Matrix Approximation using the SVD

Source:

`Matrix Methods for Computational Modeling and Data Analytics` author: Mark Embree, Virginia Tech

Most of what is in this notebook is taken directly or adapted from chapter 6 `Matrix Approximation via the SVD` (see source above).

In [15]:
```python
# imports for numerical examples

import numpy as np
import cv2
import math
import matplotlib.pyplot as plt
```

Introduction to matrix norms

definition

The norm a matrix $\mathbf{A}$ is the maximum norm of the matrix / vector product $||\mathbf{A}\cdot\mathbf{x}|| \ : \ ||\mathbf{x}|| = 1$. The constraint $||\mathbf{x}|| = 1$ is necessary. Otherwise the matrix norm could grow arbitrary large just be scaling vector $\mathbf{x}$.

Before going further into the computation of the matrix norm the topic of the matrix norm of orthogonal matrices is addressed:

Let $\mathbf{Q}$ be a orthogonal matrix with orthonormal column vectors. The quadratic norm $||\mathbf{Q}\cdot\mathbf{x}||$ is computed:

$$\begin{align}
||\mathbf{Q}\cdot\mathbf{x}||^2 = \mathbf{x}^T\cdot\mathbf{Q}^T\mathbf{Q}\cdot\mathbf{x} = \mathbf{x}^T\cdot\mathbf{x} = ||\mathbf{x}||^2 \tag{1}\\
\rightarrow \tag{2}\\
||\mathbf{Q}\cdot\mathbf{x}|| = ||\mathbf{x}|| \tag{3}
\end{align}$$

Thus premultiplication by an orthogonal matrix by some vector preserves the norm of the vector.

Now we revisit the matrix norm $||\mathbf{A}\cdot\mathbf{x}||$ by expressing $\mathbf{A}$ by its `SVD`.

$$\begin{align}
||\mathbf{A}\cdot\mathbf{x}||^2 = \mathbf{x}^T\cdot\mathbf{A}^T\mathbf{A}\cdot\mathbf{x} = \ = \mathbf{x}^T\cdot\left(\mathbf{U\Sigma V}^T\right)\cdot\mathbf{U\Sigma V}^T\cdot\mathbf{x} \tag{4}\\
= \mathbf{x}^T\cdot\mathbf{V\Sigma U}^T\mathbf{U\Sigma V}^T = \mathbf{x}^T\cdot\mathbf{V\Sigma\Sigma V}^T\cdot\mathbf{x} = ||\mathbf{\Sigma V}^T\cdot\mathbf{x}||^2 \tag{5}
\end{align}$$

Defining a new vector $\mathbf{y}$ by $\mathbf{y} = \mathbf{V}^T \cdot \mathbf{x}$ it is shown that $||\mathbf{y}|| = ||\mathbf{x}||$.

$$||\mathbf{y}||^2 = \mathbf{y}^T\mathbf{y} = \mathbf{x}^T \cdot \mathbf{V}\mathbf{V}^T \cdot \mathbf{x} = \mathbf{x}^T \cdot \mathbf{x} = ||\mathbf{x}||^2 \tag{6}$$
$$\rightarrow \tag{7}$$
$$||\mathbf{y}|| = ||\mathbf{x}|| \tag{8}$$

$$||\mathbf{A} \cdot \mathbf{x}||^2 = ||\mathbf{\Sigma} \cdot \mathbf{y}||^2 \tag{9}$$

$$||\mathbf{\Sigma} \cdot \mathbf{y}||^2 = \sigma_1^2 \cdot |\mathbf{y}_1|^2 + \cdots + \sigma_r^2 \cdot |\mathbf{y}_r|^2 \ \le \sigma_1^2 \cdot \sum_{j=1}^{r} |\mathbf{y}_j|^2 = \sigma_1^2 \cdot ||\mathbf{y}||^2 = \sigma_1^2 \tag{10}$$

