EE5471: Exploring SVD

Harishankar Ramachandran
EE Dept, IIT Madras

October 27, 2011

Reading Assignment

Chapter 4.4-4.7 and chapter 6 of Stoer.

- Pay special attention to the error analysis and why a Unitary transformation is numerically a good thing.
- In chapter 6, go through the proof that a Hermitian matrix has a complete set of eigenvectors.
- In chapter 6, the proof that the QR iteration process does converge to a diagonal matrix. This proof is quite long and confusing, which is why I gave a motivational proof in class.

Introduction

I have uploaded three python programs corresponding to the three problems here. They implement the code given in the text.

This assignment is not a programming assignment. It would take too long and you would not learn much about svd in the process. What you have to do is to tweak the various parameters and understand what svd is doing in each case. You also have to thoroughly understand the codes by viva time.

1 Reliable inversion of an illconditioned matrix

- Construct the Van der Monde matrix that arises in Taylor series estimation:

$$f(x_i) = \sum_{k=0}^{N} c_k \left(x_i - x_0\right)^k$$

Choose $x_0 = 0$ and $N = 6$. The function we wish to approximate is $f(x) = e^x + n$ at the points $x_i = j\Delta x$ where $\Delta x = 0.1$. Generate the matrix M consisting of 20 rows and 7 columns, with j going from 1 to 20.

1 ⟨*Q1matrix* 1⟩≡

```
def q1f(x,sig):
  return exp(x)+sig*randn(len(x),1)
x=arange(0.1,2.05,0.1).reshape((20,1))
b=q1f(x,0.001)
M=hstack([ones(shape(x)),x,x*x,x**3,x**4,x**5,x**6])
figure(0)
plot(x,M,x,b)
legend(["x^0","x^1","x^2","x^3","x^4",
  "x^5","x^6","b"])
title("Plot of the columns of M and of b")
```

- The least squares solution is given by

$$\vec{c} = \left(M^T M\right)^{-1} M^T \vec{f}$$

Construct $M^T M$ and find its eigen decomposition. Plot the spectrum.

2a ⟨*Q1LS* 2a⟩≡

```
A=dot(M.transpose(),M)
print eigh(A)[0]
figure(1)
contour(20*log10(A))
c=dot(inv(A),dot(M.transpose(),b))
figure(2)
semilogy(eigh(A)[0])
```

- Truncate the spectrum and keep only the eigenvalues that are greater than 10^{-6} of the largest eigenvalue, zeroing the rest. Use the truncated spectrum to construct the "pseudo inverse" of the matrix, and hence compute the coefficients.

2b ⟨*Q1inv* 2b⟩≡

```
def pinv(M,tol):
  A=dot(M.transpose(),M)
  s,v=eigh(A)
  smin=s.max()*tol
  ii=where(s<smin)
  sinv=1/s # assumes non singular
  sinv[ii]=0 # zeros the "bad singular values"
  return dot(v,dot(diag(sinv),v.transpose()))
```

- We know that the coefficients are given by

$$c_k = \frac{1}{k!}$$

Determine the error in the coefficients and the error in the reconstructed function for different noise and different truncation conditions.

3a ⟨*Q1coef* 3a⟩≡

```
c0=1/sp.gamma(arange(1,8)).reshape(7,1)
Tols=logspace(-12,-2,6)
err=zeros(6)
figure(3)
s=[]
for i in range(6):
  c=dot(pinv(M,Tols[i]),dot(M.transpose(),b))
  err[i]=(abs(c-c0)).max()
  plot(x,dot(M,c)/b-1)
  s.append("tol=%.1e" % Tols[i])
  print "Tol = %.4e, max Err=%5f" % (Tols[i],err[i])
  for k in range(len(c)):
    print "%5f\t%5f\t%5f" % (c[k],c0[k],c[k]-c0[k])
legend(s)
figure(4)
loglog(Tols,err,'b',Tols,err,'bo')
```

3b ⟨*Q1* 3b⟩≡

⟨*Q1matrix* 1⟩
⟨*Q1LS* 2a⟩
⟨*Q1inv* 2b⟩
⟨*Q1coef* 3a⟩

2 Estimate a sinusoid buried in noise

To estimate a sinusoid buried in noise where the frequency is known is a linear estimation problem.

```
p=lstsq(M,b)
```

where `M` is the two column matrix consisting of $\cos \omega t_i$ and $\sin \omega t_i$ and `b` is the vector of noisy observations. The resulting estimate is

$$f(t) = p_0 \cos \omega t + p_1 \sin \omega t$$

This is all well and good. However, what if it is the frequency that is not known?

We are going to tackle this problem next in the signal processing section, using model based estimation. However, here we try to solve the problem using SVD.

Let us first construct the data. The following function creates a sequence consisting of a chirped sinusoid plus noise.

4a ⟨*datafn* 4a⟩≡
```
def dataseq(sig,w1,w2,N,k):
  t=(arange(N+0.0)/N)*k
  y=cos(2*pi*(w1*t+w2*t*t+0.1))+sig*randn(N)
  return(y)
```

We now generate the matrix corresponding to shifting the sequence, creating a Toeplitz type of matrix. We can either rotate the samples or time shift them using older data. Here we have rotated the data. Depending on which block you include you can use either method.

