Implémentation de la méthode ACP en python

INSEA Rabat Maroc

OURAHOU MOHAMED
LACHGAR MOHAMED

Master: Systèmes d'Informations et Systèmes intelligents
Institut Nationale de Statistique et Économie Appliqué

February 27, 2022

Introduction

Dans cette partie, on va essayer d'écrire notre propre version de la fonction qui réalise l'ACP. On va commencer à expliquer comment on a fait et de commenter chaque partie du code, puis on va l'appliquer sur les mêmes données sur lesquelles on a travaillé dans ce projet et faire une petite comparaison pour voir si on a réussi à créer les composantes principales correctement.

Déclarations nécessaires et récupèration des noms des variables

Tout d'abord on inclut les bibliothèques dont on aura besoin, il s'agit de :

- **math:** pour les fonctions de mathématiques;
- **numpy:** pour calculer les valeurs et vecteurs propres en utilisant la fonction eig et les tableau numpy (np.array);

Puis, on nomme notre fonction : **ACP**. on va se contenter comme paramètre juste de la matrice des données.

Pour stocker les résultats, on va créer -s'il n'existe pas déjà- un fichier dans le répertoire courant (où se trouve notre programme) sous le nom de "resultatACP.txt" et on l'ouvre en écriture.

Déclarations nécessaires et récupèration des noms des variables

La matrice sur laquelle on travaille est supposée d'avoir la structure suivante:

- La première ligne contient les noms des variables;
- La première colonne contient les noms des individus;
- La première case est vide, pour nous elle va contenir "None".

Donc le nombre de variables (nbVars) est égal à la taille de la première ligne (nombre de cases dans cette ligne) moins 1 c'est -à -dire la case qui contient "None". Et le nombre des individus (nbInds) est la taille de la matrice (nombre de ligne) moins 1 c'est-à-dire la première ligne qui contient les noms des variables.

Déclarations nécessaires et Récupèration des noms des variables

On récupère les noms des variables en copiant simplement la première ligne et de même pour les noms des individus.

```
import math
import numpy as np
from numpy.linalg import eig

def ACP(matrice):

    fichierResultat = open("./resultatACP.txt","w")

    nbVars = len(matrice[0]) -1
    nbInds = len(matrice) - 1
    nomVars = matrice[0][:]

    nomInds=[]
    for i in range(1,nbInds+1):
        nomInds.append(matrice[i][0])
```

Calcule des moyennes et des écart-types

On calcule les moyennes et les écart-types pour chaque variable, pour cela on initialise deux listes vides. Puis on boucle sur chaque colonne pour calculer la moyenne de la variable correspondante. et de la même manière pour calculer les écart-types.

```
moyennes = []
ecarts_type=[]
for j in range(1,nbVars+1):
    moyennes.append(0)
    for i in range(1,nbInds+1):
        moyennes[j-1]+=matrice[i][j]
    moyennes[j-1]/=nbInds
    #moyennes[j-1]=math.ceil(moyennes[j-1]) #ceil pour raison de tests
for j in range(1,nbVars+1):
    ecarts_type.append(0)
    for i in range(1,nbInds+1):
        ecarts_type[j-1]+=(matrice[i][j]-moyennes[j-1])**2
    ecarts_type[j-1]/=nbInds
    ecarts_type[j-1]=math.sqrt(ecarts_type[j-1])
```

Calcule de la matrice de corrélation

Maintenant on va calculer la matrice de corrélation des variables deux à deux, cette matrice est une matrice carrée d'ordre nbVars, et chaque case de cette matrice contient le coefficient de corrélation des deux variables correspondantes. ce Coefficient se calcule de en utilisant la relation suivante:

$$cov(X_i, X_j) = \frac{1}{n} \sum_{k=1}^{n} (x_{ki} - \bar{x}_i)\left(x_{kj} - \bar{x}_j\right)$$

$$r_{ij} = \frac{cov(X_i, X_j)}{\sigma_{X_i} \sigma_{X_j}}$$

Et c'est exactement ce que fait la boucle de trois niveaux ci-dessus, et pour éviter d'avoir des valeurs très proches de 1 mais différente de 1 dans la diagonale à cause des erreurs d'arrondis, on l'a spécifié explicitement en testant si k==j ou non.

