LA COMPRESSION DES IMAGES

PRÉSENTÉE PAR ZAKI AKRAM

SOMMAIRE

- INTRODUCTION
- DÉCOMPOSITION EN VALEURS SINGULIÈRES
- TRONCATURE ET APPROXIMATION
- SIMULATION PYTHON
- DÉCOMPOSITION RANDOMISÉE
- SIMULATION PYTHON

INTRODUCTION

- Dans de nombreux domaines, les systèmes génèrent des données naturellement organisées dans des grandes matrices, ou plus généralement dans des tableaux.
- Par exemple, les valeurs des pixels dans une image en niveaux de gris peuvent être stockées dans une matrice.

Problème

- Les images contiennent généralement un grand nombre de mesures(pixels) et sont donc des éléments d'un espace vectoriel de très grande dimension.

Solution

- Les images sont compressibles, ce qui signifie que les informations pertinentes peuvent être représentées dans un sous-espace de dimension beaucoup plus faible.

Méthode étudiée

- Décomposition en valeurs singulières

Matrice (image) → Matrice décomposée → Matrice Approximée

Décomposition en valeurs singulières

- **Valeurs singulières:**

Soit A une matrice de taille n×m et de rang r,

Soient $\sigma_1^2,\ldots,\sigma_m^2$ les valeurs propres de la matrice A^TA:

Alors les r premières sont strictement positives, les m-r suivantes sont nulles.

$\sigma_1,\ldots,\sigma_m$ sont les valeurs singulières de A

Décomposition en valeurs singulières

- **Théorème:**

  Soit A une matrice de taille $n \times m$:

Il existe deux matrices orthogonales U $(n \times n)$ et $V (m \times m)$, une matrice Σ $(n \times m)$ telles que:

$$A = U \, \Sigma \, V^T$$

Décomposition réduite

Soit A une matrice de rang r:

on suppose que n>m,

donc r $\leq$ m d'où la décomposition s'écrit:

$$\mathbf{A=U}\begin{bmatrix} \hat{\Sigma} \\ \\ 0 \end{bmatrix}\mathbf{V^T}$$

C'est-à-dire:

$$\mathbf{A=\hat{U}\,\hat{\Sigma}\,V^T= \sigma_1 u_1 v_1{}^T + \ldots. + \sigma_r u_r v_r{}^T}$$

Troncature

- **Troncature au rang k:**

On suppose que

$$A=\sigma_1 u_1 v_1^T + \dots + \sigma_r u_r v_r^T$$

La troncature au rang k de A est:

$$A_k=\sigma_1 u_1 v_1^T + \dots + \sigma_k u_k v_k^T$$

Approximation du rang faible d'Eckart-Young

Soit A une matrice réelle de taille $n\times m$,

$\sigma_1,\ldots,\sigma_m$ les valeurs singulières de A:

