Test 1 Josh Boehm

October 26, 2022

Let: $A = $ `np.random.randint(-1,4,size=(3,2))`, $b = $ `np.random.randint(1,5, size=(3,1))`, and $c = $ `np.random.randint(1,5,size =(2,1))`.

[ ]:
```python
import numpy as np
import scipy as sp
from scipy.linalg import svd, qr, cholesky, ldl, lu, svd, diagsvd, pinv, solve
from numpy.linalg import cond
```

[ ]:
```python
a = np.random.randint(-1,4,size=(3,2))
B = np.random.randint(1,5, size=(3,1))
C = np.random.randint(1,5,size =(2,1))

print(f"\
Matrix A:\n{a}\n\n\
Matrix b:\n{B}\n\n\
Matrix c:\n{C}\
")
```

```
Matrix A:
[[2 1]
 [2 2]
 [1 3]]

Matrix b:
[[1]
 [1]
 [3]]

Matrix c:
[[2]
 [2]]
```

0.1 Question 1

[2 points] Solve $A^T Ax = A^T b$ using PLU factorization.

[ ]:
```python
# This step keeps the programs I have a bit simpler to utilize;
# I don't have to reorganize the programming based on whether it's A, A.T, A.T
↪@ A, or A @ A.T.
```

```python
A= a.T @ a
b = a.T @ B

P, L, U = lu(A)

x = solve(A, b)

print(f" \
The original A matrix: \n {a} \n\n \
The original right-side matrix b: \n {B} \n\n \
The matrix A^t A: \n {A} \n\n \
The right-side matrix A^t b \n {b} \n\n \
The Permutation matrix: \n {P} \n\n \
The Lower matrix: \n {L} \n\n \
The Upper matrix: \n {U} \n\n\
The solution vector x:\n{x}\n\
")
```

```
The original A matrix:
[[2 1]
[2 2]
[1 3]]

The original right-side matrix b:
[[1]
[1]
[3]]

The matrix A^t A:
[[ 9  9]
[ 9 14]]

The right-side matrix A^t b
[[ 7]
[12]]

The Permutation matrix:
[[1. 0.]
[0. 1.]]

The Lower matrix:
[[1. 0.]
[1. 1.]]

The Upper matrix:
[[9. 9.]
[0. 5.]]
```

```
The solution vector x:
[[-0.22222222]
 [ 1.        ]]
```

0.2 Question 2

[2 points] Solve $AA^Ty = Ac$ using the LDL^T factorization.

```
[ ]: A = a @ a.T
b = a @ C

L, D, P = ldl(A)
sol = solve(A, b)
z = solve(L, b)
y = solve(D, z)
x = solve(L.T, y)


print(f" \
The original A matrix: \n {a} \n\n \
The original right-side matrix c: \n {C} \n\n \
The matrix A A^t: \n {A} \n\n \
The right-side matrix A c \n {b} \n\n \
The Permutation matrix: \n {P} \n\n \
The Lower matrix: \n {L} \n\n \
The Diagonal matrix: \n {D} \n\n\
Reconstitution of A A^T:\n{L @ D @ L.T}\n\n\
Solve Lz = Ac for z:\n{z}\n\n\
Solve Dy=z for y:\n{y}\n\n\
Solve L^tx = y for x (in our case, its really y):\n{y}\n\n\
Rebuild to test validity:\n{L@D@L.T@x}\n\n\
Ac:\n{b}\n\n\
")

print(f"Condition number:\n{cond(A)}")
```

```
The original A matrix:
[[2 1]
[2 2]
[1 3]]

The original right-side matrix c:
[[2]
[2]]

The matrix A A^t:
```

```
 [[ 5  6  5]
 [ 6  8  8]
 [ 5  8 10]]

 The right-side matrix A c
 [[6]
 [8]
 [8]]

 The Permutation matrix:
 [0 2 1]

 The Lower matrix:
 [[1.  0.  0. ]
 [1.2 0.4 1. ]
 [1.  1.  0. ]]

 The Diagonal matrix:
 [[ 5.00000000e+00  0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  5.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00 -4.21884749e-16]]

Reconstitution of A A^T:
[[ 5.  6.  5.]
 [ 6.  8.  8.]
 [ 5.  8. 10.]]

Solve Lz = Ac for z:
[[ 6.00000000e+00]
 [ 2.00000000e+00]
 [-3.55271368e-16]]

Solve Dy=z for y:
[[1.2       ]
 [0.4       ]
 [0.84210526]]

Solve L^tx = y for x (in our case, its really y):
[[1.2       ]
 [0.4       ]
 [0.84210526]]

Rebuild to test validity:
[[6.]
 [8.]
 [8.]]

Ac:
```

```
[[6]
 [8]
 [8]]


Condition number:
3.5996471215481224e+16
```

```
/var/folders/90/b16ybj8s2gv__lhxf5ttmvj40000gn/T/ipykernel_10023/2976227465.py:5
: LinAlgWarning: Ill-conditioned matrix (rcond=8.42541e-18): result may not be
accurate.
  sol = solve(A, b)
/var/folders/90/b16ybj8s2gv__lhxf5ttmvj40000gn/T/ipykernel_10023/2976227465.py:7
: LinAlgWarning: Ill-conditioned matrix (rcond=8.43769e-17): result may not be
accurate.
  y = solve(D, z)
```

It's worth noting that the condition number of the matrix is quite high sometimes. It may result in poor rounding.

