Gist was adapted from the X11R4 release, copyright M.I.T. -- the complete copyright notice may be found in the (unused) file Gist/host.c.

Location: D:\PYTHON DJANGO HOPES\VALEURS PROPRES

```
python\ PYTHON\ env\ Lib\ site-packages Requirements: contourpy, cycler, fonttools, kiwisolver, numpy, packaging, pillow, pyparsing, python-dateutil Required-by: (env) PS D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\ PYTHON> pip install -r requirements.txt Requirement already satisfied: numpy in d:\python django hopes\valeurs propres python\python\env\lib\site-packages (from -r requirements.txt (line 1)) (2.4.2) Requirement already satisfied: scipy in d:\python django hopes\valeurs propres python\python\env\lib\site-packages (from -r requirements.txt (line 2)) (1.17.1) Requirement already satisfied: matplotlib in d:\python django hopes\valeurs propres python\python\env\lib\site-packages (from -r requirements.txt (line 3)) (3.10.8) Requirement already satisfied: contourpy>=1.0.1 in d:\python django hopes\valeurs propres python\python\env\lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (1.3.3) Requirement already satisfied: cycler>=0.10 in d:\python django hopes\valeurs propres python\python\env\lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (0.12.1) Requirement already satisfied: fonttools>=4.22.0 in d:\python django hopes\valeurs propres python\python\env\lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (4.61.1) Requirement already satisfied: kiwisolver>=1.3.1 in d:\python django hopes\valeurs propres python\python\env\lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (1.4.9) Requirement already satisfied: packaging>=20.0 in d:\python django
```

hopes\valeurs propres python\python\env\lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (26.0) Requirement already satisfied: pillow>=8 in d:\python django hopes\valeurs propres python\python\env\lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (12.1.1) Requirement already satisfied: pyparsing>=3 in d:\python django hopes\valeurs propres python\python\env\lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (3.3.2) Requirement already satisfied: python-dateutil>=2.7 in d:\python django hopes\valeurs propres python\python\env\lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (2.9.0.post0) 0 [notice] A new release of pip is available: 24.0 -> 26.0.1 [notice] To update, run: python.exe -m pip install --upgrade pip (env) PS D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON> python.exe -m pip install --upgrade pip Requirement already satisfied: pip in d:\python django hopes\valeurs propres python\python\env\lib\site-packages (24.0) Collecting pip Using cached pip-26.0.1-py3-none-any.whl.metadata (4.7 kB) Using cached pip-26.0.1-py3-none-any.whl (1.8 MB) Installing collected packages: pip Attempting uninstall: pip Found existing installation: pip 24.0 Uninstalling pip-24.0: Successfully uninstalled pip-24.0 Successfully installed pip-26.0.1 (env) PS D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON> pip install -r requirements.txt Requirement already satisfied: numpy in .\env\Lib\site-packages (from -r requirements.txt (line 1)) (2.4.2) Requirement already satisfied: scipy in .\env\Lib\site-packages (from -r requirements.txt (line 2)) (1.17.1) Requirement already satisfied: matplotlib in .\env\Lib\site-packages (from -r requirements.txt (line 3)) (3.10.8) Requirement already satisfied: contourpy>=1.0.1 in .\env\Lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (1.3.3) Requirement already satisfied: cycler>=0.10 in .\env\Lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (0.12.1) Requirement already satisfied: fonttools>=4.22.0 in .\env\Lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (4.61.1) Requirement already satisfied: kiwisolver>=1.3.1 in .\env\Lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (1.4.9) Requirement already satisfied: packaging>=20.0 in .\env\Lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (26.0) Requirement already satisfied: pillow>=8 in .\env\Lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (12.1.1) Requirement already satisfied: python-dateutil>=2.7 in .\env\Lib\site-packages (from matplotlib->-r requirements.txt (line 3)) (2.9.0.post0) Requirement already satisfied: six>=1.5 in .\env\Lib\site-packages (from python-dateutil>=2.7->matplotlib->-r requirements.txt (line 3)) (1.17.0) (env) PS D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON> python -m pip show matplotlib Name: matplotlib Version: 3.10.8 Summary: Python plotting package Home-page: <https://matplotlib.org> Author: John D. Hunter, Michael Droettboom Author-email: Unknown <matplotlib-users@python.org> License: License agreement for matplotlib versions 1.3.0 and later