- **La norme de Frobenius:**

$$\|A\|_F = \sqrt{\sum_{\substack{1\leq i\leq n \\ 1\leq j\leq m}} a_{ij}^2}$$

- **La norme spectrale:**

$$\|A\|_2 = \sigma_1$$

Approximation d'Eckart-Young

- **Théorème:**

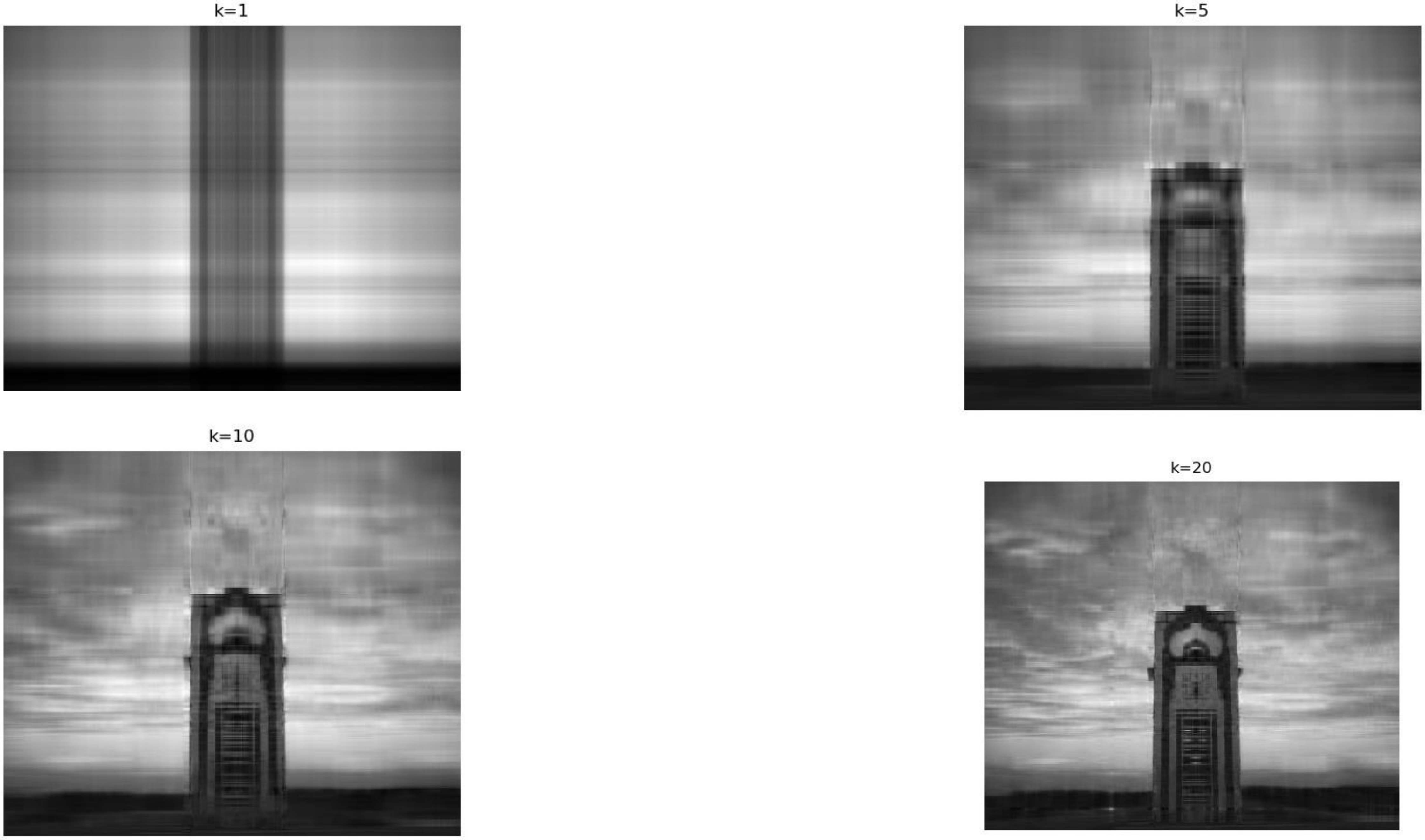
Soit A une matrice de taille n×m, on a:

