W1

March 27, 2020

Worksheet 1 Introduction to linear algebra by Prof J.Morlier February 2020

#import Base.print_matrix #import Pkg; Pkg.add("SymPy")

[157]:
```
using LinearAlgebra, SparseArrays, SuiteSparse, Random
using SymPy
import Base.print_matrix
```

[156]:
```
versioninfo()
```

```
Julia Version 1.3.1
Commit 2d5741174c (2019-12-30 21:36 UTC)
Platform Info:
  OS: macOS (x86_64-apple-darwin18.6.0)
  CPU: Intel(R) Core(TM) i5-8259U CPU @ 2.30GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-6.0.1 (ORCJIT, skylake)
```

A 2x2 Recap

[83]:
```
entries = @syms a b c d  real=true
```

[83]: (a, b, c, d)

[84]:
```
M = [a b; c d]
```

[84]:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

[85]:
```
dM= det(M)
```

[85]:

$$ad - bc$$

```
How can I invert a 3x3 matrix ?
```

[86]:
```
entries = @syms a b c d e f g h i real=true
```

[86]: (a, b, c, d, e, f, g, h, i)

[87]:
```
A = reshape([entries...],(3,3))
```

[87]:

$$\begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

[88]:
```
detA = det(A) |> simplify
```

[88]:

$$aei - afh - bdi + bfg + cdh - ceg$$

on the SVD of matrix A of size 2 x 3

[89]:
```
A=[4 11 14; 8 7 -2]; show(A)
```

[4 11 14; 8 7 -2]

Construct $B = A^T A$

[90]:
```
B=transpose(A)*A; show(B)
```

[80 100 40; 100 170 140; 40 140 200]

#Check; # B=[80 100 40; 100 170 140; 40 140 200];

[91]:
```
lambda = symbols("lambda", real=true)
```

[91]:

$$\lambda$$

[92]:
```
C=B-lambda*[1 0 0; 0 1 0; 0 0 1]
```

[92]:

$$\begin{bmatrix} 80-\lambda & 100 & 40 \\ 100 & 170-\lambda & 140 \\ 40 & 140 & 200-\lambda \end{bmatrix}$$

[93]:
```
dC=det(C)
```

[93]:

$$31200\lambda + (80-\lambda)(170-\lambda)(200-\lambda) - 2720000$$

[94]:
```
simplify(dC)
```

[94]:

$$\lambda\left(-\lambda^2 + 450\lambda - 32400\right)$$

[95]:
```
eqn = -lambda^3 + 450*lambda^2 -32400*lambda
```

[95]:

$$-\lambda^3 + 450\lambda^2 - 32400\lambda$$

[96]:
```
real_roots(eqn)
```

[96]:

$$\begin{bmatrix} 0 \\ 90 \\ 360 \end{bmatrix}$$

[97]:
```
solve(eqn)
```

[97]:

$$\begin{bmatrix} 0 \\ 90 \\ 360 \end{bmatrix}$$

[98]:
```
factor(-lambda^3 + 450*lambda^2 -32400*lambda,lambda)
```

[98]:

$$-\lambda\left(\lambda - 360\right)\left(\lambda - 90\right)$$

[99]:
```
#recap
#A=[4 11 14; 8 7 -2];
```

[100]:
```
A=[4 11 14; 8 7 -2];
```

https://docs.julialang.org/en/v1/stdlib/LinearAlgebra/#LinearAlgebra.svd

[101]:
```
U,S,V = svd(A)
```

[101]:
```
SVD{Float64,Float64,Array{Float64,2}}
U factor:
2×2 Array{Float64,2}:
 -0.948683  -0.316228
 -0.316228   0.948683
singular values:
2-element Array{Float64,1}:
 18.973665961010283
  9.486832980505136
Vt factor:
2×3 Array{Float64,2}:
 -0.333333  -0.666667  -0.666667
  0.666667   0.333333  -0.666667
```

[131]:
```julia
function fullsvd(A)
    U,s,V = svd(A, full = true)  # compute svd
    Σ = zeros(size(A))           # container for Σ
    for i=1:length(s)
       Σ[i,i] = s[i]            # place singular values in Σ
    end                        # a practical svd would never store all these zeros
    display(U);display(Σ);display(V) # display the answer
    return(U,Σ,V)                # return the answer
end
```

