ANALYSE NUMÉRIQUE - SAGEMATH/PYTHON

Ines Abdeljaoued Tej [1]

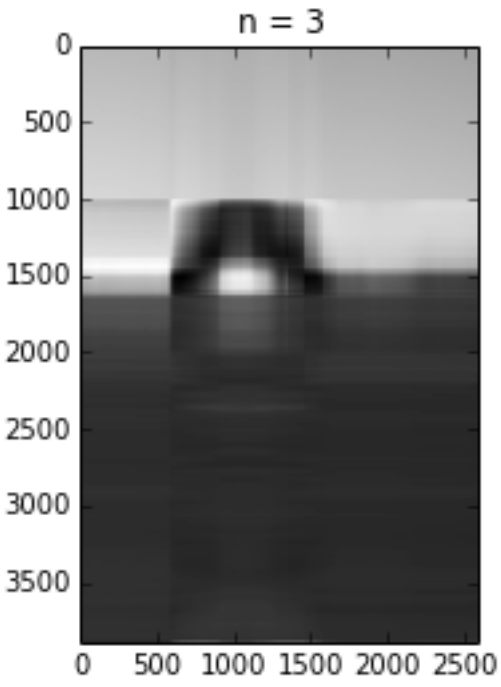
Travaux pratiques numéro 5

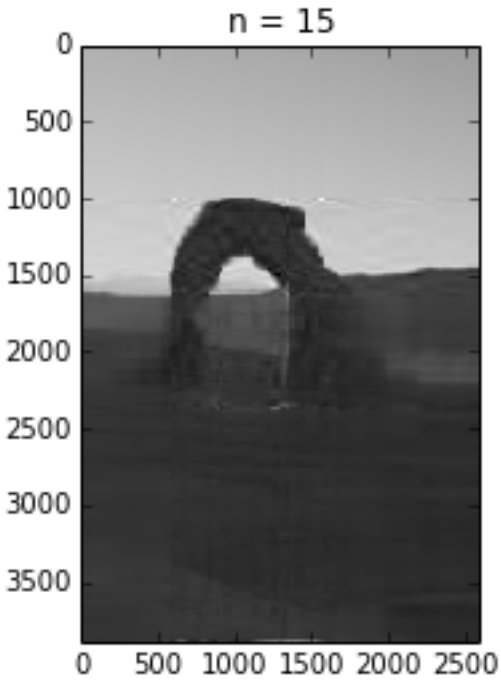
L'imagerie numérique a longtemps été un exemple d'application de la SVD. Maintenant, il existe des moyens plus élaborés et plus efficaces. Reste que la SVD est un outils intéressant pour le transfert d'image.
Une photo en noir et blanc peut être assimilée à une matrice A dont les éléments correspondent à des niveaux de gris pour les différents pixels qui constituent l'image. Chaque $a_{i,j}$ donne le niveau de gris du pixel (i, j) de l'image : $a_{i,j} \in [0, 1]$ avec par exemple $a_{i,j} = 0$ pour un pixel (i, j) blanc et $a_{i,j} = 1$ pour un pixel (i, j) noir.
Ci-dessous une image en noir et blanc de taille 2500 X 4000 pixels dont la matrice A représentative est générée par un logiciel de lecture de l'image (via un scanner ou un appareil photo numérique). le résultat renvoyé correspond à une matrice A de rang r. la matrice admet donc r valeurs singulières non nulles. Soit σ_1 la valeur maximale de ces valeurs singulières et soient $\sigma_1, \sigma_2, ...\sigma_r$ ces valeurs singulières triées par ordre décroissant. On sait alors qu'il existe une matrice U de taille 2500, une matrice V de taille 4000 telsque $A = U\ diag(\sigma_1, \sigma_2, ...\sigma_r)\ V^t$. En désignant par u_i les vecteurs colonnes de U, et par v_i les vecteurs colonnes de V, on obtient que $A = \sum_{i=1}^{r} \sigma_i\, u_i\, v_i^t$.
La connaissance de A elle-même correspond à la connaissance des 2500 X 4000 coefficients de cette matrice. Si on regarde l'expression de droite, la connaissance de A est donnée par les r valeurs singulières de A, plus r vecteurs de taille 2500 et r vecteurs de taille 4000.
Mais si on regarde de plus près les valeurs singulières non nulles $\sigma_1, \sigma_2, ...\sigma_r$, on constate que nombreuses d'entre-elles sont très petite (voir négligeable) devant les autres. On peut alors décider que $\frac{\sigma_i}{\sigma_1} > \epsilon$, où ϵ est un filtre choisi par l'utilisateur, par exemple $\epsilon = 10^{-3}$. Dans ce cas, σ_i est considérée comme nulle : on réduit ainsi le nombre de valeurs singulières à conserver dans l'expression de A selon la décomposition SVD.
Notons $\sigma_1, \sigma_2, ...\sigma_k$ les valeurs singulières conservées. Dans l'exemple proposé ci-dessous, avec $k = 3$, l'image est floue mais on constate facilement que c'est un paysage. Pour $k = 13$ l'image est déjà ressemblante à l'originale. pour $k = 80$ on peut dire que l'image obtenue est identique à l'originale. La connaissance de A est alors donnée par la connaissance de 80 valeurs singulières, plus 80 vecteurs de taille 2500, plus de 80 valeurs de taille 4000. Soit 520 080 au lieu de 10 000 000.