$$\inf_{\substack{B\ tq \\ rang(B)=k}} \|A-B\|_F = A_k$$

$$\inf_{\substack{B\ tq \\ rang(B)=k}} \|A-B\|_2 = A_k$$

Simulation python

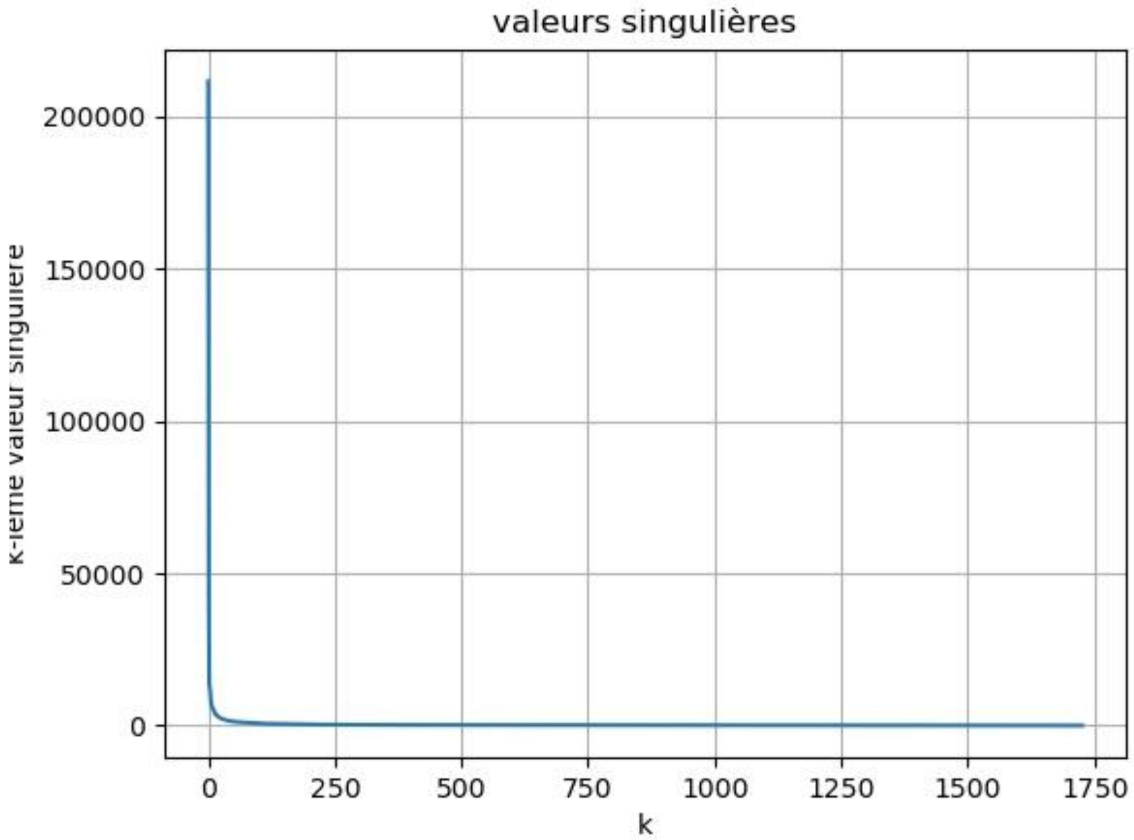
image originale

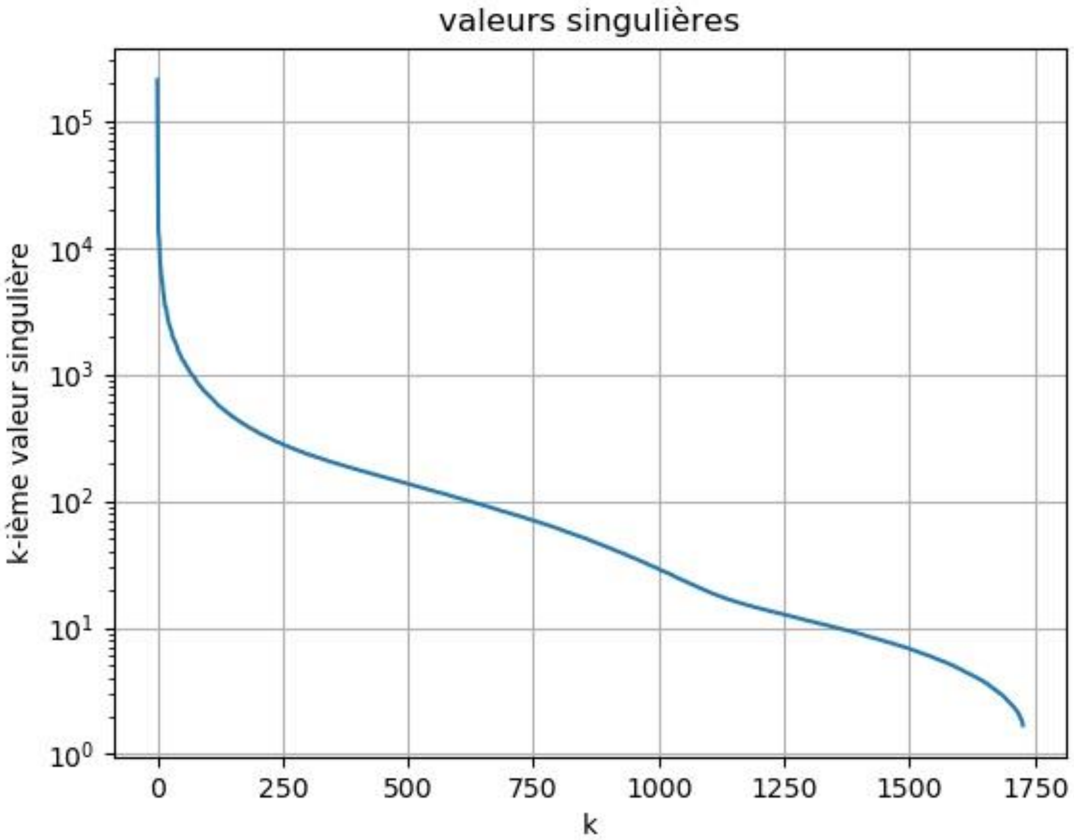

k=1
k=5
k=10
k=20

