ILI 285
Laboratorio #3

Alonso Sandoval Acevedo
asandova@alumnos.inf.utfsm.cl
201073011-5

Hernán Vargas Leighton
hvargas@alumnos.inf.utfsm.cl
201073009-3

12 de junio de 2014

1. Descripción del experimento

La descomposición en valores singulares puede ser aplicada a cualquier matriz, además nos da la información sobre que *partes* de la matriz tienen mayor importancia en ella, es por esto que es muy utilizada en la compresión de imágenes. En esté laboratorio se aplicará ésta factorización tanto en su versión reducida como completa y con ella se comprimirán diferentes archivos para probar su utilidad. Esperamos que el proceso de compresión nos de una buena calidad a un bajo peso y no *corrompa* la imagen en el proceso.

2. Desarrollo

2.1. Factorización SVD

	Factorización Reducida	**Factorización Completa**
$\hat{U}$ ó U	$\begin{bmatrix} -0{,}6083709 & 0{,}1215856 & -0{,}74505154 \\ -0{,}38579269 & 0{,}79420378 & 0{,}33725021 \\ -0{,}67742713 & -0{,}48592859 & 0{,}52754073 \\ 0{,}14879966 & 0{,}34399232 & 0{,}22991582 \end{bmatrix}$	$\begin{bmatrix} -0{,}6083709 & 0{,}1215856 & -0{,}74505154 & 0{,}24494897 \\ -0{,}38579269 & 0{,}79420378 & 0{,}33725021 & -0{,}32659863 \\ -0{,}67742713 & -0{,}48592859 & 0{,}52754073 & 0{,}16329932 \\ 0{,}14879966 & 0{,}34399232 & 0{,}22991582 & 0{,}89814624 \end{bmatrix}$
$\hat{\Sigma}$ ó Σ	$\begin{bmatrix} 4{,}81685566 & 0 & 0 \\ 0 & 3{,}40016423 & 0 \\ 0 & 0 & 1{,}49558844 \end{bmatrix}$	$\begin{bmatrix} 4{,}81685566 & 0 & 0 \\ 0 & 3{,}40016423 & 0 \\ 0 & 0 & 1{,}49558844 \\ 0 & 0 & 0 \end{bmatrix}$
V	$\begin{bmatrix} -0{,}36136581 & -0{,}19504885 & -0{,}91179532 \\ -0{,}05224848 & 0{,}98057553 & -0{,}18905484 \\ 0{,}9309591 & -0{,}02067804 & -0{,}36453748 \end{bmatrix}$	$\begin{bmatrix} -0{,}36136581 & -0{,}19504885 & -0{,}91179532 \\ -0{,}05224848 & 0{,}98057553 & -0{,}18905484 \\ 0{,}9309591 & -0{,}02067804 & -0{,}36453748 \end{bmatrix}$

a) El **significado** de las matrices generadas es por la factorización SVD reducida de $A \in \mathbb{C}^{4\times 3}$ es:

- $\hat{U}$ es una matriz con columnas ortonormales de la misma dimensión que la matriz A.
- $\hat{\Sigma}$ es una matriz diagonal con los valores singulares de la matriz A de dimensión 3×3.
- V es una matriz cuadrada unitaria con columnas ortonormales de dimensión 3×3.

b) Las **diferencias** entre las matrices generadas por la factorización reducida y la completa son:

- $\hat{U}$ pasa a ser U, es decir, se convierte en una matriz unitaria gracias a que se le agregaron los vectores ortonormales que le faltaban, por lo tanto ahora U es una matriz cuadrada de dimensiones 4×4.
- Debido al cambio de U, Σ pasa a ser $\hat{\Sigma}$ para tener las mismas dimensiones que la matriz A, con esté fin se le agregan vectores nulos como sea necesario (en este caso solo 1).

2.2. Compresión de Imágenes

NOTA: Los gráficos generados se encuentran en las siguientes páginas.

a) La siguiente tabla muestra las dimensiones de las matrices generadas por la factorización SVD de las imágenes y ficheros de texto entregados.

Archivo	U	Σ	V
fractal.png	487×487	487×487	510×510
paisaje.bmp	576×576	576×576	576×576
logo-di.png	350×350	350×350	388×388
matriz1.txt	512×512	512×512	512×512
matriz2.txt	512×512	512×512	512×512
matriz3.txt	512×512	512×512	512×512
matriz4.txt	512×512	512×512	512×512
matriz5.txt	512×512	512×512	512×512

Los valores singulares de las matrices Σ se encuentran ordenados por tamaño, decayendo dependiendo de la imagen a la cual están asociados. La función `svd` de `numpy` retorna un `array` con estos valores, los cuales, a pesar que provienen de la factorización SVD completa no incorporan los vectores nulos finales para tener dimensión $m \times n$.

