TP1.2 ACP

Taha Ez-zoury
Youness Bouallou
Marouane Elbissouri

01 November 2023

Introduction :

Au cours du TP1.1, nous avons effectivement mis en œuvre les différentes étapes de l'Analyse en Composantes Principales (ACP) en utilisant Python, ce qui nous a permis d'appliquer cette approche à nos six points. En clôturant la phase du TP1.1, nous avons élargi notre champ d'application en utilisant l'ACP que nous avions développée sur huit points, et nous avons même intégré des concepts tels que la qualité de projection et la contribution individuelle, bien que ces éléments ne figuraient pas parmi les exigences initiales du TP1.1. Cette approche a servi à garantir la fiabilité de nos résultats issus de l'implémentation et à les comparer à nos calculs manuels réalisés lors du TD.

Dans cette nouvelle étape, nous poursuivons une démarche similaire, mais nous travaillerons avec de nouvelles données auxquelles nous appliquerons l'ACP centrée et l'ACP normalisée. De plus, nous prévoyons d'apporter des ajustements mineurs à nos implémentations en Python (nos fonctions Python) utilisées dans le TP1.1. Ces modifications visent à donner un aspect plus structuré à nos implémentations et à les doter de la capacité de recevoir l'information sur le type d'ACP à appliquer, que ce soit centrée ou normalisée. Cette capacité sera explorée en détail ultérieurement dans ce TP. De plus, nous explorerons l'ACP dual qui sera appliquée à la fois sur un nuage isotrope et sur un nuage non-isotrope.

Remarque : Les commentaires dans le code seront rédigés en anglais en raison de problèmes d'affichage des caractères accentués, ce qui pourrait rendre le code moins esthétique visuellement.

1 Partie 1 : L'ACP appliqué à l'espace des variables :

Importation des bibliothèques nécessaires :

```
# For mathematical operations:
import numpy as np
# For tabular data manipulation:
import pandas as pd
# For creating plots and charts:
import matplotlib.pyplot as plt
# For advanced data visualization:
import seaborn as sb
```

1. **Préparation, Visualisation de Données et Calcul des Indicateurs Statistiques :**

1.1 Préparation et Visualisation de Données :

```
# Read data from a file named 'data_PDE20.csv':
data = pd.read_csv('data_PDE20.csv', delimiter=';', decimal=',')
# Display the first three rows:
print(data.head(3))
```

```
   Num      X1     X2    X3    X4      X5     X6      X7      X8  Unnamed: 9
0    1  303.09  24.19  0.00  3.29  179.99   8.09  360.90  120.33         NaN
1    2  281.88  38.59  4.29  1.06  192.00  10.50  353.50  117.00         NaN
2    3  277.06  34.79  0.00  6.85  183.77  38.89  343.95  114.65         NaN
```

Supprimons les colonnes inutiles "Num" et "Unnamed: 9" afin d'éviter tout problème potentiel lors des calculs ultérieurs.

```
# Remove the 'Unnamed: 9' column
del data['Unnamed: 9']
# Remove the 'Num' column
del data['Num']
```

. Visualisation de la Matrice de données:

À ce point, nous avons opté pour la conversion de la matrice de données en un tableau NumPy. L'intention derrière cette décision est de programmer nos propres indicateurs statistiques (comme la covariance, la corrélation, etc.) sans recourir aux méthodes prédéfinies pour les objets de type data.frame. Nous désignerons cette matrice par le nom "X".

```python
def data_to_matrix(data):
    return data.values
X=data_to_matrix(data)
# Display the first five rows :
print(X[:5])
```

```
[[303.09   24.19    0.      3.29  179.99    8.09  360.9   120.33 ]
 [281.88   38.59    4.29    1.06  192.     10.5   353.5   117.   ]
 [277.06   34.79    0.      6.85  183.77   38.89  343.95  114.65 ]
 [276.38   32.43    4.14    2.04  190.79   38.53  341.17  113.91 ]
 [253.8    39.5     3.04    1.    173.8    19.334 382.11  127.373]]
```

1.2 Calcul des Indicateurs Statistiques :

Nous commençons par élaborer la fonction 'moyenne(X)' pour calculer la moyenne de X. Cette fonction renverra une matrice de même dimension que X, où chaque colonne contiendra la moyenne de chaque colonne de X, répétée n fois (n étant le nombre de lignes de X).Cela vise à rendre le centrage de X plus simple, ce qui sera exploré dans la question 2 à venir.

```python
def moyenne(X):
    n,_=np.shape(X)
    a=np.ones((n,n))
    return (1/n)*np.dot(a,X)
```

. Calcul de la matrice de covariance :

```python
def cov(X):
    n,_=np.shape(X)
    #Subtracte the mean from X (centering) :
    a=X-moyenne(X)
    #Calculate the covariance matrix :
    b=(1/n)*np.dot(np.transpose(a),a)
    return b
```

. Calcul de la liste des variances et de la liste des écarts-types :

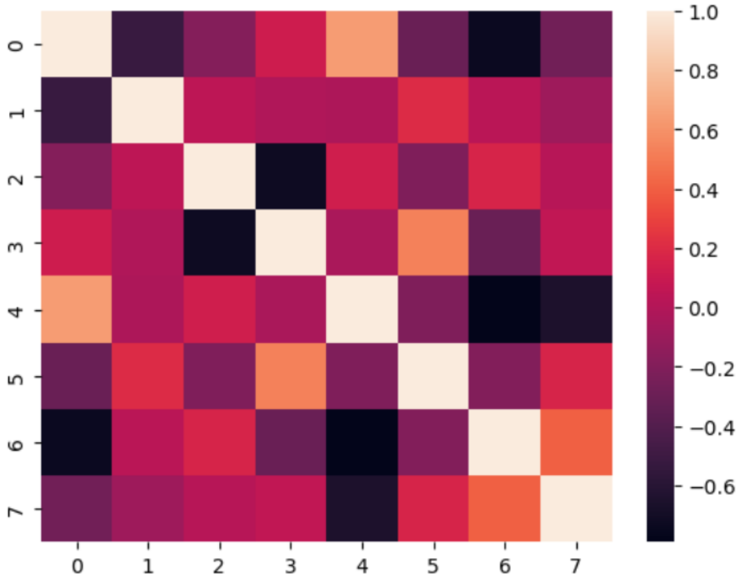
```
def variance(X):
    return np.diag(cov(X))

def sigma(X):
    return np.sqrt(variance(X))
```

. Calcul de la matrice de corrélation :

```
def corr(X):
    D=np.diag(1/sigma(X))
    a=cov(X)
    # Calculate the correlation matrix s
    s=np.dot(a,D)
    s=np.dot(D,s)
    return s
```

. Affichage de la matrice de corrélation sous la forme d'une heatmap :

```
sb.heatmap(corr(X))
plt.show()
```



Il est observé que les variables présentent généralement des corrélations faibles entre elles, la plupart affichant des valeurs inférieures à 0,5.

2. **Centrage, Normalisation de X et Détermination des Hyperplans d'Inertie Maximale**

2.1 Centrage et Normalisation de X:

```
def center(X):
    return X-moyenne(X)
# cennor stands for center and normalise
def cennor(X):
    D = np.diag(1/sigma(X))
    return np.dot(center(X),D)
```

2.2 Détermination des Hyperplans d'Inertie Maximale :

```
# Diagonalization of a matrix A:
def sorted_eig_val_vect(A):
    eigenvalues, eigenvectors = np.linalg.eig(A)
    s = sorted(eigenvalues, reverse=True)
    s = np.array(s)
    sorted_indices = np.argsort(eigenvalues)[::-1]
    sorted_eigenvectors = eigenvectors[:, sorted_indices]
    return s, sorted_eigenvectors

#This function will be used later to apply PCA to the data
def hyperplans(X):
    A = cov(X)
    return sorted_eig_val_vect(A)
```

Préalablement à la troisième partie, nous souhaitons introduire certaines notations : La liste des valeurs propres et la matrice des vecteurs propres, correspondant respectivement aux données centrées, seront notées Vpc et Vc, tandis que pour les données normalisées, nous utiliserons Vpn et Vn.

```
Vpc,Vc = hyperplans(center(X))
Vpn,Vn = hyperplans(cennor(X))
```

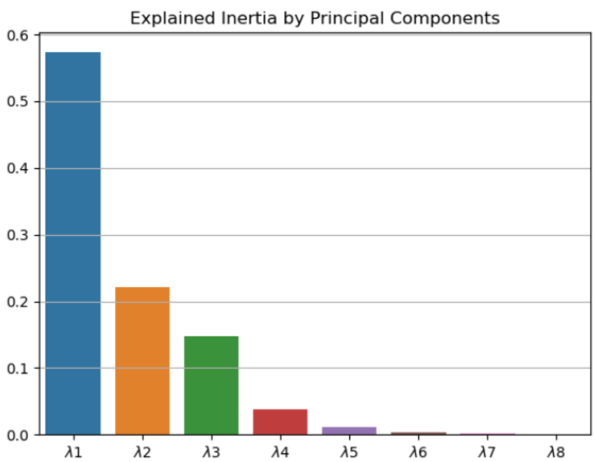
2 Partie 2 : Qualité de l'ACP :

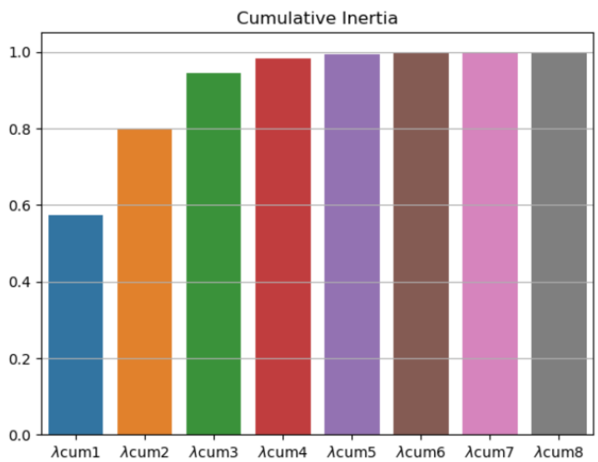
3. **Représentation de l'Inertie Expliquée et des Règles de Choix des Composantes Principales**

3.1 Analyse de l'Inertie Expliquée et Inertie Cumulée

. Centered Data Case :

```
n = len(Vpc)
fig = plt.figure(figsize=(15, 5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.set_title("Explained Inertia by Principal Components")
ax2.set_title("Cumulative Inertia")
ax1.grid()
ax2.grid()
a = sum(Vpc)
# Create a bar plot for explained inertia :
x = [f'X{i}' for i in range(1, n + 1)]
y = [Vpc[i] / a for i in range(n)]
sb.barplot(x=x, y=y, ax=ax1)
# Create a bar plot for cumulative explained inertia
x = [f'Xcum{i}' for i in range(1, n + 1)]
y = [sum(Vpc[:i + 1]) / a for i in range(n)]
sb.barplot(x=x, y=y, ax=ax2)
plt.show()
```

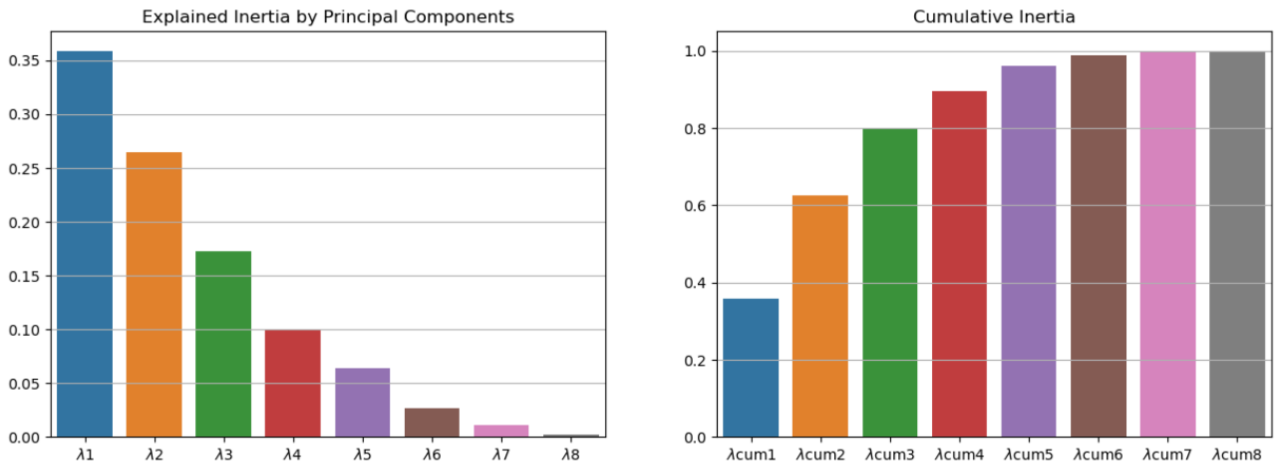




. Normalized Data Case :

```
n = len(Vpn)
fig = plt.figure(figsize=(15, 5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.set_title("Explained Inertia by Principal Components")
ax2.set_title("Cumulative Inertia")
ax1.grid()
ax2.grid()
```

```
a = sum(Vpn)
# Create a bar plot for explained inertia :
x = [f'X{i}' for i in range(1, n + 1)]
y = [Vpn[i] / a for i in range(n)]
sb.barplot(x=x, y=y, ax=ax1)
# Create a bar plot for cumulative explained inertia
x = [f'Xcum{i}' for i in range(1, n + 1)]
y = [sum(Vpn[:i + 1]) / a for i in range(n)]
sb.barplot(x=x, y=y, ax=ax2)
plt.show()
```



Une observation intéressante est que la variation entre les barreaux, représentant l'inertie expliquée, semble être moins prononcée pour l'ACP normée par rapport à l'ACP centrée. Cela suggère que l'ACP normée tend à mieux répartir l'inertie entre les composantes principales, ce qui est le résultat de la normalisation des données. Cette répartition plus uniforme de l'inertie peut avoir des implications pour la réduction de dimension et la sélection du nombre de composantes principales à conserver.

3.2 Règles de Choix des Composantes Principales

Dans cette section, nous aborderons les règles de choix des composantes principales. Lorsque l'on ne se fixe pas un seuil prédéfini pour le taux d'inertie (comme les 80% que nous avions utilisés dans le TP1.1), plusieurs règles existent pour déterminer le nombre k de composantes principales. Nous nous concentrerons sur les règles classiques, notamment la règle de Cantell, la règle de Kaiser-Guttman, ainsi que la règle de Karlis-Saporta-Spinaki, que nous avons étudiées en cours.

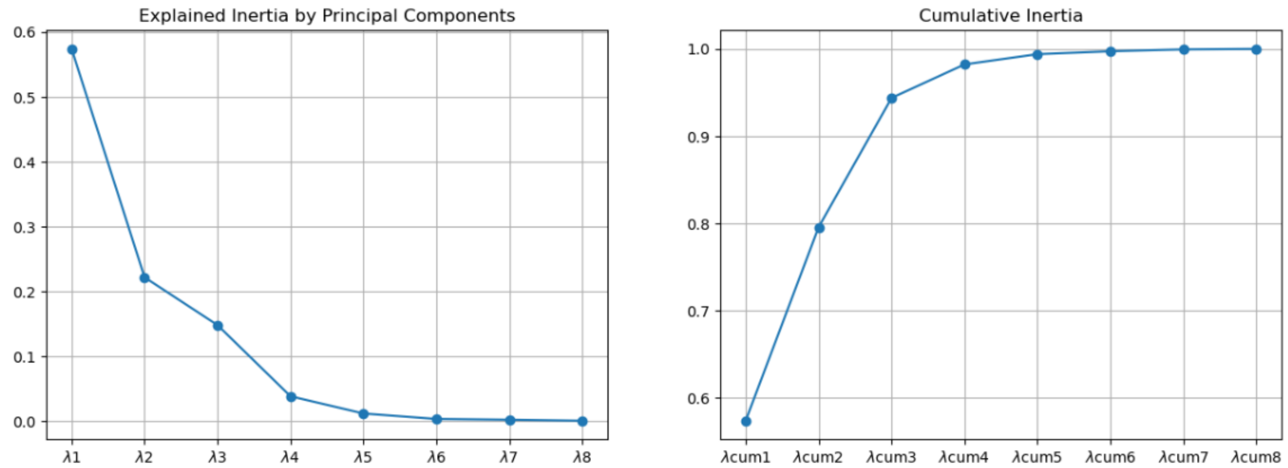
(a) .Règle de Cantell :

La règle de Cantell s'appuie sur le décompte des coudes dans les courbes d'inertie et d'inertie cumulée. Notre approche se limitera à une évaluation visuelle des coudes à partir des graphiques correspondants.

. Centered Data Case :

```
n = len(Vpc)
fig = plt.figure(figsize=(15, 5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.set_title("Explained Inertia by Principal Components")
```

```python
ax2.set_title("Cumulative Inertia")
ax1.grid()
ax2.grid()
a = sum(Vpc)
# Create a plot for explained inertia :
x = [f'X{i}' for i in range(1, n + 1)]
y = [Vpc[i]/a for i in range(n)]
ax1.plot(x,y,marker='o')
# Create a bar plot for cumulative explained inertia
x = [f'Xcum{i}' for i in range(1, n + 1)]
y = [sum(Vpc[:i + 1]) / a for i in range(n)]
ax2.plot(x,y,marker='o')
plt.show()
```



Nous pourrions avancer que k est égal à 4, bien que cette estimation demeure approximative. Il est important de noter que la visualisation d'un graphe ne fournit pas toujours des valeurs précises.

(b) **.Règle de Kaiser-Guttman :**

Cette règle est utilisé dans le cas normalisé.

. Normalised Data Case Only :

```python
def K_G(X):
    # List of eigenvalues for the normalized data
    Z = hyperplans(cennor(X))[0]
    a = Z[Z >= 1] # Keeping the values that are >=1
    return len(a)

print(K_G(X))
```

. Output : 3

(c) **.Règle de Karlis-Saporta-Spinaki :**

. Normalised Data Case :

```
def K_S_S(X):
    #keeping values > 2*np.sqrt((p-1)/(n-1))
    n,p=np.shape(X)
    Z=hyperplans(cennor(X))[0]
    a=2*np.sqrt((p-1)/(n-1))
    return len(Z[Z>a])

print(K_S_S(X))
```

. Output : 3

4. **Calcul des nouvelles coordonées et Qualité de projection**

4.1 Calcul des nouvelles coordonées :

Étant donné que la fonction 'hyperplanes(X)' renvoie des vecteurs propres unitaires, la réalisation de la projection se résume simplement à effectuer le produit scalaire entre nos points de données et ces vecteurs.

. Projection function :

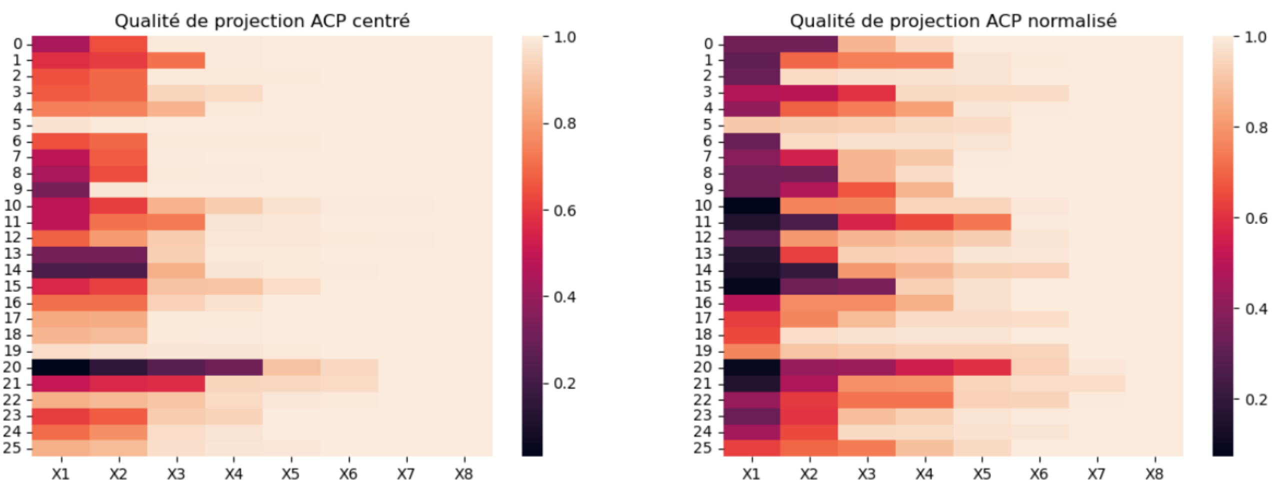
```
# Projection function:
# e = 0 means Principal Component Analysis (PCA) with centering
# e = 1 means Principal Component Analysis (PCA) with normalization
def projection(X,e,k):
    if e == 0:
        Z = center(X)
        P = hyperplans(Z)[1]
    else:
        Z = cennor(X)
        P = hyperplans(Z)[1]
    # We project onto the space of dimension k and then dimension p
    projection_k = np.dot(Z, P[:,:k])
    full_projection = np.dot(Z, P)
    return projection_k, full_projection
```

4.2 Qualité de projection :

```python
# The function returns a quality list that records the projection quality of each individual in the K-dimensional space.
def qual_proj(X, e, k):
    X_k, X_p = projection(X, e, k)
    quality = np.sum(X_k**2, axis=1) / np.sum(X_p**2, axis=1)
    return quality

def matrix_quality(X, e):
    _, p = np.shape(X)
    L = []
    for i in range(1, p+1):
        L.append(qual_proj(X, e, i))
    return np.array(L)
```

On utilise la matrice de qualité pour visualiser les qualités de projection sous forme d'une heatmap.

```python
fig = plt.figure(figsize=(15, 5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.set_title("Quality of PCA Projection (Centered)")
ax2.set_title("Quality of PCA Projection (Normalized)")
names = ['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8']
# Centered PCA
d = np.transpose(matrix_quality(X, 0))
sb.heatmap(d, ax=ax1, xticklabels=names)
# Normalised PCA
d = np.transpose(matrix_quality(X, 1))
sb.heatmap(d, ax=ax2, xticklabels=names)
plt.show()
```



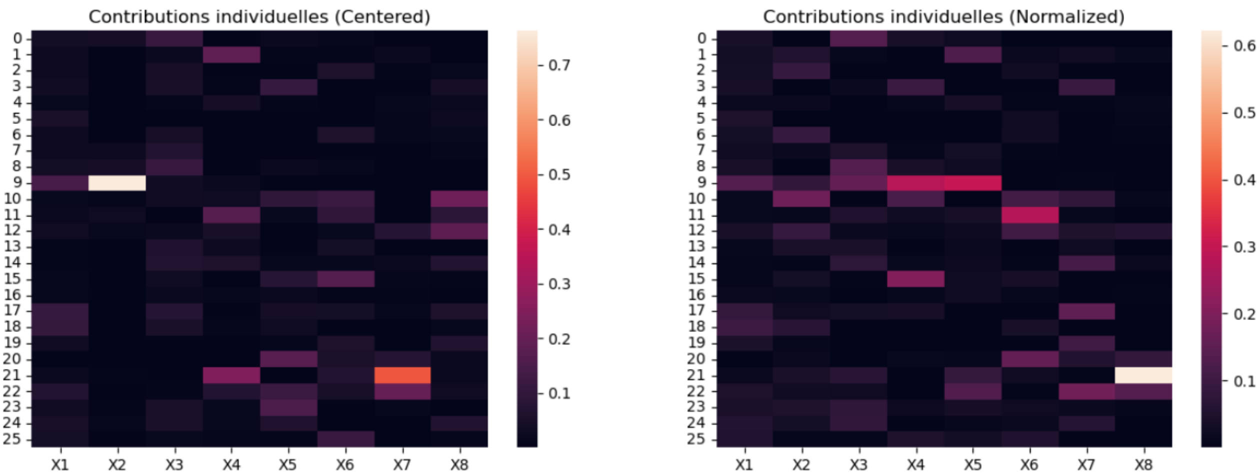
Ce que nous observons sur les heatmaps est en parfait accord avec la théorie. Nous pouvons clairement constater qu'à mesure que nous nous déplaçons vers la gauche, la qualité de la projection tend vers 1, ce qui signifie que nous projetons dans un espace similaire à l'original. De plus, il est intéressant de noter une uniformité des couleurs dans le heatmap de l'ACP normalisée, car la normalisation réduit les variations importantes.