k=100

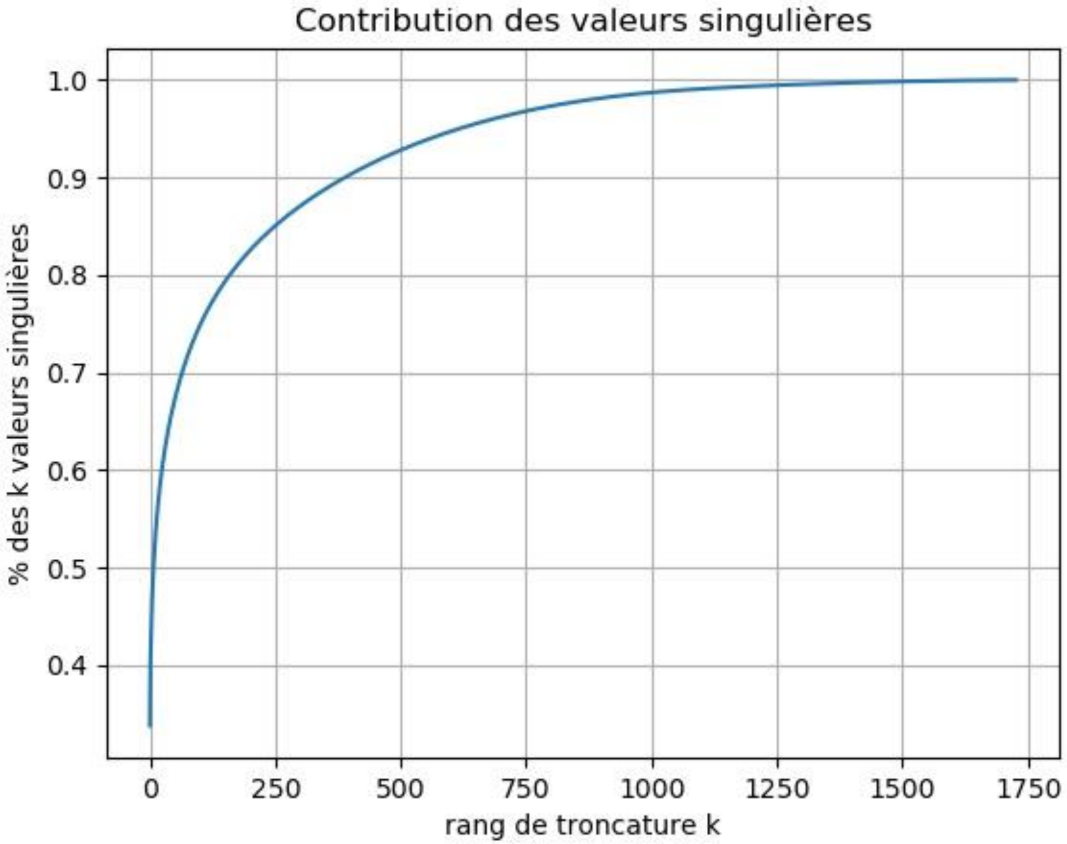
k=300

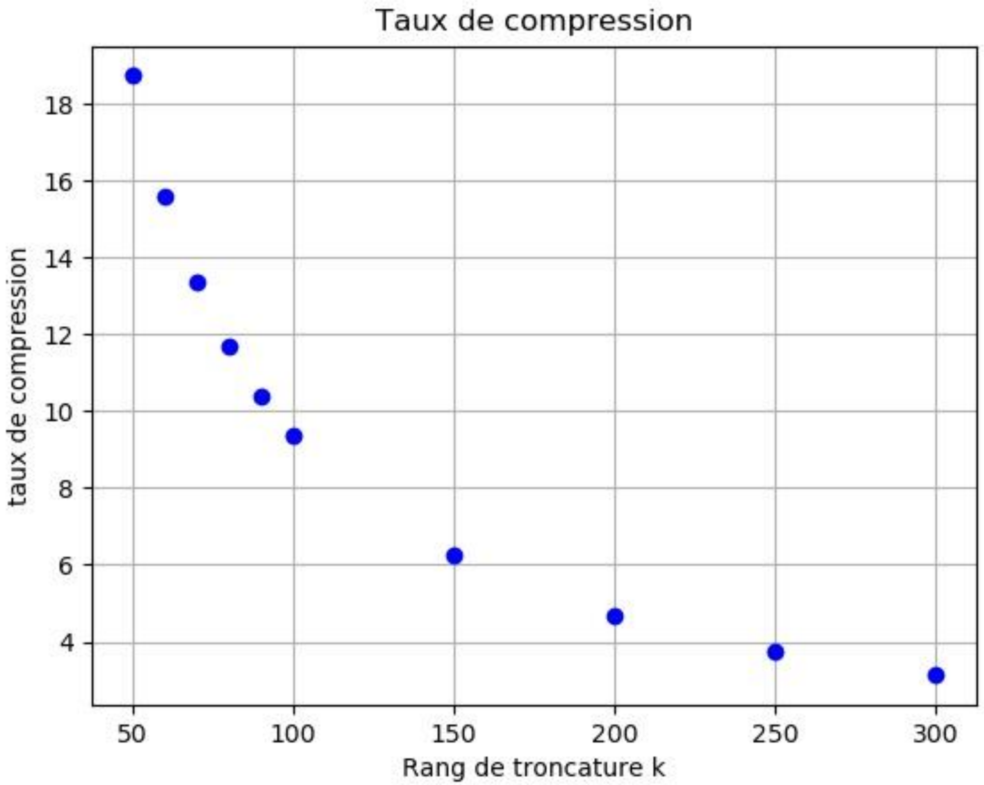
Graphes des valeurs singulières



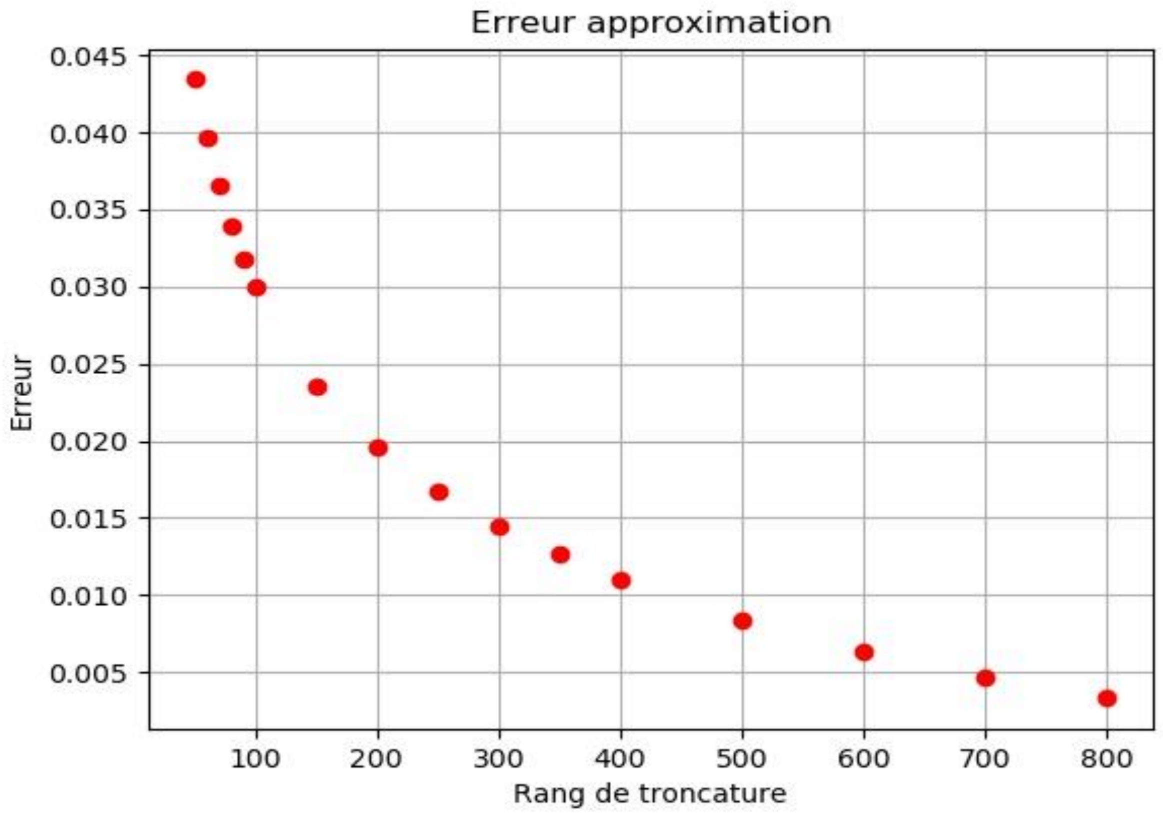


Contribution des valeurs singulières
Taux de compression





Erreur d’approximation