Calcule de la matrice de corrélation

```
matriceDeCorrelation=[]

 for j in range(0,nbVars):
     matriceDeCorrelation.append(list())
     for k in range(0,nbVars):
         matriceDeCorrelation[j].append(0)
         for i in range(0,nbInds):
             matriceDeCorrelation[j][k]+= (matrice[i+1][j+1]-moyennes[j]) *

         matriceDeCorrelation[j][k] /= nbInds
         matriceDeCorrelation[j][k] /= ecarts_type[j]*ecarts_type[k]
         if k==j :
             matriceDeCorrelation[j][k] = 1.0
```

Afichage et enregistrement des résultats

Maintenant, on va afficher la matrice de corrélation dans le console et en même temps on va l'écrire dans le fichier **"resultatACP.txt"** , et c'est la même procédure qui va être suivie les affichages qui suivent. Et on ajoute des séparateurs pour garantir la lisibilité du résultat final.
Avant d'afficher les valeurs de la matrice de corrélation, on affiche d'abord les noms des variables dans la première ligne et la première colonne.

Afichage et enregistrement de la matrice de corrélation

```
print("Matrice de corrélation des variables")
    fichierResultat.write("Matrice de corrélation des variables\n")

    tmpNomVars = ""
    for i in range(nbVars+1):
        if i == 0:
            tmpNomVars += " -------- "
        else:
            tmpNomVars += " " + nomVars[i] + " -------- "

    print(tmpNomVars)
    fichierResultat.write(tmpNomVars+"\n")

    for i in range(nbVars):
        print(nomVars[i+1]," ",matriceDeCorrelation[i][:])
        tmpp = nomVars[i+1]+" "+str(matriceDeCorrelation[i][:])+"\n"
        fichierResultat.write(tmpp)
```

Centrage et réduction des valeurs de la matrice initiale

Ce bloc de code est utilisé pour centrer et réduire les valeurs de la matrice initiale et les mettre dans une autre matrice.

```
matriceCentreReduite = matrice[:]
InertieTotaleCentreReduite = nbVars

for j in range(1,nbVars+1):
    for i in range(1,nbInds+1):
        tmp = matriceCentreReduite[i][j]-moyennes[j-1]
        tmp /= ecarts_type[j-1]
        matriceCentreReduite[i][j]= tmp
```

Calcule des valeurs et des vecteurs propres

Maintenant on va calculer les valeurs et les vecteurs propres, pour cela on va utiliser la bibliothèque numpy. Premièrement on va transformer notre matrice de corrélation qui est une liste 2D de python c'est-à-dire une liste de listes en un numpy array car la fonction qui calcule les valeurs propres n'accepte que des numpy arrays. Cette dernière est eig qui retourne un tuple contenant deux informations: la liste des valeurs propres et une liste de vecteurs propres. Puisque les vecteurs propres ne sont pas uniques, c'est-à-dire pour une même valeur propre on peut trouver plusieurs vecteurs propres alors c'est à ce stade qu'on aura des différences avec les autres fonctions de l'ACP. Il se peut que les vecteurs propres trouvés par notre fonction soient différents de ceux trouvés par les autres fonctions notamment celle de FactoMineR.Mais, en tout cas, les valeurs propres restent les mêmes.

Calcule des valeurs et des vecteurs propres

Après on transforme le résultat en des listes python régulières au lieu de numpy array. Puis, pour bien appliquer l'ACP, on doit trier les valeurs propres dans un ordre descendant (reverse = True) et aussi trier chaque vecteurs propre dans le même ordre que sont valeur propre associée. C'est pour cela qu'on a regroupé les valeurs et vecteurs propres dans un même tableau (VectsEtValsPropres) avant de faire le tri.

```
matriceDeCorrNumpy = [None]*nbVars

  for i in range(0,nbVars):
      matriceDeCorrNumpy[i]=np.array(matriceDeCorrelation[i][:])

  matriceDeCorrNumpy = np.array(matriceDeCorrNumpy)
  valspropres, vectsPropres = eig(matriceDeCorrNumpy)

  valspropres = list(valspropres)
  vectsPropres = [list(vect) for vect in vectsPropres]
```

Calcule des valeurs et des vecteurs propres

```
VectsETValsPropres = list()
    nbValsPrps = len(valspropres)

    for i in range(nbValsPrps):
        VectsETValsPropres.append( ( valspropres[i], vectsPropres[i] ) )
    VectsETValsPropres = [elm for elm in sorted(VectsETValsPropres,
    key=lambda elm: elm[0], reverse=True)]
```

