

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) :

$A = \begin{pmatrix} 4 & 1 & 1 \\ 1 & 3 & 0 \\ 1 & 0 & 2 \end{pmatrix}$

```
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 :  $A v \approx \lambda v$  pour chaque paire  $(\lambda, v)$ 
print("\nVérification  $A @ V - V @ \text{diag}(w)$  (doit être proche de 0) :")
print(A @ V - V @ np.diag(w))
```

```

# Vérification orthonormalité :  $V^T V \approx I$ 
print("\n $V^T V$  (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 :

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

- 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 $\pi \backslash \pi$ 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 :

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

ou en complexe $P = QTQ^*$ $P = Q T Q^{\top} P = QTQ^*$, 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:
        w, V = np.linalg.eig(A) # valeurs propres complexes et V orthonormale
        method = "eig (Général)"
```

```

else:
    if prefer_schur:
        try:
            from scipy.linalg import schur, rsf2csf
            # Schur réel/complexe ; en non-symétrique, Schur est une base robuste
            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  $A \cdot V \approx V \cdot \text{diag}(w)$ .
    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 $n \times n$.
- **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 $n \times nn$, 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`
- 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 ($V^*V \approx I$)

-
- affiche l'erreur de reconstruction $\| \mathbf{A}\mathbf{V} - \mathbf{V}\text{diag}(\mathbf{w}) \|$ (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 $\mathbf{BV} \approx \mathbf{V}\Lambda\mathbf{B} \approx \mathbf{V}\Lambda\mathbf{B}$

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 $O(n^3)O(n^3)O(n^3)$ (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 **nexistent 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 $\|AV - V\Lambda\|/\|AV\|$ 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)
    except:
        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 :

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

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 :

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

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

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

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

👉 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 :

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

- 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^kAk , $eAt^k\{At\}eAt$, 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)
 - 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
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))

    except Exception as e:
        self.txtEigenvalues.setPlainText(str(e))

```

```
        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

- o warning si matrice défectueuse / proche défectueuse
 - o 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 QPushButtons :

- 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 :\n{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])

# 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.complexfloating, 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\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* ; 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 :{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 "
              "'{\"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 \" \
          \"'{\"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 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 $n \geq 120$ et $n \geq 120$. 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>
</widget>

</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 fournisse 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  $\text{diag}(T)$ 
      puis vecteurs propres via  $\text{eig}(T)$  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.btnExitPDF.clicked.connect(self.export_pdf)
        self.btnExitExcel.clicked.connect(self.export_excel)
        self.btnExitJSON.clicked.connect(self.export_json)
```

```

# Onglet API
self.btnExit.clicked.connect(self.start_api)
self.btnExit.clicked.connect(self.stop_api)
self.btnExit.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.getText(
        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']:.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"]
    ))
    story.append(Spacer(1, 10))

    if "schur_T" in res:
        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]]}\"')
    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 **depuis 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  
```bat  
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 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
        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.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()
            QtWidgets.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"]
))
story.appendSpacer(1, 8)

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]]}}'"
    )
    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.