Décomposition en valeurs singulières randomisée

Soit A une matrice de taille $n \times m$:

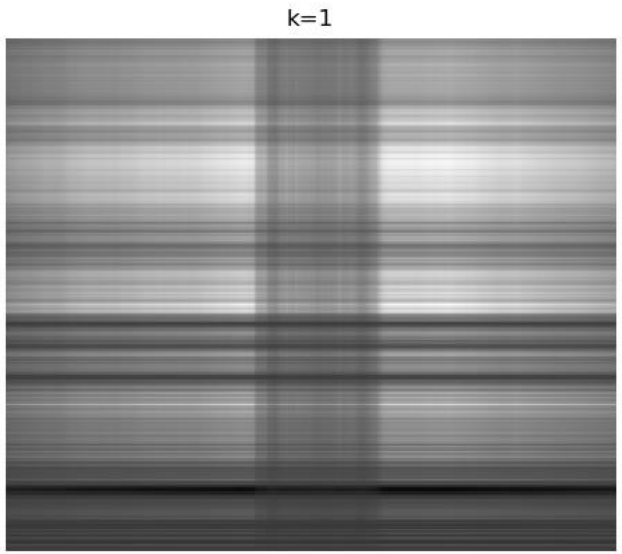
- **Etape 1:** Construction d'une matrice aléatoire P de taille $m \times r$

On note $Z=AP$.

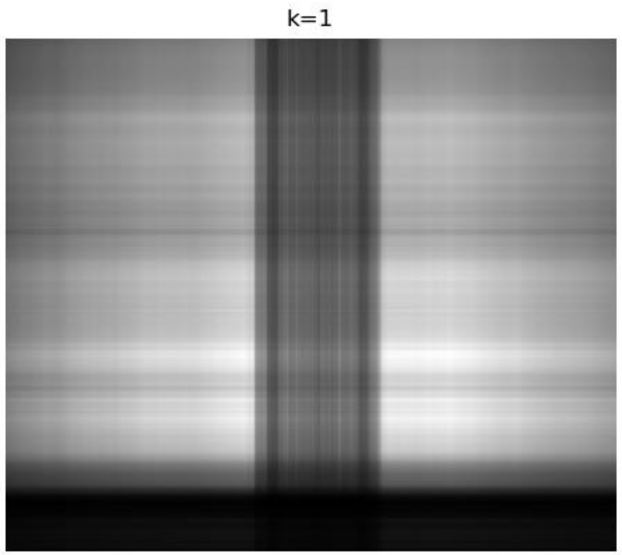
- **Etape 2:** Décomposition de Z: **$Z=QR$** où Q est orthogonale et R est triangulaire supérieure.
- **Etape 3:** On pose $Y=Q^TA$, on décompose Y en valeurs singulières : **$Y=U_1\Sigma V^T$**.
- **Etape 4:** on trouve **$U=QU_1$**

