奇异值分解

从一个例子讲起

对矩阵A= $\begin{bmatrix} 1 & 1 \\ 2 & 2 \\ 0 & 0 \end{bmatrix}$ 做奇异值分解（《统计学习方法 第2版》p283）

In [1]:
```
import numpy as np
np.set_printoptions(precision=3, suppress=True)
A=np.array([[1,1],[2,2],[0,0]])
A
```

Out[1]:
```
array([[1, 1],
       [2, 2],
       [0, 0]])
```

第一步：求A^TA的特征值和特征向量,并将特征值从大到小排序

对于A^TA(大小2x2)，由正规矩阵分解定理：

存在标准正交矩阵V，使得$V^T(A^TA)V = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}, \lambda_1 \geq 0, \lambda_2 \geq 0$ (正规矩阵半正定)

接下来就是要求:$eigenvalue = [\lambda_1, \lambda_2], eigenvector = V = [v_1, v_2]$

In [2]:
```
# 求A转置和A的乘积
Gram_A=np.matmul(A.T,A)
Gram_A
```

Out[2]:
```
array([[5, 5],
       [5, 5]])
```

In [3]:
```
# 求特征值和特征向量
eigenvalue,eigenvector=np.linalg.eig(Gram_A)
eigenvalue=np.where(eigenvalue>1e-8,eigenvalue,0)
eigenvalue,eigenvector
```

Out[3]:
```
(array([10.,  0.]),
 array([[ 0.707, -0.707],
        [ 0.707,  0.707]]))
```

In [4]:
```
# np.argsort()简单介绍
# 获得数组从最小到最大元素的下标，比如输入array([3,1,2])，那么输出的就是1，2，3在原数组中的下标[1,2,0]
np.argsort(np.array([3,1,2])),np.argsort(-1*np.array([3,1,2]))
```

Out[4]:
```
(array([1, 2, 0], dtype=int64), array([0, 2, 1], dtype=int64))
```

In [5]:
```
index=np.argsort(-1*eigenvalue)
eigenvalue=eigenvalue[index]
eigenvector=eigenvector[:,index]
eigenvalue,eigenvector
```

Out[5]:
```
(array([10.,  0.]),
 array([[ 0.707, -0.707],
        [ 0.707,  0.707]]))
```

第二步：求矩阵U，强行凑出$AV = U\Sigma$

形状：$A: mxn \quad V: nxn \quad U: mxm \quad \Sigma: mxn$

$V^T(A^TA)V = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \Rightarrow (AV)^TAV = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$,其中A是3x2的，V是2x2的，AV是3x2的

不妨设$AV = [b_1, b_2], \Rightarrow (AV)^TAV = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} = \begin{bmatrix} b_1^Tb_1 & 0 \\ 0 & b_2^Tb_2 \end{bmatrix} = \begin{bmatrix} 0.707 & 0 \\ 0 & 0 \end{bmatrix}$

所以推出：$AV = [b_1, b_2] = [b_1, 0]$,且有$b_1^Tb_1 = \lambda_1$(即b_1模长的平方)

如果$V = [v_1, v_2]$,那么$Av_1 = b_1, Av_2 = 0$

接着我们对等式稍微作个变形

$$AV = [Av_1, Av_2] = [b_1, 0] = \begin{bmatrix} \frac{b_1}{\sqrt{\lambda_1}} & u_2 & u_3 \end{bmatrix} \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix} \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

其中: $u_1 = \frac{b_1}{\sqrt{\lambda_1}}$,$u_2, u_3$没有任何要求，随便都能使上述等式成立，由于我们要强行凑$AV = U\Sigma$这个等式，并且要求U为标准正交矩阵，所以我们令u_1, u_2, u_3是R^{3x3}中的一个标准正交基

然后停下来思考一下上面的过程，是不是从最开始的正规矩阵分解，最后抽出里面部分拿出来分析，不知不觉就凑出来这么一个等式，并且除了u_2, u_3，其他都已经求出来了

只看上面最后一个式子，我们知道，V就是A^TA的特征向量，u_1就是矩阵A乘以非零特征值对应的特征向量，λ_1就是A^TA的非零特征值，这些东西我们后面就直接求解，不用一步一步分析了

对于u_2,u_3,满足如下性质:
1.与u_1线性无关
2.与u_1垂直
3.由$kerA=kerA^TA=kerA^T$可得，$u_2\in kerA=kerA^T,u_3\in kerA=kerA^T$
4.$dim<u_1>\oplus dim<u_2,u_3>=m=3$ 综上可知，$<u_2,u_3>=kerA^T$