Thus the matrix norm is the largest singular value:

$$max \ ||\mathbf{A} \cdot \mathbf{x}|| = \sigma_1$$

Low rank approximations

The dyadic form of the `SVD`

$$\mathbf{A} = \sum_{j=1}^{r} \sigma_j \cdot \mathbf{u}_j \cdot \mathbf{v}_j^T$$

provides the idea of approximating a matrix. The `k` the approximation is defined as:

$$\mathbf{A}_k = \sum_{j=1}^{k\,:k \le r} \sigma_j \cdot \mathbf{u}_j \cdot \mathbf{v}_j^T$$

Accordingly the approximation error is defined as the difference

$$\mathbf{A} - \mathbf{A}_k = \sum_{j=1}^{r} \sigma_j \cdot \mathbf{u}_j \cdot \mathbf{v}_j^T - \sum_{j=1}^{k} \sigma_j \cdot \mathbf{u}_j \cdot \mathbf{v}_j^T = \sum_{j=r+1}^{r} \sigma_j \cdot \mathbf{u}_j \cdot \mathbf{v}_j^T$$

The import fact about approximate matrix $\mathbf{A}_k$ is that it is also the best possible rank `k` approximation. (no proof provided here).

Below is an example solved with `Numpy`. The matrix is chosen such as its `k=1` approximation is an excellent match for the original matrix.

In [16]:
```
Amat = np.array([[2, 100], [0, 1]])
Umat, S, Vtmat = np.linalg.svd(Amat)

# and here is the approximation for k=1
Amat_k1 = Umat[:, 0:1] @ (S[0] * Vtmat[0:1, :])
Emat = Amat - Amat_k1

print(f"Amat         :\n{Amat}\n")
print(f"Umat         :\n{Umat}\n")
print(f"S            :\n{S}\n")
print(f"Vtmat        :\n{Vtmat}\n")
print(f"Amat_k1      :\n{Amat_k1}\n")
print(f"Emat (error) :\n{Emat}\n")
```

```
Amat         :
[[  2 100]
 [  0   1]]

Umat         :
[[ 0.99995004 -0.0099955 ]
 [ 0.0099955   0.99995004]]

S            :
[1.00024995e+02 1.99950023e-02]

Vtmat        :
[[ 0.019994   0.9998001]
 [-0.9998001  0.019994 ]]

Amat_k1      :
[[1.99980018e+00 1.00000004e+02]
 [1.99900066e-02 9.99600240e-01]]

Emat (error) :
[[ 1.99820146e-04 -3.99600349e-06]
 [-1.99900066e-02  3.99760172e-04]]
```

Low rank approximation of an image

An image is loaded from a file and converted to a matrix with gray scale values.

The image is displayed.

The image is interpreted as a matrix of which the `SVD` is computed.

The singular values are displayed. It shows that of the 1200 singular values about only 100 singular values are expected to contribute to the matrix.

Several approximations to the original image are computed from a truncated `SVD`.

1. k = 4 singular values
2. k = 20 singular values
3. k = 50 singular values
4. k = 80 singular values

For k = 50 a fairly good approximation is obtained.