5. **Contributions individuelles :**

```python
def contribution (X,e,k):
    n,_ = X.shape
    if e==0:
        vp = hyperplans(center(X))[0][:k]
    else:
        vp = hyperplans(cennor(X))[0][:k]
    # The new coordinates in the set of dim=k
    X_k = projection (X,e,k)[0]
    # Calculate the contribution matrix
    c=(1/n)*(X_k**2)
    # Divide  by the corresponding eigenvalue
    A=c/vp
    return A
```

Visualisation :

```python
d=contribution(X,0,8)
fig = plt.figure(figsize=(15, 5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.set_title("Contributions individuelles (Centered)")
ax2.set_title("Contributions individuelles (Normalized)")
names = ['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8']
sb.heatmap(d, ax=ax1, xticklabels=names)
d=contribution(X,1,8)
sb.heatmap(d, ax=ax2, xticklabels=names)
plt.show()
```



Il est crucial de souligner que la présence d'un individu avec une contribution exceptionnellement élevée, représentée en blanc sur les deux heatmaps (Heatmap 1 pour l'ACP centrée et Heatmap 2 pour l'ACP normalisée), n'est pas souhaitable. Cette contribution excessive peut devenir un facteur d'instabilité dans l'analyse. Dans de nombreux cas, il pourrait être envisagé d'éliminer les individus présentant une contribution disproportionnée afin de garantir la robustesse des résultats et une interprétation plus équilibrée des composantes. Cette considération sera explorée en détail dans la partie 3.

6. **Validation du Premier Plan Factoriel avec l'Utilisation de l'ACP du Module sklearn**

Nous allons à présent comparer nos résultats avec ceux obtenus en utilisant la fonction PCA intégrée de Python. Pour ce faire, nous allons vérifier si nous avons obtenu le même premier plan factoriel (un plan à deux dimensions) généré par les deux premiers vecteurs propres.

.Centered Data Case : PCA from sklearn

```python
from sklearn.decomposition import PCA
# Create a PCA instance
pca = PCA(n_components=2)
# Apply PCA to our centered data
Z = pca.fit_transform(center(X))
# Get the eigenvectors
eigenvectors = pca.components_
print(eigenvectors)
```

```
[[ 0.60539488 -0.05008526 -0.0103714   0.00983442  0.28056837 -0.06080149
  -0.59148033 -0.44555562]
 [-0.46057614  0.14552594  0.01696215 -0.02416068  0.08615615 -0.04896718
   0.19904668 -0.84639241]]
```

.Centered Data Case : Our PCA

```python
# Compute the principal components using the hyperplans function on
   ↪ centered data
Z = hyperplanes(center(X))
# Extract and print the first two principal components
principal_components = np.transpose(Z[1][:, :2])
print(principal_components)
```

```
[[ 0.60539488 -0.05008526 -0.0103714   0.00983442  0.28056837 -0.06080149
  -0.59148033 -0.44555562]
 [-0.46057614  0.14552594  0.01696215 -0.02416068  0.08615615 -0.04896718
   0.19904668 -0.84639241]]
```

Nous avons effectivement obtenu le même premier plan factoriel, ce qui confirme la validité de notre code.

.Normalised Data Case : PCA from sklearn

```
Z=pca.fit_transform(cennor(X))
eigenvectors=pca.components_
print(eigenvectors)
```

```
[[ 0.51230648 -0.15355077 -0.09802888  0.10649843  0.52700652 -0.09495124
  -0.52045452 -0.36795126]
 [ 0.02488198 -0.06088817  0.53501712 -0.62970936  0.15769264 -0.49468335
   0.15829354 -0.13513285]]
```

.Normalised Data Case : Our PCA

```
Z=hyperplans(cennor(X))
print(np.transpose(Z[1][:,:2]))
```

```
[[-0.51230648  0.15355077  0.09802888 -0.10649843 -0.52700652  0.09495124
   0.52045452  0.36795126]
 [ 0.02488198 -0.06088817  0.53501712 -0.62970936  0.15769264 -0.49468335
   0.15829354 -0.13513285]]
```

Malgré l'omission d'un signe négatif dans le facteur, le résultat reste valable car nous obtenons le même axe. Par conséquence, nous obtenons le même premier plan.

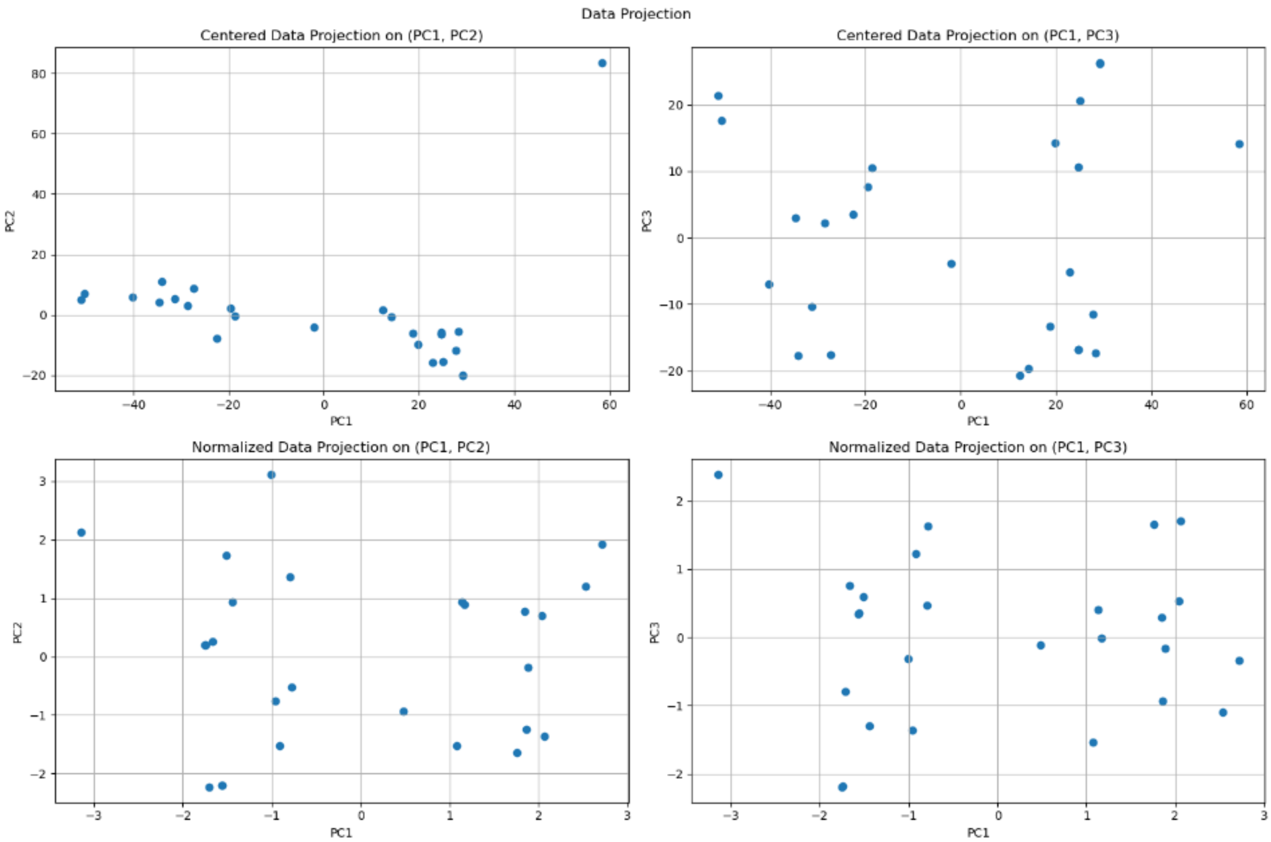
7. **Visualisation des Nouveaux Individus dans le Nouvel Espace en Utilisant les Deux Premiers Plans Factoriels :**

Nous représentons graphiquement les individus dans le plan (CP1, CP2) et dans le plan (CP1, CP3).

.Normalised Data Case : Our PCA

```
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('Data Projection')
# Create subplots for centered data with CP1, CP2
Z = projection(X, 0, 3)[0]
ax1 = axes[0, 0]
ax1.set_title('Centered Data Projection on (PC1, PC2)')
ax1.set_xlabel('PC1')
ax1.set_ylabel('PC2')
ax1.scatter(Z[:, 0], Z[:, 1])
ax1.grid()
# Create subplots for centered data with CP1, CP3
ax2 = axes[0, 1]
ax2.set_title('Centered Data Projection on (PC1, PC3)')
ax2.set_xlabel('PC1')
ax2.set_ylabel('PC3')
ax2.scatter(Z[:, 0], Z[:, 2])
ax2.grid()

```

```
# Create subplots for normalized data with CP1, CP2
Z = projection(X, 1, 3)[0]
ax3 = axes[1, 0]
ax3.set_title('Normalized Data Projection on (PC1, PC2)')
ax3.set_xlabel('PC1')
ax3.set_ylabel('PC2')
ax3.scatter(Z[:, 0], Z[:, 1])
ax3.grid()
# Create subplots for normalized data with CP1, CP3
ax4 = axes[1, 1]
ax4.set_title('Normalized Data Projection on (PC1, PC3)')
ax4.set_xlabel('PC1')
ax4.set_ylabel('PC3')
ax4.scatter(Z[:, 0], Z[:, 2])
ax4.grid()
plt.tight_layout()
plt.show()
```



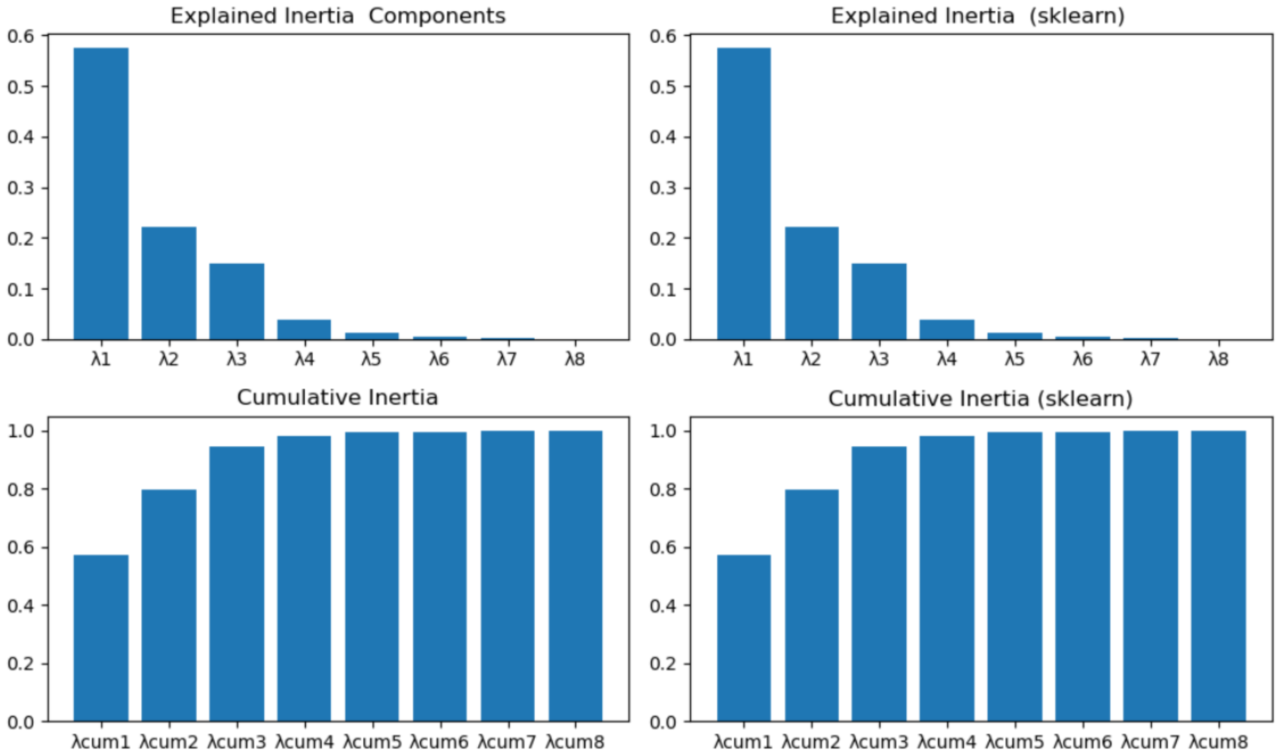
Observons que, pour l'ACP centrée, la projection sur le plan (CP1, CP2) révèle une anomalie notable : un point isolé dans un coin tandis que tous les autres points semblent regroupés dans une autre partie du plan. Cette observation concorde avec le heatmap de la contribution individuelle où le 10e individu était identifié comme contribuant significativement à CP2. Le point isolé représente donc ce dernier, ayant une influence importante sur CP2. Cette singularité rend la projection esthétiquement moins satisfaisante et entrave la visualisation globale du nuage de points. Par conséquent, il est préconisé d'éliminer les points ayant une contribution significative avant d'appliquer l'ACP centrée afin d'éviter tout impact déformant sur l'analyse.

En ce qui concerne l'ACP normalisée, la dispersion des points apparaît plus homogène, ce qui peut être attribué au lissage résultant de la division par les écarts types (normalisation).

.Une dernière comparaison avec l'ACP du module sklearn :

.Centered Data Case : Our PCA

```
pca = PCA()

# Apply PCA to our data
pca.fit_transform(center(X))
E = pca.explained_variance_
fig, axes = plt.subplots(2, 2, figsize=(10, 6))

R = hyperplans(center(X))
n = len(R[0])
x = [f'$\lambda${i}' for i in range(1, n+1)]
a = sum(R[0])
y = [R[0][i]/a for i in range(n)]

axes[0, 0].bar(x, y)
axes[0, 0].set_title("Explained Inertia  Components")
a = sum(E)
y = [E[i]/a for i in range(n)]
axes[0, 1].bar(x, y)
axes[0, 1].set_title("Explained Inertia  (sklearn) ")

a = sum(R[0])
x = [f'$\lambda$cum{i}' for i in range(1, n+1)]
y = [sum(R[0][:i+1])/a for i in range(n)]
axes[1, 0].bar(x, y)
axes[1, 0].set_title('Cumulative Inertia')
a = sum(E)
y = [sum(E[:i+1])/a for i in range(n)]
axes[1, 1].bar(x,y)
axes[1, 1].set_title('Cumulative Inertia (sklearn)')

plt.tight_layout()
plt.show()
```



✓Implémentation Valide ✓

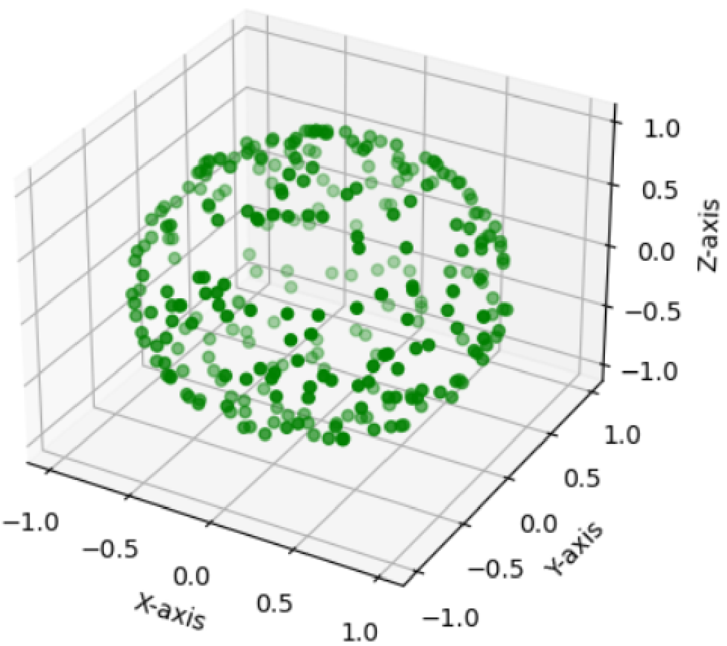
3 Partie 3 : Etude de la forme du nuage initiale et réduction de dimension :

8. **Nuage isotrope :**

   8.1 **Génération d'un nuage isotrope : (de 300 individus)**

.

```
n = 300
s = 5
np.random.seed(s)
S = np.random.randn(3, n)
X, Y, Z = S
#Normalize the data points to get a sphere :
a = np.sqrt(np.sum(S**2, axis=0))
A = S / a
X, Y, Z = A
# Create a 3D scatter plot :
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X, Y, Z, c='g', marker='o')
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_zlabel('Z-axis')
plt.show()
```



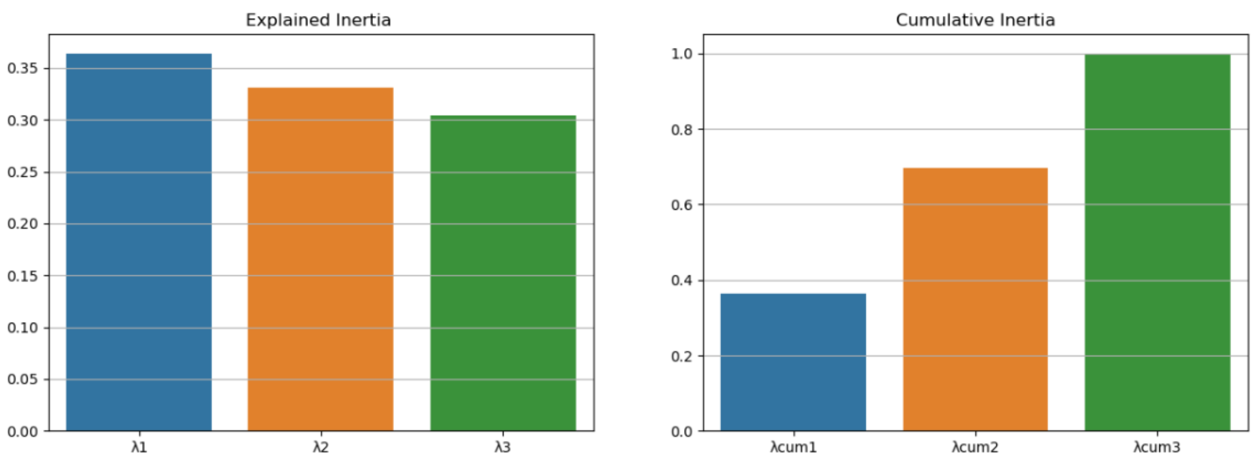
```
.
# Create the data matrix :
X = np.transpose(A)
```

Nous continuerons à travailler avec cette matrice de données pour le reste de cette question, en notant qu'elle est déjà normalisée.