小结:
$V=[v_1,v_2]$
$u_1=Av_1/\sqrt{\lambda_1}$
$<u_2,u_3>=KerA^T$
$\Sigma=\begin{bmatrix}\sqrt{\lambda_1} & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}$

In [6]:
```
# 求V
V=eigenvector
V
```

Out[6]:
```
array([[ 0.707, -0.707],
       [ 0.707,  0.707]])
```

In [7]:
```
# 求u1
mask=eigenvalue>0
rank=np.sum(mask)
rank,mask,eigenvalue[mask],V[:,mask]
```

Out[7]:
```
(1,
 array([ True, False]),
 array([10.]),
 array([[0.707],
        [0.707]]))
```

In [8]:
```
u_1=np.matmul(A,V[:,mask])/np.sqrt(eigenvalue[mask])
u_1
```

Out[8]:
```
array([[0.447],
       [0.894],
       [0.   ]])
```

In [9]:
```
# 求sigma
sigma=np.zeros((3,2))
sigma[0:rank,0:rank]=np.diag(np.sqrt(eigenvalue[mask]))
sigma
```

Out[9]:
```
array([[3.162, 0.   ],
       [0.   , 0.   ],
       [0.   , 0.   ]])
```

In [10]:
```
# 求u_2,u_3
print("很难求,用solve,lstsq,pinv通通不行,对于齐次方程,求的都是最小二乘解.由于对于SVD来说这个东西不需要求解,这里就不求了.")
```

```
很难求,用solve,lstsq,pinv通通不行,对于齐次方程,求的都是最小二乘解.由于对于SVD来说这个东西不需要求解,这里就不求了.
```

第三步：无损重建

$$AV=A[v_1,v_2]=[u_1\quad u_2\quad u_3]\begin{bmatrix}\sqrt{\lambda_1} & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}\quad\Rightarrow\quad A=[u_1\quad u_2\quad u_3]\begin{bmatrix}\sqrt{\lambda_1} & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}V^T=[u_1\quad u_2\quad u_3]\begin{bmatrix}\sqrt{\lambda_1} & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}v_1^T\\ v_2^T\end{bmatrix}=\sqrt{\lambda_1}u_1v_1^T$$

因此，求矩阵A，只需要向量u_1,v_1和非零奇异值，其他东西不需要，上面这个东西类似矩阵的谱分解，对于一般的mxn矩阵A，设$rankA=r$,那么有以下矩阵奇异值分解

$$A=U\Sigma V^T=[u_1\quad u_2\quad\cdots\quad u_r\quad\cdots\quad u_m]\begin{bmatrix}\sigma_1 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0\\ 0 & \sigma_2 & 0 & \cdots & 0 & 0 & \cdots & 0\\ 0 & 0 & \sigma_3 & \cdots & 0 & 0 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \ddots\\ 0 & 0 & 0 & \cdots & \sigma_r & 0 & \cdots & 0\\ 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0\\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\ 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0\end{bmatrix}\begin{bmatrix}v_1^T\\ v_2^T\\ v_3^T\\ \vdots\\ v_r^T\\ v_{r+1}^T\\ \vdots\\ v_n^T\end{bmatrix}=\sigma_1u_1v_1^T+\sigma_2u_2v_2^T+\cdots+\sigma_ru_rv_r^T$$

其中,$\sigma_i=\sqrt{\lambda_i},u_i=Av_i/\sigma_i,v_i$是$A^TA$对应特征值$\lambda_i$的特征向量,$1\le i\le r$.

数据压缩(秩k重建)

令$\tilde{A}=\sigma_1 u_1 v_1^T+\sigma_2 u_2 v_2^T+\cdots+\sigma_k u_k v_k^T, 1\le k\le r$. 说明:
(1).保存矩阵$\tilde{A}$只需要k个奇异值，k个mx1向量，k个nx1向量，总共需要k（m+n+1）个数，而保存A需要mxn个数，
假如m=n=1000，k=100，$\tilde{A}$需要100x（1000+1000+1）=200100个数，A需要1000x1000=1000000个数，节省了80%的存储空间
(2).把矩阵A看成向量，考虑向量的2范数，也就是矩阵的F范数，那么秩k重建的矩阵与原来的矩阵的F范数距离为：$||A-\tilde{A}||_F=(\sigma_{k+1}^2+\sigma_{k+2}^2+\cdots+\sigma_r^2)^{\frac{1}{2}}$
(3).根据上面，可以取前k个奇异值，使其平方和占全部的95%，从而保证信息丢失的不多