===== 1. This LICENSE AGREEMENT is between the Matplotlib Development Team ("MDT"), and the Individual or Organization ("Licensee") accessing and otherwise using matplotlib software in source or binary form and its associated documentation. 2. Subject to the terms and conditions of this License Agreement, MDT hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use matplotlib alone or in any derivative version, provided, however, that MDT's License Agreement and MDT's notice of copyright, i.e., "Copyright (c) 2012- Matplotlib Development Team; All Rights Reserved" are retained in matplotlib alone or in any derivative version prepared by Licensee. 3. In the event Licensee prepares a derivative work that is based on or incorporates matplotlib or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to matplotlib . 4. MDT is making matplotlib available to Licensee on an "AS IS" basis. MDT MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, MDT MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF MATPLOTLIB WILL NOT INFRINGE ANY THIRD PARTY RIGHTS. 5. MDT SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF MATPLOTLIB FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING MATPLOTLIB , OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. 6. This License Agreement will automatically terminate upon a material breach of its terms and conditions. 7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between MDT and Licensee. This License Agreement

does not grant permission to use MDT trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party. 8. By copying, installing or otherwise using matplotlib , Licensee agrees to be bound by the terms and conditions of this License Agreement. License agreement for matplotlib versions prior to 1.3.0 ===== 1. This LICENSE AGREEMENT is between John D. Hunter ("JDH"), and the Individual or Organization ("Licensee") accessing and otherwise using matplotlib software in source or binary form and its associated documentation. 2. Subject to the terms and conditions of this License Agreement, JDH hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use matplotlib alone or in any derivative version, provided, however, that JDH's License Agreement and JDH's notice of copyright, i.e., "Copyright (c) 2002-2011 John D. Hunter; All Rights Reserved" are retained in matplotlib alone or in any derivative version prepared by Licensee. 3. In the event Licensee prepares a derivative work that is based on or incorporates matplotlib or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to matplotlib. 4. JDH is making matplotlib available to Licensee on an "AS IS" basis. JDH MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, JDH MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF MATPLOTLIB WILL NOT INFRINGE ANY THIRD PARTY RIGHTS. 5. JDH SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF MATPLOTLIB FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING MATPLOTLIB , OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. 6. This License Agreement will automatically terminate upon a material breach of its terms and conditions. 7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between JDH and Licensee. This License Agreement does not grant permission to use JDH trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party. 8. By copying, installing or otherwise using matplotlib, Licensee agrees to be bound by the terms and conditions of this License Agreement. ---- This binary distribution of Matplotlib can also bundle the following software (depending on the build): Name: AMS Fonts Files: matplotlib/tests/cmr10.pfb Description: Type-1 version of one of Knuth's Computer Modern fonts License: OFL-1.1 The cmr10.pfb file is a Type-1 version of one of Knuth's Computer Modern fonts. It is included here as test data only, but the following license applies. Copyright (c) 1997, 2009, American Mathematical Society (<http://www.ams.org>). All Rights Reserved. "cmb10" is a Reserved Font Name for this Font Software. "cmbsy10" is a Reserved Font Name for this Font Software. "cmbsy5" is a Reserved Font Name for this Font Software. "cmbsy6" is a Reserved Font Name for this Font Software. "cmbsy7" is a Reserved Font Name for this Font Software. "cmbsy8" is a Reserved Font Name for this Font Software. "cmbsy9" is a Reserved Font Name for this Font Software. "cmbx10" is a Reserved Font Name for this Font Software. "cmbx12" is a Reserved Font Name for this Font Software. "cmbx5" is a Reserved Font Name for this Font Software. "cmbx6" is a Reserved Font Name for this Font Software. "cmbx7" is a Reserved Font Name for this Font Software. "cmbx8" is a Reserved Font Name for this Font Software. "cmbx9" is a Reserved Font Name for this Font Software. "cmbxsl10" is a Reserved Font Name for this Font Software. "cmbxti10" is a Reserved Font Name for this Font Software. "cmcsc10" is a Reserved Font Name for this Font Software. "cmcsc8" is a Reserved Font Name for this Font Software. "cmcsc9" is a Reserved Font Name for this Font Software. "cmdunh10" is a Reserved Font Name for this Font Software. "cmex10" is a Reserved Font Name for this Font Software. "cmex7" is a Reserved Font Name for this Font Software. "cmex8" is a Reserved Font Name for this Font Software. "cmex9" is a Reserved Font Name for this Font Software. "cmff10" is a Reserved Font Name for this Font Software. "cmfi10" is a Reserved Font Name for this Font Software. "cmfib8" is a Reserved Font Name for this Font Software. "cminch" is a Reserved Font Name for this Font Software. "cmitt10" is a Reserved Font Name for this Font Software. "cmmi10" is a Reserved Font Name for this Font Software. "cmmi12" is a Reserved Font Name for this Font Software. "cmmi5" is a Reserved Font Name for this Font Software. "cmmi6" is a Reserved Font Name for this Font Software. "cmmi7" is a Reserved Font Name for this Font Software. "cmmi8" is a Reserved Font Name for this Font Software. "cmmi9" is a Reserved Font Name for this Font Software. "cmmib10" is a Reserved Font Name for this Font Software. "cmmib5" is a Reserved Font Name for this Font Software. "cmmib6" is a Reserved Font Name for this Font Software.