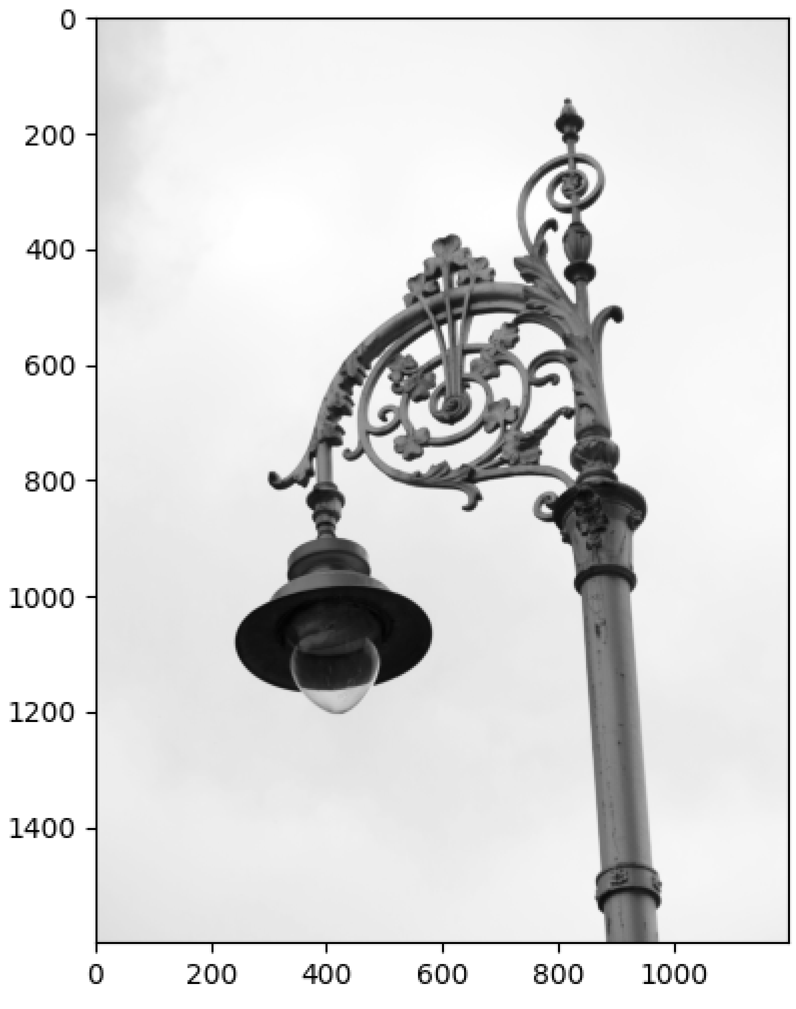
For k = 80 it is barely impossible to differentiate between the original image and its approximation.

In [18]:
```
imgFile1 = "images/lamp_dublin_1.png"
img1 = cv2.imread(imgFile1, cv2.IMREAD_GRAYSCALE)

Nx1 = img1.shape[1]
Ny1 = img1.shape[0]
print(f"size of image: {img1.size} ; shape of image: {img1.shape}")
```

```
size of image: 1920000 ; shape of image: (1600, 1200)
```

In [66]:
```
fig1 = plt.figure(1, figsize=[6, 6])
ax_f11 = fig1.add_subplot(1, 1, 1)
# plot of image
ax_f11.imshow(img1, cmap='Greys_r' );
```



In [22]:
```
# compute SVD

Umat_lamp, S_lamp, Vtmat_lamp = np.linalg.svd(img1)
```

