SVD

December 24, 2016

1 The SVD decomposition

Àlex Escolà Nixon

1.0.1 1. Eigenvalue decomposition

1. Code a simple algorithm to compute SVD decomposition of a matrix A using the eigenvalues/eigenvectors of AtA and AAt.

```
In [453]: import numpy as np
          from numpy.linalg import svd,matrix_rank,cond,pinv
          from scipy.linalg import eig,norm
```

```
In [429]: m=4
          n=2
```

```
In [430]: A=np.array([[2,1,0,0],[4,3,0,0]]).T
          AAt=A.dot(A.T)
          AtA=A.T.dot(A)
          A
```

```
Out[430]: array([[2, 4],
                 [1, 3],
                 [0, 0],
                 [0, 0]])
```

```
In [431]: vaps_AAt,veps_AAt=eig(AAt,right=True)
          vaps_AtA,veps_AtA=eig(AtA,right=True)
```

```
In [432]: vaps_AAt=np.real(vaps_AAt)
          veps_AAt=np.real(veps_AAt)
          vaps_AtA=np.real(vaps_AtA)
          veps_AtA=np.real(veps_AtA)
```

```
In [433]: L_AAt=np.argsort(vaps_AAt)[::-1]
          L_AtA=np.argsort(vaps_AtA)[::-1]
```

In [434]:
```
S=np.zeros((m,n))
for i in range(0,n):
    S[i,i]=np.sqrt(vaps_AAt[i])

U=np.zeros((m,m))
for i in range(0,m):
    U[i]=veps_AAt[L_AAt[i]]

V=np.zeros((n,n))
for i in range(0,n):
    V.T[i]=veps_AtA[L_AtA[i]]
```

In [435]:
```
V[:,1]=-V[:,1]
print U.dot(S.dot(V))
```

```
[[ 2.  4.]
 [ 1.  3.]
 [ 0.  0.]
 [ 0.  0.]]
```

2. Use the scipy.linalg.svd function to get the SVD decomposition of A.

In [436]:
```
U,S_ar,V = svd(A)
print U
print S_ar
print V
```

```
[[-0.81741556 -0.57604844  0.          0.        ]
 [-0.57604844  0.81741556  0.          0.        ]
 [ 0.          0.          1.          0.        ]
 [ 0.          0.          0.          1.        ]]
[ 5.4649857   0.36596619]
[[-0.40455358 -0.9145143 ]
 [-0.9145143   0.40455358]]
```

3. Write a program that uses SVD decomposition to compute

In [437]:
```
A=np.array([[2,1],[4,3]]).T
```

In [438]:
```
U,S_ar,V = svd(A)
```

In [439]:
```
S=np.diag(S_ar)
```

In [440]:
```
S
```

Out[440]:
```
array([[ 5.4649857 ,  0.        ],
       [ 0.        ,  0.36596619]])
```

(a) the rank(A) We know from theory that, supposing the following for the eigenvalues of a matrix A:

$$\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0 \tag{1}$$

Then the rank of A is r. In our case A is full rank:

```
In [445]: rank = len(np.diag(S[S>1e-6]))
          print rank
          print matrix_rank(A)
```

```
2
2
```

(b) the 2-norm of A

```
In [446]: Norm2 = np.diag(S).max()
          print Norm2, norm(S,2)
```

```
5.46498570422 5.46498570422
```

(c) the Frobenius norm of A For any matrix A, the sum of squares of the singular values equals the Frobernius norm.

```
In [447]: Frob_norm = np.sqrt(sum(np.diag(S**2)))
          print Frob_norm, norm(A,'fro')
```

```
5.47722557505 5.47722557505
```

(d) the condition number k2(A) The ratio C of the largest to smallest singular value in the singular value decomposition of a matrix

```
In [451]: Cond_number = S.max()/S[S!=0].min()
          print Cond_number, cond(A)
```

```
14.9330343737 14.9330343737
```

