

Rapport final - Projet ALN

Mathis Verstrepen Matthieu Fontaine

25 Mai 2023

1 Introduction

Toutes les fonctions nécessaires au calcul de l'ACP par la méthode historique et par la méthode moderne ont été implémentées.

Chaque fonction possède 3 tests unitaires différents.

Chaque méthode de calcul de l'ACP possède un test de bout en bout.

Chaque fonction est typée et documentée selon la convention de la docstring Python.

2 ACP méthode historique

2.1 Hessenberg

Nous avons programmé et testé notre version de la décomposition de Hessenberg.

Cette fonction nous permet d'obtenir la matrice tridiagonale nécessaire pour calculer l'ACP par la méthode historique.

Le paramètre *calc_q* a été implémenté.

Fichier : `fonctions/eighvals/hessenberg.py`.

Commande de test : `python3 test_unitaire.py T1_Hessenberg`.

```
# Compute the Hessenberg decomposition of a square matrix

# Args:
# A (np.array): Matrix to decompose (must be square).
# calc_q (bool, optional): Whether to compute the orthogonal matrix Q. Defaults to False.

# Returns:
# np.array: Hessenberg matrix H.
# np.array: Orthogonal matrix Q. Only returned if calc_q is True.

def hessenberg(A: np.array, calc_q: bool = False) -> tuple[np.array, np.array]:
```

2.2 Gershgorin

Nous avons programmé et testé notre version du calcul de l'intervalle des valeurs propres d'une matrice tridiagonale par la méthode des disques de Gershgorin.

Cette fonction nous permet d'obtenir l'intervalle des valeurs propres d'une matrice tridiagonale, nécessaire pour calculer par la suite les valeurs propres de cette même matrice.

```
Fichier : functions/eighvals/gershgorin_trig.py.
Commande de test : python3 test_unitaire.py T2_Gershgorin.

# Compute an interval containing all eigenvalues of a tridiagonal matrix

# Args:
#     C (np.array): Tridiagonal matrix.

# Returns:
#     np.float64: Lower bound of the interval, obtained via Gershgorin circle theorem
#     np.float64: Upper bound of the interval, obtained via Gershgorin circle theorem

def gershgorin_trig(C: np.array) -> tuple[np.float64, np.float64]:
```

2.3 Bisection

Nous avons programmé et testé notre version du calcul des valeurs propres d'une matrice tridiagonale par la méthode de la bisection.

```
Fichier : functions/eighvals/bisection.py.
Commande de test : python3 test_unitaire.py T3_Bisection.

# Compute the eigenvalues of a matrix A using the bisection method.
# Recursively calls itself until the interval is small enough.

# Args:
#     A (np.array): Matrix to compute the eigenvalues of.
#     binf (np.float64): Inferior bound of the interval.
#     bsup (np.float64): Superior bound of the interval.
#     e (np.float32): Precision of the computation. Defaults to np.finfo(np.float32).eps
#                     (machine precision in 32 bits).

# Returns:
#     np.array: Eigenvalues of A in the interval [binf, bsup].
#     Empty array if the interval does not contain any eigenvalue.

def bisection(A: np.array, binf: np.float64, bsup: np.float64,
              precision: np.float32 = np.finfo(np.float32).eps) -> np.array:
```

Cette fonction est dépendante de deux autres fonctions que nous avons également programmé et testé nous-même:

2.3.1 Séquence de Sturm

Nous avons programmé et testé notre version du calcul de la séquence de Sturm d'une matrice avec un décalage μ des diagonales.

Fichier : `functions/eighvals/bissection.py`.
Commande de test : `python3 test_unitaire.py T4_SturmSeq`.

```
# Compute the Sturm sequence of a matrix A.

# Args:
#     A (np.array): Matrix to compute the Sturm sequence of.
#     mu (int): Adjusts the matrix A by subtracting mu from the diagonal.

# Returns:
#     np.array: Sturm sequence of A.

def sturm_seq(A: np.array, mu: int) -> np.array:
```

2.3.2 Comptage changement de signe

Nous avons programmé et testé une fonction pour calculer le nombre de changements de signe dans une séquence de Sturm.

