Abgabe - Übungsblatt [4]

Angewandte Mathematik: Numerik

[Felix Lehmann] [Markus Menke]

1. Dezember 2020

Aufgabe 1

Here comes your text . . .

Aufgabe 2

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

def ComputeSVD(A:np.matrix):
    """Given a matrix A use the eigen value decomposition
        to compute a SVD
        decomposition. It returns a tuple (U,Sigma,V) of
           np.matrix objects."""
    k = np.linalg.matrix_rank(A)
    B = A.transpose()@A
    w, V = eig(B)

    #Eigenwerte und Vektoren neu sortieren
    idx = np.argsort(w)
    w = w[idx]
    V = V[:,idx]

    #S berechnen
    S = np.zeros(A.shape)
    for i in range(S.shape[0]):
        for j in range(S.shape[1]):
            if i==j:
                S[i][j] = sqrt(w[i])

    #U berechnen
    U = np.zeros((k+1,k+1))
    for i in range(k):
        U[:,i] = (1/S[i][i] * A * V[:,i]).flat
    U = np.linalg.qr(U)[0]
```

```python
    return np.asmatrix(U),np.asmatrix(S),np.asmatrix(V)


def PseudoInverse(A:np.matrix):
    """Given a matrix A use the SVD of A to compute the
        pseudo inverse. It returns the pseudo inverse as a
         np.matrix object."""
    U,S,V = ComputeSVD(A)
    #invert S
    S_inv = S.H
    for i in range(S_inv.shape[0]):
        S_inv[i,i] = 1/S_inv[i,i]
    return np.asmatrix(V*S_inv*U.H)


def LinearSolve(A:np.matrix, b:np.ndarray):
    """Given a matrix A and a vector b this function
        solves the linear equations
       A*x=b by solving the least squares problem of
           minimizing |A*x-b| and
       returns the optimal x."""
    x = PseudoInverse(A)*b
    return x


if (__name__ == "__main__"):
    # Try the SVD decomposition
    A = np.matrix([
        [1.0,1.0,1.0],
        [1.0,2.0,3.0],
        [1.0,4.0,9.0],
        [1.0,8.0,27.0]
    ])
    U, Sigma, V = ComputeSVD(A)
    print(U)
    print(Sigma)
    print(V)
    print("If the following numbers are nearly zero, SVD
        seems to be working.")
    print(np.linalg.norm(U*Sigma*V.H - A))
    print(np.linalg.norm(U.H*U-np.eye(4))) #Testet eure
        Framewroks doch mal auf Fehler...
    print(np.linalg.norm(V.H*V-np.eye(3)))
    # Try solving a least squares system
    b = np.matrix([1.0,2.0,3.0,4.0]).T
    x = LinearSolve(A,b)
    print("If the following number is nearly zero, linear
        solving seems to be working.")
    print(np.linalg.norm(x-np.linalg.lstsq(A,b,rcond=None
        )[0]))
```

Aufgabe 3

Here comes your text ...

Aufgabe 4

```python
import numpy
import numpy as np
from PIL import Image
from matplotlib import cm, pyplot

def Compress(Image, ComponentCount):
    """This function uses a singular value decomposition
       to compress an image.
      \param Image An quadratic array providing the image
          . Entries provide the
                brightness of indidividual pixels, rows
                    correspond to scanlines.
      \param ComponentCount The number of singular values
           to be maintained in the
                compressed representation.
      \return A tuple (U,SingularValues,V,
          CompressionRatio) such that U*Sigma*V^*
                provides an approximation to the original
                    image when Sigma is a
                diagonal matrix with SingularValues on its
                    main diagonal.
                CompressionRatio should provide the
                    quotient of the number of scalars
                in Image and the number of scalars in the
                    returned representation of
                Image."""
    U,S,V = np.linalg.svd(Image)
    result_U = U[:,:ComponentCount]
    result_S = S[:ComponentCount]
    result_V = V[:ComponentCount,:]

    comp = (U.size + S.size + V.size) / (result_U.size +
        result_S.size + result_V.size)

    return np.asmatrix(result_U),np.asarray(result_S),np.
        asmatrix(result_V),int(comp)


def Decompress(U,SingularValues,V):
```

```python
    """Given a compressed representation of an image as
        produced by Compress() this
        function reconstructs the original image
            approximately and returns it."""
    return U * np.diag(SingularValues) * V


if(__name__=="__main__"):
    # Define the task
    ImageFileNameList=["Lena","Stoff","Stoff2"]
    ComponentCountList=[1,4,8,32,64]
    # Iterate over all tasks and generate one large plot
    PlotIndex=1
    for ImageFileName in ImageFileNameList:
        ImagePath="./"+ImageFileName+".png"
        img=Image.open(ImagePath)
        # Convert to numpy array
        imgmat = np.array(list(img.getdata(band=0)),
            float)
        # Reshape according to orginal image dimensions
        imgmat.shape = (img.size[1], img.size[0])
        imgmat = np.matrix(imgmat)
        for ComponentCount in ComponentCountList:
            # Define a subplot for this decompressed
                image
            Axes=pyplot.subplot(len(ImageFileNameList),
                len(ComponentCountList),PlotIndex)
            Axes.set_xticks([])
            Axes.set_yticks([])
            Axes.set_title(ImageFileName+", p="+str(
                ComponentCount))
            PlotIndex+=1
            # Apply compression
            U,SingularValues,V,CompressionRatio=Compress(
                imgmat,ComponentCount)
            # Apply decompression and show the result
            DecompressedImage=Decompress(U,SingularValues
                ,V)
            pyplot.imshow(DecompressedImage,cmap='gray')
            # Compute and print the compression ratio
            print("Compression ratio for p="+str(
                ComponentCount)+" is "+str(
                CompressionRatio)+":1.")
        print("")
    pyplot.show()
```

Im stark komprimierten Bild verbleiben Intänsitäts werte der Zeilen und Spalten. Da „Stoff“ hauptsächlich horizontale und vertikale Bestandteile hat, lässt sich dies so besser komprimieren.