(e) the pseudoinverse A+ of A

```
In [454]: pseudo_inv=V.dot(np.diag(1./np.diag(S)).T).dot(U.T)
          print pseudo_inv
          print
          print pinv(A)
```

```
[[ 1.5 -2. ]
 [-0.5  1. ]]

[[ 1.5 -2. ]
 [-0.5  1. ]]
```

1.0.2 Use of SVD to solve Least Square problems

```
In [455]: A = np.array([[2,3,5,1],[6,3,1,4],[8,2,1,9],[5,9,11,3]])
          A
```

```
Out[455]: array([[ 2,  3,  5,  1],
                 [ 6,  3,  1,  4],
                 [ 8,  2,  1,  9],
                 [ 5,  9, 11,  3]])
```

```
In [456]: b=np.array([4,5,2,3],ndmin=2).T
          b.shape
```

```
Out[456]: (4, 1)
```

```
In [457]: U,S_ar,V = svd(A)
          S=np.diag(S_ar)
```

The pseudo inverse is calculated using the previus method as:

```
In [458]: pseudo_inv=V.dot(np.diag(1./np.diag(S)).T).dot(U.T)
```

And the solution to the least squares problem by means of using the SVD decomposition is as follows:

```
In [459]: x = pseudo_inv.dot(b)
          print x
          print norm(x,2)
```

```
[[ 1.65484989]
 [ 3.76630751]
 [ 2.10160849]
 [-2.23860076]]
5.13338991527
```

1.0.3 Computing the SVD decomposition

```
In [460]: A = np.array([[2,3,5,3],[3,7,2,1],[5,2,4,9],[7,2,2,9]])
          A
```

```
Out[460]: array([[2, 3, 5, 3],
                 [3, 7, 2, 1],
                 [5, 2, 4, 9],
                 [7, 2, 2, 9]])
```

```
In [461]: def create_h(A):
              zeros1=np.zeros((len(A.T),len(A.T)))
              zeros2=np.zeros((len(A),len(A)))
              H1=np.concatenate((zeros1,A.T),axis=1)
              H2=np.concatenate((A,zeros2.T),axis=1)
              return np.concatenate((H1,H2),axis=0)
```

```
In [462]: H = create_h(A)
```

1. Write a routine to check that:

(a) the eigenvalues of H are si, i = 1 . . . n, where si, i = 1. . . n are the singular values of A

```
In [463]: Ua,Sa,Va = svd(A)
          Uh,Sh,Vh = svd(H)


          HHt=H.dot(H.T)
          vaps_HHt,veps_HHt=eig(HHt,right=True)
          vaps_HHt=np.real(vaps_HHt)

          print 'Singular values of A:'
          print Sa**2
          print
          print 'Eigenvalues of H:'
          print sorted(vaps_HHt,reverse=True)[0::2]
```

```
Singular values of A:
[ 312.89730709   47.64292149   12.92845516    0.53131627]

Eigenvalues of H:
[312.89730708691718, 47.642921489508865, 12.928455156133376, 0.53131626744104332]
```