Simulation python

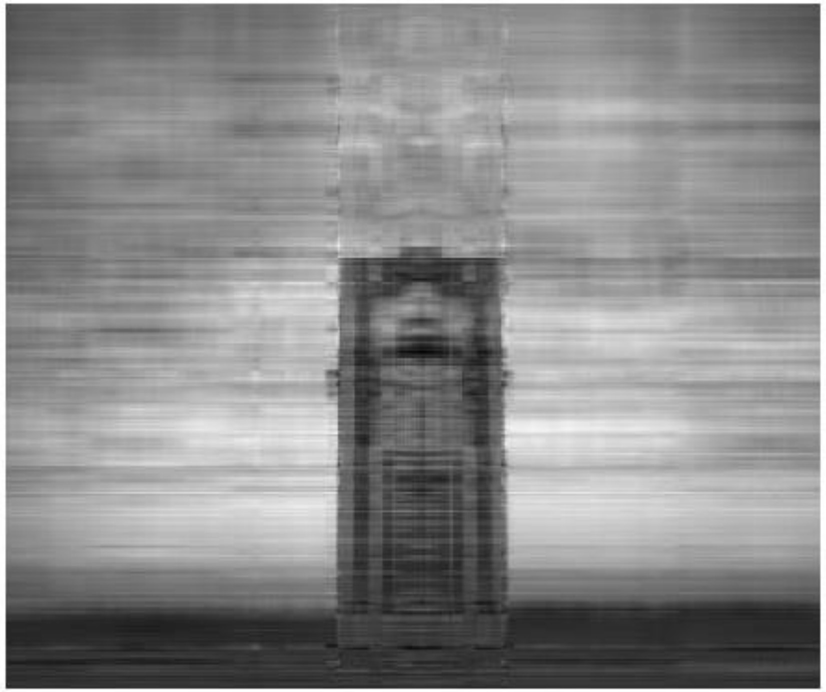
Randomisée



Normale



k=10



k=10

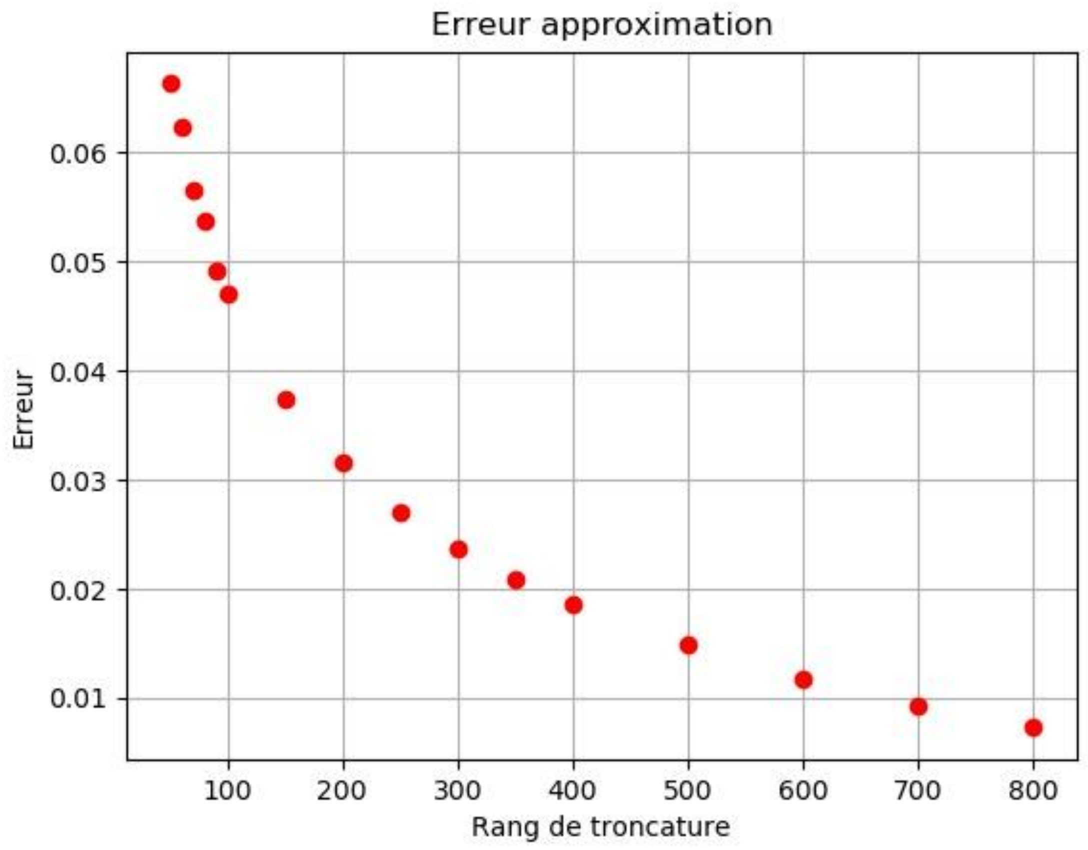
k=100

k=100

k=300

k=300

Erreur de l'approximation



Annexe

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.image import imread
def decomposition(A):
    return np.linalg.svd(A,full_matrices=True)
def decomposition_reduite(A):
    return np.linalg.svd(A,full_matrices=False)
def troncature(A,k):
    U,Sigma,VT=decomposition_reduite(A)
    Sigma=np.diag(Sigma)
    Ak=U[:,:k]@Sigma[0:k,:k]@VT[:k,:]
    return Ak
def Frobenius(A):
    return np.linalg.norm(A,ord='fro')
def norme2(A):
    return np.linalg.norm(A,ord=2)
def erreur_approx(A,k):
    Ak=troncature(A,k)
    return Frobenius(A-Ak)/Frobenius(A)
def taux_comp(A,k):
    n=A.shape[0]
    m=A.shape[1]
    return (m*n)/(k*(m+n+1))
```

```python
def aleatoire(A,k):
    return np.random.randn(A.shape[1],k)
def qr(A):
    return np.linalg.qr(A,mode='reduced')
def decomposition_rapide(A,k):
    P=aleatoire(A,k)
    Z=A@P
    Q,R=qr(Z)
    Y=np.transpose(Q)@A
    U1,Sigma,VT=decomposition_reduite(Y)
    U=Q@U1
    return U,Sigma,VT
def troncature_rapide(A,k):
    U,Sigma,VT=decomposition_rapide(A,k)
    Sigma=np.diag(Sigma)
    Ak=U[:,:k]@Sigma[0:k,:k]@VT[:k,:]
    return Ak
def erreur_rapide(A,k):
    Ak=troncature_rand(A,k)
    return Frobenius(A-Ak)/Frobenius(A)
```

```python

A=imread(r'C:\Users\Akram\Desktop\test.jpg')
B=imread(r'C:\Users\Akram\Desktop\photo.jpg')
B=np.mean(B,-1)
A=np.mean(A,-1)
plt.imshow(A)
plt.set_cmap('gray')
plt.axis('off')
plt.title('image originale')
plt.show()


# ####################Approximations:######################""
#décomposition normale
j=0
for k in (1,5,10,20,100,300):
    Ak=troncature(A,k)