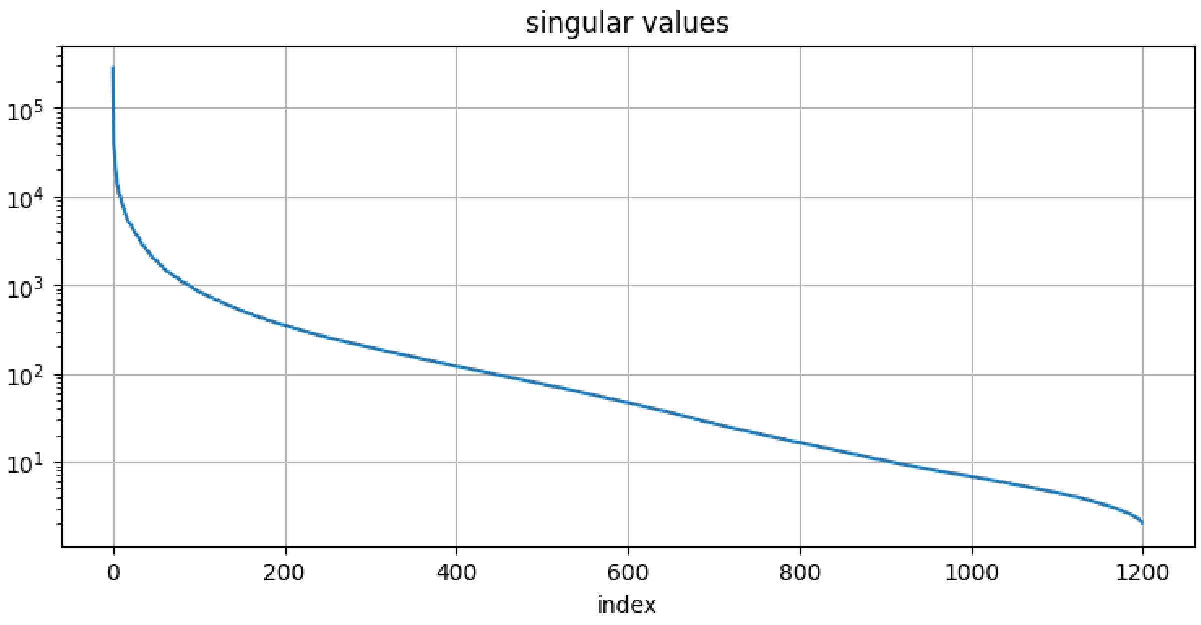
In [35]:
```
print(f"Umat_lamp.shape    : {Umat_lamp.shape}")
print(f"S_lamp.shape       : {S_lamp.shape}")
print(f"Vtmat_lamp.shape  : {Vtmat_lamp.shape}")
```

```
Umat_lamp.shape   : (1600, 1600)
S_lamp.shape      : (1200,)
Vtmat_lamp.shape  : (1200, 1200)
```

In [65]:
```
# display singular values

fig2 = plt.figure(2, figsize=[9, 4])
ax_f21 = fig2.add_subplot(1, 1, 1)
ax_f21.grid(True)
# plot of image

ax_f21.semilogy(S_lamp)
ax_f21.set_xlabel('index')
ax_f21.set_title('singular values');
```



In [57]:
```
# compute approximations to the original image using truncated SVD

k = 4
Umat_lamp_k4 = np.zeros(img1.shape)
for m in range(k):
    Umat_lamp_k4 = Umat_lamp_k4 + S_lamp[m] * np.outer(Umat_lamp[:, m], Vtmat_lamp[

k = 20
Umat_lamp_k20 = np.zeros(img1.shape)
for m in range(k):
    Umat_lamp_k20 = Umat_lamp_k20 + S_lamp[m] * np.outer(Umat_lamp[:, m], Vtmat_lam

k = 50
Umat_lamp_k50 = np.zeros(img1.shape)
for m in range(k):
    Umat_lamp_k50 = Umat_lamp_k50 + S_lamp[m] * np.outer(Umat_lamp[:, m], Vtmat_lam

k = 80
Umat_lamp_k80 = np.zeros(img1.shape)
```

```
for m in range(k):
    Umat_lamp_k80 = Umat_lamp_k80 + S_lamp[m] * np.outer(Umat_lamp[:, m], Vtmat_lam
```