It has been checked through the code above, that indeed the eigenvalues of H are the sigular values of A (the singular values must be squared)

```
In [464]: np.matrix.round(veps_HHt,2)

Out[464]: array([[ 0.52,  0.73,  0.44,  0.04,  0.  ,  0.  ,  0.  ,  0.  ],
                 [ 0.31, -0.34,  0.27, -0.84,  0.  ,  0.  ,  0.  ,  0.  ],
                 [ 0.34,  0.29, -0.86, -0.26,  0.  ,  0.  ,  0.  ,  0.  ],
                 [ 0.72, -0.52, -0.03,  0.47,  0.  ,  0.  ,  0.  ,  0.  ],
                 [ 0.  ,  0.  ,  0.  ,  0.  ,  0.33,  0.34,  0.75,  0.46],
                 [ 0.  ,  0.  ,  0.  ,  0.  ,  0.29,  0.85, -0.4 , -0.18],
                 [ 0.  ,  0.  ,  0.  ,  0.  ,  0.62, -0.24,  0.26, -0.69],
                 [ 0.  ,  0.  ,  0.  ,  0.  ,  0.64, -0.33, -0.46,  0.52]])

In [465]: np.matrix.round(Uh,2)

Out[465]: array([[ 0.52,  0.  ,  0.01, -0.04, -0.08,  0.44, -0.73,  0.01],
                 [ 0.31,  0.  , -0.19,  0.82, -0.05,  0.26,  0.34, -0.  ],
                 [ 0.34,  0.  , -0.06,  0.25,  0.15, -0.84, -0.29,  0.  ],
                 [ 0.72,  0.  ,  0.11, -0.45,  0.01, -0.03,  0.51, -0.01],
                 [-0.  , -0.33,  0.33,  0.08, -0.74, -0.13, -0.01, -0.46],
                 [ 0.  , -0.29,  0.83,  0.19,  0.4 ,  0.07,  0.  ,  0.18],
                 [ 0.  , -0.62, -0.23, -0.05, -0.26, -0.05,  0.01,  0.69],
                 [ 0.  , -0.64, -0.32, -0.07,  0.45,  0.08, -0.01, -0.52]])
```

```
In [466]: np.matrix.round(np.concatenate((-Va.T,-Ua),axis=0),2)
```

```
Out[466]: array([[ 0.52,  0.04, -0.44, -0.73],
                 [ 0.31, -0.84, -0.27,  0.34],
                 [ 0.34, -0.26,  0.86, -0.29],
                 [ 0.72,  0.47,  0.03,  0.52],
                 [ 0.33, -0.34,  0.75, -0.46],
                 [ 0.29, -0.85, -0.4 ,  0.18],
                 [ 0.62,  0.24,  0.26,  0.69],
                 [ 0.64,  0.33, -0.46, -0.52]])
```

Ex.2 House(x) function:

```
In [467]: def house(x):
              n=x.shape[0]
              s=np.dot(x[1:n],x[1:n].T)
              v=np.zeros(n)
              v[0]=1
              for i in range (1,n):
                  v[i]=x[i]
              if(s<1.e-14):
                  bet=0
              else:
                  mu=np.sqrt(x[0]*x[0]+s)
                  if(x[0]<=0):
                      v[0]=x[0]-mu
                  else:
                      v[0]=-s/(x[0]+mu);
                  bet=2*v[0]*v[0]/(s+v[0]*v[0])
                  v=v/v[0]
              return v,bet
```

Ex.3 Write functions PA(bet,v,A) and AP(bet,v,A) that perform the previous updating computations

```
In [468]: def PA(bet,v,A):
              v=v.reshape(v.shape[0],-1)
              return A-np.dot(v,bet*np.dot(A.T,v).T)
```

```
In [469]: def AP(bet,v,A):
              v=v.reshape(v.shape[0],-1)
              return A-np.dot(bet*np.dot(A,v),v.T)
```

Ex.4 Write a function bidiag(A) that performs the bidiagonalization of A by applying Householder transformations

```
In [470]: def bidiag(A):
              m=A.shape[0]
              n=A.shape[1]
```

```
        for i in range(n):
            x=A[i:,i]
            v,bet=house(x)
            if i==0:
                A=PA(bet,v,A)
            else:
                A[i:,i:]=PA(bet,v,A[i:,i:])
            if i!=n-1:
                x=A[i,i+1:]
                v,bet=house(x)
                A[i:,i+1:]=AP(bet,v,A[i:,i+1:])
        return A
```

```
In [471]: print np.matrix.round(bidiag(H),2)
```

```
[[  9.33  14.7   -0.    -0.     0.     0.     0.     0.  ]
 [  0.     3.65   2.96   0.     0.     0.     0.     0.  ]
 [  0.     0.     2.09  -4.93  -0.    -0.    -0.    -0.  ]
 [  0.     0.     0.     4.49  -0.    -0.    -0.    -0.  ]
 [ -0.     0.    -0.     0.     6.86  13.7    0.     0.  ]
 [ -0.     0.    -0.    -0.    -0.     9.31   4.11   0.  ]
 [ -0.     0.     0.    -0.     0.     0.     4.83  -3.37]
 [ -0.     0.    -0.     0.    -0.     0.    -0.    -1.04]]
```

le (just the dimension m+n and the two arrays containing the bidiagonal of H).

```
In [472]: A = np.array([[2,3,5,1,5],[6,3,1,4,3],[8,2,1,9,1],[5,9,11,3,9]])
```