0.3 Question 3

[4 points] Solve $A^T Ax = A^T b$ and $AA^T y = Ac$ using the Cholesky factorization.

0.3.1 $A^T Ax = A^T b$ variant

The below is a matrix that seemed to work with Cholesky's. Feel free to use the same or search for your own!

[ ]:
```
aposdef = np.array([[2,1],[2,2],[1,3]])
bposdef = np.array([[1],[1],[3]])
# print(aposdef)
# print(bposdef)
```

[ ]:
```
a = aposdef
b = bposdef

A = a.T @ a
b = a.T @ B

L = cholesky(A)
y = solve(L, b)
x = solve(L.T, y)

print(f" \
The original A matrix: \n {aposdef} \n\n \
The matrix A^t A: \n {A} \n\n \
The original right-side matrix b: \n {bposdef} \n\n \
The right-side matrix A^t b: \n {b} \n\n \
```

```
The Cholesky Lower matrix: \n {L} \n\n \
Solve A^T A x = L L^t x =  A^T b\n\n \
1st: Solve Ly =  b for y:\n{y}\n\n \
2nd: Solve L^tx = y for x: \n {x} \n\n \
Reconstruct to check:\n {L @ L.T @ x}\n\n \
The A^T b matrix for comparison: \n {b}")
```

```
The original A matrix:
[[2 1]
[2 2]
[1 3]]

The matrix A^t A:
[[ 9  9]
[ 9 14]]

The original right-side matrix b:
[[1]
[1]
[3]]

The right-side matrix A^t b:
[[ 7]
[12]]

The Cholesky Lower matrix:
[[3.         3.        ]
[0.         2.23606798]]

Solve A^T A x = L L^t x =  A^T b

1st: Solve Ly =  b for y:
[[-3.03322981]
 [ 5.36656315]]

2nd: Solve L^tx = y for x:
[[-1.0110766 ]
[ 3.75650161]]

Reconstruct to check:
[[ 7.]
[12.]]

The A^T b matrix for comparison:
[[ 7]
[12]]
```

0.3.2 AA^T variant

[ ]:
```
aposdef = np.array([[3,-1],[1,2],[-1,0]])
cposdef = np.array([[1],[1]])

a = aposdef
C = cposdef

A = a @ a.T
b = a @ C

L = cholesky(A)
y = solve(L, b)
x = solve(L.T, y)

print(f" \
The original A matrix: \n {aposdef} \n\n \
The matrix A^t A: \n {A} \n\n \
The original right-side matrix b: \n {bposdef} \n\n \
The right-side matrix A^t b: \n {b} \n\n \
The Cholesky Lower matrix: \n {L} \n\n \
Solve A^T A x = L L^t x =  A^T b\n\n \
1st: Solve Ly =  b for y:\n{y}\n\n \
2nd: Solve L^tx = y for x: \n {x} \n\n \
Reconstruct to check:\n {L @ L.T @ x}\n\n \
The A^T b matrix for comparison: \n {b}")
```

```
The original A matrix:
[[ 3 -1]
[ 1  2]
[-1  0]]

The matrix A^t A:
[[10  1 -3]
[ 1  5 -1]
[-3 -1  1]]

The original right-side matrix b:
[[1]
[1]
[3]]

The right-side matrix A^t b:
[[ 2]
[ 3]
[-1]]

The Cholesky Lower matrix:
```

```
[[ 3.16227766e+00  3.16227766e-01 -9.48683298e-01]
 [ 0.00000000e+00  2.21359436e+00 -3.16227766e-01]
 [ 0.00000000e+00  0.00000000e+00  1.05367121e-08]]

 Solve A^T A x = L L^t x =  A^T b

 1st: Solve Ly =  b for y:
[[-27116075.39571395]
 [-13558036.5910598 ]
 [-94906265.62425154]]

 2nd: Solve L^tx = y for x:
 [[-8.57485595e+06]
 [-4.89991718e+06]
 [-9.92630119e+15]]

 Reconstruct to check:
 [[ 1.99999999]
 [ 3.        ]
 [-1.        ]]

 The A^T b matrix for comparison:
 [[ 2]
 [ 3]
 [-1]]
```

0.3.3 Question 4

[2 Points] Find the Eigenvalues and corresponding Eigenvectors of $A^T A$.

[ ]:
```python
from numpy.linalg import eig

eig(a.T @ a)
```

[ ]:
```
(array([11.16227766,  4.83772234]),
 array([[ 0.98708746,  0.16018224],
        [-0.16018224,  0.98708746]]))
```

[1 Point] What do you know about the Eigenvalues of AA^T?