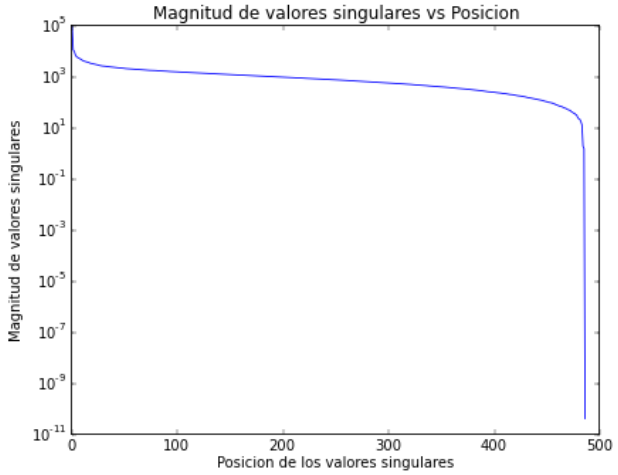
b) Dada la observación de los distintos gráficos para la compresión de las imágenes, notamos que en la mayoría la pendiente de calidad se acentúa decayendo más rápido luego de considerar el 10 % de los valores singulares, por lo tanto, creemos que éste es el tope para que la imagen sea aceptable. Sobre el 30 %, no hay cambios muy notorios, por lo que hasta aquí no tendríamos una perdida significativa de calidad.
Cabe destacar que las curvas de esté gráfico son prácticamente una el inverso aditivo de la otra.

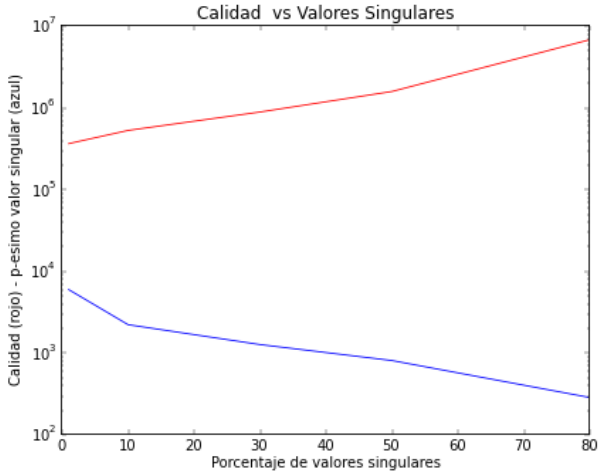
c) Observando los gráficos, es muy notorio que la razón de comprensión decae más rápido bajo el 10 % de los valores singulares, lo cual es congruente con el decaimiento de la calidad de la imagen. Cabe destacar que sobre el 10 % el decaimiento se muestra "suave", aumentando la pendiente considerablemente luego de este valor.
Caso especial es el archivo `matriz1.txt` ya que, debido a que todos sus valores singulares son de la misma magnitud, no disminuye mucho su calidad con respecto a la compresión.

3. Conclusiones

Hemos logrado analizar varios casos de compresión de imágenes por medio de la factorización SVD, ésto nos permite comprimir la imagen para que ocupe un menor espacio en el disco duro, pero por ello debemos pagar el precio en la calidad de dicha imagen. En general, notamos que con un margen de compresión del 10 %, la perdida de calidad es lo suficientemente pequeña como para ser aceptable.
Gracias a los gráficos de las matrices generadas por los ficheros de texto, podemos ver casos especiales en los cuales los valores singulares se comportan de manera tal que se hace posible una compresión muy mayor a lo que sucede normalmente, pero creemos que no es muy común en la generalidad de las imágenes.
Como vimos en los gráficos, la compresión puede disminuir notablemente el tamaño de las imágenes, pero esto solo se notará si disponemos de los medios necesarios para guardar dicho archivo como su descomposición y no como una matriz, lo que implicaría un proceso de visualización más complejo.