1 A faire sur Python :

1. Donnez la décomposition en valeurs sigulières de la matrice $A = 1, 1, 1, -1$

1. inestej@gmail.com, http://bit.ly/1JEWZiK





2. Trouvez la commande qui permet d'effectuer la décomposition en valeurs singulières de A sur SAGEmath
3. Téléchargez une image sous format .png
4. Calculez la SVD de cette image en utilisant les logiciels numpy et pylab.

1.1 Correction

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time

from PIL import Image


img = Image.open('input/img_6847.jpg') #ici l'image chargée est input/img_6847.jpg
imggray = img.convert('LA')
plt.figure(figsize=(9, 6))
plt.imshow(imggray);

#On la transforme en une matrice numpy
imgmat = np.array(list(imggray.getdata(band=0)), float)
imgmat.shape = (imggray.size[1], imggray.size[0])
imgmat = np.matrix(imgmat)
```

```
plt.figure(figsize=(9,6))
plt.imshow(imgmat, cmap='gray');

#Le calcul de la SVD se fait comme suit :
U, sigma, V = np.linalg.svd(imgmat)

#Ici la reconstruction de l'image
reconstimg = np.matrix(U[:, :1]) * np.diag(sigma[:1]) * np.matrix(V[:1, :])
plt.imshow(reconstimg, cmap='gray');

for i in xrange(2, 4):
    reconstimg = np.matrix(U[:, :i]) * np.diag(sigma[:i]) * np.matrix(V[:i, :])
    plt.imshow(reconstimg, cmap='gray')
    title = "n = %s" % i
    plt.title(title)
    plt.show()

for i in xrange(5, 51, 5):
    reconstimg = np.matrix(U[:, :i]) * np.diag(sigma[:i]) * np.matrix(V[:i, :])
    plt.imshow(reconstimg, cmap='gray')
    title = "n = %s" % i
    plt.title(title)
    plt.show()
```

1.2 Un code plus intéracif

```
import numpy
import pylab
U, s, V = numpy.linalg.svd(matrix(2,2,[1,1,1,-1]))
U, s, V

import pylab
import numpy

# Read in a png image as a *numpy array* : pylab.imread
# Convert to greyscale : numpy.mean(...,2)
#A_image = numpy.mean(pylab.imread(DATA + 'coffee.png'), 2)
#A_image = numpy.mean(pylab.imread(DATA + 'CapturFiles_11.png'), 2)
A_image = numpy.mean(pylab.imread(DATA + 'chelsea.png'), 2)

# Use numpy to compute the singular value decomposition of the image
u,s,v = numpy.linalg.svd(A_image)
S = numpy.zeros(A_image.shape)
S[:len(s),:len(s)] = numpy.diag(s)
n = A_image.shape[0]
```

```
# The SVD is a factorization of A_image as a product:  U * S * V
# We verify that this is indeed the case.
# WARNING u * S is *NOT* matrix multiplication in numpy,
# it's componentwise multiplication.  Instead use "numpy.dot" to multiply.

@interact
def svd_image(i = ( "Eigenvalues(quality)",(20,(1..A_image.shape[0])))):

    # Thus the following expression involves only the first i eigenvalues,
    # and the first i rows of v and the first i columns of u.
    # Thus we use i*m + i*n + i = i*(m+n+1) storage, compared to the
    # original image that uses m*n storage.
    A_approx = numpy.dot(numpy.dot(u[:,:i], S[:i,:i]), v[:i,:])
    p = show(graphics_array([matrix_plot(A_image), matrix_plot(A_approx)]), axes=False)


    # Of course, nothing is ever exactly right with floating point matrices...
    # so we check for closeness.
    print numpy.allclose(A_approx, A_image, 1e-1, 1e-1)
```