Afichage et enregistrement des valeurs propres

On affiche et on écrit dans le fichier.

```
print("Valeurs et vecteurs propres")
fichierResultat.write("Valeurs et vecteurs propres\n")
for i in range(nbValsPrps):
    print("Valeur prore",i," : ",  VectsETValsPropres[i][0] )
    tmpp= "Valeur prore"+ str(i) + " : " +  str(VectsETValsPropres[i][0])+"\n"
    fichierResultat.write(tmpp)
    print("Vecteur prore",i," : ", VectsETValsPropres[i][1] )
    tmpp= "Vecteur prore"+ str(i) + " : " +  str(VectsETValsPropres[i][1])+"\n"
    fichierResultat.write(tmpp)
```

Calcule des coordonnées de chaque individu dans les différentes axes

Maintenant, on va calculer les coordonnées de chaque individus dans les différentes axes, cela ce fait par le biais d'un produit scalaire entre les vecteurs lignes de la matrice de données centrée réduite et le premier vecteur propre pour avoir la première composante principale (premier axe), et avec le second vecteur propre pour la deuxième composante principale etc...

```
MatriceDesAxes =[]
for i in range(nbInds):
    MatriceDesAxes.append(list())
    for j in range(nbValsPrps):
        tmp=0
        for k in range(nbValsPrps):
            tmp += matriceCentreReduite[i+1][k+1] * VectsETValsPropres[j][1][k]
            MatriceDesAxes[i].append(tmp)
```

Afichage et enregistrement des coordonnées de chaque individu dans les différentes axes

Et on affiche sur le console, écrit dans le fichier en ajoutant les noms des axes et des individus.

```
print("Coordonnées des individus")
fichierResultat.write("Coordonnées des individus\n")

tmpNomsAxes = "--------"
for i in range(nbVars):
    tmpNomsAxes += " " + nomAxes[i] + " -------- "

print(tmpNomsAxes)
fichierResultat.write(str(tmpNomsAxes)+"\n")
for i in range(nbInds):
    print(nomInds[i]," ", MatriceDesAxes[i][:])
    tmpp= nomInds[i]+" "+ str(MatriceDesAxes[i][:])+"\n"
    fichierResultat.write(tmpp)
```

Calcule des pourcentages des variances pour chaque valeur propre

Ce pourcentage se calcule en divisant la valeur propre par l'inertie totale qui est égale au nombre de variables car on travaille sur des données centrées et réduites. Et on calcule le pourcentage de variance cumulé.

```
TableuPourcentage=[None]*3
TableuPourcentage[0]=[ val[0] for val in VectsETValsPropres]
TableuPourcentage[1]=[ val[0]/InertieTotaleCentreReduite for val in  VectsETVal
TableuPourcentage[2]=[  ]
for i in range(0,nbValsPrps):
    if i==0:
        TableuPourcentage[2].append(TableuPourcentage[1][0])
    else:
        TableuPourcentage[2].append(TableuPourcentage[2][i-1]+
        TableuPourcentage[1][i])
```

Afichage et enregistrement des pourcentages

```
print("Pourcentage d'inertie des valeurs propres et pourcentage
d'inertie cumulé")
fichierResultat.write("Pourcentage d'inertie des valeurs propres et
pourcentage d'inertie cumulé\n")
print("Valeurs Propres: ", TableuPourcentage[0][:] )
fichierResultat.write("Valeurs Propres: "+str(TableuPourcentage[0][:])+"\n" )

print("% inertie: ",TableuPourcentage[1][:])
fichierResultat.write("% inertie: "+str(TableuPourcentage[1][:])+"\n")

print("% inertie cumulé: ",TableuPourcentage[2][:])
fichierResultat.write("% inertie cumulé: "+str(TableuPourcentage[2][:])+"\n")
```

Calcule des contributions des individus dans un axe

La contribution d'un individu dans un axe se calcule en divisant le carré de son coordonnées dans cet axe par la somme de toutes les coordonnées des individus (lui même inclus) dans cet axe et on multiplie par 100 pour avoir le pourcentage. C'est ce qu'on fait dans ce bloc de code pour tous les individus et tous les axes.

```
SommeCarreCoord = []
for j in range(nbVars):
    tmp = 0
    for i in range(nbInds):
        tmp+= MatriceDesAxes[i][j]**2
    SommeCarreCoord.append(tmp)
ContribIndvs = []
for i in range(nbInds):
    ContribIndvs.append(list())
    for j in range(nbVars):
        ContribIndvs[i].append(((MatriceDesAxes[i][j]**2)/SommeCarreCoord[j])*1
```