Figura 1: Gráficos para fractal.png





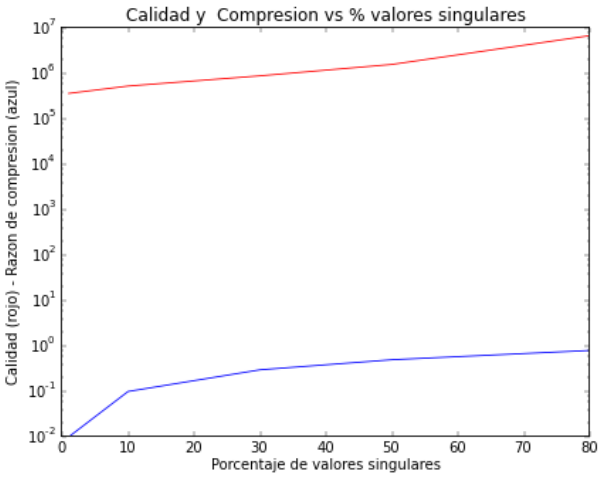
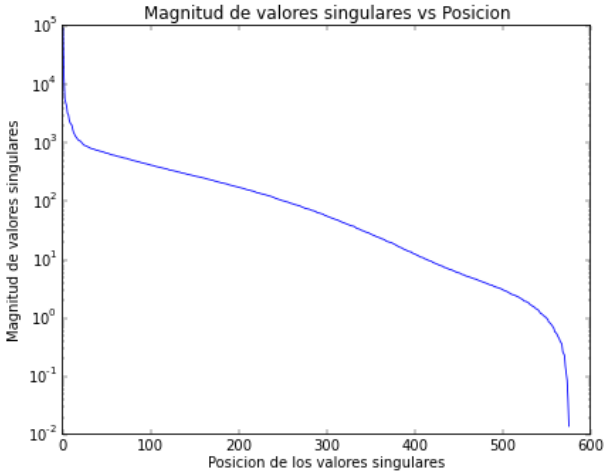
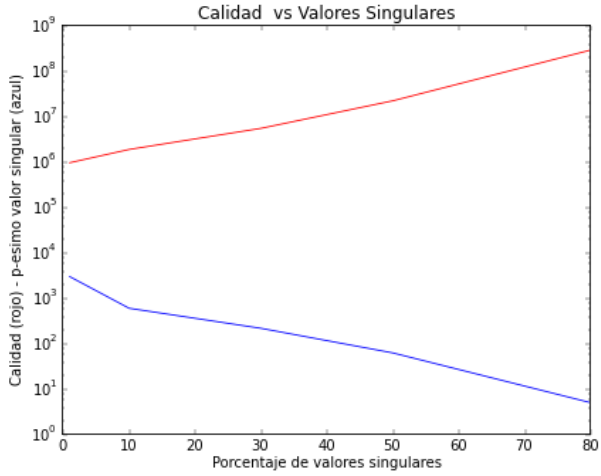


Figura 2: Gráficos para paisaje.png





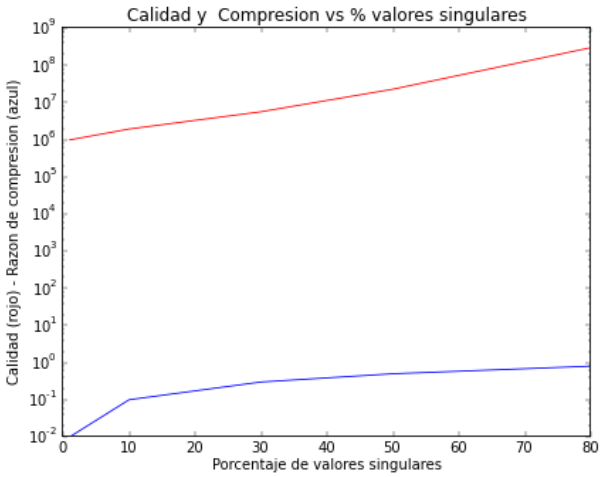
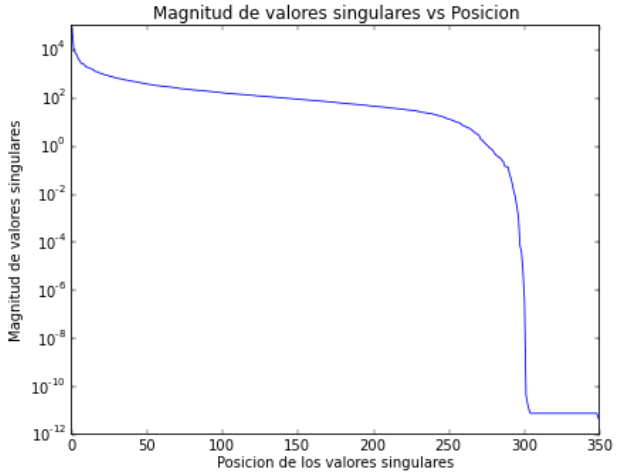
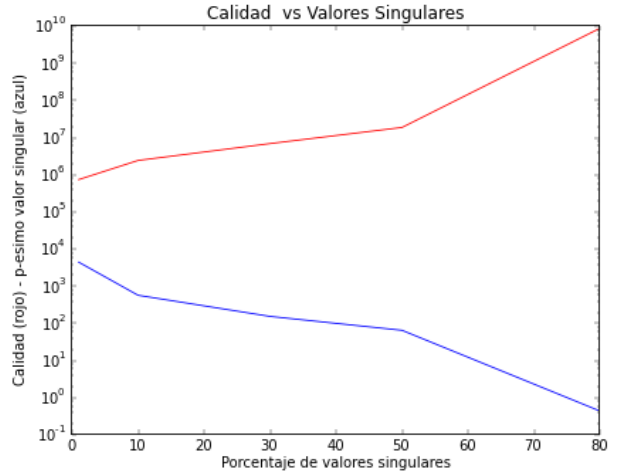