8.2 ACP appliquée au nuage isotrope :

Tout comme dans la partie précédente, nous commençons l'application de l'ACP en calculant les inerties expliquées.

```
.
fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(121)
ax2 = fig add_subplot(122)
ax1.set_title("Explained Inertia ")
ax1.grid()
ax2.set_title('Cumulative Inertia')
ax2.grid()
# Get the eigenvalues of the dataset :
Vp = hyperplans(X)[0]

n = len(Vp)
a = sum(Vp)
x = [f'$\lambda${i}' for i in range(1, n+1)]
y = [Vp[i]/a for i in range(n)]
sb.barplot(x=x, y=y, ax=ax1)

x = [f'$\lambda$cum{i}' for i in range(1, n+1)]
y = [sum(Vp[:i+1])/a for i in range(n)]
sb.barplot(x=x, y=y, ax=ax2)
plt.show()
```

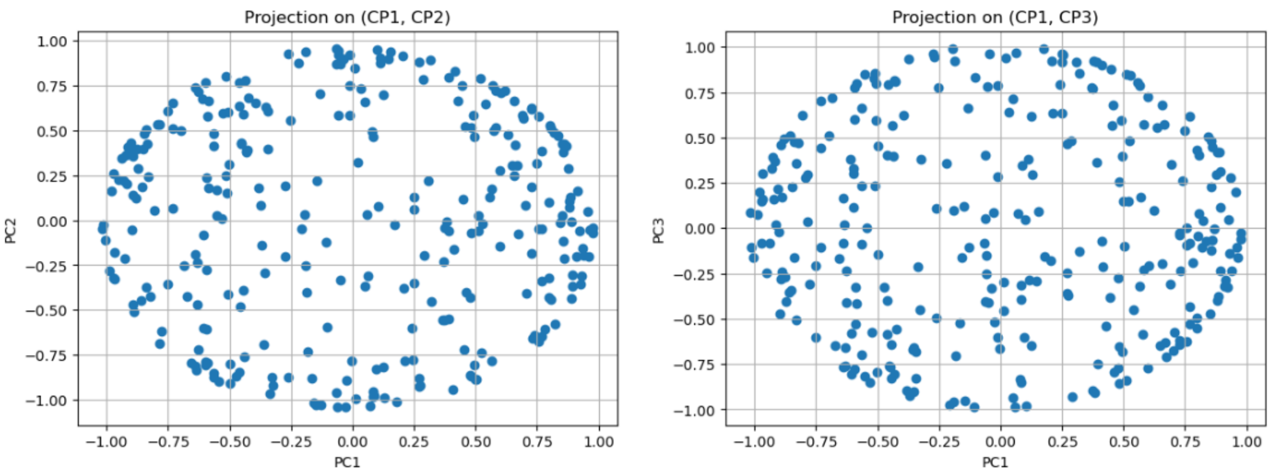


Les barplots représentant l'inertie expliquée et l'inertie cumulative révèlent une distribution relativement uniforme de la variance expliquée par chaque composante principale. Cette observation suggère que chacune des composantes principales contribue de manière significative à l'explication de la variance totale des données.

Ce constat de répartition équilibrée de la variance entre les composantes principales implique que chaque composante transporte une quantité d'informations équivalente en ce qui concerne la réduction de dimension. De plus, ce résultat est cohérent avec la nature isotrope du nuage de points généré, où aucune direction particulière ne privilégie la variance.

.Projection du nuage sur les plans (CP1,CP2) et (CP1,CP3):

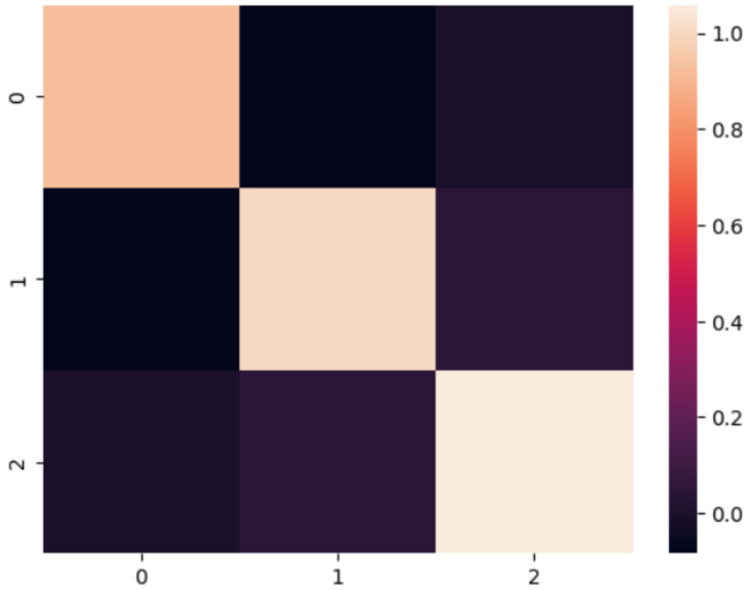
```
#Matrix of new coordinates :
Z = projection(X, 0, 3)[0]
fig = plt.figure(figsize=(15, 5))
ax1 = fig.add_subplot(121)
ax1.set_title('Projection on (CP1, CP2)')
ax1.set_xlabel('PC1')
ax1.set_ylabel('PC2')
ax2 = fig.add_subplot(122)
ax2.set_title('Projection on (CP1, CP3)')
ax2.set_xlabel('PC1')
ax2.set_ylabel('PC3')
ax1.scatter(Z[:, 0], Z[:, 1])
ax2.scatter(Z[:, 0], Z[:, 2])
ax1.grid()
ax2.grid()
plt.show()
```



Les projections des données sur ces composantes principales prennent la forme d'ellipses, démontrant ainsi l'équitable contribution de chaque composante à la variance totale des données. Cette configuration est en accord avec l'isotropie supposée de notre nuage de points initial. De plus, selon la loi des grands nombres, à mesure que le nombre d'individus augmente vers l'infini, la forme devrait tendre vers un cercle parfait.

.**Visualisation de la matrice de covariance-corrélation:**

```
#Heatmap of the correlation matrix (300 individuals):
sb.heatmap(cov(X))
plt.show()
```



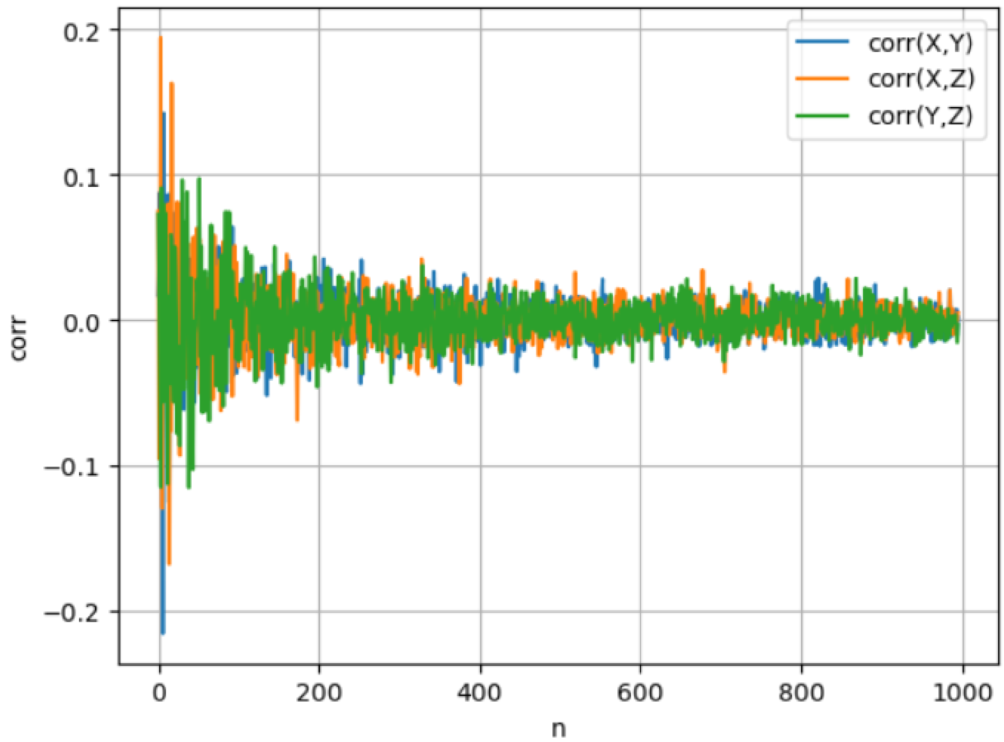
Il est clair que la heatmap de la matrice de corrélation est en parfait accord avec le fait que nous travaillons avec un vecteur gaussien (nuage isotropique).

8.3 Evolution de la matrice corrélation avec le nombre d'individus:

.Fonction pour créer un nuage isotropique sphérique :

```
#n is the number of points
def nuage(n):
    s=5
    np.random.seed(s)
    S=np.random.randn(3,n)
    a=np.sqrt(np.sum(S**2,axis=0))
    return np.transpose(S/a)
```

```
.
L=[]
M=[]
N=[]
for n in range(4,1000):
    T=cov(nuage(n))
    L.append(T[0][1])
    M.append(T[0][2])
    N.append(T[1][2])
plt.grid()
plt.xlabel('n')
plt.ylabel('corr')
plt.plot(L,label='corr(X,Y)')
plt.plot(M,label='corr(X,Z)')
plt.plot(N,label='corr(Y,Z)')
plt.legend()
plt.show()
```

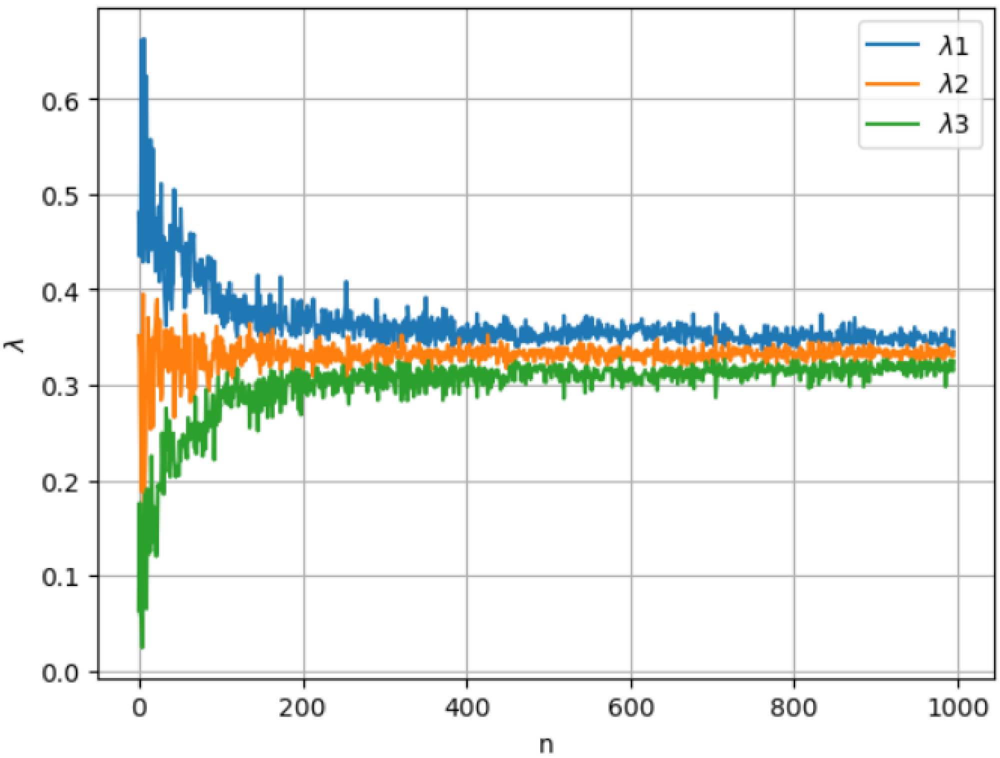


Nous constatons que la stabilité de la corrélation s'affirme à mesure que n augmente, résultant de l'application de la loi des grands nombres. La corrélation converge vers zéro étant donné que les trois variables, pour lesquelles nous calculons les représentations sur n individus, sont, par construction, indépendantes.

8.4 **Cascade des valeurs propres:**

.

```
L=[]
M=[]
N=[]
for n in range(4,1000):
    #eignvalues
    T=hyperplans(nuage(n))[0]
    L.append(T[0])
    M.append(T[1])
    N.append(T[2])
plt.grid()
plt.xlabel('n')
plt.ylabel('$\lambda$')
plt.plot(L,label='$\lambda$1')
plt.plot(M,label='$\lambda$2')
plt.plot(N,label='$\lambda$3')
plt.legend()
plt.show()
```

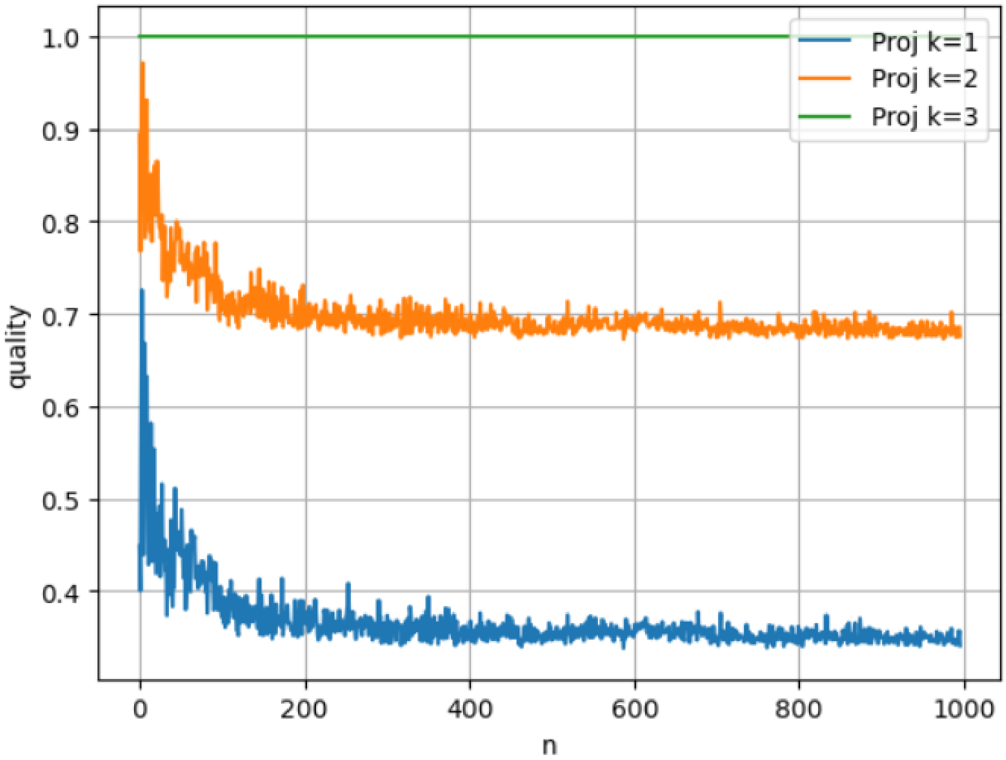


En cas de retour de S par la fonction nuage(n) au lieu de S/a, l'observation serait une convergence des lambdas vers 1, indiquant ainsi la variance de la loi gaussienne, conformément à la loi des grands nombres. Cependant, dans la situation actuelle, on remarque que les lambdas convergent également vers une constante inférieure. Il convient de vérifier s'il s'agit effectivement de la variance de la loi.

8.5 Cascade des moyennes de qualités de projections:

.

```
L=[]
M=[]
N=[]
for n in range(4,1000):
    T=matrix_quality(nuage(n),0)
    L.append(np.mean(T[0]))
    M.append(np.mean(T[1]))
    N.append(np.mean(T[2]))
plt.grid()
plt.xlabel('n')
plt.ylabel('quality')
plt.plot(L,label='Proj k=1')
plt.plot(M,label='Proj k=2')
plt.plot(N,label='Proj k=3')
plt.legend(loc='upper right')
plt.show()
```



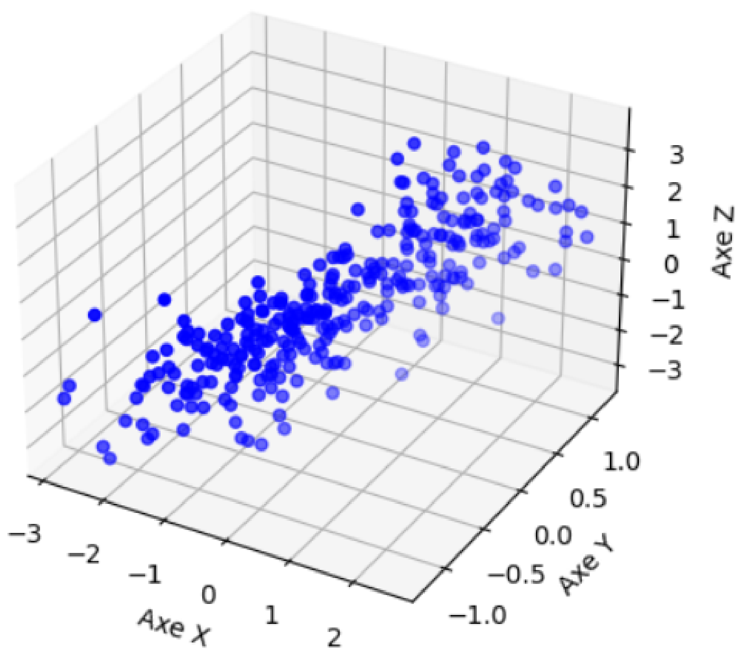
Selon le graphique, la qualité de la projection (pour les espaces de dimensions 2 ou 1) se stabilise après un certain rang. Au-delà d'un certain rang, toutes les composantes auront une importance équivalente, ce qui rend la projection sur un seul axe et la projection sur deux axes susceptibles de conduire à une perte d'information.

9. **Nuage non isotrope :**

9.1 **Génération d'un nuage non isotrope : (de 300 individus)**

.

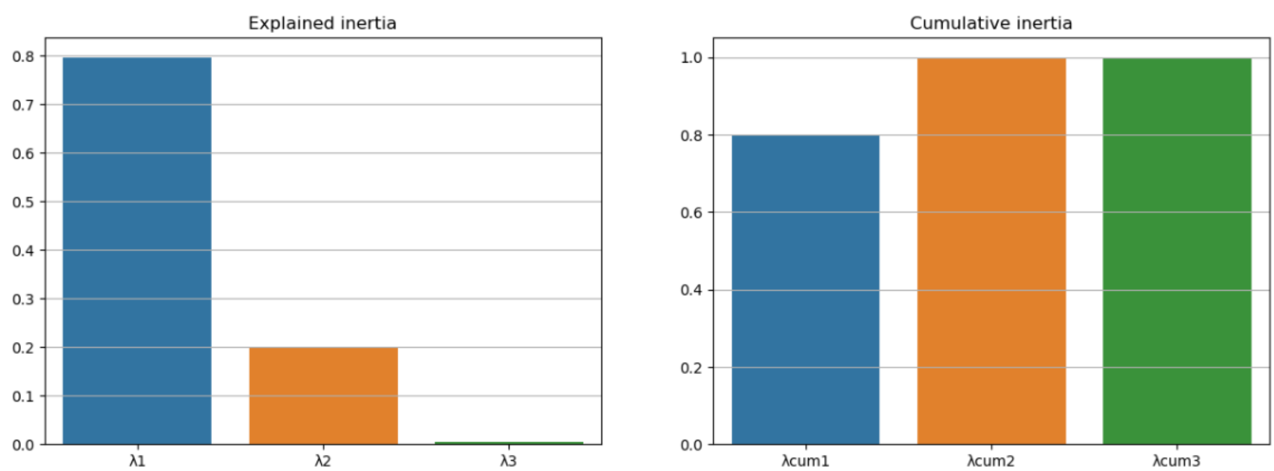
```
n=300
s=5
np.random.seed(s)
X = sorted(np.random.randn(n))
Y = np.random.randn(n)
Z= np.random.randn(n) + np.arctan(X,Y)
S=np.array([X,Y,Z])
# Center the data
S=center(np.transpose(S))
X=S[:,0]
Y=S[:,1]
Z=S[:,2]
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X, Y, Z, c='b', marker='o')
ax.set_xlabel('Axe X')
ax.set_ylabel('Axe Y')
ax.set_zlabel('Axe Z')
plt.show()
```



```
.
# Create the data matrix :
X = S
```

9.2 **ACP centré appliquée au nuage non isotrope :**

```
.
fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.set_title("Explained inertia")
ax1.grid()
ax2.set_title('Cumulative inertia')
ax2.grid()
#Get the eigenvalues of the dataset :
Vp = hyperplans(X)[0]

n=len(Vp)
x = [f'$\lambda${i}' for i in range(1, n+1)]
a = sum(Vp)
y = [Vp[i]/a for i in range(n)]
sb.barplot(x=x, y=y,ax=ax1)

x = [f'$\lambda$cum{i}' for i in range(1, n+1)]
y = [sum(Vp[:i+1])/a for i in range(n)]
sb.barplot(x=x, y=y,ax=ax2)
plt.show()
```

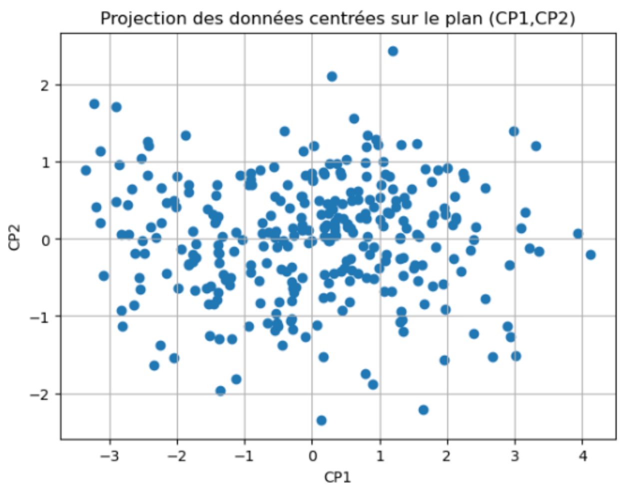


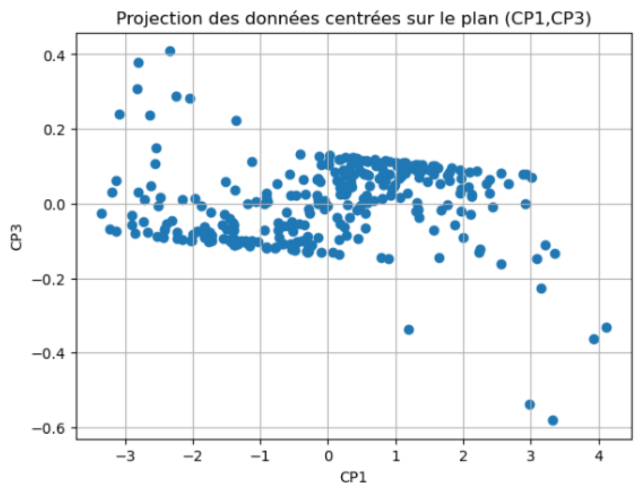
Dans le cas du nuage non isotrope, on observe que la première composante prédomine, expliquant 80% de l'inertie totale. Cela rend la projection sur un seul axe envisageable.