Afichage et enregistrement des contributions des individus dans les axes

```
print("contributions des individus")
fichierResultat.write("contributions des individus\n")
print(tmpNomsAxes)
fichierResultat.write(str(tmpNomsAxes)+"\n")
for i in range(nbInds):
    print(nomInds[i]," ", ContribIndvs[i][:])
    fichierResultat.write(nomInds[i]+" "+str(ContribIndvs[i][:])+"\n")
```

La qualité de représentation des individus dans un axe

La qualité de représentation d'un individu dans un axe se calcule en divisant le carré de sa coordonnées dans cet axe sur la somme des carrés de ses coordonnées dans tous les autres axes (lui même inclus) et on multiplie par 100 pour avoir le pourcentage.
C'est ce qu'on fait pour tous les individus et tous les axes dans ce bloc de code.
Maintenant on va se concentrer sur les variables et calculer leur coordonnées dans les axes, leurs contributions et leurs qualités de représentation. C'est la même démarche présentée ci-dessus qu'on va suivre donc on n'a pas besoin d'inclure le code ici - En tout cas un lien vers le code source complet de cette fonction va être fourni après -.
Cependant, on doit noter que les coordonnées des variables dans les axes se calculent de la même manière que celle des individus mais en se basant sur la matrice de corrélation des variables au lieu de celle de données.

Afichage et enregistrement des contributions des individus dans les axes

```
for i in range(nbInds):
    tmp = 0
    for j in range(nbVars):
        tmp+= MatriceDesAxes[i][j]**2
    SommeCarreCos2.append(tmp)
Cos2Indvs = []
for i in range(nbInds):
    Cos2Indvs.append(list())
    for j in range(nbVars):
        Cos2Indvs[i].append(((MatriceDesAxes[i][j]**2)/SommeCarreCos2[i])*100)

print("Qualités de représentation des individus")
fichierResultat.write("Qualités de représentation des individus\n")
print(tmpNomsAxes)
for i in range(nbInds):
    print(nomInds[i]," ", Cos2Indvs[i][:])
    fichierResultat.write(nomInds[i]+" "+ str(Cos2Indvs[i][:]) + "\n")
```

Application sur les données de notre projet

On va essayer maintenant d'appliquer cette fonction sur les mêmes données qu'on a utiliser dans ce projets : **"Indicateurs sociaux : Emploi, activité et chômage"** .
Pour simplifier et gagner du temps, on ne va pas créer une fonction qui lit les données à partir d'un fichier excel mais on va juste entrer les données manuellement.

Application sur les données de notre projet

Voici comment les données sont représentée en une matrice (liste 2d) en python et l'appel de la fonction:

```
matriceTest = [
["None","PA15+","PA15+Urb","PA15+Rur","PAO","PAOUrb","PAORur","AgrForPêche","Indus","BatTrav","comm","TraEntCo
["2002-08",10849,5532,5318,9710,4586,5124,5.3,21.9,10.0,20.9,6.2,35.6,0.1,1139,27.7,9.8,23.6,23.5,23.7,2.4,11.
["2008",11267,5874,5393,10189,5013,5176,5.5,20.9,11.2,19.9,6.8,35.6,0.2,1078,27.5,8.3,20.0,20.6,19.0,2.6,12.2]
["2009",11314,5916,5398,10284,5101,5184,5.0,20.2,11.8,19.9,6.6,36.0,0.2,1029,27.6,7.7,18.6,19.1,17.8,2.5,11.8]
["2010",11415,5966,5449,10405,5169,5235,4.8,20.2,12.4,20.2,6.9,35.3,0.2,1037,28.3,8.1,18.1,18.5,17.5,2.4,11.4]
["2011",11538,5553,5237,10509,5272,5237,4.9,20.2,12.4,20.6,7.1,35.1,0.2,1028,30.6,7.0,18.3,17.8,19.0,2.3,11.1]
["2012",11549,6655,5404,10511,5320,5190,5.1,18.7,11.9,20.5,6.8,36.8,0.2,1038,29.1,6.9,18.2,17.9,18.6,2.4,10.6]
["2013",11705,6217,5488,10625,5346,5278,4.9,18.4,10.9,21.0,6.6,38.0,0.2,1081,27.8,8.1,18.2,18.1,18.4,2.4,9.8]
]

started = time.time()

ACP(matriceTest)

print("temps d'execution : ",time.time()-started)
```