Figura 3: Gráficos para logo-di.bmp





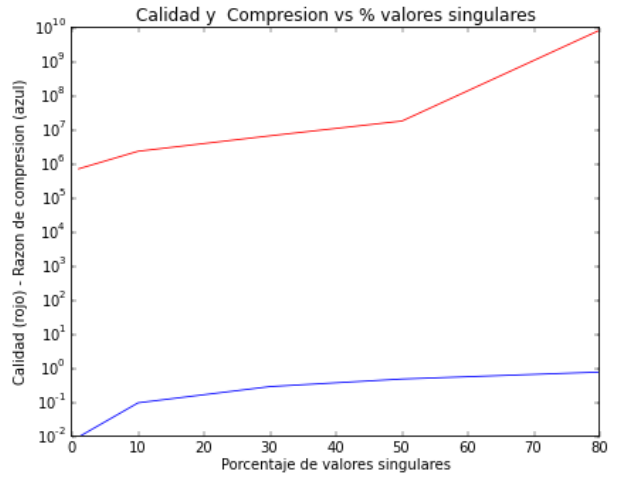
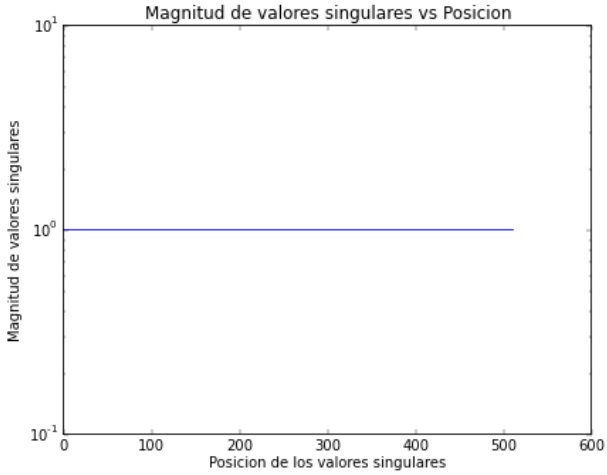
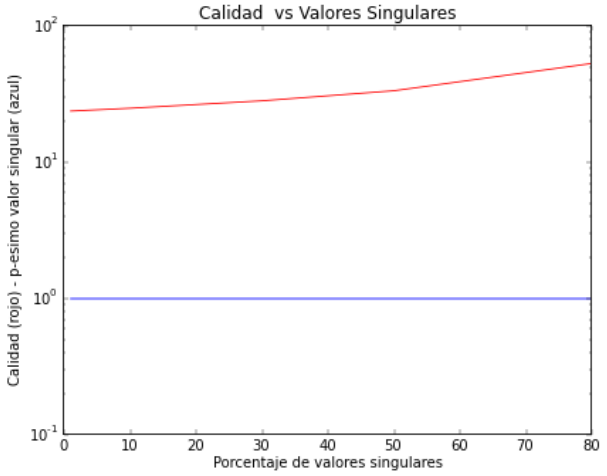