is a Reserved Font Name for this Font Software. "wncyi10" is a Reserved Font Name for this Font Software. "wncyr10" is a Reserved Font Name for this Font Software. "wncysc10" is a Reserved Font Name for this Font Software. "wncyss10" is a Reserved Font Name for this Font Software. This Font Software is licensed under the SIL Open Font License, Version 1.1. This license is copied below, and is also available with a FAQ at: <http://scripts.sil.org/OFL>

SIL OPEN FONT LICENSE Version 1.1 - 26 February 2007

PREAMBLE The goals of the Open Font License (OFL) are to stimulate worldwide development of collaborative font projects, to support the font creation efforts of academic and linguistic communities, and to provide a free and open framework in which fonts may be shared and improved in partnership with others. The OFL allows the licensed fonts to be used, studied, modified and redistributed freely as long as they are not sold by themselves. The fonts, including any derivative works, can be bundled, embedded, redistributed and/or sold with any software provided that any reserved names are not used by derivative works. The fonts and derivatives, however, cannot be released under any other type of license. The requirement for fonts to remain under this license does not apply to any document created using the fonts or their derivatives.

DEFINITIONS "Font Software" refers to the set of files released by the Copyright Holder(s) under this license and clearly marked as such. This may include source files, build scripts and documentation. "Reserved Font Name" refers to any names specified as such after the copyright statement(s). "Original Version" refers to the collection of Font Software components as distributed by the Copyright Holder(s). "Modified Version" refers to any derivative made by adding to, deleting, or substituting -- in part or in whole -- any of the components of the Original Version, by changing formats or by porting the Font Software to a new environment. "Author" refers to any designer, engineer, programmer, technical writer or other person who contributed to the Font Software.

PERMISSION & CONDITIONS Permission is hereby granted, free of charge, to any person obtaining a copy of the Font Software, to use, study, copy, merge, embed, modify, redistribute, and sell modified and unmodified copies of the Font Software, subject to the following conditions:

- 1) Neither the Font Software nor any of its individual components, in Original or Modified Versions, may be sold by itself.
- 2) Original or Modified Versions of the Font Software may be bundled, redistributed and/or sold with any software, provided that each copy contains the above copyright notice and this license. These can be included either as stand-alone text files, human-readable headers or in the appropriate machine-readable metadata fields within text or binary files as long as those fields can be easily viewed by the user.
- 3) No Modified Version of the Font Software may use the Reserved Font Name(s) unless explicit written permission is granted by the corresponding Copyright Holder. This restriction only applies to the primary font name as presented to the users.
- 4) The name(s) of the Copyright Holder(s) or the Author(s) of the Font Software shall not be used to promote, endorse or advertise any Modified Version, except to acknowledge the contribution(s) of the Copyright Holder(s) and the Author(s) or with their explicit written permission.
- 5) The Font Software, modified or unmodified, in part or in whole, must be distributed entirely under this license, and must not be distributed under any other license.

The requirement for fonts to remain under this license does not apply to any document created using the Font Software.

TERMINATION This license becomes null and void if any of the above conditions are not met.