Temps d'exécution de la fonction

On calcule aussi le temps d'exécution de cette fonction, ça donne :

```
------------
temps d'execution :  0.0499720573425293
```

Donc c'est un bon temps d'exécution si on ajoute à cela le fait que la matrice est de dimensions 8 lignes x 22 colonnes. mais bien sur ce temps va augmenter lorsque la taille de la matrice augmente. Le résultat est dans le fichier accessible à partir de ce lien :
`https://drive.google.com/file/d/1F3WRjWbdQjegbNBbcXYFNDYz4ep1GqzC/view?usp=sharing`
Maintenant on va essayer de comparer les résultats qu'on a trouvés avec celle retourné par la fonction PCA de FactoMineR.

Comparaison des résultats: Les valeurs et les vecteurs propres

Commençons par les valeurs et les vecteurs propres:

- **PCA de FactoMineR:**

```
> resultACP$eig
        eigenvalue percentage of variance cumulative percentage of variance
comp 1 11.8387135              56.374826                            56.37483
comp 2  3.9646567              18.879317                            75.25414
comp 3  3.0196455              14.379265                            89.63341
comp 4  1.1106243               5.288687                            94.92210
comp 5  0.7712067               3.672413                            98.59451
comp 6  0.2951533               1.405492                           100.00000
```

PCA ne retourne pas les vecteurs propres.

- **Notre Implémentation:**

Nous avons généré toutes les valeurs propres (21 en total car on a 21 variables), mais on va se concentrer sur les 6 premiers:

Comparaison des résultats: Les valeurs et les vecteurs propres

```
Valeurs et vecteurs propres
Valeur prore0 : 12.100544449121259
Vecteur prore0 : [0.2762896526269812, 0.10225636392610997, 0.014
Valeur prore1 : 3.946225420739247
Vecteur prore1 : [0.16414213299712688, 0.2301628308446578, -0.21
Valeur prore2 : 3.1281394665191726
Vecteur prore2 : [0.08816350848599068, 0.2823776166120053, -0.37
Valeur prore3 : 0.9815825009617075
Vecteur prore3 : [0.2833323010909296, 0.057084480178708735, 0.06
Valeur prore4 : 0.6054857961861179
Vecteur prore4 : [0.284313003237785 47, 0.04303936331593 2576, -0.
```

Donc on a trouvé les mêmes valeurs propres (on a juste une très petite différence qu'on peut négliger dans quelques une) Nous, on affiche les vecteurs propres mais la fonction PCA ne les affiche pas, et puisque les vecteurs propres ne sont pas uniques alors on ne peut pas être sûr qu'ils sont les mêmes.

Comparaison des résultats: Les poursentages des variances

- **Pour le pourcentage des variances:**

```
------------------------------------------------------------------------
Pourcentage d'inertie des valeurs propres et pourcentage d'inertie cumulé
Valeurs Propres: [12.100544449121259, 3.946225420739247, 3.1281394665191726, 0.981582
% inertie: [0.5762164023391075, 0.18791549622567844, 0.1489590222151987, 0.0467420238
% inertie cumulé: [0.5762164023391075, 0.764131898564786, 0.9130909207799847, 0.95983
```

```
cumulé
65191726, 0.9815825009617075, 0.6054857961861179, 0.2380223664725042,
1987, 0.04674202385531940 6, 0.028832656961243708, 0.011334398403452581
207799847, 0.9598329446353041, 0.9886656015965478, 1.0000000000000004,
```

Et encore c'est les mêmes valeurs retournées par les deux fonctions: **PCA de FactoMineR** et **Notre implémentation.**

Comparaison des résultats: Les coordonnées,les contributions et les qualités de représentation des variables

Passant maintenant aux variables et comparant les coordonnées, contributions et qualités de représentation pour juste la première axe pour simplifier:

PCA de FactoMineR:

$coord

	Dim.1
PA15.	0.96327556
PA15.Urb.....	0.57339855
PA15.Rur	0.29536346
PAO	0.98672741
PAOUrb	0.99029697
PAORur	0.83061226
AgrForPêche	-0.69758050
Indus	-0.85144593
BatTrav	0.78329152
Comm	-0.09782985
TraEntCommu	0.78378001
Serv	0.33216650
MalDes	0.64594915
PAC	-0.82711315
TFPAC	0.50444986
ChSansDipUrb	-0.88576596
ChNivMoyUrb	-0.97762170
ChNivSupUrb	-0.99630006
ChDipUrb	-0.86751900
ChSansDipRur	-0.34626769
ChDipRur	-0.57440908

$cos2

	Dim.1
PA15.	0.92789980
PA15.Urb.....	0.32878590
PA15.Rur	0.08723958
PAO	0.97363098
PAOUrb	0.98068809
PAORur	0.68991673
AgrForPêche	0.48661855
Indus	0.72496017
BatTrav	0.61354561
Comm	0.00957068
TraEntCommu	0.61431110
Serv	0.11033458
MalDes	0.41725031
PAC	0.68411616
TFPAC	0.25446966
ChSansDipUrb	0.78458134
ChNivMoyUrb	0.95574419
ChNivSupUrb	0.99261380
ChDipUrb	0.75258921
ChSansDipRur	0.11990131
ChDipRur	0.32994579

$contrib

	Dim.1
PA15.	7.83784315
PA15.Urb.....	2.77720971
PA15.Rur	0.73690081
PAO	8.22412819
PAOUrb	8.28373865
PAORur	5.82763261
AgrForPêche	4.11040061
Indus	6.12363974
BatTrav	5.18253615
Comm	0.08084223
TraEntCommu	5.18900213
Serv	0.93198119
MalDes	3.52445650
PAC	5.77863595
TFPAC	2.14947054
ChSansDipUrb	6.62725165
ChNivMoyUrb	8.07304096
ChNivSupUrb	8.38447354
ChDipUrb	6.35701854
ChSansDipRur	1.01279004
ChDipRur	2.78700712

Comparaison des résultats: Les coordonnées,les contributions et les qualités de représentation des variables

Notre implémentation:

Coordonnées des Variables

	Axe 1	Axe
PA15+	[0.9940270297648329,	-0
PA15+Urb	[0.6082116591878555,	
PA15+Rur	[0.44888024624821654	
PAO	[1.0112132223349093,	-0.3
PAOUrb	[1.0000690369514331,	-
PAORur	[0.9285232135688304,	-
AgrForPêche	[-0.8446689617366	
Indus	[-0.9307936884446947,	0
BatTrav	[0.7112026784692941,	
comm	[0.04165014436087494,	0.
TraEntCommu	[0.63522118957682	
Serv	[0.4846471114882531,	-0.
MalDes	[0.8420780386226373,	-
PAC	[-0.7785079533512188,	0.3
TFPAC	[0.39683377750603044,	-
ChSansDipUrb	[-0.800126557811	
ChNivMoyUrb	[-0.9925010083465	
ChNivSupUrb	[-1.0108403374434	
ChDipUrb	[-0.8768877579216228	
ChSansDipRur	[-0.380441748380	
ChDipRur	[-0.7061905266744815	

Qualité de représentation des

	Axe 1	Axe
PA15+	[9.586614422453845,	1.:
PA15+Urb	[7.6773985203822015	
PA15+Rur	[6.734500039633233,	
PAO	[9.574372845839205,	1.29
PAOUrb	[9.24601072387446,	1.:
PAORur	[11.37044859304946,	1
AgrForPêche	[12.7556058556228	
Indus	[10.092198333632169,	1
BatTrav	[7.062563252638931,	:
comm	[0.09598382494989195,	0
TraEntCommu	[5.72621359004019	
Serv	[7.575428694650762,	0.5
MalDes	[8.183412839866525,	1
PAC	[7.659214100998921,	1.60
TFPAC	[2.951686324505052,	0.4
ChSansDipUrb	[6.8337735567276	
ChNivMoyUrb	[9.6213697266089]	
ChNivSupUrb	[9.3570537650307:	
ChDipUrb	[9.552890823510483,	
ChSansDipRur	[4.1638020150673	
ChDipRur	[10.020925937613924	