Figura 4: Gráficos para matriz1.txt





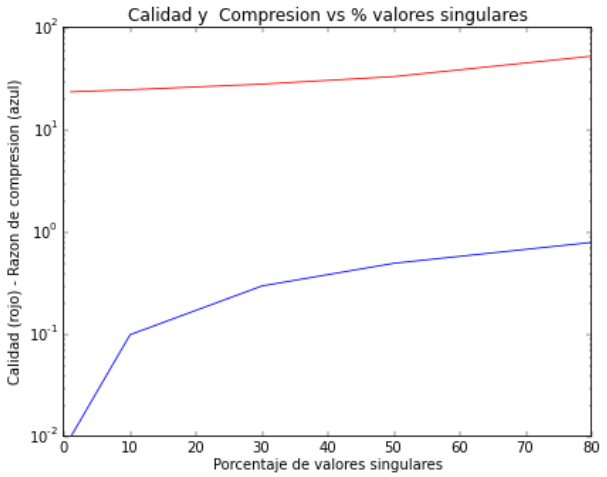
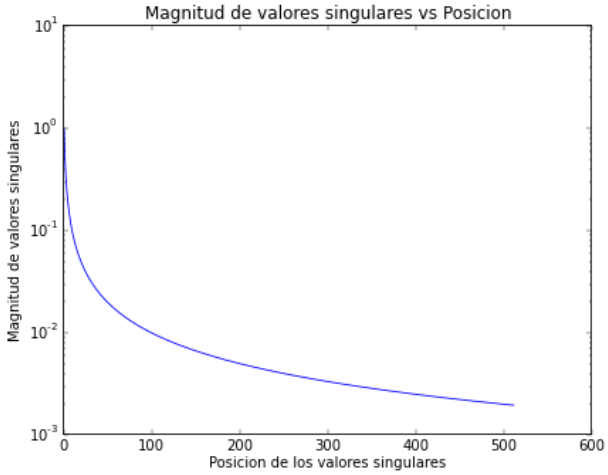
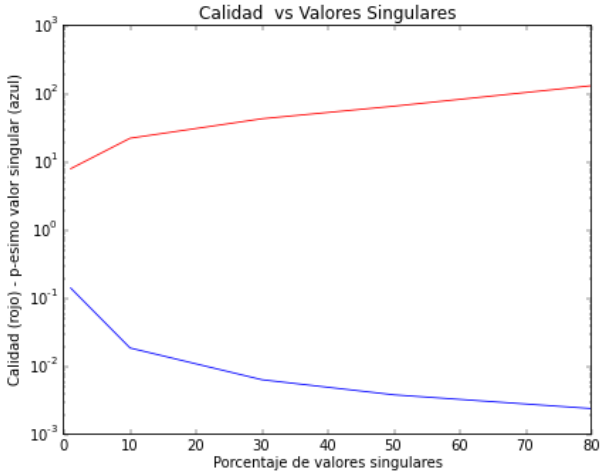


Figura 5: Gráficos para matriz2.txt





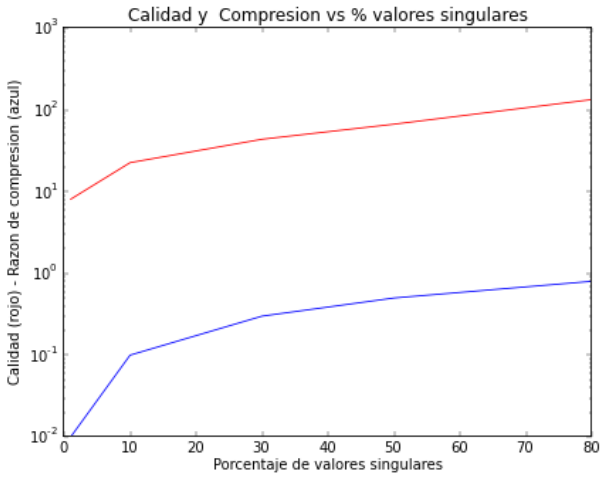
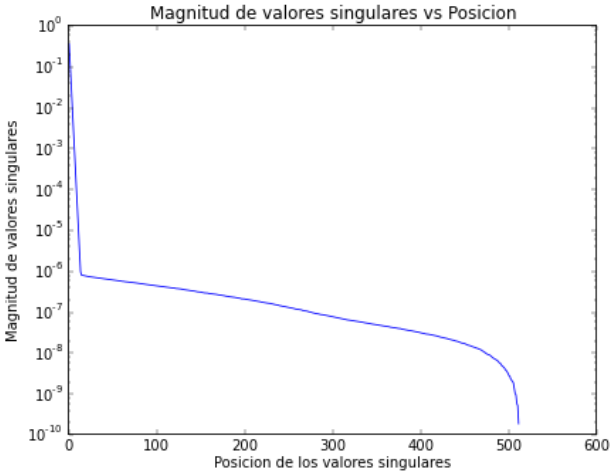
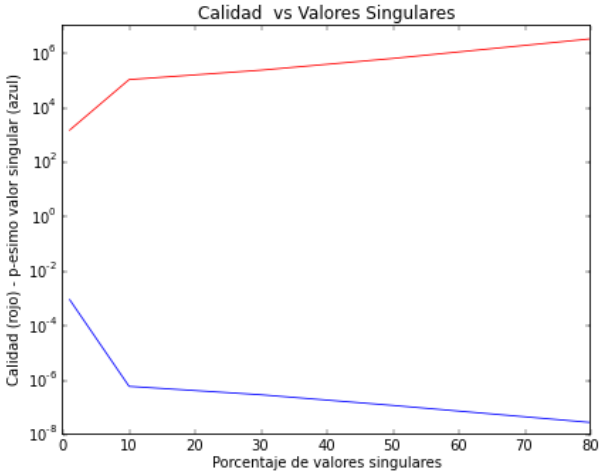


