Majeure *Science des données* 2020\_21

Compte-rendu du TP n°1 - Analyse en composantes principales [[1]](#footnote-1)

**Noms du groupe : AMINI Nada ESSAYEGH Nour**

1. **Programmer l’ACP sur l’espace de variables :**

*Fournir les pseudo code et les scripts développés et répondre aux questions :*

1. - Visualiser la matrice de dimension (*n,p*) ( *n* individus et *p* variables)
2. - Construire les indicateurs statistiques classiques : variance, covariance, écart-type

# Question 1

import pandas as pd

D = pd.read\_csv("data\_PDE20.txt", delimiter = " ")

print("La data frame est: \n")

print(D)

# On a modifié les espaces du fichier .txt donné. Plusieurs espaces séparaient certains chiffres, on les a tous convertit à un seul espace «  » qu’on a par la suite utilisé comme délimiteur. Ensuite on a modifié la virgule des nombres décimaux en point car sinon le code affichait des erreurs

# Question 2

# Construction des indicateurs statistiques classiques :

# La moyenne :

import numpy as np

def moyenne(df):

n, m = df.shape

moyenne = [sum(df.loc[:,"X{}".format(i+1)])/n for i in range(m)]

return moyenne

m1 = moyenne(D)

print("\nLa moyenne de D est: \n" + str(m1))

# La variance et l'écart-type

def variance(df):

n, m = df.shape

moy = moyenne(df)

variance = [sum(df.loc[:,"X{}".format(i+1)]\*\*2)/n - moy[i]\*\*2 for i in range(m)]

ecart\_type = [np.sqrt(sum(df.loc[:,"X{}".format(i+1)]\*\*2)/n - moy[i]\*\*2) for i in range(m)]

return variance,ecart\_type

v1 = variance(D)[0]

e1 = variance(D)[1]

print("\nLa variance de D est: \n" + str(v1))

print("\nL'écart-type de D est: \n" + str(e1))

# La covariance

def covariance(df,i,j):

moy = moyenne(df)

n = df.shape[0]

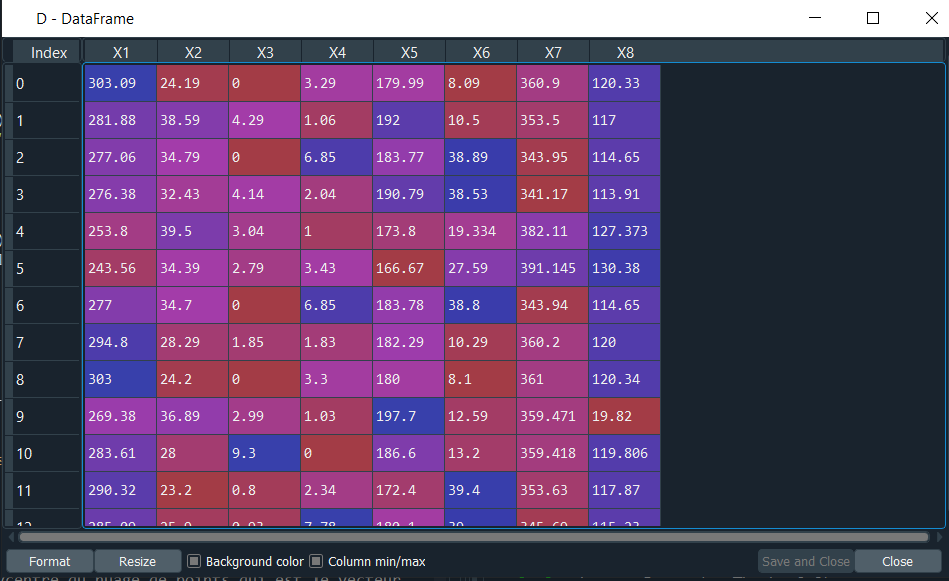
return sum((df.loc[:,"X{}".format(i+1)]- moy[i])\*(df.loc[:,"X{}".format(j+1)]- moy[j]))/n

cov = covariance(D, 1, 5)