On suppose que la séquence de Sturm fournie commence toujours bien par 1.

Fichier : `functions/eighvals/bissection.py`.
Commande de test : `python3 test_unitaire.py T5_CountSignChange`.

```
# Compute the number of sign changes in a Sturm sequence.

# Args:
#     seq (np.array): Sequence of determinants.

# Returns:
#     int: Number of sign changes.

def count_sign_changes(seq: np.array) -> int:
```

2.4 Puissance Inverse

Nous avons programmé et testé notre version du calcul d'un vecteur propre associé à une valeur propre d'une matrice par la méthode de la puissance inverse.

Fichier : `functions/eighvals/puissance_inverse.py`.
Commande de test : `python3 test_unitaire.py T6_puissance_inverse`.

```
# Compute the eigenvector of a matrix A using the inverse method and the given eigenvalue mu.

# Args:
#     A (np.array): Matrix to compute the eigenvector of.
#     mu (np.float64): One of the eigenvalues of A.
#     epsilon (np.float32): Precision of the computation.

# Returns:
```

```
#      np.array: Eigenvector of A corresponding to the eigenvalue mu.

def puissance_inverse(A: np.array, mu: np.float64, epsilon: np.float32 = np.finfo(np.float32).eps):
```

Cette fonction est dépendante de deux autres fonctions que nous avons également nous-même programmé et testé :

2.4.1 Solve Triangular

Nous avons programmé et testé notre version de la fonction `solve_triangular`. Les paramètres `lower`, `overwrite_b` et `unit_diagonal` ont été implémentés.

Fichier : `functions/eighvals/puissance_inverse.py`.

Commande de test : `python3 test_unitaire.py T7_solve_triangular`.

```
# Solve a linear system of equations with a triangular matrix.

# Args:
#      A (np.array): Triangular matrix. A from Ax = b.
#      b (np.array): Vector of constants. b from Ax = b.
#      lower (bool): If True, the matrix is lower triangular, otherwise it is upper triangular.
#      overwrite_b (bool): If True, the function overwrites the vector b.
#      unit_diagonal (bool): If True, the diagonal of the matrix is assumed to be 1.
#                          Defaults to False.

# Returns:
#      np.array: Solution of the linear system of equations.

def solve_triangular(A: np.array, b: np.array, lower: bool, overwrite_b: bool,
                    unit_diagonal: bool = False):
```

2.4.2 Factorisation LU

Nous avons programmé et testé notre version de la décomposition LU d'une matrice carrée.

Fichier : `functions/eighvals/puissance_inverse.py`.

Commande de test : `python3 test_unitaire.py T8_lu_factor`.

```
# Compute the LU factorization of a matrix A.

#      Args:
#      A (np.array): Matrix to factorize.

#      Returns:
#      tuple[np.array, np.array]: Tuple containing the factorized matrix and the pivot vector.

def lu_factor(A: np.array) -> tuple[np.array, np.array]:
```

2.5 Eigh

Cette fonction est une "wrapper function", qui sert à exécuter facilement tout le déroulé d'obtention des valeurs propres et des vecteurs propres par les fonctions précédentes.

Fichier : `functions/eighvals/eigh.py`.

Commande de test : `python3 test_unitaire.py T9_eigh`.

```
# Compute the eigenvalues and eigenvectors of a symmetric matrix A.
# Same as numpy.linalg.eigh

# Args:
#     A (np.array): Symmetric matrix to compute the eigenvalues and eigenvectors.

# Returns:
#     tuple[np.array, np.array]: Eigenvalues and eigenvectors of A.

def eigh( A : np.array ) -> tuple[np.array, np.array]:
```

3 ACP méthode moderne

3.1 Bidiagonalisation

Nous avons programmé et testé notre version du calcul de la bidiagonalisation d'une matrice.

Fichier : `functions/svd/bidiagonal.py`.

Commande de test : `python3 test_unitaire.py T10_bidiagonal`.

```
# Compute the bidiagonal decomposition of a matrix A.

# Args:
#     A (np.array): Matrix to decompose.

# Returns:
#     tuple[np.array, list[np.array], list[np.array]]: B, VL, VR such that A = VL * B * VR.

def bidiagonal(A: np.array) -> tuple[np.array, list[np.array], list[np.array]]:
```

Cette fonction est dépendante d'une autre fonction :

3.1.1 Reflecteur

Cette fonction calcule la matrice de réflexion associée à un vecteur v .