Figura 6: Gráficos para matriz3.txt





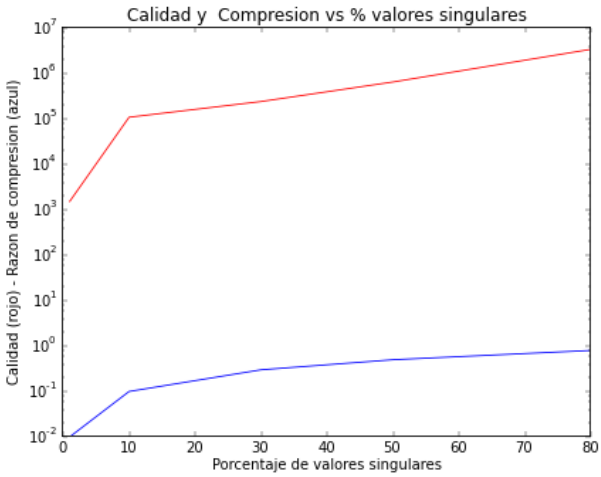
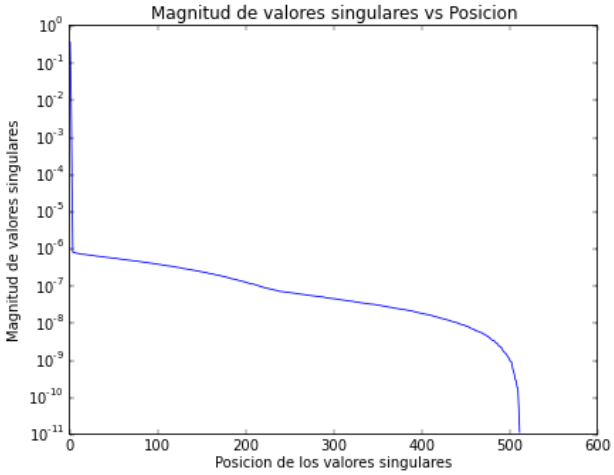
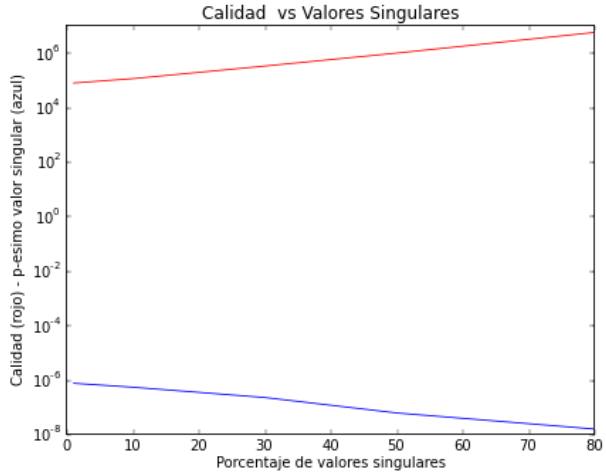