把上述奇异值分解算法做成一个接口函数svd（）

输入：一个矩阵A,秩k
输出：U,Σ,V
说明：这里的$U=[u_1,u_2,\ldots,u_k], V=[v_1,v_2,\ldots,v_k]^T$

In [11]:
```
def SVD(matrixA, k=None):

    #求A的Gram矩阵
    Gram_A = np.dot(matrixA.T, matrixA)

    # 求特征值和特征向量
    eigenvalue,eigenvector=np.linalg.eig(Gram_A)
    eigenvalue=np.where(eigenvalue>1e-9,eigenvalue,0)

    # 特征值按从大到小排序
    index=np.argsort(-1*eigenvalue)
    eigenvalue=eigenvalue[index]
    eigenvector=eigenvector[:,index]

    # 求A的秩r
    mask=eigenvalue>0
    rank=np.sum(mask)

    # 确定保留多少奇异值
    if k==None:k=rank #如果没有指定k，就按秩r来算

    # 求V,形状kxn
    V=eigenvector.T[:k,:]

    # 求U,形状mxk
    V_1=eigenvector[:,:k]
    U=matrixA.dot(V_1)/np.sqrt(eigenvalue[:k])

    #求Sigma,形状kxk
    Sigma=np.diag(np.sqrt(eigenvalue[:k]))

    return U, Sigma, V
```

案例：图像压缩

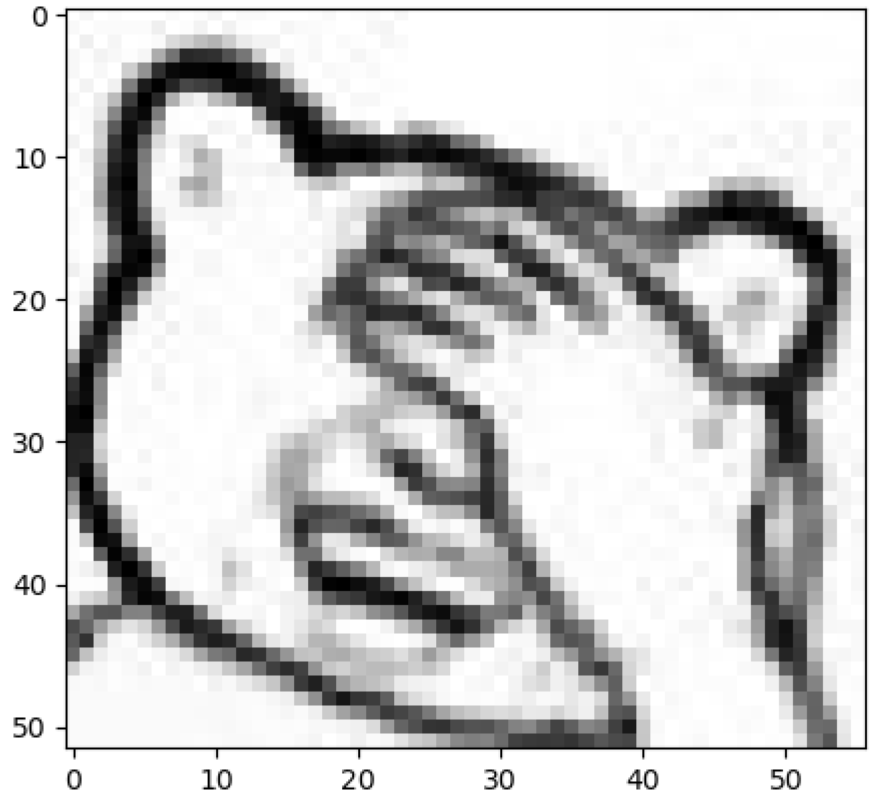
In [12]:
```
from PIL import Image
import matplotlib.pyplot as plt
```