DISCLAIMER THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL THE COPYRIGHT HOLDER BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Name: BaKoMa Fonts
Files: matplotlib/mpl-data/fonts/ttf/cm*.ttf matplotlib/mpl-data/fonts/afm/cm*.afm
Description: Computer Modern Fonts in PostScript Type 1 and TrueType font formats.
License: BaKoMa Fonts Licence
BaKoMa Fonts Licence -----
This licence covers two font packs (known as BaKoMa Fonts Collection, which is available at CTAN: fonts/cm/ps-type1/bakoma/) :
1) BaKoMa-CM (1.1/12-Nov-94) Computer Modern Fonts in PostScript Type 1 and TrueType font formats.
2) BaKoMa-AMS (1.2/19-Jan-95) AMS TeX fonts in PostScript Type 1 and TrueType font formats.
Copyright (C) 1994, 1995, Basil K. Malyshev. All Rights Reserved. Permission to copy and distribute these fonts for any purpose is hereby granted without fee, provided that the above copyright notice, author statement and this permission notice appear in all copies of these fonts and related documentation. Permission to modify

and distribute modified fonts for any purpose is hereby granted without fee, provided that the copyright notice, author statement, this permission notice and location of original fonts (<http://www.ctan.org/tex-archive/fonts/cm/ps-type1/bakoma>) appear in all copies of modified fonts and related documentation. Permission to use these fonts (embedding into PostScript, PDF, SVG and printing by using any software) is hereby granted without fee. It is not required to provide any notices about using these fonts. Basil K. Malyshev INSTITUTE FOR HIGH ENERGY PHYSICS IHEP, OMVT Moscow Region 142281 PROTVINO RUSSIA E-Mail: bakoma@mail.ru or malyshev@mail.ihep.ru Name: ColorBrewer Color Schemes Files: lib/matplotlib/_cm.py Description: Color schemes from ColorBrewer License: Apache-2.0 Apache-Style Software License for ColorBrewer software and ColorBrewer Color Schemes Copyright (c) 2002 Cynthia Brewer, Mark Harrower, and The Pennsylvania State University. Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0> Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Name: Courier 10 Files: matplotlib/tests/Courier10PitchBT-Bold.pfb Description: Courier 10 font, used in tests. License: Bitstream-Charter The Courier10PitchBT-Bold.pfb file is a Type-1 version of Courier 10 Pitch BT Bold by Bitstream, obtained from <<https://ctan.org/tex-archive/fonts/courierten>>. It is included here as test data only, but the following license applies. (c) Copyright 1989-1992, Bitstream Inc., Cambridge, MA. You are hereby granted permission under all Bitstream propriety rights to use, copy, modify, sublicense, sell, and redistribute the 4 Bitstream Charter (r) Type 1 outline fonts and the 4 Courier Type 1 outline fonts for any purpose and without restriction; provided, that this notice is left intact on all copies of such fonts and that Bitstream's trademark is acknowledged as shown below on all unmodified copies of the 4 Charter Type 1 fonts. BITSTREAM CHARTER is a registered trademark of Bitstream Inc. Name: JSXTools resize observer Files: Description: Minimal polyfill for the ResizeObserver API License: CC0-1.0 # CC0 1.0 Universal ## Statement of Purpose The laws of most jurisdictions throughout the world automatically confer exclusive Copyright and Related Rights (defined below) upon the creator and subsequent owner(s) (each and all, an "owner") of an original work of authorship and/or a database (each, a "Work"). Certain owners wish to permanently relinquish those rights to a Work for the purpose of contributing to a commons of creative, cultural and scientific works ("Commons") that the public can reliably and without fear of later claims of infringement build upon, modify, incorporate in other works, reuse and redistribute as freely as possible in any form whatsoever and for any purposes, including without limitation commercial purposes. These owners may contribute to the Commons to promote the ideal of a free culture and the further production of creative, cultural and scientific works, or to gain reputation or greater distribution for their Work in part through the use and efforts of others. For these and/or other purposes and motivations, and without any expectation of additional consideration or compensation, the person associating CC0 with a Work (the "Affirmer"), to the extent that he or she is an owner of Copyright and Related Rights in the Work, voluntarily elects to apply CC0 to the Work and publicly distribute the Work under its terms, with knowledge of his or her Copyright and Related Rights in the Work and the meaning and intended legal effect of CC0 on those rights. 1. Copyright and Related Rights. A Work made available under CC0 may be protected by copyright and related or neighboring rights ("Copyright and Related Rights"). Copyright and Related Rights include, but are not limited to, the following: 1. the right to reproduce, adapt, distribute, perform, display, communicate, and translate a Work; 2. moral rights retained by the original author(s) and/or performer(s); 3. publicity and privacy rights pertaining to a person's image or likeness depicted in a Work; 4. rights protecting against unfair competition in regards to a Work, subject to the limitations in paragraph 4(i), below; 5. rights protecting the extraction, dissemination, use and reuse of data in a Work; 6. database rights (such as those arising under Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, and under any national implementation thereof, including any amended or successor version of such directive); and 7. other similar, equivalent or corresponding rights throughout the world based on applicable law or treaty, and any national implementations thereof. 2. Waiver. To the greatest extent permitted by, but not in contravention of, applicable law, Affirmer hereby overtly, fully, permanently, irrevocably and unconditionally waives, abandons, and surrenders all of Affirmer's Copyright and Related Rights and associated claims and causes of action, whether now known or unknown (including existing as well as future claims and causes of action), in the Work (i) in all territories worldwide, (ii) for the maximum duration provided by applicable law or treaty