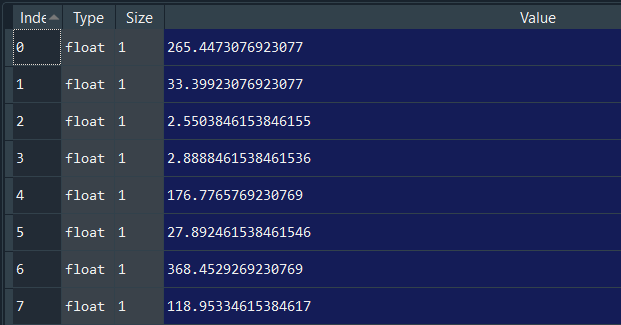
print("\nLa covariance entre X1 et X5: " + str(cov))

Vos résultats sur le cas d’étude :

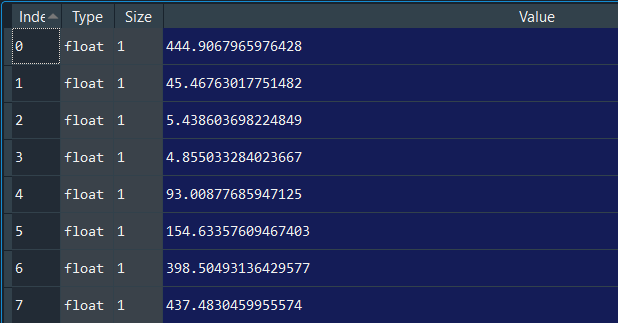
Voici un aperçu de la data frame affichée par le code :



La moyenne de cette data frame est :



La variance de cette data frame est :



3. Le code qui permet de centrer le nuage dans l’espace initial

4. Le code qui permet d’obtenir le nouvel espace projeté : les hyperplans sont associés aux *p* directions de l’espace, chaque axe factoriel est de vecteur propre (respectivement matrice) et une valeur propre λ (respectivement matrice) :

**Cas ACP centrée** et cas de **l’ACP centrée- normée** (standardisée)

# Question 3

# ACP centrée

# La translation se fait en trouvant le barycentre du nuage de points qui est le vecteur des moyennes et en le soustrayant aux coordonnées des points dans

import copy as cp

n,m = D.shape

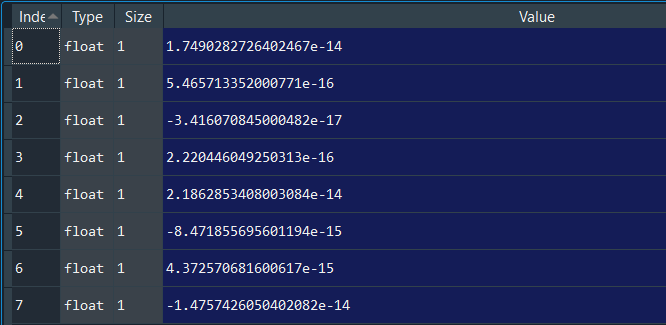
D1 = cp.deepcopy(D)

for i in range(m):

D1.loc[:,"X{}".format(i+1)] = D1.loc[:,"X{}".format(i+1)] - m1[i]

# Verification

m2 = moyenne(D1) # On remarque que les valeurs de la data frame tendent vers zéro et donc on a bien centré nos points dans



# Question 4 :

# On commence par déterminer la matrice de variance-covariance de la data frame

def matrice\_covariance(D):

n,m = D.shape

Mat\_cov = np.zeros((m,m))

var = variance(D)[0]

for i in range(m):

for j in range(i,m):

if i == j:

Mat\_cov[i][j] = var[i]

else:

Mat\_cov[i][j] = covariance(D,i,j)

Mat\_cov[j][i] = covariance(D,i,j)

return Mat\_cov

Mat = matrice\_covariance(D1)

val\_propre, vec\_propre = np.linalg.eig(Mat)

vec\_propre = np.transpose(vec\_propre)

# L'axe factoriel de chaque hyperplan i est de vecteur normal vec\_propre[i] et de valeur d'inertie projetée correspondante val\_propre[i]

print("L'axe factoriel de l'hyperplan 1 de plus grande inertie projeté est de vecteur normal: " + str(vec\_propre[0]))

print("L'inertie projetée associée est: " + str(val\_propre[0]))

5. Quelle est la valeur de l’inertie projetée totale ?

*Formule et résultat sur le cas exemple*

# Dans le cas général, la valeur de l’inertie projetée totale est la somme des valeurs propres associé à la matrice de variance-covariance de la data frame donnée.