[131]:
```
fullsvd (generic function with 1 method)
```

https://github.com/mitmath/1806/blob/master/summaries.md

[132]:
```julia
function rankrsvd(A)
    U,s,V = svd(A, full = true)  # compute svd
    r = sum(s.>1e-8)             # rank = how many positive?
    U  = U[:,1:r]
    Σ  = Diagonal(s[1:r])         # Diagonal matrix of singular values
    V  = V[:,1:r]
    display(U );display(Σ );display(V ) # display the answer
    return(U ,Σ ,V )                # return the answer
end
```

[132]:
```
rankrsvd (generic function with 1 method)
```

[135]:
```julia
U,S,V = fullsvd(A) #matlab notation
```

```
2×2 Array{Float64,2}:
 -0.948683  -0.316228
 -0.316228   0.948683


2×3 Array{Float64,2}:
 18.9737  0.0      0.0
  0.0     9.48683  0.0


3×3 Adjoint{Float64,Array{Float64,2}}:
 -0.333333   0.666667  -0.666667
 -0.666667   0.333333   0.666667
 -0.666667  -0.666667  -0.333333
```

[135]:
```
([-0.9486832980505135 -0.3162277660168378; -0.3162277660168378
0.9486832980505138], [18.973665961010283 0.0 0.0; 0.0 9.486832980505136 0.0],
[-0.3333333333333332 0.6666666666666665 -0.6666666666666666; -0.6666666666666665
0.3333333333333334 0.6666666666666669; -0.6666666666666665 -0.6666666666666671
-0.3333333333333332])
```

[136]:
```
print_matrix(stdout, U);
```

```
-0.9486832980505135  -0.3162277660168378
-0.3162277660168378   0.9486832980505138
```

[137]:
```
print_matrix(stdout,S);
```

```
18.973665961010283  0.0                0.0
 0.0                9.486832980505136  0.0
```

[138]:
```
print_matrix(stdout, V);
```

```
-0.3333333333333332   0.6666666666666665  -0.6666666666666666
-0.6666666666666665   0.3333333333333334   0.6666666666666669
-0.6666666666666665  -0.6666666666666671  -0.3333333333333332
```

Need to check that $(B - \lambda_i I)v_i = 0$

[139]:
```
#check for lambda_1
 360*V[:,1]
```

[139]:
```
3-element Array{Float64,1}:
 -119.99999999999996
 -239.99999999999994
 -239.99999999999994
```

[140]:
```
alpha=360
```

[140]:
```
360
```

[141]:
```
(B)*V[:,1]
```

[141]:
```
3-element Array{Float64,1}:
 -119.99999999999997
 -239.99999999999994
 -239.99999999999994
```

[142]:
```
#check done
```

Pseudo inverse and linear system

[143]:
```
b=[1;2]
```

[143]:
```
2-element Array{Int64,1}:
 1
 2
```

[144]:
```
rang_A=rank(A)
```

[144]: 2

[145]:
```
#Non inversible == square matrix
# 1/A
# inv(A)
#Least square possible using pinv

pinv(A)
```

[145]:
```
3×2 Array{Float64,2}:
 -0.00555556   0.0722222
  0.0222222    0.0444444
  0.0555556   -0.0555556
```

[146]:
```
xstar=pinv(A)*b
```

[146]:
```
3-element Array{Float64,1}:
  0.13888888888888884
  0.11111111111111109
 -0.0555555555555557
```

[147]:
```
#check 1
```

[148]:
```
pseudoA1=V*pinv(S)*U'
```

[148]:
```
3×2 Array{Float64,2}:
 -0.00555556   0.0722222
  0.0222222    0.0444444
  0.0555556   -0.0555556
```

[152]:
```
xstar1=pseudoA1*b
```

[152]:
```
3-element Array{Float64,1}:
  0.13888888888888884
  0.11111111111111109
 -0.055555555555555684
```

[153]:
```
St=Sn[1:2,1:2]
```

[153]:
```
2×2 Array{Float64,2}:
 18.9737  0.0
  0.0     9.48683
```

[154]:
```
pseudoA2=V*[inv(St'*St)*St'; 0 0 ]*U'
```

[154]:
```
3×2 Array{Float64,2}:
 -0.00555556   0.0722222
```

```
0.0222222    0.0444444
0.0555556   -0.0555556
```

[155]:
```
xstar2=pseudoA2*b
```

[155]:
```
3-element Array{Float64,1}:
  0.13888888888888884
  0.11111111111111109
 -0.055555555555555684
```