(including future time extensions), (iii) in any current or future medium and for any number of copies, and (iv) for any purpose whatsoever, including without limitation commercial, advertising or promotional purposes (the "Waiver"). Affirmer makes the Waiver for the benefit of each member of the public at large and to the detriment of Affirmer's heirs and successors, fully intending that such Waiver shall not be subject to revocation, rescission, cancellation, termination, or any other legal or equitable action to disrupt the quiet enjoyment of the Work by the public as contemplated by Affirmer's express Statement of Purpose. 3. Public License Fallback. Should any part of the Waiver for any reason be judged legally invalid or ineffective under applicable law, then the Waiver shall be preserved to the maximum extent permitted taking into account Affirmer's express Statement of Purpose. In addition, to the extent the Waiver is so judged Affirmer hereby grants to each affected person a royalty-free, non transferable, non sublicensable, non exclusive, irrevocable and unconditional license to exercise Affirmer's Copyright and Related Rights in the Work (i) in all territories worldwide, (ii) for the maximum duration provided by applicable law or treaty (including future time extensions), (iii) in any current or future medium and for any number of copies, and (iv) for any purpose whatsoever, including without limitation commercial, advertising or promotional purposes (the "License"). The License shall be deemed effective as of the date CC0 was applied by Affirmer to the Work. Should any part of the License for any reason be judged legally invalid or ineffective under applicable law, such partial invalidity or ineffectiveness shall not invalidate the remainder of the License, and in such case Affirmer hereby affirms that he or she will not (i) exercise any of his or her remaining Copyright and Related Rights in the Work or (ii) assert any associated claims and causes of action with respect to the Work, in either case contrary to Affirmer's express Statement of Purpose. 4. Limitations and Disclaimers. 1. No trademark or patent rights held by Affirmer are waived, abandoned, surrendered, licensed or otherwise affected by this document. 2. Affirmer offers the Work as-is and makes no representations or warranties of any kind concerning the Work, express, implied, statutory or otherwise, including without limitation warranties of title, merchantability, fitness for a particular purpose, non infringement, or the absence of latent or other defects, accuracy, or the present or absence of errors, whether or not discoverable, all to the greatest extent permissible under applicable law. 3. Affirmer disclaims responsibility for clearing rights of other persons that may apply to the Work or any use thereof, including without limitation any person's Copyright and Related Rights in the Work. Further, Affirmer disclaims responsibility for obtaining any necessary consents, permissions or other rights required for any use of the Work. 4. Affirmer understands and acknowledges that Creative Commons is not a party to this document and has no duty or obligation with respect to this CC0 or use of the Work. For more information, please see