Fichier : `functions/svd/bidiagonal.py`.

Commande de test : `python3 test_unitaire.py T11_reflecteur`.

```
# Compute the reflection matrix associated to a vector v.
```

```

# Args:
#     p (int): Number of rows of the matrix.
#     v (np.array): Vector to compute the reflection matrix.

# Returns:
#     np.array: Reflection matrix associated to v.

def reflecteur(p: int, v: np.array) -> np.array:

```

3.1.2 SVD

Nous avons programmé et testé notre propre version de la fonction `svd`. A noter que ladite fonction utilise un intervalle légèrement moins précis que celui donné par Gershgorin: on peut voir dans le code que l'intervalle est initialement celui fourni par Gershgorin, mais élargi de 0,0001 aux deux extrêmes. Il s'agit d'une mesure de précaution prise à cause d'un bug qui semble se produire seulement sur certaines versions de Python, qui conduisait à la disparition d'une des valeurs propres.

Fichier : `functions/svd/svd.py`.

Commande de test (un seul test): `python3 test_unitaire.py T12_svd`.

```

# Compute the singular values and singular vectors of a matrix A.

# Args:
#     A (np.array): Matrix to decompose.

# Returns:
#     tuple[np.array, np.array, np.array]: U, Sigma, Vt such that A = U * Sigma * Vt.

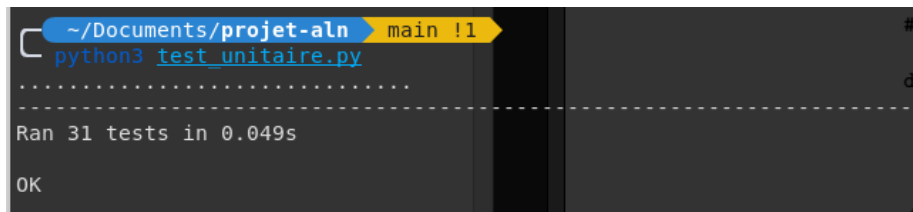
def svd(A: np.array) -> tuple[np.array, np.array, np.array]:

```

Cette fonction est dépendante de la fonction `reflecteur` précisée précédemment.

4 Tests et Résultats

Tout les tests mis en place sont effectués avec succès :



```

~/.Documents/projet-aln main !1
python3 test_unitaire.py
.....
Ran 31 tests in 0.049s
OK

```

Figure 1: Test de toutes les fonctions décrites dans ce rapport.

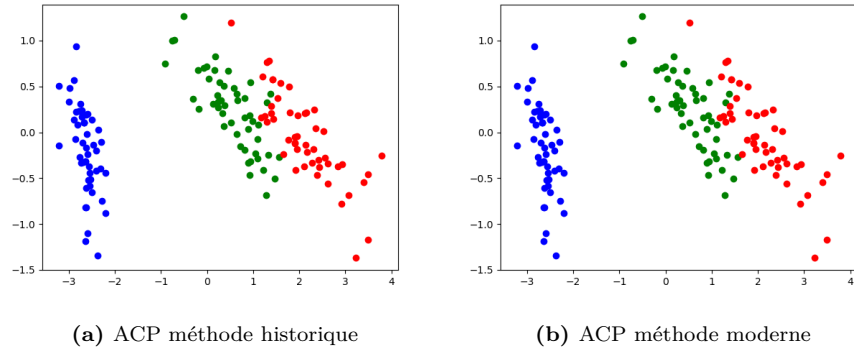


Figure 2: Résultats des ACP effectuées avec nos fonctions

Pour voir l'utilisation de ces fonctions sur le cas général de l'ACP appliqué aux iris, utilisez les programmes suivants:

- ACP méthode historique : `acp_iris_eigh.py`
- ACP méthode moderne : `acp_iris_svd.py`

Les résultats sont concluants et nous obtenons des graphiques similaires ou extrêmement proches de ceux produits par le fichier original fourni.