Figura 7: Gráficos para matriz4.txt





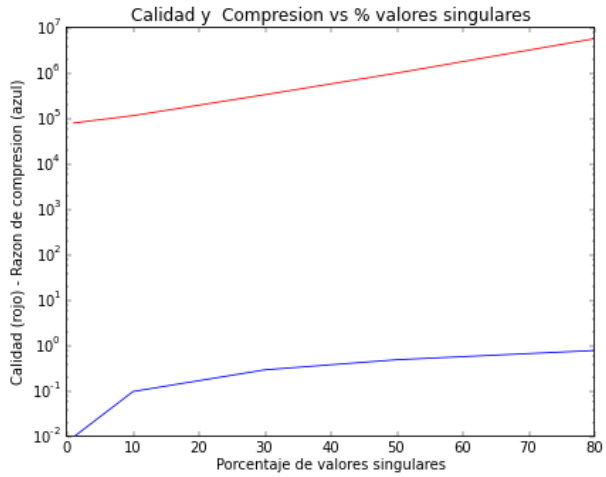
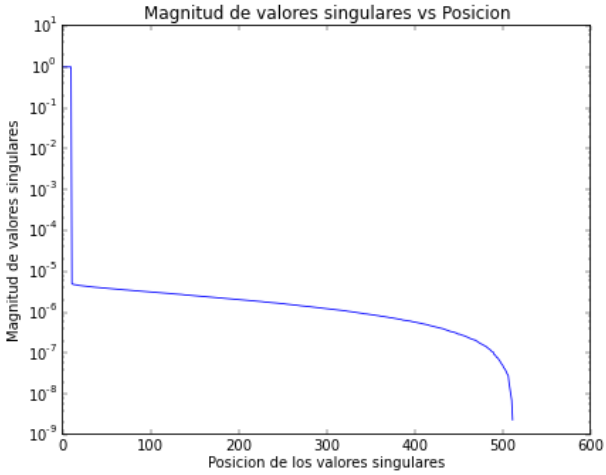
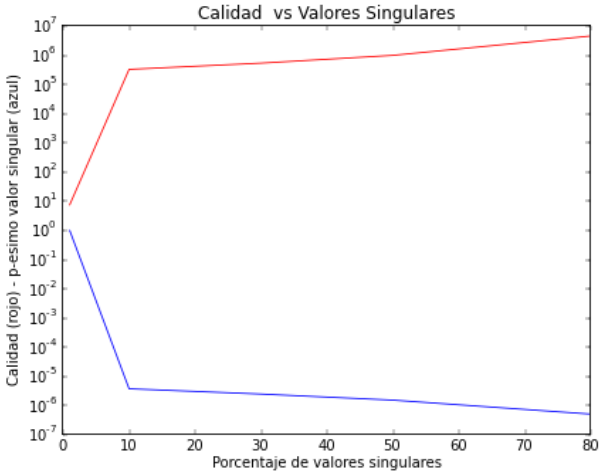
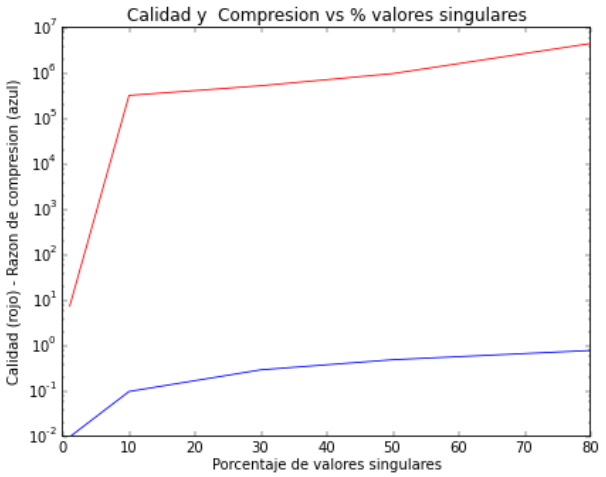


Figura 8: Gráficos para matriz5.txt







4. Anexo

../Codigos/Laboratorio3.py

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#SVD
import numpy
from numpy.linalg import svd
from numpy import matrix, array
#Manejo de imágenes
from PIL import Image
#Gráficos
import matplotlib.pyplot as plt

### Parte 0: Datos ###
e_mach = 10**(-16)
A = matrix([[0, 1, 3],
            [1, 3, 1],
            [2,-1, 3],
            [0, 1,-1]])
#Imágenes a ocupar
imgs = {'fractal': "Dataset/imagenes/fractal.png",
        'logo-di': 'Dataset/imagenes/logo-di.png',
        'paisaje': 'Dataset/imagenes/paisaje.bmp' }
#matrices formato txt
mTxt = {"matriz1":  "Dataset/matrices/matriz1.txt",
        "matriz2":  "Dataset/matrices/matriz2.txt",
        "matriz3":  "Dataset/matrices/matriz3.txt",
        "matriz4":  "Dataset/matrices/matriz4.txt",
        "matriz5":  "Dataset/matrices/matriz5.txt" }

### Parte 1: Factorización SVD ###
#Retorna el SVD, full indica si será completo o reducido.
def getSVD(A, verbose=False, full=True):
    u,z,v = svd(A, full_matrices = 1 if full else 0)
    if verbose: print 'U:', u.shape, ' Z:', z.shape, ' V:', v.shape
    return u,z,v
#Iteración para imprimir.
for name, bol in (('reducida', False), ('completa', True)):
    print 'Factorizacion', name,  'de A:'
    u,z,v = getSVD(A,True,bol)
    print 'U:\n',u,'\nZ:\n',z,'\nV:\n',v,'\n'

### Parte 2: Compresión de imágenes ###
#imgToMatx toma una imagen Image y retorna una matriz.
def imgToMatx(img):
    x, y = img.size
    array = list(img.getdata())
    return matrix([array[x*i:x*(i+1)] for i in range(y)])

#txtToMatx toma la ruta a un fichero de texto, retorna una matriz con él.
def txtToMatx(fdir):
    f = open(fdir, "r")
    # matrix puede leer desde string si las lineas están separadas por ';'
    data = f.read().replace('\n',';')[:-1]
```

```
    f.close()
    return matrix(data)

#imgOpen abre una imagen y la retorna como una matriz.
def imgOpen(fdir, mode='F'):
    img = Image.open(fdir).convert(mode)
    return imgToMatx(img)