Functions to create H and bidiagonalize it encapsulated in one:

```
In [473]: def bidiagonalize(A):
              H=create_h(A)
              return H,bidiag(H)
```

```
In [474]: H,B = bidiagonalize(A)
          print np.matrix.round(B,1)
```

```
[[ 11.4  15.9   0.    0.    0.    0.    0.    0.    0. ]
 [  0.    9.9  10.6   0.   -0.    0.    0.    0.    0. ]
 [  0.    0.    2.5   1.7   0.    0.    0.    0.    0. ]
 [  0.    0.    0.    0.    1.2  -0.   -0.   -0.   -0. ]
 [  0.    0.    0.    0.    0.   10.8   0.    0.    0. ]
 [ -0.    0.   -0.   -0.    0.   17.9   6.4   0.    0. ]
 [ -0.   -0.   -0.    0.    0.    0.    9.9   4.8   0. ]
 [ -0.   -0.    0.   -0.    0.    0.    0.    1.2   1.2]
 [ -0.    0.    0.    0.    0.    0.    0.    0.    1.1]]
```

```
In [475]: H

Out[475]: array([[  0.,    0.,    0.,    0.,    0.,    2.,    6.,    8.,    5.],
                 [  0.,    0.,    0.,    0.,    0.,    3.,    3.,    2.,    9.],
                 [  0.,    0.,    0.,    0.,    0.,    5.,    1.,    1.,   11.],
                 [  0.,    0.,    0.,    0.,    0.,    1.,    4.,    9.,    3.],
                 [  0.,    0.,    0.,    0.,    0.,    5.,    3.,    1.,    9.],
                 [  2.,    3.,    5.,    1.,    5.,    0.,    0.,    0.,    0.],
                 [  6.,    3.,    1.,    4.,    3.,    0.,    0.,    0.,    0.],
                 [  8.,    2.,    1.,    9.,    1.,    0.,    0.,    0.,    0.],
                 [  5.,    9.,   11.,    3.,    9.,    0.,    0.,    0.,    0.]])
```

Ex.6 Write a program that implements the qds algorithm

```
In [476]: q_aux=np.zeros(len(H))
          e_aux=np.zeros(len(H))
          a=np.diagonal(B,offset=0)
          b=np.diagonal(B,offset=1)
          q=a**2
          e=b**2

In [477]: def qds(H,q_aux,e_aux,q,e):
              while norm(e,np.inf)>1.e-14:
                  for j in range(len(A.T)-1):
                      q_aux[j] = q[j] + e[j] - e_aux[j-1]
                      e_aux[j] = e[j] * (q[j+1]/q_aux[j])
                  q_aux[len(A.T)] = q[len(A.T)] - e_aux[len(A.T)-1]
                  q=q_aux
                  e=e_aux
              return q_aux
```

Obtained values of the square of the singular values of B with the dqs algorithm:

```
In [478]: q_aux = qds(H,q_aux,e_aux,q,e)
          q_aux[0:len(A.T)]

Out[478]: array([ 476.5027563 ,  120.76523989,    4.17980061,    1.55220319,    0.
```

As it can be seen, the values indeed give a good approximation for the tolerance mentioned in the staement, of the squared of the singular values of B, which are computed directly from B bellow:

```
In [479]: Ua,Sa,Va = svd(H)
          print np.matrix.round(Sa**2)[::2]

[ 477.   121.    4.    2.    0.]
```