.Projection du nuage sur les plans (CP1,CP2) et (CP1,CP3):

.

```
#Matrix of new coordinates :
Z = projection(X, 0, 3)[0]
fig = plt.figure(figsize=(15, 5))
ax1 = fig.add_subplot(121)
ax1.set_title('Projection on (CP1, CP2)')
ax1.set_xlabel('PC1')
ax1.set_ylabel('PC2')
ax2 = fig.add_subplot(122)
ax2.set_title('Projection on (CP1, CP3)')
ax2.set_xlabel('PC1')
ax2.set_ylabel('PC3')
ax1.scatter(Z[:, 0], Z[:, 1])
ax2.scatter(Z[:, 0], Z[:, 2])
ax1.grid()
ax2.grid()
plt.show()
```

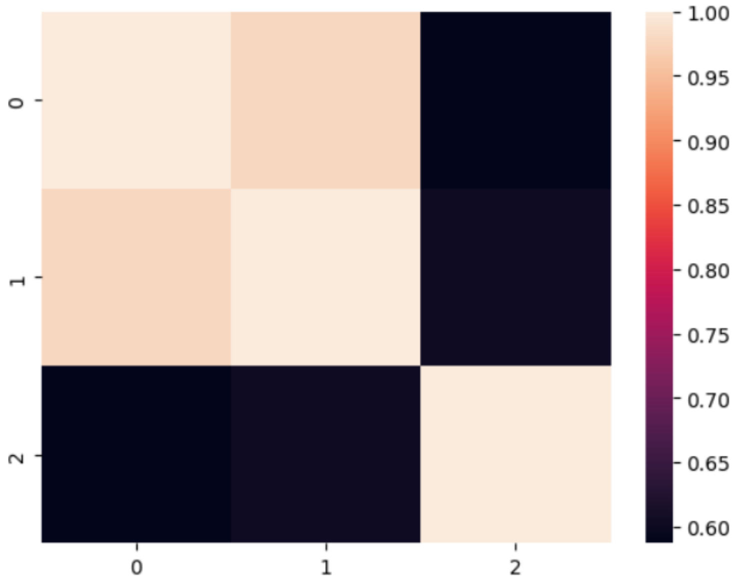




En observant les graphiques, on remarque que la projection sur le plan (CP1, CP2) transmet pratiquement l'intégralité de l'information du nuage, alors que la projection sur le plan (CP1, CP3) révèle une altération de l'information, ce qui est cohérent étant donné que CP3 a une inertie expliquée très faible.

.Visualisation de la matrice de corrélation:

```
#Heatmap of the correlation matrix (300 individuals):
sb.heatmap(corr(X))
plt.show()
```



La forte intercorrelation entre les deux premières variables donne une explication au taux d'inertie élevé expliqué par λ_1.

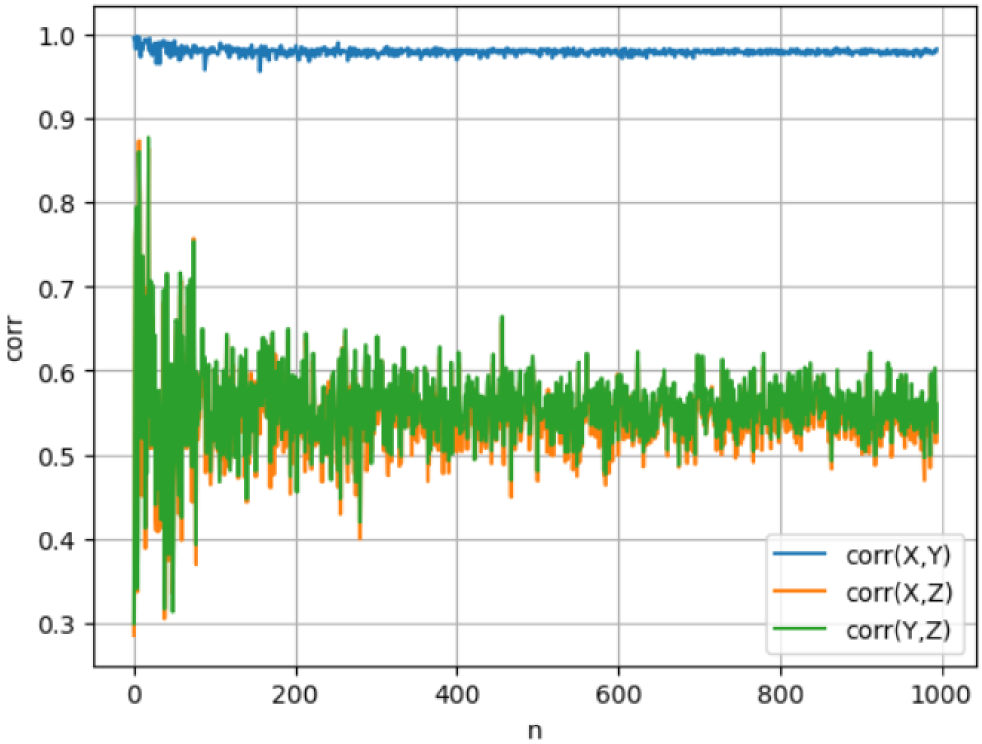
9.3 Evolution de la matrice corrélation avec le nombre d'individus:

.Fonction pour créer un nuage non isotropique :

```
#n is the number of points
def nuage(n):
    X = sorted(np.random.normal(0, 1, (n)))
    Y = np.random.normal(0, 1, (n))
    Z= np.random.normal(0, 1, (n)) + np.arctan(X,Y)
    S=np.array([X,Y,Z])
    #Center data
    S=center(np.transpose(S))
    return S
```

.

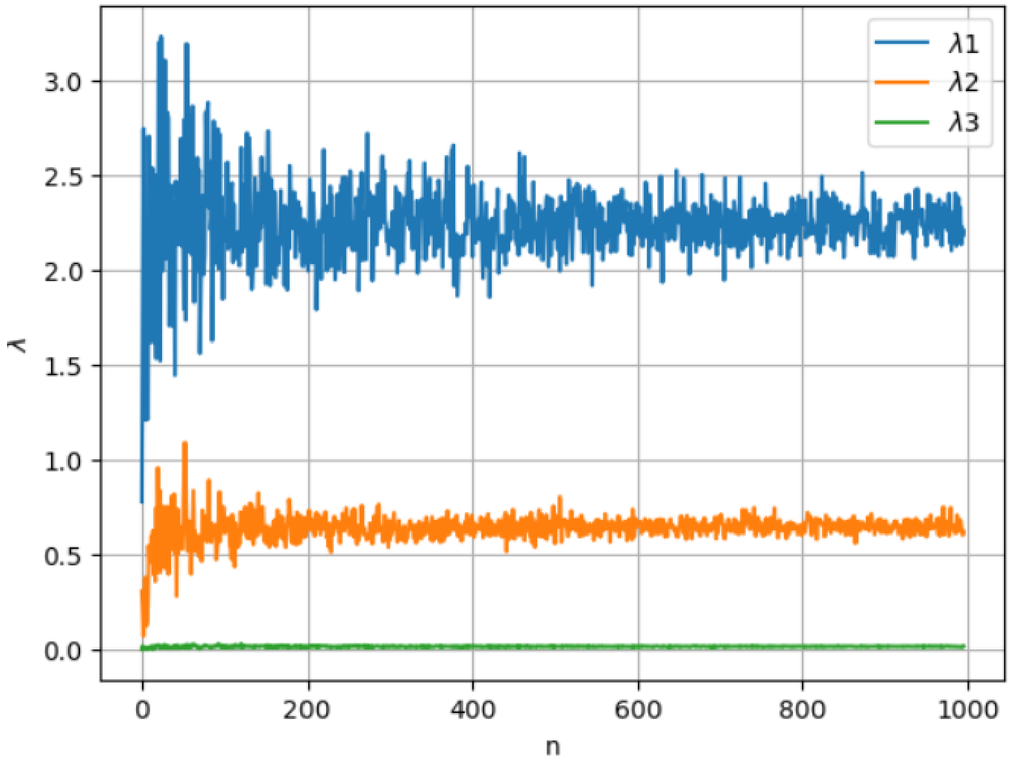
```
L=[]
M=[]
N=[]
for n in range(5,1000):
    T=corr(nuage(n))
    L.append(T[0][1])
    M.append(T[0][2])
    N.append(T[1][2])
plt.grid()
plt.xlabel('n')
plt.ylabel('corr')
plt.plot(L,label='corr(X,Y)')
plt.plot(M,label='corr(X,Z)')
plt.plot(N,label='corr(Y,Z)')
plt.legend()
plt.show()
```



On observe que X et Y sont étroitement liées, indépendamment de la taille de l'échantillon. En revanche, les corrélations avec Z tendent à approcher environ 0.55 à mesure que la taille de l'échantillon augmente, suggérant ainsi que Z représente une variable moins fortement corrélée avec X et Y.

9.4 Cascade des valeurs propres:

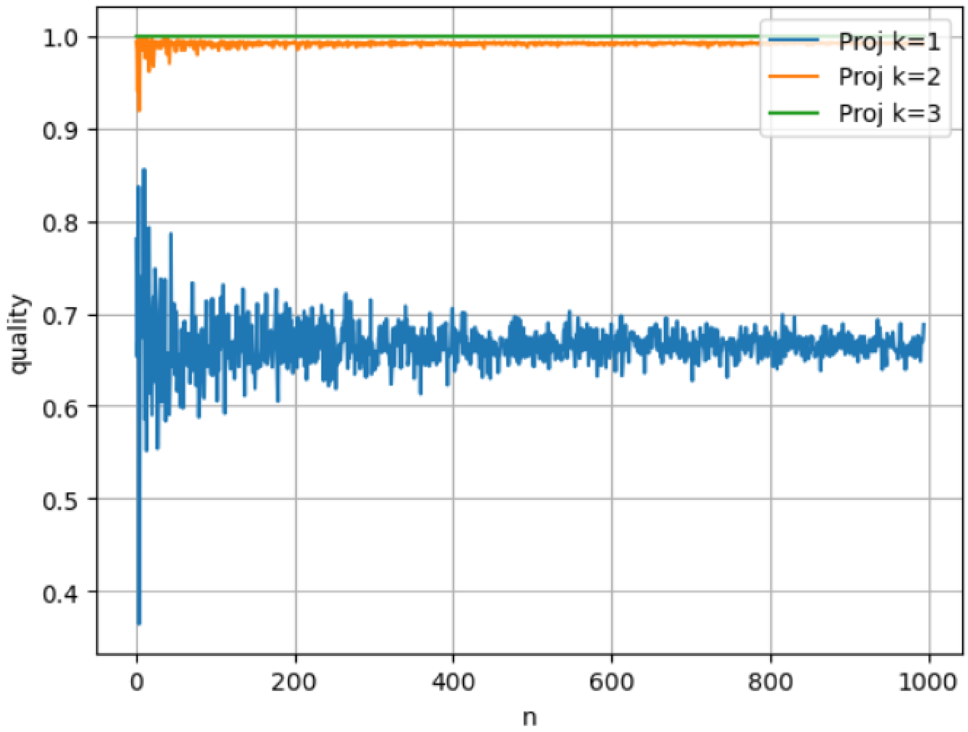
```
L=[]
M=[]
N=[]
for n in range(4,1000):
    #eignvalues
    T=hyperplans(nuage(n))[0]
    L.append(T[0])
    M.append(T[1])
    N.append(T[2])
plt.grid()
plt.xlabel('n')
plt.ylabel('$\lambda$')
plt.plot(L,label='$\lambda$1')
plt.plot(M,label='$\lambda$2')
plt.plot(N,label='$\lambda$3')
plt.legend()
plt.show()
```



La valeur λ_3, qui représente l'inertie apportée par le troisième axe factoriel, est négligeable, comme prévu. En effet, la troisième composante principale explique peu la variance, comme observé précédemment avec le nuage n = 300. Les valeurs λ_1 et λ_2 convergent, par la loi des grands nombres, vers les variances des deux premières composantes principales. Ces valeurs représentent l'inertie apportée par ces deux axes, ce qui est cohérent avec l'étude du nuage n = 300.

9.5 Cascade des moyennes de qualités de projections:

```
.
L=[]
M=[]
N=[]
for n in range(4,1000):
    T=matrix_quality(nuage(n),0)
    L.append(np.mean(T[0]))
    M.append(np.mean(T[1]))
    N.append(np.mean(T[2]))
plt.grid()
plt.xlabel('n')
plt.ylabel('quality')
plt.plot(L,label='Proj k=1')
plt.plot(M,label='Proj k=2')
plt.plot(N,label='Proj k=3')
plt.legend(loc='upper right')
plt.show()
```



En fonction de la tolérance à une qualité de projection d'environ 60% ou non, il est possible de décider de valider l'ACP avec un seul axe principal ou d'opter pour la représentation des données dans le premier plan factoriel CP1/CP2. Ce dernier présente une qualité de projection convergent vers une valeur supérieure à 95%.

10. Nuage non isotrope et points extrémaux :

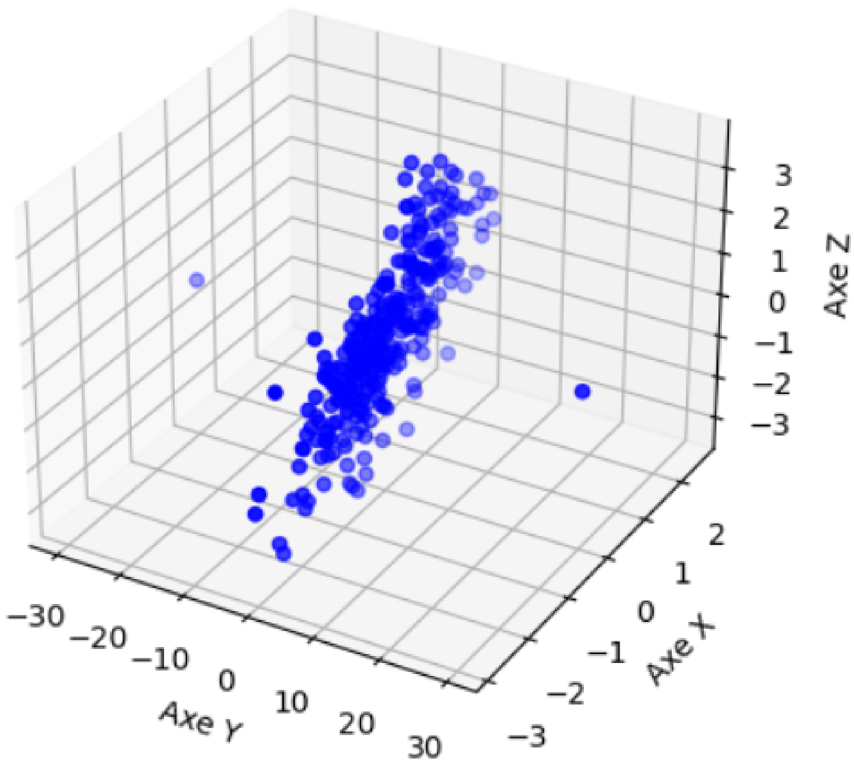
10.1 Ajout de deux points extrémaux au nuage non isotrope :

.Generating the data matrix:

```
n = 300
s = 5
np.random.seed(s)
X = sorted(np.random.normal(0, 1, n))
Y = np.random.normal(0, 1, n)
Z = np.random.normal(0, 1, n) + np.arctan(X, Y)
c = np.mean(Z)
# Adding points: (0, -10, c) and (0, 10, c)
X.extend([0, 0])
Y = Y.tolist()
Y.extend([-10, 10])
Z = Z.tolist()
Z.extend([c, c])
S = np.array([X, Y, Z])
S = np.transpose(S)
#Generate the centered data matrix
X = center(S)
```

. Visualisation du nuage 3D:

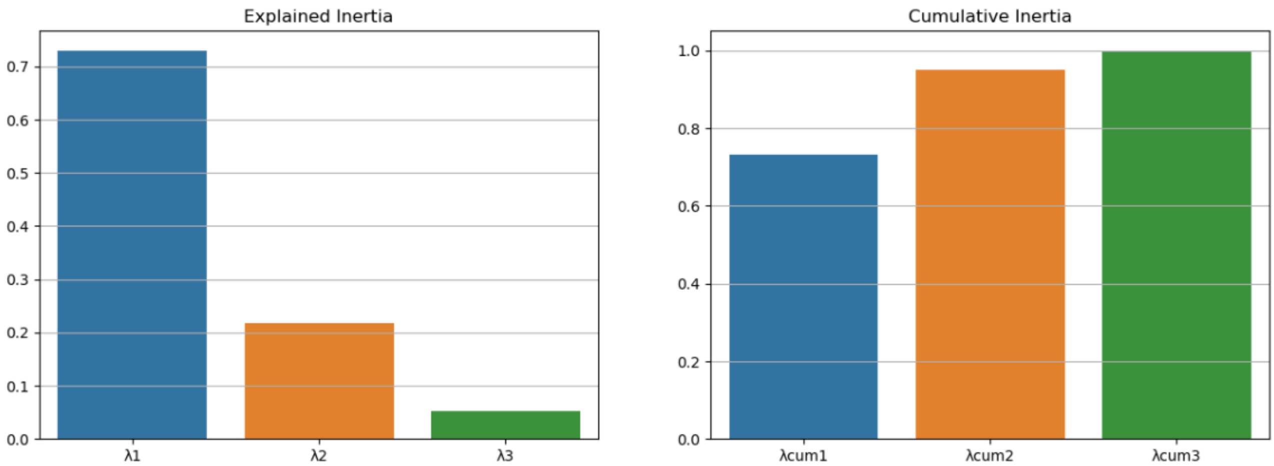
```
x = X[:, 0]
y = X[:, 1]
z = X[:, 2]
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x, y, z, c='b', marker='o')
ax.set_xlabel('X Axis')
ax.set_ylabel('Y Axis')
ax.set_zlabel('Z Axis')
plt.show()
```



10.2 Application de l'ACP centrée et normalisée:

. PCA Centered Case:

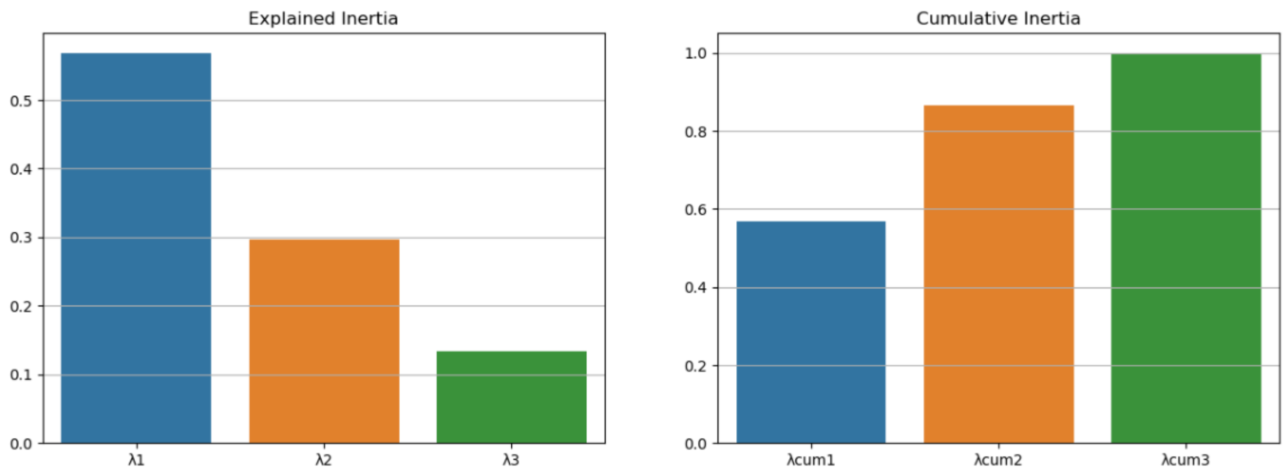
```
fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.set_title("Explained Inertia")
ax1.grid()
ax2.set_title('Cumulative Inertia')
ax2.grid()
#Eignvalues
Vpc = hyperplans(X)[0]
n=len(Vpc)
x = [f'$\lambda${i}' for i in range(1, n+1)]
a = sum(Vpc)
y = [Vpc[i]/a for i in range(n)]
sb.barplot(x=x, y=y,ax=ax1)
x = [f'$\lambda$cum{i}' for i in range(1, n+1)]
y = [sum(Vpc[:i+1])/a for i in range(n)]
sb.barplot(x=x, y=y,ax=ax2)
plt.show()
```



La figure montre clairement une réduction de l'inertie expliquée par la première composante, tandis qu'une augmentation de l'inertie expliquée est observée pour les deux autres composantes. Notamment, on remarque une augmentation significative de l'inertie dans la deuxième composante, c'est la direction où deux points extrêmes ont été ajoutés. Ces observations mettent en évidence la sensibilité de l'ACP à la présence de points extrêmes (outliers), ce qui conduit à une distorsion de l'information. En conséquence, il semble peu approprié d'appliquer l'ACP à un échantillon de données comprenant des points extrêmes, à moins que ceux-ci ne soient exclus préalablement pour garantir l'intégrité de l'analyse.