#compress toma una matriz m y un valor de compresión y retorna m comprimida.
#Si el parámetro last_value es True retornará además el ultimo valor singular,
#esto es útil solo para los gráficos.
def compress(matx, compress_value, verbose=False, last_value=False):
    if compress_value > 1 or compress_value < 0:
        raise ValueError('Compresión debe ser un valor entre 0 y 1')
    u,z,v = getSVD(matx,verbose)
    r = z.shape[0]              #Dimensión de matriz con valores singulares.
    result = matrix("0")
    #Rearmando matriz (propiedad 5) descartando un % de valores singulares.
    for k in range(0,r):
        result = result + ((u[:,k]*z[k])*v[k])
        if k > r*compress_value:
            lvalue = z[k]
            break
    if last_value: return result, lvalue
    else: return result

#quality determina la calidad entre dos matrices de la misma imagen.
def quality(mtx,mtx2):
    dmtx = svd(mtx - mtx2)
    mtx = svd(mtx)
    fnorm = 0
    for x in mtx[1]:            #Norma de Frobenius para la imagen original.
        fnorm = x**2 + fnorm
    den = fnorm**0.5 + fnorm    #Denominador.
    fnorm = 0
    for x in dmtx[1]:           #Norma de Frobenius para la diferencia.
        fnorm = x**2 + fnorm
    num = fnorm**0.5            #Numerador.
    return den/num              #Inversa de la división.

## Para la creación de gráficos: ##
#Gráfico 1, recibe una matriz mtx y gráfica sus valores singulares
def graph1(mtx, filename, path="../Informe/Graficos/"):
    u, svalue, v = getSVD(mtx,True)
    size = svalue.size
    plt.plot(range(0,size),svalue)
    plt.yscale('log')
    plt.ylabel('Magnitud de valores singulares')
    plt.xlabel('Posicion de los valores singulares')
    plt.title('Magnitud de valores singulares vs Posicion')
    plt.savefig(path+filename+'-svalue.png', dpi = 60)
    print('Grafico "'+path+filename+'-svalue.png" guardado.')
    plt.hold(False)

#Gráfico 2: Recibe una matriz mtx y crea los gráficos de calidad v/s el valor
```

```
#           singular y calidad v/s tamaño de la imagen.
#Advertencia: Esta función se basa en comprimir varias veces la imagen por lo
#             que puede ser realmente lenta.
def graph2(mtx, filename, path="../Informe/Graficos/"):
    qlty = [0.8,0.5,0.3,0.1,0.01]
    perc = [80, 50, 30, 10, 1]
    v_quality, v_last, size_compress = [],[],[]
    m,n = mtx.shape
    p = svd(mtx)[1].size
    calcSize = lambda x: 4*x*(m+n+1)
    orig_size = calcSize(p)
    #Bucle con varias compresiones de la imagen.
    for c in qlty:
        mtx2, pC = compress(mtx, c, last_value=True)
        v_quality.append(quality(mtx, mtx2))
        v_last.append(pC)
        size_compress.append(calcSize(p*c)/orig_size)
    #Creando el gráfico de calidad:
    plt.plot(perc, v_quality, 'r', perc, v_last, 'b')
    plt.yscale('log')
    plt.ylabel('Calidad (rojo) - p-esimo valor singular (azul)')
    plt.xlabel('Porcentaje de valores singulares')
    plt.title('Calidad  vs Valores Singulares')
    plt.savefig(path+filename+'-quality.png', dpi = 60)
    print('Grafico "'+path+filename+'-quality.png" guardado.')
    plt.hold(False)
    #Creando el gráfico de tamaño:
    plt.plot(perc, v_quality, 'r', perc, size_compress, 'b')
    plt.yscale('log')
    plt.ylabel('Calidad (rojo) - Razon de compresion (azul)')
    plt.xlabel('Porcentaje de valores singulares')
    plt.title('Calidad y  Compresion vs % valores singulares')
    plt.savefig(path+filename+'-size.png', dpi = 60)
    print('Grafico "'+path+filename+'-size.png" guardado.')
    plt.hold(False)

ENABLE_PLOT = True #True creará los graficos, demorará mucho más.

#Bucle para las imágenes.
for key in imgs:
    print 'Para', key, 'tenemos:'
    m = imgOpen(imgs[key])
    if ENABLE_PLOT:
        graph1(m, key)
        graph2(m, key)
    else: getSVD(m,True)
#Bucle para las matrices.
for key in mTxt:
    print 'Para', key, 'tenemos:'
    m = txtToMatx(mTxt[key])
    if ENABLE_PLOT:
        graph1(m, key)
        graph2(m, key)
    else: getSVD(m,True)
```

HV/AS/LaTeX 2_ε