<http://creativecommons.org/publicdomain/zero/1.0/>. Name: QHull Files: matplotlib/_qhull.*.so Description: Convex hull, Delaunay triangulation, Voronoi diagrams, Halfspace intersection License: Qhull Qhull, Copyright (c) 1993-2020 C.B. Barber Arlington, MA and The National Science and Technology Research Center for Computation and Visualization of Geometric Structures (The Geometry Center) University of Minnesota email: qhull@qhull.org This software includes Qhull from C.B. Barber and The Geometry Center. Files derived from Qhull 1.0 are copyrighted by the Geometry Center. The remaining files are copyrighted by C.B. Barber. Qhull is free software and may be obtained via http from www.qhull.org. It may be freely copied, modified, and redistributed under the following conditions: 1. All copyright notices must remain intact in all files. 2. A copy of this text file must be distributed along with any copies of Qhull that you redistribute; this includes copies that you have modified, or copies of programs or other software products that include Qhull. 3. If you modify Qhull, you must include a notice giving the name of the person performing the modification, the date of modification, and the reason for such modification. 4. When distributing modified versions of Qhull, or other software products that include Qhull, you must provide notice that the original source code may be obtained as noted above. 5. There is no warranty or other guarantee of fitness for Qhull, it is provided solely "as is". Bug reports or fixes may be sent to qhull_bug@qhull.org; the authors may or may not act on them as they desire. Name: Qt4 Editor Files: matplotlib/backends/qt_editor Description: Module creating PyQt4 form dialogs/layouts to edit various type of parameters License: MIT Module creating PyQt4 form dialogs/layouts to edit various type of parameters formlayout License Agreement (MIT License) ----- Copyright (c) 2009 Pierre Raybaut Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. """ Name: Solarized Files: matplotlib/mpl-data/stylelib/Solarize_Light2.mplstyle Description: Solarized color scheme/style License: MIT <https://github.com/altercation/solarized/blob/master/LICENSE> Copyright (c) 2011 Ethan Schoonover Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software. THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. Name: Stix fonts Files: matplotlib/mpl-data/fonts/ttf/STIX*.ttf Description: STIX fonts License: TERMS AND CONDITIONS 1. Permission is hereby granted, free of charge, to any person obtaining a copy of the STIX Fonts-TM set accompanying this license (collectively, the "Fonts") and the associated documentation files (collectively with the Fonts, the "Font Software"), to reproduce and distribute the Font Software, including the rights to use, copy, merge and publish copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so same, subject to the following terms and conditions (the "License"). 2. The following copyright and trademark notice and these Terms and Conditions shall be included in all copies of one or more of the Font typefaces and any derivative work created as permitted under this License: Copyright (c) 2001-2005 by the STI Pub Companies, consisting of the American Institute of Physics, the American Chemical Society, the American Mathematical Society, the American Physical Society, Elsevier, Inc., and The Institute of Electrical and Electronic Engineers, Inc. Portions copyright (c) 1998-2003 by MicroPress, Inc. Portions copyright (c) 1990 by Elsevier, Inc. All rights reserved. STIX Fonts-TM is a trademark of The Institute of Electrical and Electronics Engineers, Inc. 3. You may (a) convert the Fonts from one format to another (e.g., from TrueType to PostScript), in which case the normal and reasonable distortion that occurs during such conversion shall be permitted and (b) embed or include a subset of the Fonts in a document for the purposes of allowing users to read text in the document that utilizes the Fonts. In each case, you may use the STIX Fonts-TM mark to designate the resulting Fonts or subset of the Fonts. 4. You may also (a) add glyphs or characters to the Fonts, or modify the shape of existing glyphs, so long as the base set of glyphs is not removed and (b) delete glyphs or characters from the Fonts, provided that the resulting font set is distributed with the following disclaimer: "This [name] font does not include all the Unicode points covered in the STIX Fonts-TM set but may include others." In each case, the name used to denote the resulting font set shall not include the term "STIX" or any similar term. 5. You may charge a fee in connection with the distribution of the Font Software, provided that no copy of one or more of the individual Font typefaces that form the STIX Fonts-TM set may be sold by itself. 6. THE FONT SOFTWARE IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK OR OTHER RIGHT. IN NO EVENT SHALL MICROPRESS OR ANY OF THE STI PUB COMPANIES BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING, BUT NOT LIMITED TO, ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM OR OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE. 7. Except as contained in the notice set forth in Section 2, the names MicroPress Inc. and STI Pub Companies, as well as the names of the companies/organizations that compose the STI Pub Companies, shall not be used in