. PCA Normalised Case:

```python
# Normalise Data Matrix
X=cennor(X)
fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.set_title("Explained Inertia")
ax1.grid()
ax2.set_title('Cumulative Inertia')
ax2.grid()
#Eignvalues :
Vpn = hyperplans(X)[0]
n=len(Vpn)
a = sum(Vpn)
x = [f'$\lambda${i}' for i in range(1, n+1)]
y = [Vpn[i]/a for i in range(n)]
sb.barplot(x=x, y=y,ax=ax1)
x = [f'$\lambda$cum{i}' for i in range(1, n+1)]
y = [sum(Vpn[:i+1])/a for i in range(n)]
sb.barplot(x=x, y=y,ax=ax2)
plt.show()
```



Lors de l'application de l'ACP normalisée à un nuage de données non isotrope, les résultats ont révélé des changements significatifs par rapport à l'ACP centrée. Une observation préoccupante est la forte diminution de l'inertie expliquée par la première composante, contrastant avec une augmentation considérable pour les deux autres composantes. Ces variations importantes témoignent de la sensibilité accrue de l'ACP normalisée dans ce contexte spécifique.

En particulier, la présence de points extrêmes dans l'analyse a accentué cette sensibilité. La normalisation, combinée à la présence de ces points, a entraîné une sorte d'uniformisation des écarts entre les composantes (qui n'est pas cohérente), induisant ainsi une réduction significative de l'inertie expliquée par la composante principale initiale.

Cette observation souligne davantage l'importance d'éliminer les points extrêmes.

4 Partie 4 : Etude de la forme du nuage initiale sur la réduction de dimension dans les deux espaces :

11.

Avant d'entamer cette section, nous désirons introduire des modifications aux fonctions déjà utilisées afin de prendre en compte la pénalisation de la matrice des données par $\sqrt{n}$, où n représente le nombre d'individus, pour pouvoir élaborer l'implémentation de l'ACP dual.

11.1 Implémentation de l'ACP dual :

. We recall that :

```

def sorted_eig_val_vect(A):
    eigenvalues, eigenvectors = np.linalg.eig(A)
    s = sorted(eigenvalues, reverse=True)
    s = np.array(s)
    sorted_indices = np.argsort(eigenvalues)[::-1]
    sorted_eigenvec = eigenvectors[:, sorted_indices]
    s = s.astype(np.float64)
    sorted_eigenvectors =sorted_eigenvec.astype(np.float64)
    return s, sorted_eigenvectors
```

astype(np.float64) is eliminating the imaginary part. When applying ACP to 3 individuals with 300 parameters, the function return the values with an imaginary part equal to 0, so removing the imaginary part will not affect our analysis.

.Two new functions are constructed :

```
# X matrix must be centered and scaled by the square root
# of the number of individuals
def cov_dual(X):
    # Calculates the covariance matrix
    b = np.dot(np.transpose(X), X)
    return b

def hyperplans_dual(X):
    # Computes eigenvalues and eigenvectors from the
    # covariance matrix (X matrix must be preprocessed)
    A = cov_dual(X)
    return sorted_eig_val_vect(A)
```

Pour standardiser la matrice de données X, nous appliquerons la fonction "cennor" définie dans la section 1 (Consultez la page 5).

.The projection function :

```
def projection_dual(X,k):
    # We project onto the space of dimension k and then
    # dimension p
    P = hyperplans_dual(X)[1]
    projection_k = np.dot(X, P[:,:k])
    full_projection = np.dot(X, P)
    return projection_k, full_projection
```

11.2 Application de l'ACP dual sur le nuage isotrope :

11.2.1 Génération du nuage isotrope :

```
n = 300
s = 5
np.random.seed(s)
S = np.random.randn(3,n)

# Normalize the data points to get a sphere :
a = np.sqrt(np.sum(S**2, axis=0))

# Penalise with sqrt(n):
X = cennor(np.transpose(S/a))/np.sqrt(n)
```

X est la matrice des individus.

11.2.2 Comparaison entre l'ACP et l'ACP dual :

. Eignvalues :

```
# diagonalisation in space R^p :
Vp,_=hyperplans_dual(X)
# diagonalisation in space R^n :
Vn,_=hyperplans_dual(np.transpose(X))
```

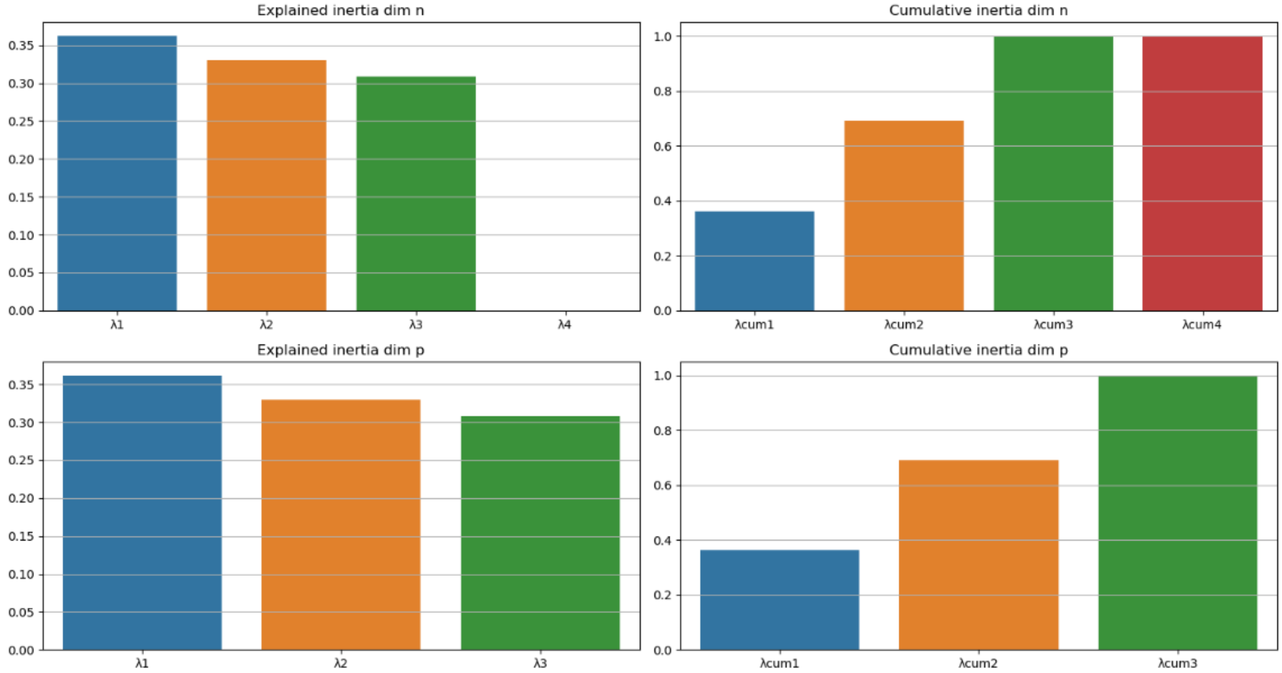
```
1 Vp
array([1.08576131, 0.98948012, 0.92475858])

1 Vn[0:3]
array([1.08576131, 0.98948012, 0.92475858])
```

Il est évident que les trois premières valeurs propres fournies par l'ACP dual correspondent étroitement à celles de l'ACP classique.

. Visualisation des inerties expliquées :

. Code 1 :

```
# Create a figure with a 2x2 subplot layout and set the figure
    ↪ size
fig,((ax1, ax2),(ax3, ax4))=plt.subplots(2,2,figsize=(15, 8))
# Set titles for each subplot
ax1.set_title("Explained inertia dim n")
ax2.set_title('Cumulative inertia dim n')
ax3.set_title("Explained inertia dim p")
ax4.set_title('Cumulative inertia dim p')
# Set grids for each subplot
ax1.grid()
ax2.grid()
ax3.grid()
ax4.grid()
# Eignvalues :
n = 4
x = [f'$\lambda${i}' for i in range(1, n + 1)]
a = sum(Vp)
y = [Vn[i] / a for i in range(n)]
sb.barplot(x=x, y=y, ax=ax1)
x = [f'$\lambda$cum{i}' for i in range(1, n + 1)]
y = [sum(Vn[:i + 1]) / a for i in range(n)]
sb.barplot(x=x, y=y, ax=ax2)
n = len(Vp)
x = [f'$\lambda${i}' for i in range(1, n + 1)]
a = sum(Vp)
y = [Vp[i] / a for i in range(n)]
sb.barplot(x=x, y=y, ax=ax3)
x = [f'$\lambda$cum{i}' for i in range(1, n + 1)]
y = [sum(Vp[:i + 1]) / a for i in range(n)]
sb.barplot(x=x, y=y, ax=ax4)
plt.tight_layout()
# Show the plot
plt.show()
```



Avant de poursuivre, nous tenons à spécifier que si nous faisons référence à un code en utilisant la notation "Code" suivi d'un chiffre, cela signifie que nous avons l'intention de l'utiliser ultérieurement.

11.2.3 Projection des trois individus variables :

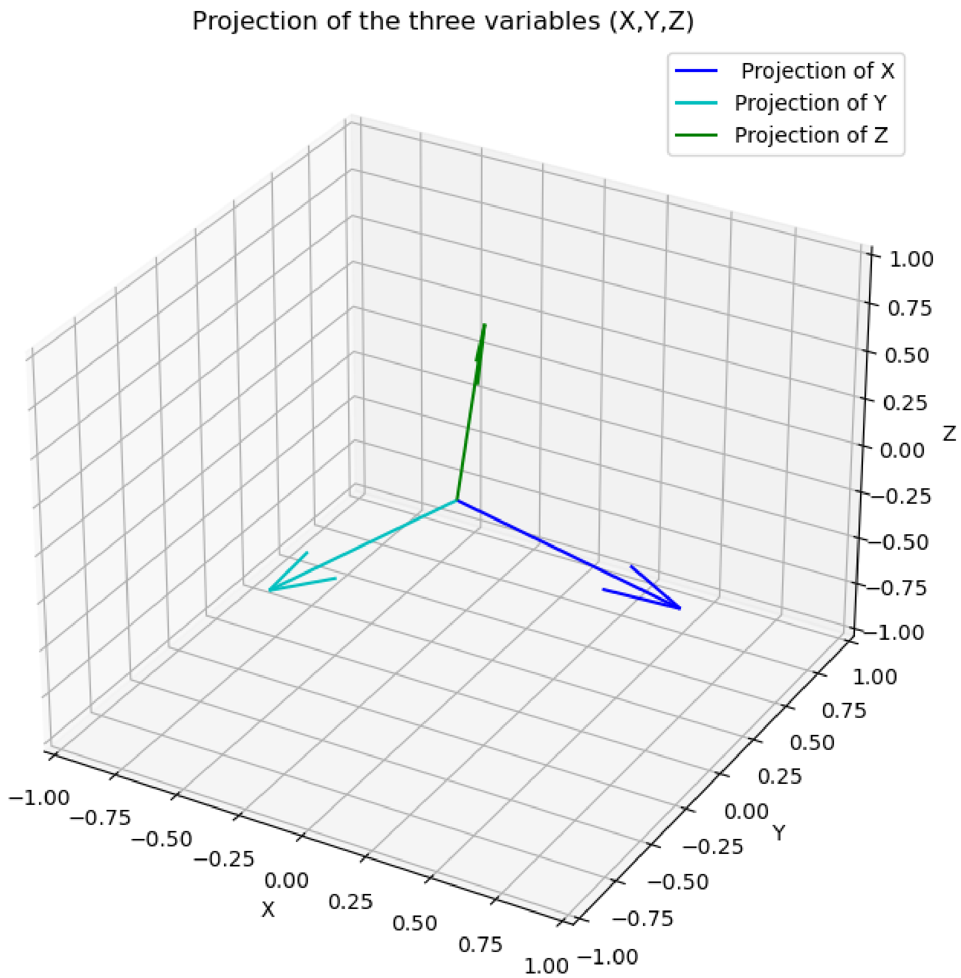
```
# Matrix of variables :
A = np.transpose(X)
# Projection :
Vn,_ = projection_dual(A,3)
```

. Visualisation des trois nouvelles variables en 3D:

. Code 2 :

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111, projection='3d')

# Define the components of the vectors
vector1 = Vn[0]
vector2 = Vn[1]
vector3 = Vn[2]

# Origin of the vectors
origin = np.zeros(3)

# Draw the vectors
ax.quiver(origin[0], origin[1], origin[2],
          vector1[0], vector1[1], vector1[2], color='b', label=
            'Projection of X')
ax.quiver(origin[0], origin[1], origin[2],
```

```
            vector2[0], vector2[1], vector2[2], color='c', label='Projection of Y')
ax.quiver(origin[0], origin[1], origin[2],
            vector3[0], vector3[1], vector3[2], color='g', label='Projection of Z')

# Set the axis limits:
ax.set_xlim([-1, 1])
ax.set_ylim([-1, 1])
ax.set_zlim([-1, 1])

# Label the axes
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

# Display the figure
plt.legend()
plt.title('Projection of the three variables (X,Y,Z)')
plt.show()
```

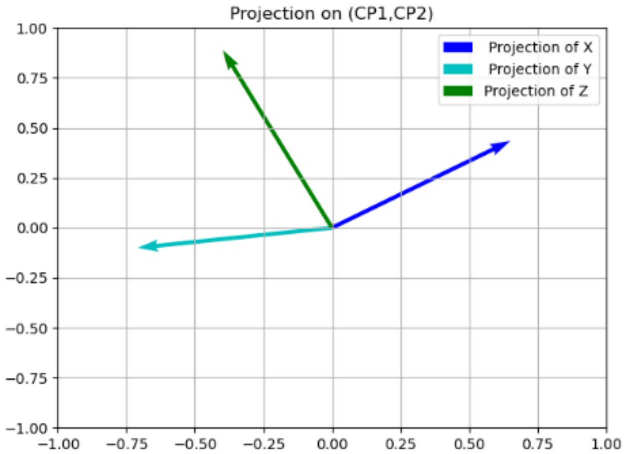


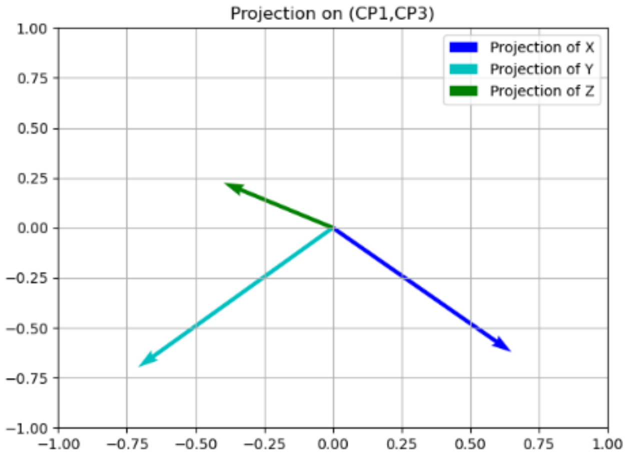
Il est recommandé de représenter nos individus en trois dimensions car les trois composantes principales de l'ACP duale expliquent presque le même pourcentage d'inertie. De plus, on observe qu'aucune direction n'est privilégiée, car les angles entre les vecteurs sont presque égaux.

.Visualisation des trois nouvelles variables dans les premiers plans factoriels:

. Code 3 :

```
fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.set_title("Projection on (CP1,CP2)")
ax2.set_title("Projection on (CP1,CP3)")
# Define the components of the vectors
vector1 = Vn[0]
vector2 = Vn[1]
vector3 = Vn[2]

ax1.grid()
ax2.grid()
# Set the limits of the axes:
ax1.set_xlim([-1, 1])
ax1.set_ylim([-1, 1])
ax2.set_xlim([-1, 1])
ax2.set_ylim([-1, 1])

# Origin of the vectors
origin = np.zeros(2)

# Draw the vectors
ax1.quiver(origin[0], origin[1], vector1[0], vector1[1],
           color='b', angles='xy', scale_units='xy', scale=1, label=' Projection of X')
ax1.quiver(origin[0], origin[1], vector2[0], vector2[1],
           color='c', angles='xy', scale_units='xy', scale=1, label=' Projection of Y')
ax1.quiver(origin[0], origin[1], vector3[0], vector3[1],
           color='g', angles='xy', scale_units='xy', scale=1, label='Projection of Z')

ax2.quiver(origin[0], origin[1], vector1[0], vector1[2],
           color='b', angles='xy', scale_units='xy', scale=1, label='Projection of X')
ax2.quiver(origin[0], origin[1], vector2[0], vector2[2],
           color='c', angles='xy', scale_units='xy', scale=1, label='Projection of Y')
ax2.quiver(origin[0], origin[1], vector3[0], vector3[2],
           color='g', angles='xy', scale_units='xy', scale=1, label='Projection of Z')

ax1.legend()
ax2.legend()

# Display the figure
plt.show()
```





Les deux figures affichent presque la même forme de vecteur, décalée et avec une certaine rotation, ce qui suggère qu'il est essentiel de ne négliger aucune composante pour préserver l'information.

11.3 Formules de passage :

. Defining functions to transform eignvectors in R^p to eignvectors in R^n and vice versa :

```python
# X should be a matrix of normalized data and penalized by sqrt
    (n)
def passage(X):
    # a list of eignvalues
    # b matrix of eignvectors
    a, b = hyperplans_dual(X)
    c = np.dot(X, b) / np.sqrt(a)
    return c
```

```python
# Checking transformation from n to p:
# X is the matrix of individuals
print(hyperplans_dual(X)[1])
```

```
[[-0.62495484  0.43698254  0.64689853]
 [ 0.68061785 -0.10086146  0.72566267]
 [ 0.38234904  0.89379708 -0.23438426]]
```

```python
# np.transpose(X) is the matrix of variables
print(passage(np.transpose(X))[:,:3])
```

```
[[ 0.62495484  0.43698254 -0.64689853]
 [-0.68061785 -0.10086146 -0.72566267]
 [-0.38234904  0.89379708  0.23438426]]
```

En regroupant les vecteurs par colonnes, on observe que les mêmes vecteurs apparaissent multipliés par ϵ dans l'ensemble $\{-1, 1\}$, ce qui confirme la validité de notre fonction de passage.

11.4 Ajout d'invidus et de variables supplémentaires (Cas du nuage isotrope):

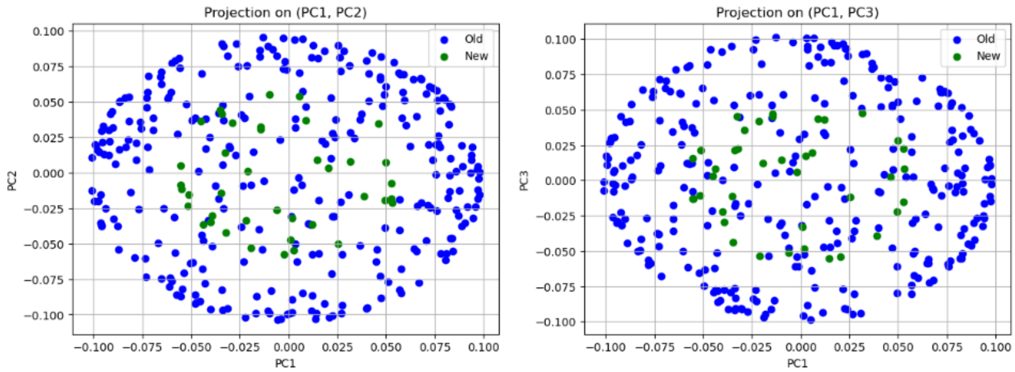
. Adding 40 individuals and 2 variables :

```
n, _ = np.shape(X)
# 40 additional individuals:
new_individuals = np.random.normal(0, 1, (40, 3))
U = np.transpose(new_individuals)
t = np.sqrt(np.sum(U**2, axis=0))
new_individuals = np.transpose(U / t)/np.sqrt(n)

# 2 additional variables
s = 5
np.random.randn(s)
new_variables = np.random.normal(0, 1, (2, 300)) / np.sqrt(n)
```

.Visualisation de la projection des nouveaux individus :

. Code 4 :