    plt.figure(j+1)
    j+=1
    plt.imshow(Ak)
    plt.set_cmap('gray')
    plt.axis('off')
    plt.title('k='+str(k))
    plt.show()

# décomposition rapide:
j=0
for k in (1,5,10,20,100,300):
    Ak=troncature_rapide(A,k)
    plt.figure(j+1)
    j+=1
    plt.imshow(Ak)
    plt.set_cmap('gray')
    plt.axis('off')
    plt.title('k='+str(k))
    plt.show()
```

```python
plt.plot(np.cumsum(np.diag(Sigma))/np.sum(np.diag(Sigma)))
plt.grid()
plt.xlabel('rang de troncature k')
plt.ylabel('% des k valeurs singulières')
plt.title('Contribution des valeurs singulières')
plt.show()

##################################

plt.plot(np.diag(Sigma))
plt.grid()
plt.xlabel('k')
plt.ylabel('k-ième valeur singulière')
plt.title('valeurs singulières')
plt.show()

##################################

plt.semilogy(np.diag(Sigma))
plt.grid()
plt.xlabel('k')
plt.ylabel('k-ième valeur singulière')
plt.title('valeurs singulières')
plt.show()

##################################

L=[50,60,70,80,90,100,150,200,250,300]
T=[]
for elt in L:
    T.append(taux_comp(A,elt))
plt.plot(L,T,'bo')
plt.title('Taux de compression')
plt.xlabel('Rang de troncature k')
plt.ylabel('taux')
plt.grid()
plt.show()
```

```python

L=[50,60,70,80,90,100,150,200,250,300,350,400,500,600,700,800]
T=[]
for elt in L:
    T.append(erreur_approx(A,elt))

plt.plot(L,T,'ro'))
plt.grid()
plt.xlabel('rang de troncature k')
plt.ylabel('erreur')
plt.title('erreur approximation')
plt.show()

#####################################

L=[50,60,70,80,90,100,150,200,250,300,350,400,500,600,700,800]
T=[]
for elt in L:
    T1.append(erreur_rapide(A,elt))
plt.plot(L,T,'ro')
plt.grid()
plt.xlabel('rang de troncature k')
plt.ylabel('erreur')
plt.title('erreur approximation')
plt.show()

```