print("la valeur de l'inertie projetée totale est: "+ str(sum(val\_propre)))

# Sur le cas exemple, la valeur de l’inertie projetée totale est : 1584.298

*Comparaison entre ACP centrée et ACP normée*

def acp\_normee(df):

d = cp.deepcopy(df)

n,m = df.shape

moy = moyenne(df)

var = variance(df)[0]

# Centrage

for i in range(m):

d.loc[:,"X{}".format(i+1)] = d.loc[:,"X{}".format(i+1)] - moy[i]

# Réduction

for i in range(m):

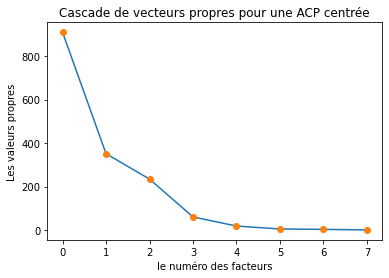
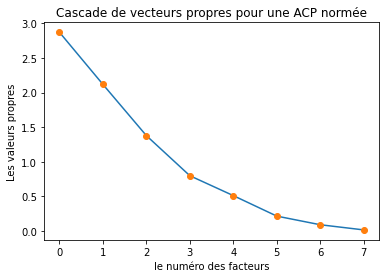
d.loc[:,"X{}".format(i+1)] = d.loc[:,"X{}".format(i+1)] / var[i]

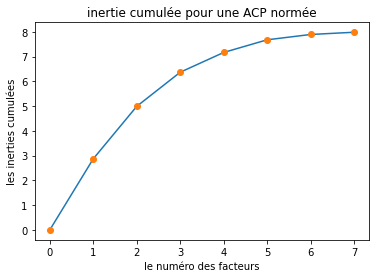
return d

# L’ACP normée nous permet d’éviter l’erreur de surinterprétation des données. On peut avoir une variance très grande entre certaines valeurs et une autre trop petite entre d’autres valeurs. Par conséquent les grandes variances sont susceptibles d’écraser les plus petites, le centrage et la réduction permettent de pallier ce problème et bien interpréter les résultats.

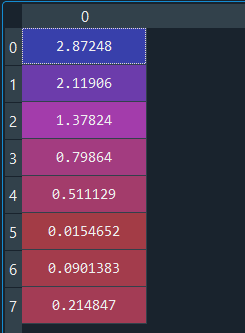
6. Donner la cascade des valeurs propres ? combien y en a-t-il ?

Il y a dans le cas exemple 8 valeurs propres. On peut voir graphiquement que les trois premières valeurs propres sont les plus significatives



*Combien retenez-vous de composantes dans l’exemple ?*



* Suivant la règle de **Kaiser-Guttman**, on retient les 3 premières valeurs propre et donc on projette sur les hyperplans de vecteurs normaux vec\_propre[1]/[2]/[3].
* Suivant la règle de **Karlis-Saporta-Spinaki** on prend les valeurs propres qui sont supérieurs ou égales = 1,05. Comme pour la règle précédente on retient les 3 premières valeurs propres et leurs vecteurs propres correspondants.

7. Comment obtenez-vous les nouvelles coordonnées des points dans l’espace réduit avec k < P

*Code de calcul*

# On obtient les nouvelles coordonnées en multipliant la matrice des individus/variables par la troncature jusqu’à la colonne k de la matrice des vecteurs propres U.

def coord\_Rk(matcov,D\_normee,k)

val\_p ,vec\_p = np.linalg.eig(matcov)

return D\_normee.dot(np.transpose(vec\_p[:k]))

8. Donner les nouvelles coordonnées pour *k*=5 des 8 premiers individus puis leur qualité de projection

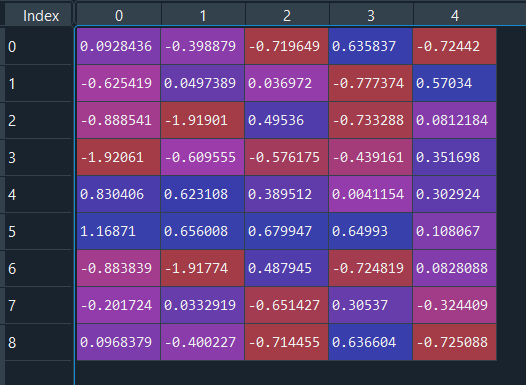
*Coordonnées de 8 premiers individus et de leur qualité de projection*