```
# Projection of the old individuals
Z1 = projection_dual(X, 3)[0]
fig = plt.figure(figsize=(15, 5))
ax1 = fig.add_subplot(121)
ax1.set_title('Projection on (PC1, PC2)')
ax1.set_xlabel('PC1')
ax1.set_ylabel('PC2')
ax2 = fig.add_subplot(122)
ax2.set_title('Projection on (PC1, PC3)')
ax2.set_xlabel('PC1')
ax2.set_ylabel('PC3')
ax1.scatter(Z1[:, 0], Z1[:, 1], c='b',label='Old')
ax2.scatter(Z1[:, 0], Z1[:, 2], c='b',label='Old')
ax1.grid()
ax2.grid()

# Projection of the new individuals
E = hyperplans_dual(X)[1]
Z2 = np.dot(new_individuals, E)
ax1.scatter(Z2[:, 0], Z2[:, 1], c='g',label='New')
ax2.scatter(Z2[:, 0], Z2[:, 2], c='g',label='New')

ax1.legend()
ax2.legend()
plt.show()
```



Étant donné l'isotropie du nuage, la projection sur les plans factoriels ne présente pas une pertinence élevée, car aucune direction n'est privilégiée.

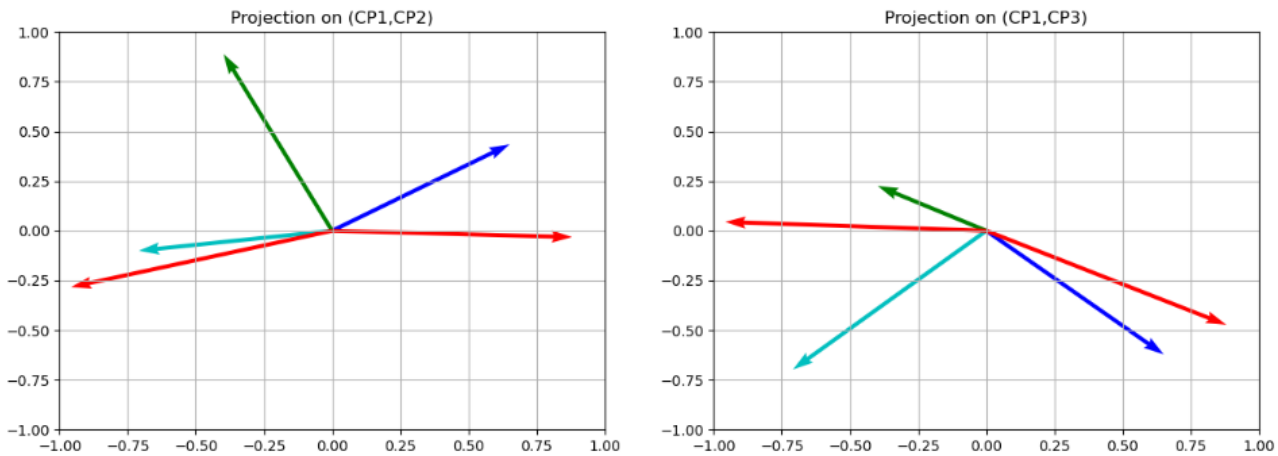
Cependant, il est à noter que les nouveaux individus semblent être représentés aussi efficacement que les anciens individus dans les premiers plans factoriels de l'ACP réalisée avant l'ajout des individus supplémentaires. Cela suggère une robustesse du modèle d'ACP face à l'intégration d'individus similaires.

.Visualisation de la projection des nouveaux variables :

. Code 5 :

```
A = np.transpose(X)
Vn,_ = projection_dual(A,3)
B= hyperplans_dual(A)[1][:,:3]
Vn_new = np.dot(new_variables,B)
# Visualisation de la projection des nouvelles variables :

fig = plt.figure(figsize=(15,5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.set_title("Projection on (CP1,CP2)")
ax2.set_title("Projection on (CP1,CP3)")
# Define the components of the vectors

vector1 = Vn[0]
vector2 = Vn[1]
vector3 = Vn[2]
vector4 = Vn_new[0]/np.sqrt(np.sum(Vn_new[0]**2))
vector5 = Vn_new[1]/np.sqrt(np.sum(Vn_new[1]**2))
ax1.grid()
ax2.grid()
# Set the limits of the axes:
ax1.set_xlim([-1, 1])
ax1.set_ylim([-1, 1])
ax2.set_xlim([-1, 1])
ax2.set_ylim([-1, 1])

# Origin of the vectors
origin = np.zeros(2)

# Draw the vectors
ax1.quiver(origin[0], origin[1], vector1[0], vector1[1],
           color='b', angles='xy', scale_units='xy', scale=1,
              ↪ label='Variable X')
```

```python
ax1.quiver(origin[0], origin[1], vector2[0], vector2[1],
           color='c', angles='xy', scale_units='xy', scale=1, label='Variable Y')
ax1.quiver(origin[0], origin[1], vector3[0], vector3[1],
           color='g', angles='xy', scale_units='xy', scale=1, label='Variable Z')
ax1.quiver(origin[0], origin[1], vector4[0], vector4[1],
           color='r', angles='xy', scale_units='xy', scale=1, label='Variable Z1')
ax1.quiver(origin[0], origin[1], vector5[0], vector5[1],
           color='r', angles='xy', scale_units='xy', scale=1, label='Variable Z2')


ax2.quiver(origin[0], origin[1], vector1[0], vector1[2],
           color='b', angles='xy', scale_units='xy', scale=1, label='Variable X')
ax2.quiver(origin[0], origin[1], vector2[0], vector2[2],
           color='c', angles='xy', scale_units='xy', scale=1, label='Variable Y')
ax2.quiver(origin[0], origin[1], vector3[0], vector3[2],
           color='g', angles='xy', scale_units='xy', scale=1, label='Variable Z')
ax2.quiver(origin[0], origin[1], vector4[0], vector4[2],
           color='r', angles='xy', scale_units='xy', scale=1, label='Variable Z1')
ax2.quiver(origin[0], origin[1], vector5[0], vector5[2],
           color='r', angles='xy', scale_units='xy', scale=1, label='Variable Z2')

plt.show()
```



Les deux vecteurs en rouge représentent les deux variables que nous avons ajoutées au nuage initial.

Il est observé que ces deux vecteurs adoptent des directions et des sens aléatoires, indépendants des trois variables initiales. Cela est dû au fait que nous travaillons avec un nuage isotrope.

11.5 Application de l'ACP dual sur le nuage non isotrope :

11.5.1 Génération du nuage isotrope :

```
n=300
s=5
np.random.seed(s)
xo = sorted(np.random.randn(n))
yo = np.random.randn(n)
zo = np.random.randn(n) + np.arctan(xo,yo)
X = np.array([xo,yo,zo])
X = cennor(np.transpose(X))/np.sqrt(n)
```

X est la matrice des individus.

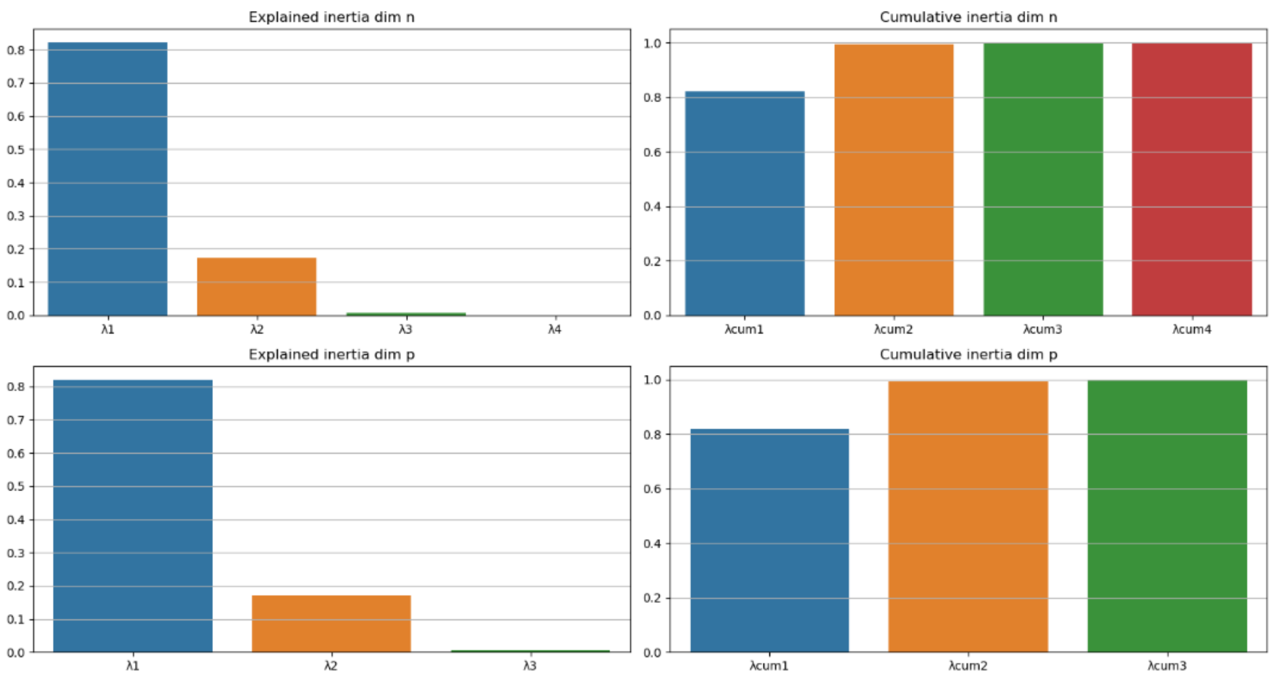
11.5.2 Comparaison entre l'ACP et l'ACP dual :

. Eignvalues :

```
# diagonalisation in space R^p :
Vp,_=hyperplans_dual(X)
# diagonalisation in space R^n :
Vn,_=hyperplans_dual(np.transpose(X))
```

.Visualisation des inerties expliquées :

En utilisant Code 1, on obtient :



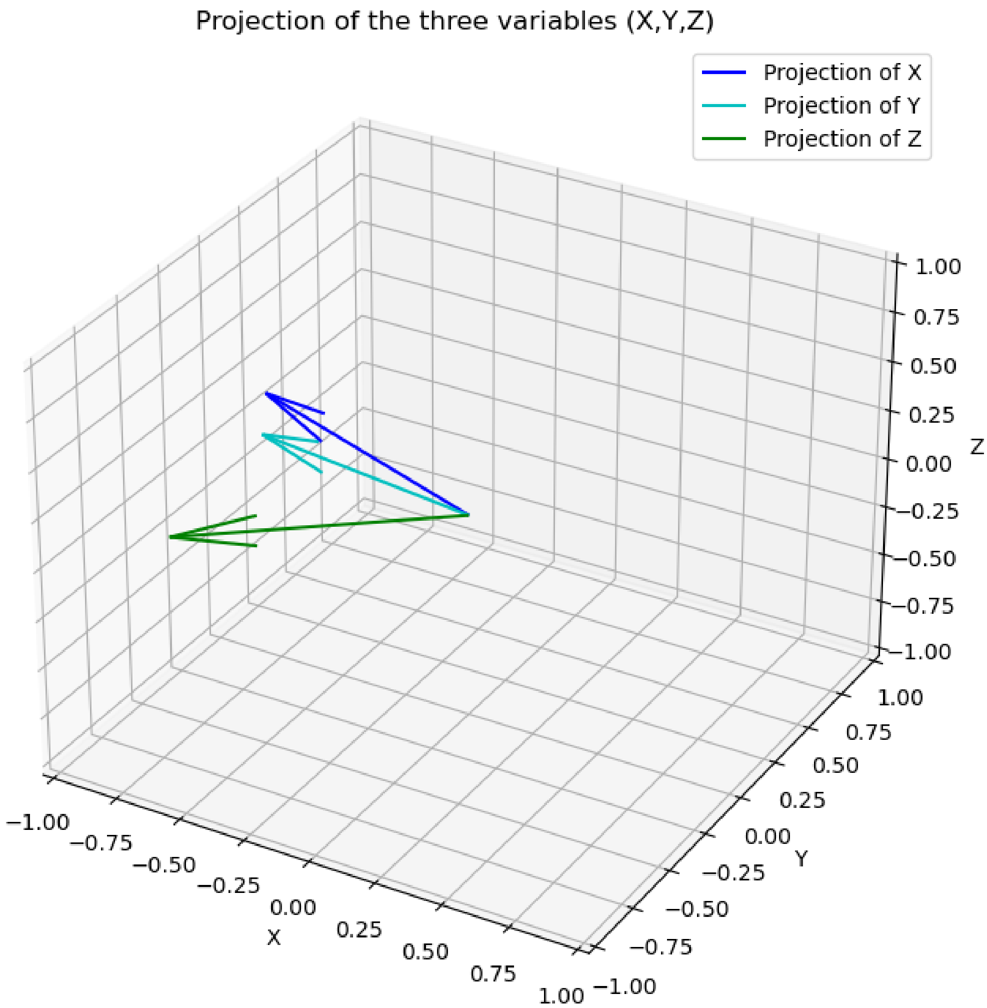
La première composante représente plus de 80% de l'information, tandis que les autres composantes sont moins explicatives dans les deux cas. Ce phénomène est courant pour des ensembles de points non isotropes..

11.5.3 Projection des trois individus variables :

```
# Matrix of variables :
A = np.transpose(X)
# Projection :
Vn,_ = projection_dual(A,3)
```

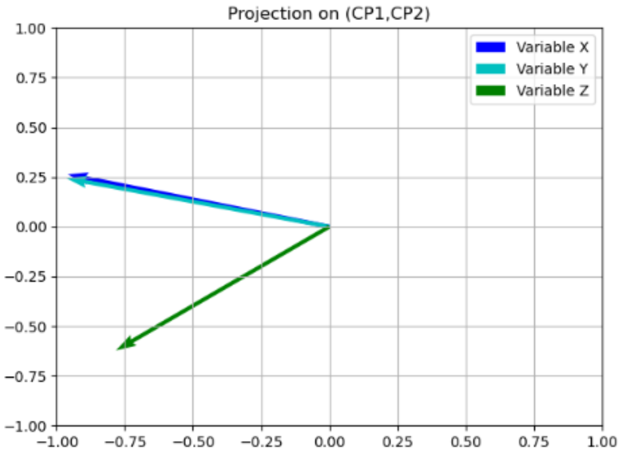
. Visualisation des trois nouvelles variables en 3D:

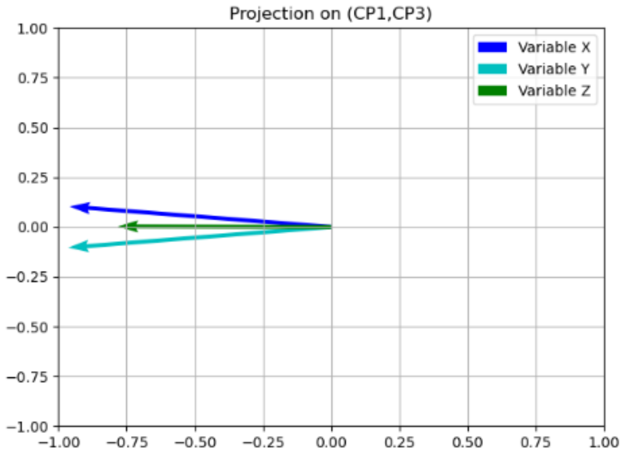
On fait appel à Code 2 :



Comme évoqué la-dessus, la première direction contient la plupart de l'information des 3 points, c'est pour cela il semble que les vecteurs suivent la même direction.

.Visualisation des trois nouvelles variables dans les premiers plans factoriels: On utilise Code 3 :





Le cosinus de l'angle entre deux vecteurs dans l'espace des variables correspond à la corrélation entre ces variables : il est clair que la variable X (en bleu foncé) et la variable Y (en bleu clair) sont fortement liées, et ces deux variables présentent un coefficient de corrélation d'environ 0,5 avec la variable Z (en vert). Ces résultats concordent avec les calculs antérieurs et sont conformes à la définition des trois variables.

En définitive, la projection sur le premier plan factoriel permet une réduction presque totale de la dimension sans perte significative de données.

Vu le taux d'inertie trop faible de la composante 3 par rapport aux 2 autres composantes, il semble que les vecteurs sont presque unidirectionnels dans la figure à droite.

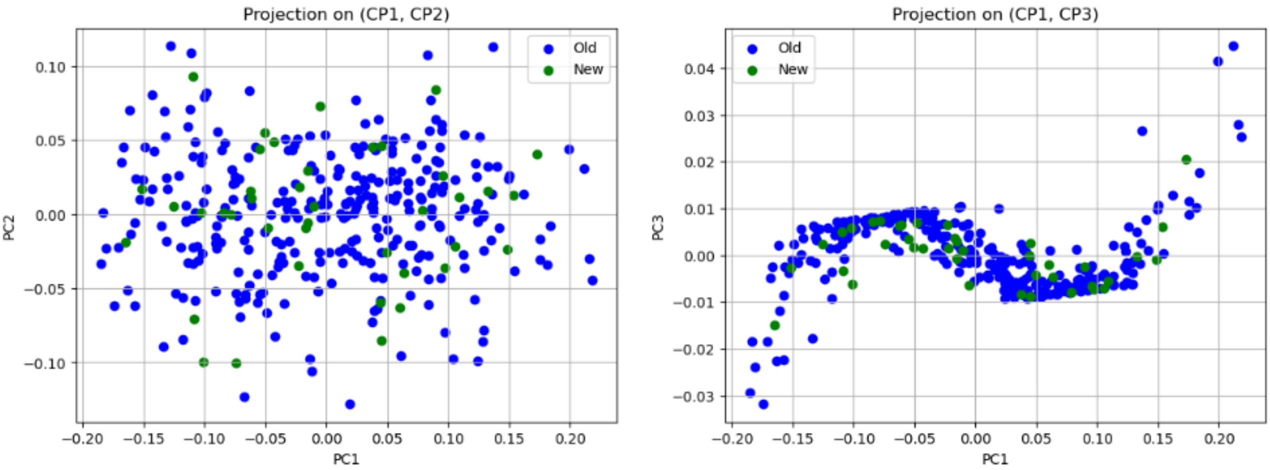
11.6 Ajout d'invidus et de variables supplémentaires (Cas du nuage non isotrope):

. Adding 40 individuals and 2 variables :

```
n, _ = np.shape(X)
# 40 additional individuals:
x = sorted(np.random.randn(40))
y = np.random.randn(40)
z = np.random.randn(40) + np.arctan(x,y)
S = np.array([x,y,z])
new_individuals = cennor(np.transpose(S))/np.sqrt(n)

# 2 additional variables
s=5
np.random.randn(s)
i=xo-5*yo
o=xo+10*zo
new_variables = np.array([i,o])
```

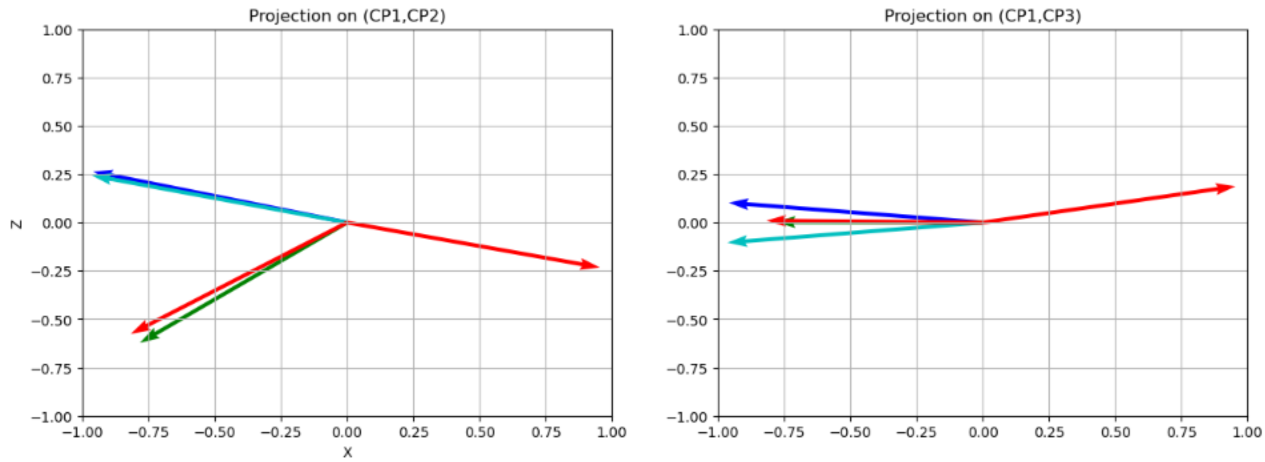
.Visualisation de la projection des nouveaux individus :

Résultat d'exécution du Code 4:



Il est observable que la projection des nouveaux points suit la même tendance de projection que les anciens points dans les deux plans. Cela démontre la robustesse du modèle de l'ACP pour ce nuage de données.

.Visualisation de la projection des nouveaux variables : Résultat du code 5:



Les deux nouvelles variables sont indiquées en rouge.

En raison de l'écart notable du taux d'inertie expliquée entre la deuxième et la troisième composante, on observe des dispersions différentes en projetant sur (CP1, CP3) par rapport à (CP1, CP2), bien que ces dispersions ne soient pas considérablement altérées. Cela s'explique par le fait que plus de 80% de l'inertie est contenue dans CP1. Néanmoins, le plan (CP1, CP2), qui contient presque 98% de l'information, représente davantage la dispersion actuelle des points.

Le vecteur rouge proche du vecteur vert (Z) dénote une forte corrélation avec Z, ce qui est conforme à la construction initiale.

Quant à l'autre vecteur rouge, sa corrélation avec les deux autres est apparente du fait qu'ils partagent presque la même direction. Cette observation s'accorde également avec la conception initiale de ce vecteur.

5 Partie 5 : ACP sur données réelles environnementales :

Dans cette dernière partie du TP, nous allons essayer d'appliquer l'ACP à des données réelles. Il s'agit d'étudier l'impact environnemental d'un site industriel situé dans la Loire. Pour cela, nous disposons des résultats de plusieurs campagnes de mesure portant sur 14 composés organiques volatils à l'extérieur du site. Ces campagnes sont globalement divisées en deux types : BF, réalisées avant la mise en activité du site, et CA, menées après la mise en activité du site.

L'objectif de cette partie est de voir si l'ACP nous permettra de bien visualiser ces mesures en 3D et de bien les distinguer par campagne.

5.1 Gestion de données manquantes :

Premièrement, on supprime les colonnes non exploitables dans la suite, il s'agit des colonnes : 'indice', 'TYPE', 'Localisation'

```
.
# Read Excel file
data = pd.read_excel('TP4_covC1234_DS19_20.xlsx', decimal=',')

# Delete unnecessary columns in the rest of the study
columns_to_drop = ['indice', 'TYPE', 'Localisation']
data = data.drop(columns=columns_to_drop)
```

Pour gérer une donnée manquante, on choisit d'attribuer une valeur à cette donnée au lieu de supprimer toute la ligne

Pour une variable X, les données manquantes sont remplacées par la moyenne de cette variable uniquement pour la campagne concernée, plutôt que par la moyenne de cette variable sur l'ensemble de l'échantillon. Cette approche permet de mieux refléter la réalité, car on peut voir la grande différence entre les mesures des campagnes BF et CA pour toutes les variables.

NB: - Étant donné que les valeurs manquantes (NA) ne sont pas explicitement déclarées dans le tableau, nous considérons toute valeur égale à zéro comme une donnée manquante, après observation des données. Cette hypothèse doit encore être validée.

- D'autres méthodes peuvent être envisagées pour estimer les valeurs manquantes, telles que l'estimation par le modèle de forêt aléatoire.