Contribution des Variables

PA15+	[7.778883715129527, 6.54
PA15+Urb	[2.912261532336831, 2
PA15+Rur	[1.586287417124447, 1
PAO	[8.050194450785456, 7.2507
PAOUrb	[7.873736089291748, 7.1
PAORur	[6.787446556346971, 6.6
AgrForPêche	[5.616864706421187
Indus	[6.8206811986711955, 5.7
BatTrav	[3.982058707569126, 5.
comm	[0.013656956102730284, 0.
TraEntCommu	[3.176660807938825
Serv	[1.8491500294674688, 0.97
MalDes	[5.582459405290353, 5.2
PAC	[4.771412883710672, 6.6316
TFPAC	[1.2397615233394843, 1.1
ChSansDipUrb	[5.04008966688057
ChNivMoyUrb	[7.755017902883083
ChNivSupUrb	[8.044258526765027
ChDipUrb	[6.053532877115508, 6
ChSansDipRur	[1.13945513306535
ChDipRur	[3.9261299137644223,

Comparaison des résultats: Les coordonnées,les contributions et les qualités de représentation des variables

- **Pour les coordonnées:** Les deux fonctions ont trouvé presque les mêmes coordonnées pour les variables dans l'axe 1.
- **Pour les qualité de représentation:** les qualités de représentations trouvées par notre implémentation sont très inférieures à celles trouvées par FactoMineR.
- **Pour les contributions:** les deux fonctions ont trouvées presque les mêmes contributions.

Passant maintenant aux individus et comparons les coordonnées, contributions et qualités de représentation pour juste la première axe pour simplifier:

Comparaison des résultats: Les coordonnées ,les contributions et les qualités de représentation des individus

PCA de FactoMineR:

```
> resultACP$ind
$coord
             Dim.1
2002-08 -7.51552541
2008    -1.87234196
2009    -0.03364808
2010     1.66317844
2011     2.44889958
2012     2.76135251
2013     2.54808491

$cos2
                Dim.1
2002-08 0.9406616713
2008    0.2196983063
2009    0.0001832634
2010    0.3791149327
2011    0.2773133586
2012    0.6107627872
2013    0.2773186962

$contrib
                Dim.1
2002-08 68.157890799
2008     4.230267068
2009     0.001366212
2010     3.337913997
2011     7.236680531
2012     9.201129662
2013     7.834751733
```

Comparaison des résultats: Les coordonnées ,les contributions et les qualités de représentation des individus

Notre implémentation:

```
Coordonnées des individus
-------- Axe 1  --------
2002-08 [-2.0626615182291
2008 [-1.028893883717529,
2009 [0.1562251211729538,
2010 [0.6851573024448079,
2011 [0.44363779476859105
2012 [0.6403900006855523,
2013 [1.1661451828747402,
```

```
Qualités de représentatic
2002-08 [8.18341283986653
2008 [8.620805997049134,
2009 [0.6943877307624807,
2010 [12.289890004287303,
2011 [0.9425633824179115,
2012 [3.0686469257830487,
2013 [9.259429705963107,
```

```
contributions des individu
-------- Axe 1  --------
2002-08 [54.72928230315831
2008 [13.617738541440374,
2009 [0.31395366757043586,
2010 [6.038713174378392, 1
2011 [2.5317504514654012,
2012 [5.27536974793157, 2.
2013 [17.493192114055503,
```

Comparaison des résultats: Les coordonnées ,les contributions et les qualités de représentation des individus

- **Pour les coordonnées:** Les coordonnées qu'on a trouvé en utilisant notre implémentation sont un peu différentes de celles trouvées par la fonction PCA de FactoMineR.
- **Pour les qualité de représentation:** Les qualités de représentation de FactoMineR sont très bonnes en les comparant aux résultats trouvées par notre fonction.
- **Pour les contributions:** Les deux fonctions ont trouvé presque les mêmes contributions.

Conclusion

On a réussit à faire une implémentation de l'ACP en utilisant python, cette fonction donne des résultats acceptables mais celle du package FactoMineR donne de très bonnes qualité de représentation et elle présente des graphiques qui ne sont pas moins importantes que juste les résultats de calculs, la chose qu'on a pas encore ajouter à notre fonction.
Pour voir, tester ou même améliorer notre implémentation vous pouvez trouver son code source ici : `https://github.com/lachmed/ACP/blob/master/MaVersionAcp.py`