*Fournir le code*

nouveau\_coord = coord\_Rk(Mat, acp\_normee(D),5)

nouveau\_coord = nouveau\_coord.loc[:8,:]

Aperçu des nouvelles coordonnées :



# Qualité de la projection

def quali\_representation(D\_normee,coord,i):

norme\_xi = np.linalg.norm(coord.loc[i,:])

M = matrice\_covariance(D\_normee)

m = coord.shape[1]

val,vec = np.linalg.eig(M)

s = 0

for j in range(m):

axe\_j = vec[j]

axe\_j = axe\_j[:m]

proj\_xi\_j = coord.loc[i,:].dot(np.transpose(axe\_j))

s += proj\_xi\_j \*\*2

return s/(norme\_xi\*\*2)

# Qualités de projection de chaque individu

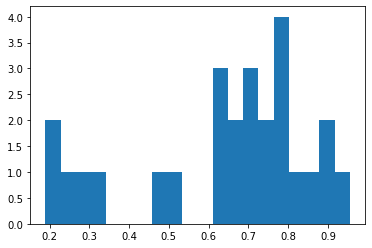
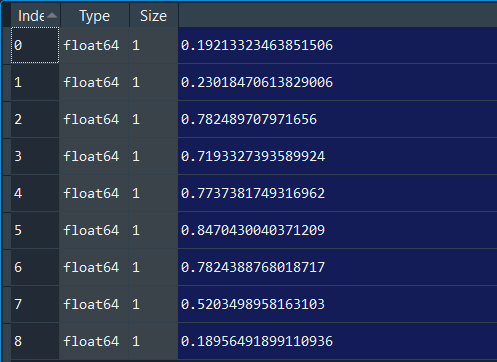
 On sait que plus l’angle est proche de 0 (et donc le cos proche de1) plus l’individu est bien projeté. Sur l’histogramme on voit bien que la majorité des individus ont une qualité de projection >= 0.6 et donc sont bien représenté.

Figure : Histogramme de la qualité de projection de chaque individu i



9. Fournir la contribution de ces 8 individus aux 5 premiers axes factoriels

*Calcul et résultats : sur ces 8 individus lesquels ont la plus forte contribution ?*

# Contribution des 8 individus

def contribution\_j(coord,i,j):

n,m = coord.shape

M = matrice\_covariance(coord)

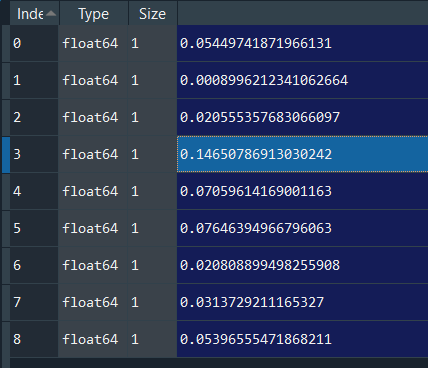
val,vec = np.linalg.eig(M)

axe\_j = vec[j]

proj\_xi\_j = coord.loc[i,:].dot(np.transpose(axe\_j))

return (proj\_xi\_j \*\* 2)/(n\*val[j])

# Evaluons la plus forte contribution des individus par rapport au premier axe factoriel

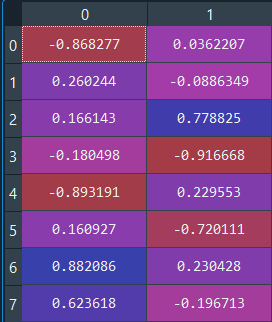
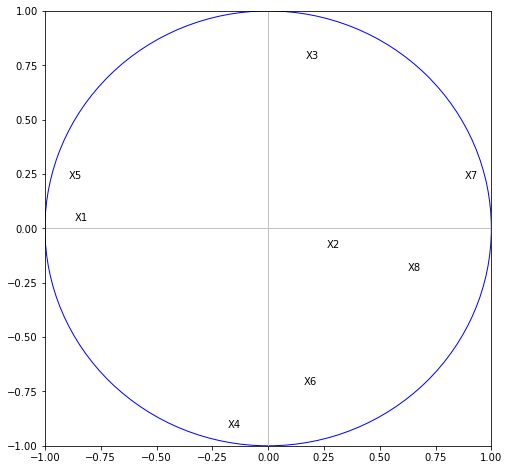


On remarque que l’individu 4 est celui qui a la plus forte contribution. C’est l’individu 4 qui a le plus d’importance dans la définition du premier vecteur propre.