```
#Function which assigns to each missing data the average during
    ↪ the given period
def impute_missing_values_by_category(data, category_column):
    # Copy data to avoid modifying the original
    data_imputed = data.copy()

    for column in data.columns:
        if column != category_column and column != 'SAISON':
            # Calculate the mean of the variable for each
                ↪ category
            category_means = data.groupby(category_column)[
                ↪ column].mean()

            # Identify indices of missing values (represented
                ↪ by 0)
```

```
            missing_indices = data_imputed[column] == 0

            # Replace missing values with the mean of the corresponding category
            data_imputed.loc[missing_indices, column] = data_imputed.loc[missing_indices, category_column].map(category_means)

    return data_imputed

# Using the function
data_corrige1 = impute_missing_values_by_category(data, 'Campagne')
```

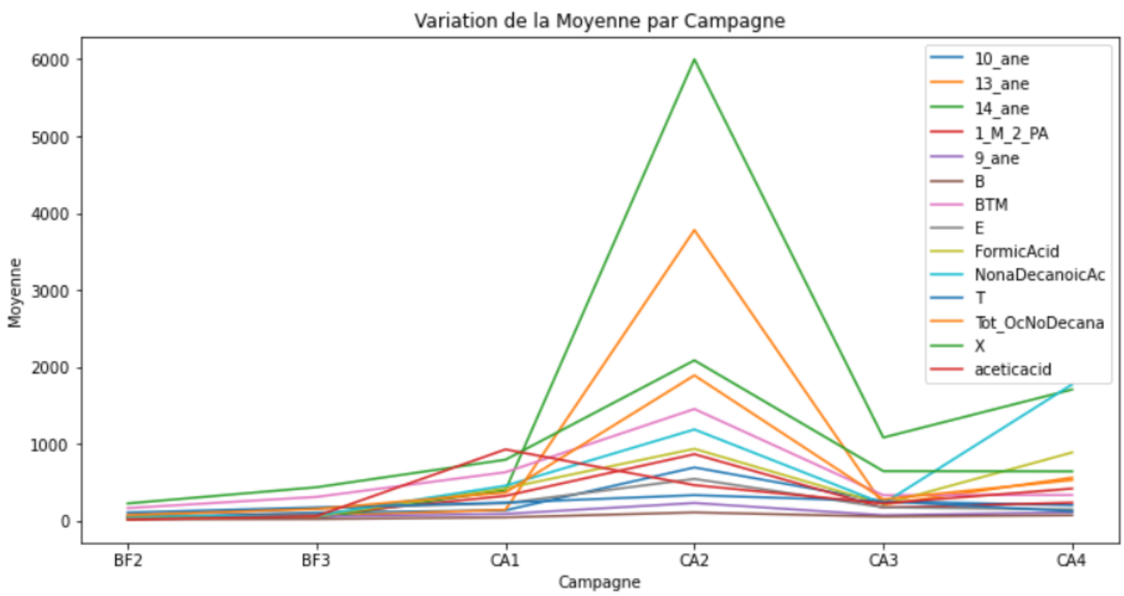
Maintenant on procéde à l'analyse statistique des données.

5.2 Analyse Statistique

Commençant par une analyse des moyennes des mesures de chaque composé organique sur chaque compagne et saison:

```
# Calculer le tableau de moyennes : moyenne de chaque variable par chaque compagne
mean_table = data_corrige1.pivot_table(index=['Campagne', 'SAISON'], aggfunc='mean')

print(mean_table)
# Afficher le tableau de moyennes
plt.figure(figsize=(14, 6))
sb.heatmap(mean_table, annot=True, cmap='viridis')
plt.show()

# Creer un graphique lin aire pour chaque variable
plt.figure(figsize=(12, 6))
for variable in mean_table.columns:
    plt.plot(mean_table.index.levels[0], mean_table[variable], label=variable)


plt.xlabel('Campagne')
plt.ylabel('Moyenne')
plt.title('Variation de la Moyenne par Campagne')
plt.legend()
plt.show()
```

```
                       10_ane       13_ane       14_ane    1_M_2_PA       9_ane  \
Campagne SAISON
BF2      hiver      38.095633    21.414714    42.922017   48.731687   22.407794
BF3      hiver     102.494935    58.145027    30.761745   73.203228   52.480000
CA1      hiver     133.188390   140.238788   396.021912  320.845806   86.275471
CA2      été       690.685507  3779.349060  5999.949335  863.214067  228.061505
CA3      hiver     249.324598   205.654314  1079.762165  167.674146   68.959656
CA4      été       199.658865   558.271084  1704.504659  232.596805  107.425136

                            B          BTM           E  FormicAcid  \
Campagne SAISON
BF2      hiver      14.583022   161.704140   67.970605   29.866504
BF3      hiver      19.354274   307.478029  157.683310   55.655680
CA1      hiver      41.425457   628.427551  225.705478  419.277411
CA2      été       105.556478  1450.934957  541.438629  933.103144
CA3      hiver      52.001739   329.594021  171.947333  213.366315
CA4      été        68.582970   332.063869  144.557390  885.480959

                 NonaDecanoicAc           T  Tot_OcNoDecana            X  \
Campagne SAISON
BF2      hiver        52.691406  104.335816       70.857729   222.529374
BF3      hiver        66.235082  169.982896      149.464736   433.258081
CA1      hiver       453.218301  232.504208      364.740891   791.127223
CA2      été        1184.752086  330.473288     1888.401752  2083.814435
CA3      hiver       220.386903  252.191829      276.281357   643.087978
CA4      été        1771.808397  119.890215      526.405290   640.177626
```



Il est observé qu'après la mise en activité du site, la concentration des composés volatils a significativement augmenté, surtout lors de la deuxième mesure. Par exemple, la concentration du composé 14_ane est passée de 400 à 6000, soit une augmentation de 15 fois.

En appliquant le même analyse sur les écarts types :

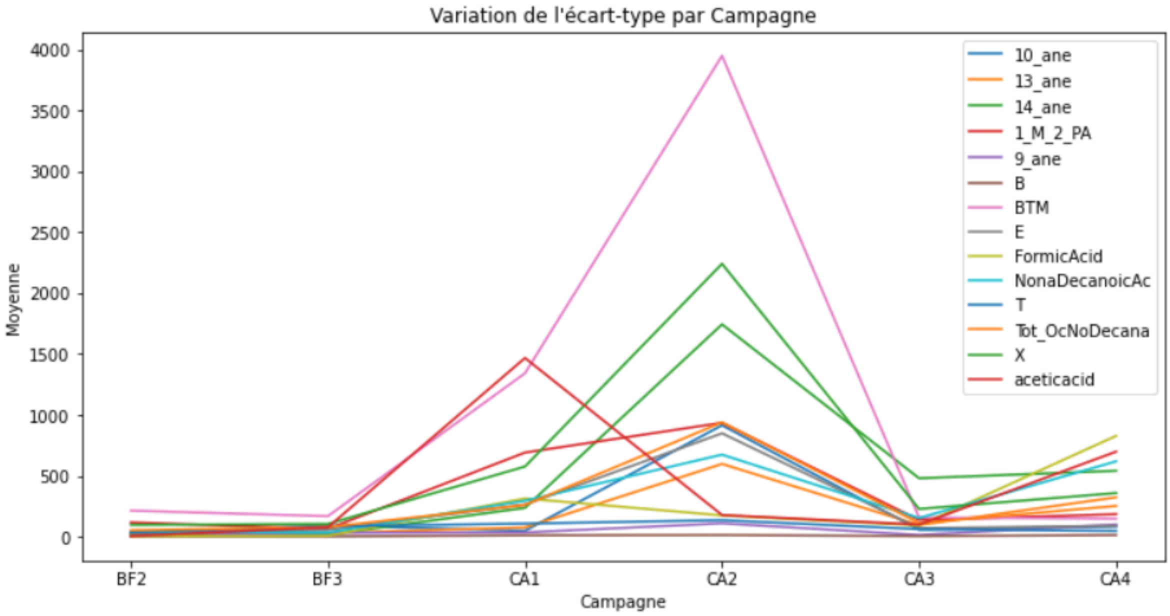
```
# Calculate standard deviation table
std_table = data.pivot_table(index=['Campagne', 'SAISON'],
    ↪ aggfunc='std')

# Show standard deviation table
print(std_table)
```

```python
plt.figure(figsize=(14, 6))
sb.heatmap(std_table, annot=True, cmap='viridis')
plt.show()

# Create a linear graph for each variable
plt.figure(figsize=(12, 6))
for variable in std_table.columns:
    plt.plot(std_table.index.levels[0], std_table[variable],
    ↪ label=variable)

plt.xlabel('Campagne')
plt.ylabel('Moyenne')
plt.title('Variation de l\' cart -type par Campagne')
plt.legend()
plt.show()
```

Campagne	SAISON	10_ane	13_ane	14_ane	1_M_2_PA	9_ane \
BF2	hiver	14.416253	13.670617	34.078403	118.468215	5.897971
BF3	hiver	93.065739	31.029644	18.343182	65.632581	33.067545
CA1	hiver	50.002170	77.160859	237.546559	691.568915	36.273317
CA2	été	915.504764	598.265319	1743.607022	935.850115	110.899129
CA3	hiver	58.790000	96.809514	480.642518	138.023870	15.723494
CA4	été	84.803853	324.033532	542.175875	186.131493	101.322501

Campagne	SAISON	B	BTM	E	FormicAcid \
BF2	hiver	4.734951	214.956614	32.323463	5.256121
BF3	hiver	3.217718	169.638767	56.201735	9.520422
CA1	hiver	12.555213	1342.352733	264.971228	314.826746
CA2	été	14.954645	3947.517071	849.761735	175.847156
CA3	hiver	6.316959	158.390298	71.478691	93.354658
CA4	été	14.554742	149.398661	90.454348	828.418503

Campagne	SAISON	NonaDecanoicAc	T	Tot_OcNoDecana	X \
BF2	hiver	42.674690	37.710805	54.012889	98.056288
BF3	hiver	35.423214	86.342073	79.840961	106.120372
CA1	hiver	296.304494	108.726783	261.644017	577.128302
CA2	été	674.366762	136.864151	939.581127	2240.554741
CA3	hiver	154.313387	63.338080	112.708157	228.876783
CA4	été	619.638208	48.509628	253.702961	359.342216



Même remarque que l'analyse de la moyenne, la différence des écarts types entre les mesures avant et après l'exploitation de site est importante.

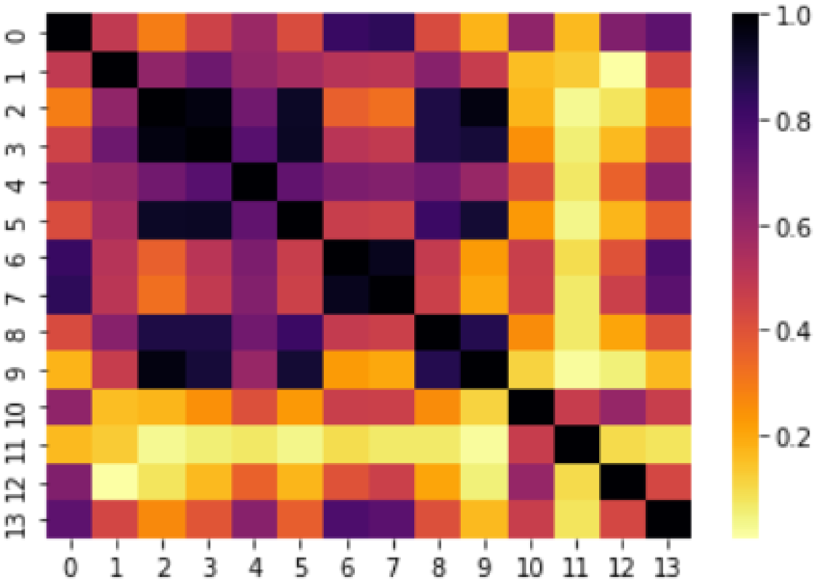
5.3 Application de l'ACP :

Avant de procéder à l'ACP, on supprime les colonnes 'Compagne' et 'SAISON':

```
# Suppression des colonnes 'Compagne' et 'SAISON'
columns_to_drop = ['Campagne', 'SAISON']
data_pour_ACP = data_corrige1.drop(columns=columns_to_drop)
```

Ensuite, nous normalisons nos données étant donné que les mesures sont effectuées à des échelles différentes:

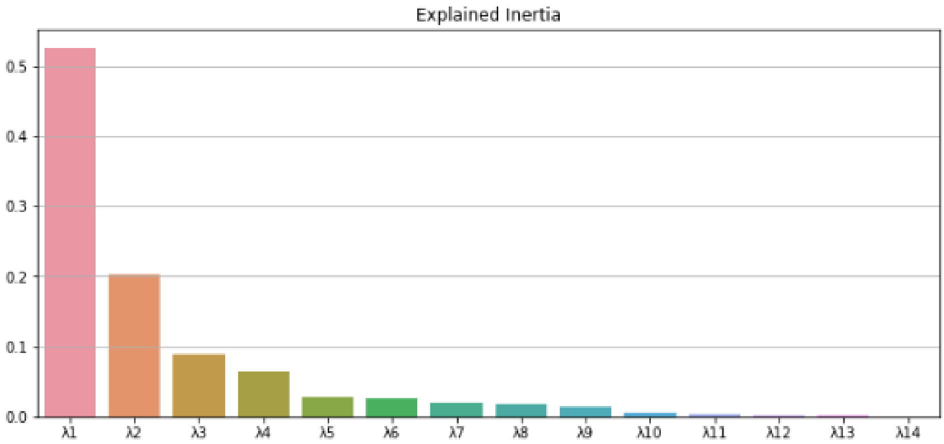
```
# Normalizing the data and visualizing the correlation between
    ↪ the variables

X=data_to_matrix(data_pour_ACP)
X=cennor(X)
sb.heatmap(cov(X),cmap="inferno_r")
plt.show()
```

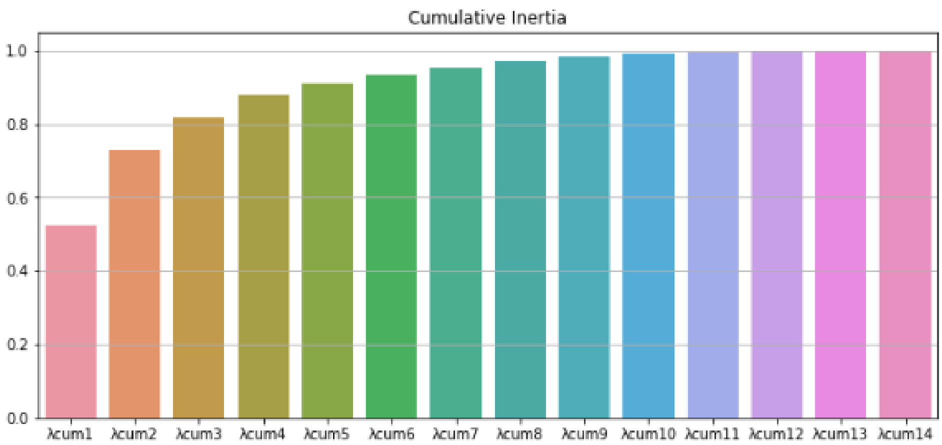


Il est observé une corrélation significative entre différentes variables, ce qui suggère une probable réduction importante des dimensions après l'application de l'ACP.

Visualisons les inerties expliquée par chaque composante, en appliquant l'ACP programmée avant:

```
fig = plt.figure(figsize=(25,5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
ax1.set_title("Explained Inertia")
ax1.grid()
ax2.set_title('Cumulative Inertia')
ax2.grid()
Vpn = hyperplans(X)[0]
n=len(Vpn)
x = [f'  {i}' for i in range(1, n+1)]
a = sum(Vpn)
y = [Vpn[i]/a for i in range(n)]
sb.barplot(x=x, y=y,ax=ax1)
x = [f' cum {i}' for i in range(1, n+1)]
y = [sum(Vpn[:i+1])/a for i in range(n)]
sb.barplot(x=x, y=y,ax=ax2)
plt.show()
```



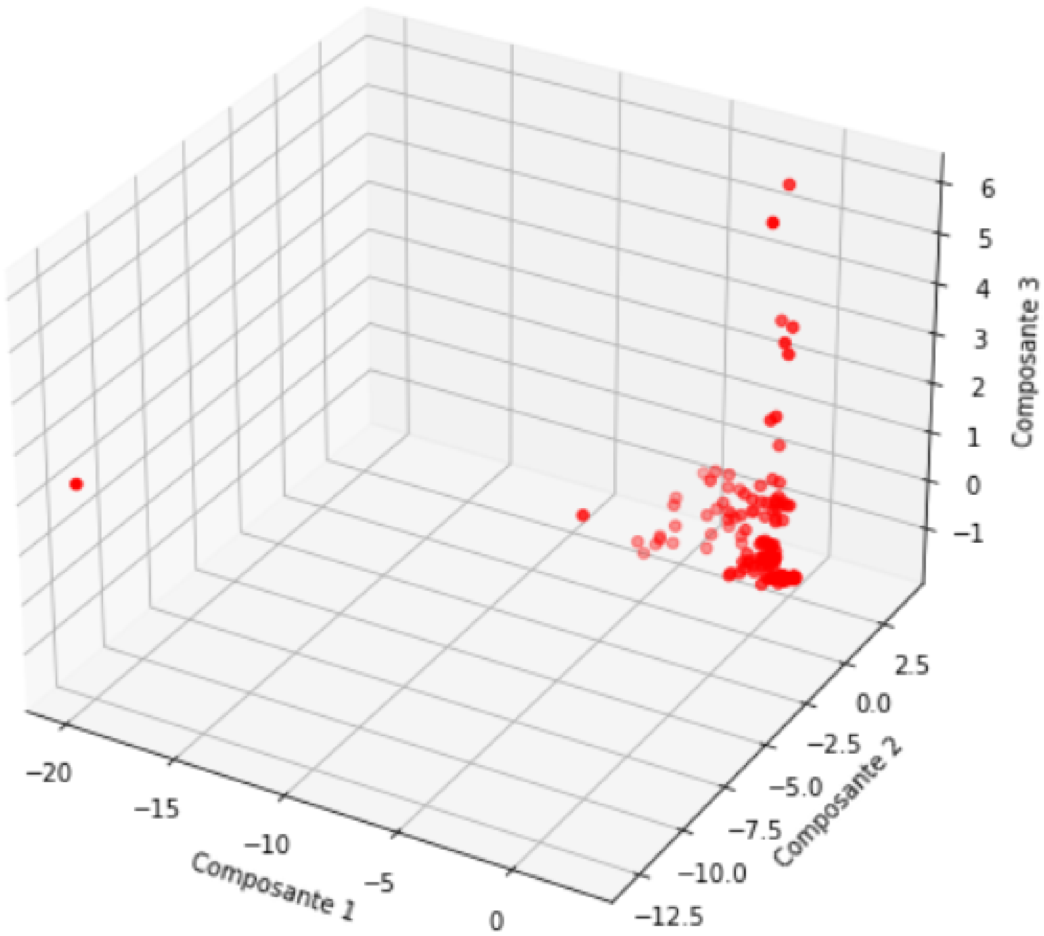


On peut voir que l'inértie cumulée que par la première composante dépasse 50% de l'inertie totale, ce qui nous amenera à une bonne réduction de dimesion.

En appliquant la Régle de Kaiser-Guttman on trouve que k=3.

On projete les points sur les composantes principales :

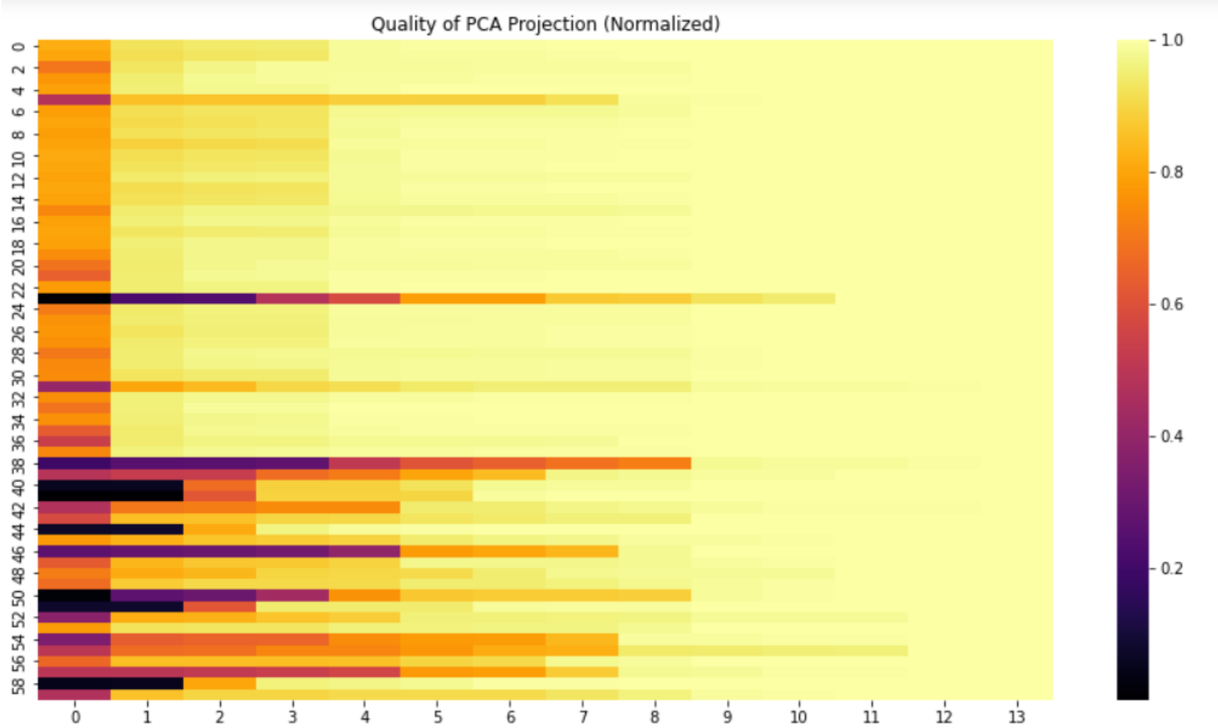
```
#on projecte :
X_k, X_p = projection(X, 1, 3)
# Cr er une figure 3D
fig = plt.figure(figsize=(15, 8))