In [13]:
```
im=np.array(Image.open("./data/svd.jpg").convert("L"))
im.shape
```

Out[13]:
```
(52, 56)
```

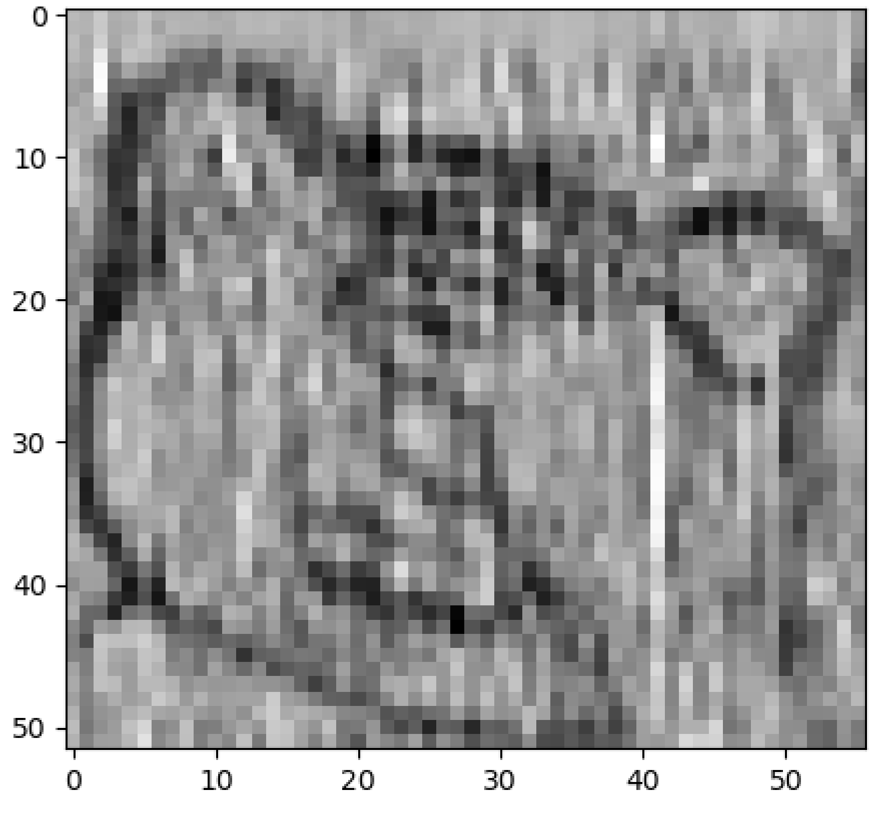
In [14]:
```
plt.figure()
plt.imshow(im,cmap="gray")
```

Out[14]:
```
<matplotlib.image.AxesImage at 0x19124790d60>
```



In [15]:
```
a,b,c=SVD(im)
im_2=a.dot(b).dot(c)
plt.imshow(im_2,cmap="gray")
print("这里无损重建，结果却有损，是因为这里numpy求解特征值特征向量的时候，数值有点问题，后面换torch的库就没问题了")
```

这里无损重建，结果却有损，是因为这里numpy求解特征值特征向量的时候，数值有点问题，后面换torch的库就没问题了



In [18]:
```
import torch
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

def SVD(matrixA, k=None):
    # 求A的Gram矩阵
    Gram_A = torch.matmul(matrixA.T, matrixA)

    # 求特征值和特征向量
    eigenvalue, eigenvector = torch.linalg.eigh(Gram_A)
    eigenvalue = torch.where(eigenvalue > 1e-9, eigenvalue, 0)

    # 特征值按从大到小排序
    index = torch.argsort(-1 * eigenvalue)
    eigenvalue = eigenvalue[index]
    eigenvector = eigenvector[:, index]

    # 求A的秩r
    mask = eigenvalue > 0
    rank = torch.sum(mask)

    # 确定保留多少奇异值
    if k is None: k = rank  # 如果没有指定k，就按秩r来算

    # 求V，形状kxn
    V = eigenvector.T[:k, :]

    # 求U，形状mxk
    V_1 = eigenvector[:, :k]
    U = torch.matmul(matrixA,V_1) / torch.sqrt(eigenvalue[:k])

    # 求Sigma，形状kxk
    Sigma = torch.diag(torch.sqrt(eigenvalue[:k]))

    # 返回三个矩阵
```

```
    return U, Sigma, V

# 测试

im=np.array(Image.open("./data/svd.jpg").convert("L"))
im=torch.tensor(im,dtype=torch.float32)
plt.figure()
plt.imshow(im.numpy(),cmap="gray")

fig,axes=plt.subplots(2,10)
fig.set_size_inches(20, 5)
for i in range(2):
    for j in range(10):
        a,b,c=SVD(im,k=i*2+j)
        im_rebuild=torch.matmul(a,b)
        im_rebuild=torch.matmul(im_rebuild,c)
        axes[i][j].imshow(im_rebuild.numpy(),cmap="gray")
```