10.Calculer la corrélation entre chacune des *p* variables avec les *p* composante :

*Code de calcul*

*Pour les deux premières composantes : quelles sont les variables les plus corrélées à ces deux composantes : fournir vos résultats pour le cas ACP centrée et normée*

def correlation(df):

n,m = df.shape

cor = np.zeros((m,m))

M = matrice\_covariance(df)

val , vec = np.linalg.eig(M)

vec = np.transpose(vec)

for i in range(m):

cor[:,i] = np.sqrt(val[i]) \* vec[i]

return cor

# Cas de l’ACP normée : plus l’angle entre les variable Xi est proche de 0 (ou 180), plus les variables sont corrélées. Par exemple, d’après le cercle, X1 et X7 sont négativement corrélés tandis que X1 et X3 sont décorrélés.

# En regardant les valeurs des corrélations suivant les deux composantes on remarque que X4, X5, et X7 sont les plus corrélés.

# Cas de l’ACP centrée :

# On a modifié le code pour qu’il prenne en considération si la data frame entrée est normée ou juste centrée.

def correlation(df, normee = "oui"):

n,m = df.shape

cor = np.zeros((m,m))

M = matrice\_covariance(df)

ecart = variance(df)[1]

val , vec = np.linalg.eig(M)

vec = np.transpose(vec)

if normee == "oui":

for i in range(m):

cor[:,i] = np.sqrt(val[i]) \* vec[i]

else:

for i in range(m):

cor[:,i] = (np.sqrt(val[i]) \* vec[i] )/ ecart[i]

return cor

11. Comparaison des ACP centrée et normée (standardisée)

L’ACP normée nous permet d’éviter l’erreur de surinterprétation des données. On peut avoir une variance très grande entre certaines valeurs et une autre trop petite entre d’autres valeurs. Par conséquent les grandes variances sont susceptibles d’écraser les plus petites, le centrage et la réduction permettent de pallier ce problème et bien interpréter des résultats.

12. Validation par rapport aux packages de R : dudi.pca (ade4) ou FactoMineR

# Comparaison avec les fonctions prédéfinies de Python

# Pour créer une ACP centrée normée

# Classe pour standardisation

from sklearn.preprocessing import StandardScaler

# Instanciation

sc = StandardScaler()

# Transformation: centrage-réduction

Z = sc.fit\_transform(D)

print(Z)

# Moyenne

print(np.round(np.mean(Z,axis=0),4))

# Ecart-type

print(np.std(Z,axis=0,ddof=0))

# Classe pour l'ACP

from sklearn.decomposition import PCA

# Instanciation

ACP = PCA(svd\_solver='full')

# Calculs

coord = ACP.fit\_transform(Z)

# Nombre de composantes calculées

print(ACP.n\_components\_)

# Variance expliquée

print(ACP.explained\_variance\_)

# Valeur corrigée

eigval = (n-1)/n\*ACP.explained\_variance\_

print(eigval)

# Il faut appliquer une correction

# Proportion de variance expliquée

print(ACP.explained\_variance\_ratio\_)

plt.plot(np.arange(1,m+1),eigval)

plt.title("Scree plot")

plt.ylabel("Eigen values")

plt.xlabel("Factor number")

plt.show()

# Cumul de variance expliquée

plt.plot(np.arange(1,m+1),np.cumsum(ACP.explained\_variance\_ratio\_))

plt.title("Explained variance vs. # of factors")

plt.ylabel("Cumsum explained variance ratio")

plt.xlabel("Factor number")

plt.show()

**Conclusion:**

On peut faire une première validation de nos fonctions car elles retournent les mêmes résultats que les fonctions ACP prédéfinies de Python.

1. *Dans les cadres : les réponses aux questions et éléments de compte-rendu* [↑](#footnote-ref-1)