advertising or otherwise to promote the sale, use or other dealings in the Font Software without the prior written consent of the respective company or organization. 8. This License shall become null and void in the event of any material breach of the Terms and Conditions herein by licensee. 9. A substantial portion of the STIX Fonts set was developed by MicroPress Inc. for the STI Pub Companies. To obtain additional mathematical fonts, please contact MicroPress, Inc., 68-30 Harrow Street, Forest Hills, NY 11375, USA - Phone: (718) 575-1816. Name: Yorick Colormaps Files: lib/matplotlib/_cm.py Description: Gist/Yorick colormaps License: BSD-style license for gist/yorick colormaps. Copyright: Copyright (c) 1996. The Regents of the University of California. All rights reserved. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. This work was produced at the University of California, Lawrence Livermore National Laboratory under contract no. W-7405-ENG-48 between the U.S. Department of Energy and The Regents of the University of California for the operation of UC LLNL. DISCLAIMER This software was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately-owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes. AUTHOR David H. Munro wrote Yorick and Gist. Berkeley Yacc (byacc) generated the Yorick parser. The routines in Math are from LAPACK and FFTPACK; MathC contains C translations by David H. Munro. The algorithms for Yorick's random number generator and several special functions in Yorick/include were taken from Numerical Recipes by Press, et. al., although the Yorick implementations are unrelated to those in Numerical Recipes. A small amount of code in Gist was adapted from the X11R4 release, copyright M.I.T. -- the complete copyright notice may be found in the (unused) file Gist/host.c. Location: D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON\env\Lib\site-packages Requires: contourpy, cycler, fonttools, kiwisolver, numpy, packaging, pillow, pyparsing, python-dateutil Required-by: (env) PS D:\PYTHON DJANGO HOPES\VALEURS PROPRES python\PYTHON>

Parfait — ton log montre que **matplotlib est bien installé** dans ton venv :

- Location: ... \env\lib\site-packages
- Requirement already satisfied: matplotlib ... (3.10.8)

Donc si VS Code/Pylance affiche encore :

```
Import "matplotlib.pyplot" could not be resolved
```

le problème n'est **plus** l'installation : c'est **l'interpréteur Python sélectionné dans VS Code** (ou le serveur Pylance qui n'a pas rechargé).

1) Pourquoi `where python` ne répond rien ?

Sous **PowerShell**, `where` n'est pas toujours fiable comme sous cmd (c'est un alias vers autre chose). Utilise plutôt :

```
where.exe python
python -c "import sys; print(sys.executable)"
```

Le 2e doit afficher un chemin du type :

```
...\\env\\Scripts\\python.exe
```

Correctif (2 minutes) : VS Code + Pylance

A) Sélectionner le bon interpréteur

Dans VS Code :

1. Ctrl+Shift+P
2. **Python: Select Interpreter**
3. Choisis celui dans ton dossier env :
D:\\PYTHON DJANGO HOPES\\VALEURS PROPRES python\\PYTHON\\env\\Scripts\\python.exe

B) Redémarrer Pylance

Toujours dans VS Code :

- Ctrl+Shift+P → **Python: Restart Language Server**
ou
 - Ctrl+Shift+P → **Developer: Reload Window**
-

Test immédiat (dans le terminal intégré VS Code)

Avec le venv activé :

```
python -c "import matplotlib.pyplot as plt; print('OK matplotlib')"
```

Si ça affiche OK matplotlib, Python est OK et Pylance finira par se synchroniser.

Si PyLance continue malgré tout (solution radicale)

1. Ferme VS Code
 2. Supprime le cache Python (optionnel mais efficace) :
 - supprime le dossier `.vscode/` du projet si tu en as un
 3. Rouvre VS Code, re-sélectionne l'interpréteur `env`
-

Dernier point : tu n'as pas besoin de matplotlib pour EigenGUI

Si ton projet actuel est **l'interface PyQt6** (calcul valeurs/vecteurs propres + exports), **matplotlib n'est pas utilisé**.

Donc tu peux aussi enlever complètement `matplotlib` de `requirements.txt` et supprimer tout import `matplotlib` dans ton code (ça simplifie).

Si tu me dis :

- tu utilises **VS Code** ou un autre IDE ?
- et colle la sortie de :

```
python -c "import sys; print(sys.executable)"
```

je te dis exactement si VS Code pointe au bon endroit.