# Ajouter un sous-plot 3D    la figure
ax = fig.add_subplot(111, projection='3d')
x = X_k[:, 0]
y = X_k[:, 1]
z = X_k[:, 2]
ax.scatter(x, y, z, c='r', marker='o')
ax.set_xlabel('Composante 1')
ax.set_ylabel('Composante 2')
ax.set_zlabel('Composante 3')

# Afficher le plot
plt.show()
```



En examinant la figure, on remarque la présence d'un point extrême bien défini, ainsi que d'autres points similaires. Celui-ci semble contribuer de manière significative à la première et à la deuxième composante. Nous pourrons l'identifier plus tard en examinant la matrice de contribution des points. Cependant, avec ces points extrêmes, la visualisation des mesures est limitée ; nous envisagerons leur élimination ultérieurement.

Visualisons la qualité de la projection pour les 60 premiers individus:

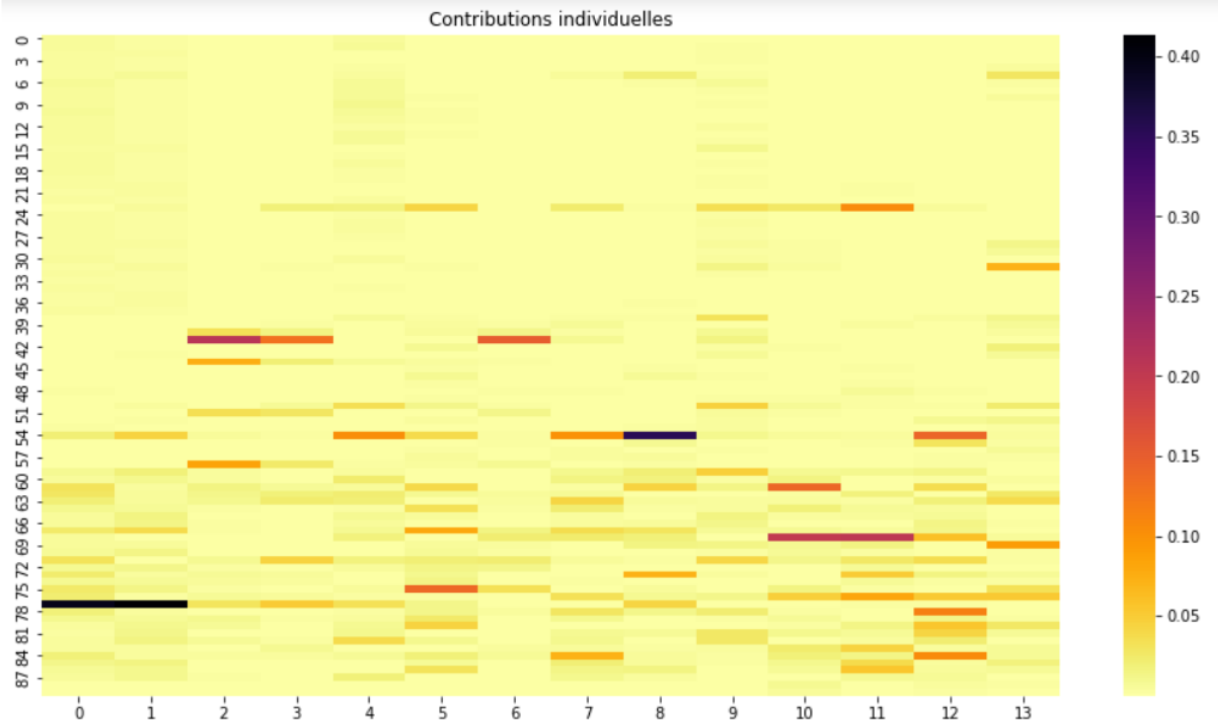
```
#Quality of projection

d = np.transpose(matrix_quality(X, 1))[:60]
fig = plt.figure(figsize=(15, 8))
plt.title("Quality of PCA Projection (Normalized)")
sb.heatmap(d, cmap="inferno")


plt.show()
```



Globalement les individus sont assez bien projetés sur les 3 premières composantes.

Analysons la contribution des points sur les composantes (On visualise ici que les 90 premièrs points):

```
#Contribution :
d=contribution(X,1,14)[:90]
fig = plt.figure(figsize=(15, 8))
plt.title("Contributions individuelles")
sb.heatmap(d, cmap="inferno_r")


plt.show()
```



On peut voir que la mesure N°76 contribue le plus à la première et la deuxième composantes.

Essayons maintenant d'éliminer tous les points extremaux :

On analysant la matrice de contribution, on considère tous les points qui conribue plus que 1% à la première composante comme points extremaux.

NB : à 4% on aura que le point N°76 qui est extremal.

```
seuil=0.01
indices_most_contributed = np.where(d[:, 0] > seuil)[0]
```

Indices des mesures les plus contribuants à la première composante :

[54 61 62 63 64 67 71 73 75 76 77 78 79 84 85]

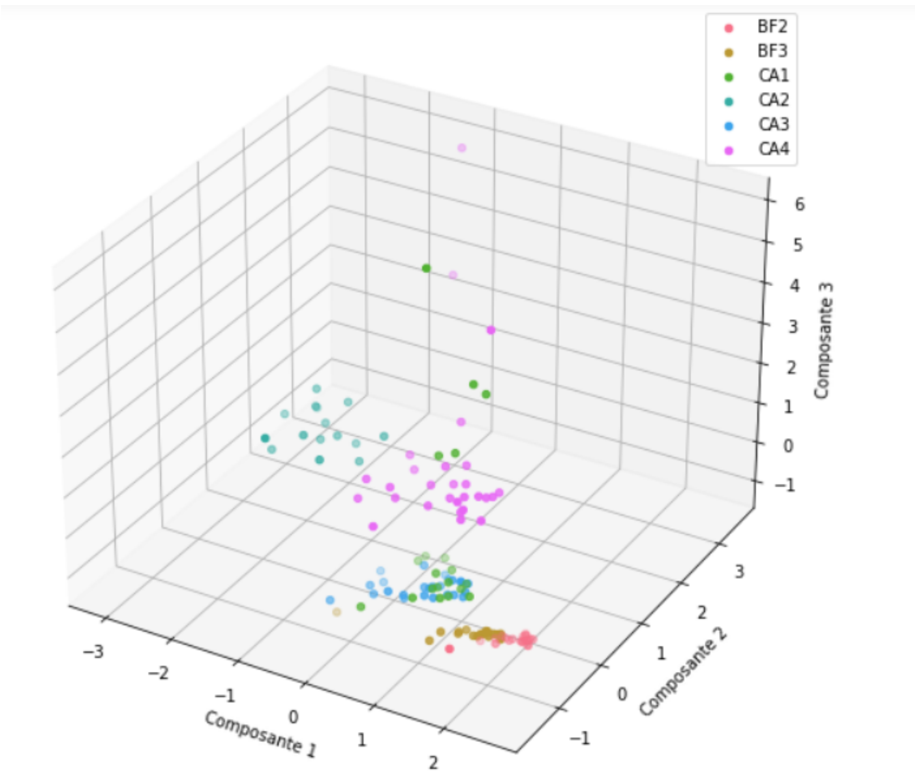
Donc on enlevera ces mesures premièrement à partir des données projetées, et deuxièment à partir des données initiales et on refait l'ACP pour voir s'il y a une différence.

```
#Remove rows from projected individuals

# List of index to delete
indices_a_supprimer = indices_most_contributed

# Delete rows with specified indices
X_k0 = np.delete(X_k, indices_a_supprimer, axis=0)
# Removing rows of extremal individuals from the data:
data_corrige2 = data_corrige1.drop(indices_a_supprimer)


# Ajouter les colonnes de Campagne et saison au donnees projetees:
```

```
# Selection of the last two columns: : Campagne et saison
dernieres_colonnes = data_corrige2.iloc[:, -2:]
# Conversion des deux derni res colonnes en une array numpy
dernieres_colonnes_array = np.array(dernieres_colonnes)
#Ajouter les colonnes au individus projet s avant
result = np.concatenate((X_k0, dernieres_colonnes_array ), axis =1)
```

Maintenant on visualise les données projetées après suppression des points extremaux :

```
df = pd.DataFrame(result)
# Cr ez une figure 3D
fig = plt.figure(figsize=(12, 9))
ax = fig.add_subplot(111, projection='3d')

# Liste des cat gories uniques
categories = df[4].unique()

# Assignez une couleur unique   chaque cat gorie
palette = sb.color_palette("husl", n_colors=len(categories))

# Parcourez les donn es et tracez les points avec des couleurs bas es sur la cat gorie
for i, cat in enumerate(categories):
    subset = df[df[4] == cat]
    ax.scatter(subset[0], subset[1], subset[2], label=f'{cat}', c=[palette[i]])

# Ajoutez des l gendes
ax.legend()

# Ajoutez des  tiquettes  d'axe
ax.set_xlabel('Composante 1')
ax.set_ylabel('Composante 2')
ax.set_zlabel('Composante 3')

plt.show()
```



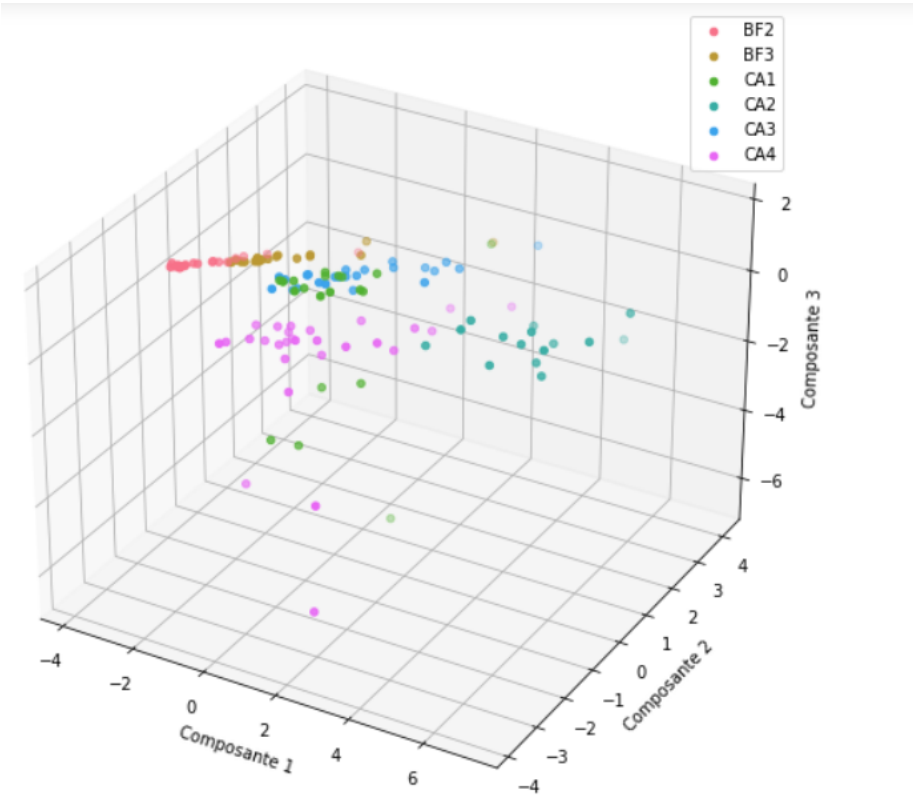
Donc, On peut voir une bonne distinction des données par chaque compagne.

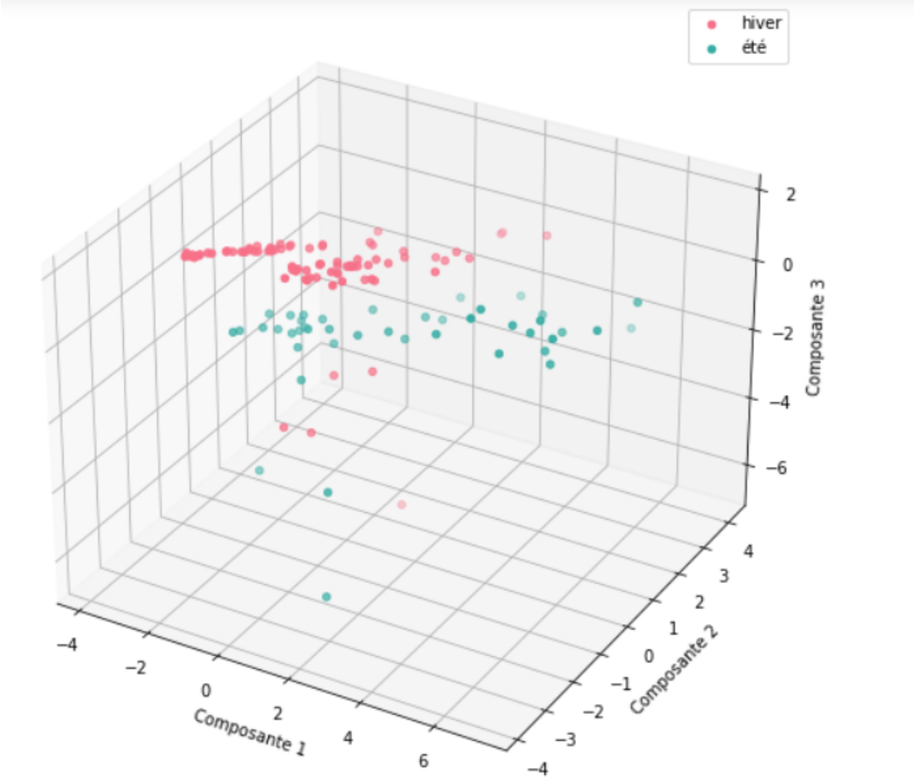
Maintenant on supprime les points extrémaux à partir des données initiales, puis on refait l'ACP

```
#Supprimer les lignes a partir des donnees initiales puis refaire l'ACP

X1=data_to_matrix(data_pour_ACP)
# Liste d'indices   supprimer
indices_a_supprimer = indices_most_contributed

# Supprimer les lignes avec les indices specifies
X1 = np.delete(X1, indices_a_supprimer, axis=0)
X1=cennor(X1)
#on projecte :
X_k1, X_p = projection(X1, 1, 3)   # La fonction 'projection' applique l'ACP dedans puis projecte.

# Ajouter les colonnes de Campagne et saison :

# Selection des deux dernieres colonnes : Campagne et saison
dernieres_colonnes = data_corrige2.iloc[:, -2:]
# Conversion des deux derni res colonnes en une array numpy
dernieres_colonnes_array = np.array(dernieres_colonnes)
#Ajouter les colonnes au individus projetes avant
result1 = np.concatenate((X_k1, dernieres_colonnes_array ), axis=1)
```

```
df1 = pd.DataFrame(result1)
# Cr ez une figure 3D
```

```
fig = plt.figure(figsize=(12, 9))
ax = fig.add_subplot(111, projection='3d')

# Liste des cat gories uniques
categories = df1[4].unique()

# Assignez une couleur unique   chaque cat gorie
palette = sb.color_palette("husl", n_colors=len(categories))

# Parcourez les donn es et tracez les points avec des couleurs bas es sur la cat gorie
for i, cat in enumerate(categories):
    subset = df1[df1[4] == cat]
    ax.scatter(subset[0], subset[1], subset[2], label=f'{cat}', c=[palette[i]])

# Ajoutez des l gendes
ax.legend()

# Ajoutez des  tiquettes  d'axe
ax.set_xlabel('Composante 1')
ax.set_ylabel('Composante 2')
ax.set_zlabel('Composante 3')

plt.show()
```





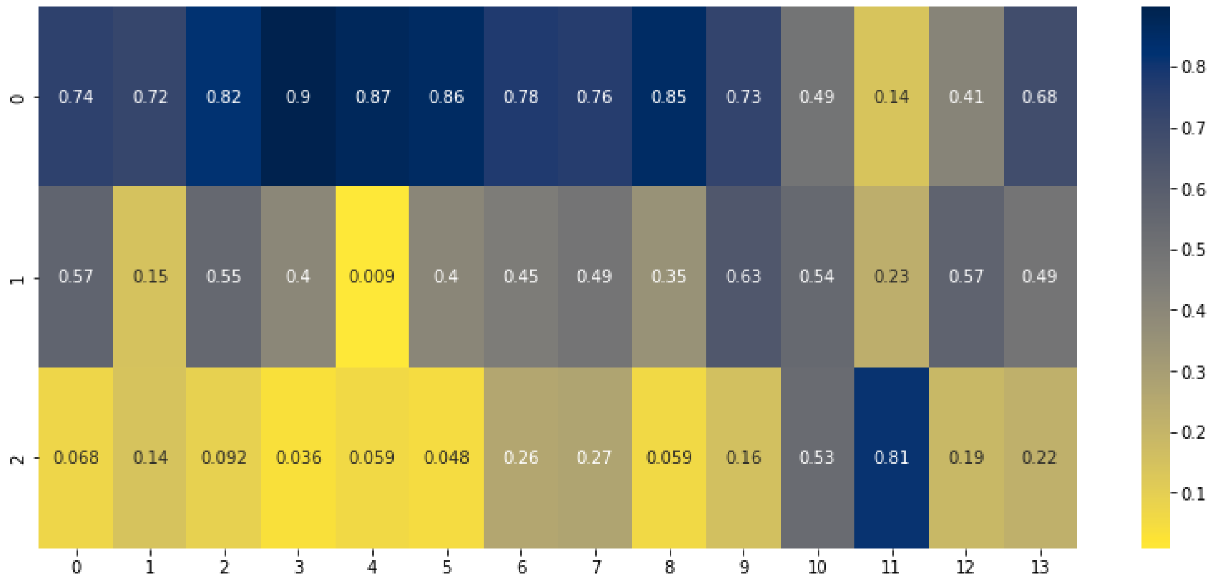
Cette approche nous a aussi permis de bien visualiser les données par chaque compagne et chaque saison.

Conclusion : une réduction du nombre de variables par ACP est une méthode bien adaptée pour donner une signature des composantes pour chaque période (été, hiver, avant activité, apres activité)

Les composantes principales sont des combinaisons linéaires des variables initiales. Maintenant on s'intéresse au regroupements des variables sur chaque composante.

Pour cela on calcule les charges factorielles; les corrélations entre les anciens variables et les nouveaux variables :

```
def calculer_correlation(v1, v2):
    return np.abs(np.corrcoef(v1, v2)[0, 1])
L = []
for i in range(3):
    S = [calculer_correlation(X_k[:, i], X[:, j]) for j in range(14)]
    L += S

L = np.array(L).reshape(3, 14)
plt.figure(figsize=(14, 6))
sb.heatmap(L, annot=True, cmap="cividis_r")
plt.show()
```



On définit 0.5 comme seuil de signification pour les charges factorielles afin de déterminer quelles variables sont les plus importantes pour chaque composante. Si une charge factorielle dépasse 0,5 cela peut indiquer une contribution significative de la variable à la composante.

On constate que la première composante regroupe 11 variables importantes, alors que la deuxième regroupe 5, tandis que la dernière regroupe que 2 variables importantes.