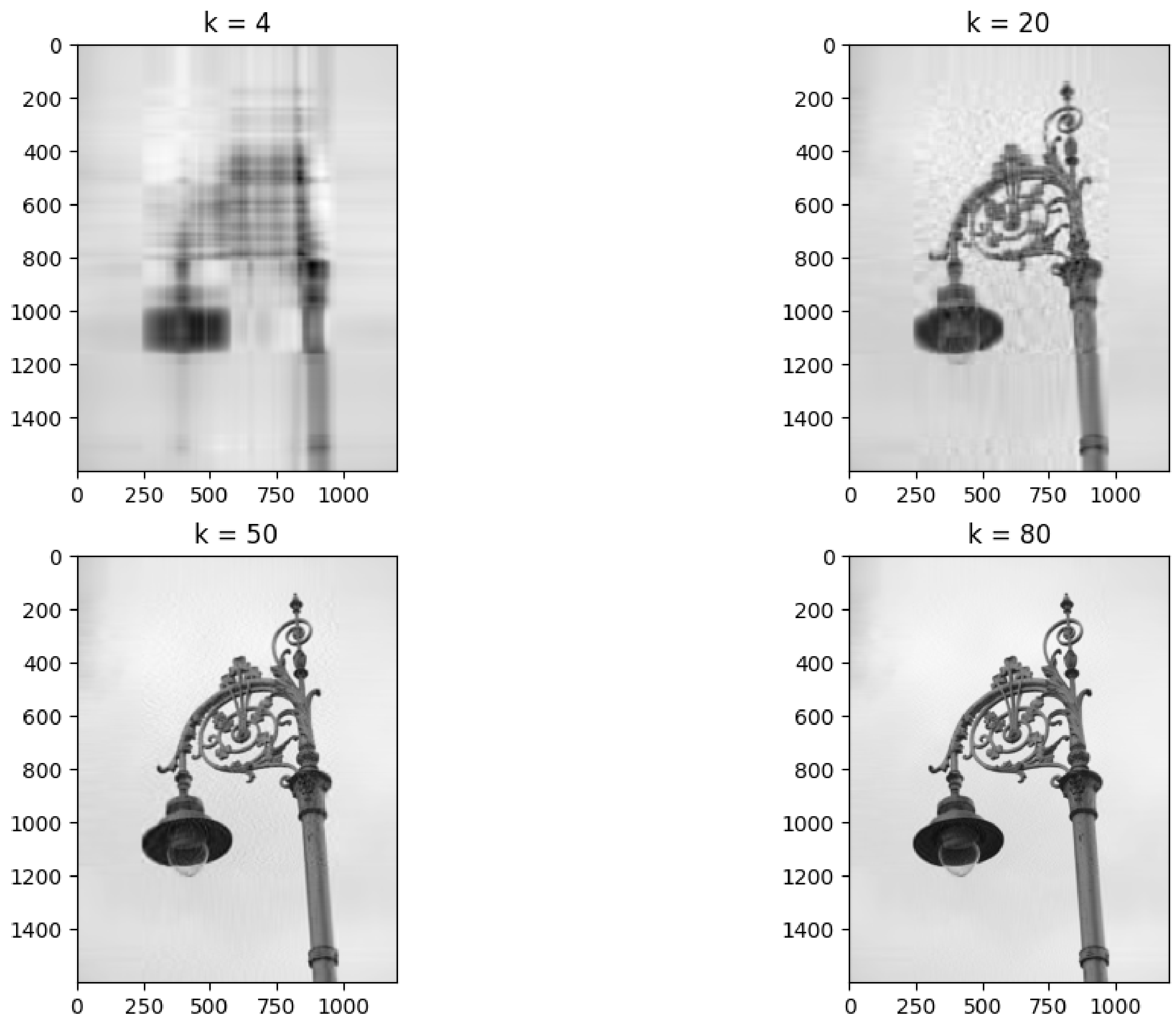
In [64]:
```
# display approximated images

fig3 = plt.figure(3, figsize=[12, 8])
ax_f31 = fig3.add_subplot(2, 2, 1)
# plot of image
ax_f31.imshow(Umat_lamp_k4, cmap='Greys_r' )
ax_f31.set_title('k = 4')

ax_f32 = fig3.add_subplot(2, 2, 2)
# plot of image
ax_f32.imshow(Umat_lamp_k20, cmap='Greys_r' )
ax_f32.set_title('k = 20')

ax_f33 = fig3.add_subplot(2, 2, 3)
# plot of image
ax_f33.imshow(Umat_lamp_k50, cmap='Greys_r' )
ax_f33.set_title('k = 50')

ax_f34 = fig3.add_subplot(2, 2, 4)
# plot of image
ax_f34.imshow(Umat_lamp_k80, cmap='Greys_r' )
ax_f34.set_title('k = 80');
```


k = 4
k = 20
k = 50
k = 80
0
200
400
600
800
1000
1200
1400
0
250
500
750
1000

```
In [ ]:
```

```
In [ ]:
```