4b ⟨*datamatrixNP* 4b⟩≡
```
def datamatrix(sig,w1,w2,N,k):
  y=dataseq(sig,w1,w2,2*N,2*k)
  A=zeros((N,N))
  for i in range(N):
    A[i,:]=y[i:i+N]
  return(A)
```

4c ⟨*datamatrixP* 4c⟩≡
```
def datamatrix(sig,w1,w2,N,k):
  y=dataseq(sig,w1,w2,N,k)
  A=zeros((N,N))
  for i in range(N):
    A[i,:]=y
    t=y[0];y[0:-1]=y[1:];y[-1]=t
  return(A)
```

Our parameters are

4d ⟨*Q2params* 4d⟩≡
```
sig=0.1   # noise level (signal is unit amplitude)
w1=1      # frequency at t=0
w2=0.05    # frequency chirp, ie coef of t^{2}
N=100      # number of samples
k=2       # number of wavelengths to sample
n0=4      # number of eigenvalues to keep
```

We now apply SVD to this problem. First construct the matrix and decompose it via svd.

5a ⟨*Q2* 5a⟩≡
```
⟨datafn 4a⟩
⟨datamatrixNP 4b⟩
⟨Q2params 4d⟩
A=datamatrix(sig,w1,w2,N,k)
u,s,v=svd(A)
```

Plot the spectrum. This spectrum will depend on the amount of noise added, since noise has a flat spectrum.

5b ⟨*Q2* 5a⟩+≡
```
figure(0)
semilogy(s,'ro')
title("svd spectrum")
```

Reconstruct after zeroing the noise eigenvalues. This is not the same as the pseudoinverse. Here we are creating a cleaned up matrix A itself. That is what the line

```
B=dot(u,dot(diag(s),v))
```

does.

5c ⟨*Q2* 5a⟩+≡
```
figure(1)
s[n0:]=0 # zero the noise eigenvalues
B=dot(u,dot(diag(s),v))
figure(1)
plot(B[0,:],'ko',A[0,:],'r+',dataseq(0,w1,w2,N,k),'g')
title("Original signal and reconstruction")
legend(["svd reconstruction","data","signal"])
```

Here we plot the svd eigenvectors that have been retained.

5d ⟨*Q2* 5a⟩+≡
```
figure(2)
str=[]
for i in range(n0):
  plot(v[i,:])
  str.append(r"$i=%d$" % i)
title("Profile of eigenvectors")
legend(str)
```

3 Image Compression

SVD is frequently used to compress images. I have uploaded a landscape image to analyse and compress. Here is the python code.

First we open the jpg file using the tools of the Image module.

5e ⟨*Q3* 5e⟩≡
```
from scipy import mat, dot
from PIL import Image
im=Image.open("test.jpg")
```

The image needs to be converted into a form that we can apply numerical algorithms. We use the `asarray` function to do this. Further, we save back the file as jpg to have a reference size to compare our compressions.

6a ⟨*Q3* 5e⟩+≡
```
A = asarray(im)
misc.imsave("test1.jpg", A)
```

We set up for plotting and we create a 3-D matrix to hold the compressed image.

6b ⟨*Q3* 5e⟩+≡
```
print A.shape
rows,cols,colors=A.shape
str=["red","green","blue"] # legends
figure(0)
d = 0.001
A1=zeros((rows,cols,3))
```

We perform svd on each of the three colour planes. Note that the standard svd creates a square matrix U. But only the first n columns of U are useful as the rest of the columns correspond to zero eigenvalues. Setting `full_matrices=0` means that the routine returns an $m \times n$ matrix for U, which is what we want. Then,

$$A = U \cdot \mathrm{diag}(s) \cdot V$$

In the following code, eigenvalues that are less than a fraction d less than the maximum eigenvalue are all set to zero and the matrix reconstructed.

6c ⟨*Q3* 5e⟩+≡
```
for i in range(3):
  matrix=A[:,:,i]
  U,s,V = svd(matrix,full_matrices=0)
  semilogy(s,'ro')
  l = s.max()
  ii=where(s<d*l);ii=ii[0]
  s[ii]=0
  print len(ii)/rows.__float__()
  A1[:,:,i]=dot(dot(U,diag(s)),V)
```

Just because we have reduced the information in the matrices does not mean that the jpg file will be better compressed. The jpg algorithm depends on the DCT and it so turns out that the compression algorithm is not able to compress the file well.

6d ⟨*Q3* 5e⟩+≡
```
misc.imsave("test2.jpg", A1)
show()
```

In practice, an svd compression algorithm only keeps the columns of U and V that correspond to the non-zero eigenvalues. Thus the compression is directly given by the numbers printed out by the program and not just an indication. These vectors, of course, are themselves compressed to get the best compression.

- Get this program working
- Vary d, the threshold value to zero small eigenvalues
- Obtain the compression fraction as a function of d and interpret it from the spectrum of the eigenvalues
- Understand what d value recovers the details of the photo and therefore captures the image fully.

7 ⟨* 7⟩≡

```
from scipy import *
from matplotlib.pyplot import *
from scipy.linalg import *
import scipy.special as sp
⟨Q1 3b⟩
show()
```