We know them to be equal to the above, $A^T A$, in addition to more zero eigenvalues; the quantity of how many extra eigenvalues depends on the difference between the number of rows, m, and the number of columns, n, which results in this case to be 1.

0.4 Question 5

[4 points] Find the QR factorization of A and use the factorization to solve $Ax = b$ and $A^T y = c$.

0.4.1 QR factorization

[ ]:
```
A = a
b = B
c = C
```

0.4.2 Solving $Ax = b$ with QR factorizations.

[ ]:
```
Q,R = qr(A, mode = 'economic')

x = solve(R, Q.T @ b)
print(f"\
The Matrix A:\n{A}\n\n\
The Matrix b:\n{b}\n\n\
The Q matrix:\n{Q}\n\n\
The R matrix:\n{R}\n\n\
Solving Ax = QRx = b for x:\n{x}\n\n\
")
```

```
The Matrix A:
[[ 3 -1]
 [ 1  2]
 [-1  0]]

The Matrix b:
[[3]
 [2]
 [4]]

The Q matrix:
[[-0.90453403  0.32824398]
 [-0.30151134 -0.94370143]
 [ 0.30151134  0.0410305 ]]

The R matrix:
[[-3.31662479  0.30151134]
 [ 0.          -2.21564684]]

Solving Ax = QRx = b for x:
[[0.66666667]
 [0.33333333]]
```

0.4.3 Solving $A^T y = c$ with QR factorizations.

```
[ ]: Q, R = qr(A.T, mode='economic')
x = x = solve(R.T @ R, R.T @ c)


print(f" \
The A transpose matrix: \n {A.T} \n\n \
The QR factorization:\n\n\
Q matrix:\n{Q}\n\n\
R matrix:\n{R}\n\n\
Solving for x by using QR y = c:\n\
{x}\n\n\
")
```

```
 The A transpose matrix:
 [[ 3  1 -1]
 [-1  2  0]]

 The QR factorization:

Q matrix:
[[-0.9486833   0.31622777]
 [ 0.31622777  0.9486833 ]]

R matrix:
[[-3.16227766 -0.31622777  0.9486833 ]
 [ 0.          2.21359436 -0.31622777]]

Solving for x by using QR y = c:
[[-1.40210586]
 [-0.0685974 ]
 [-3.64245943]]



/var/folders/90/b16ybj8s2gv__lhxf5ttmvj40000gn/T/ipykernel_10023/3290188184.py:2
: LinAlgWarning: Ill-conditioned matrix (rcond=5.74937e-19): result may not be
accurate.
  x = x = solve(R.T @ R, R.T @ c)
```

0.5 Question 6

[4 points] Find the SVD factorization of A and use the factorization to solve $Ax = b$ and $A^T y = c$.

0.5.1 SVD factorization of A

[ ]:
```
m = 3; n = 2
# Enter m-Rows & n-Columns Of Coefficient Matrix A:
U, sigma, VT = svd(A)
Sigma = diagsvd(sigma, m, n)

print(f"\
------------------------------------\n\
Perform Singular Value Decomposition\n\
------------------------------------\n\
The A matrix:\n\
{A}\n\
------------------------------------\n\
The ({m} x {m}) U Matrix\n\
{U}\n\
------------------------------------\n\
The ({m} x {n}) Singular Values of A\n\
{Sigma}\n\
------------------------------------\n\
The ({n} x {n}) V^T Matrix\n\
{VT}\n\
------------------------------------\n\
Reconstructed Original ({m} x {n}) Matrix A From SVD Factors\n\
{U @ Sigma @ VT}\n\
------------------------------------\n\
")
```

```
------------------------------------
Perform Singular Value Decomposition
------------------------------------
The A matrix:
[[ 3 -1]
 [ 1  2]
 [-1  0]]
------------------------------------
The (3 x 3) U Matrix
[[-0.93428467  0.23029998  0.27216553]
 [-0.19955794 -0.9703907   0.13608276]
 [ 0.29544675  0.07282725  0.95257934]]
------------------------------------
The (3 x 2) Singular Values of A
[[3.3409995  0.        ]
 [0.         2.19948229]
 [0.         0.        ]]
------------------------------------
The (2 x 2) V^T Matrix
[[-0.98708746  0.16018224]
```

```
 [-0.16018224 -0.98708746]]
------------------------------------
Reconstructed Original (3 x 2) Matrix A From SVD Factors
[[ 3.00000000e+00 -1.00000000e+00]
 [ 1.00000000e+00  2.00000000e+00]
 [-1.00000000e+00 -3.31756994e-17]]
------------------------------------
```

0.5.2 $Ax = b$ with SVD factorization.

[ ]:
```
PseudoA = (VT).T @ pinv(Sigma) @ U.T
x = PseudoA @  b
print(f"\
------------------------------------\n\
Solve A x = b Using The Pseudo-Inverse of A\n\
From  x = A^+ b = V Sigma^+ U^T b\n\
------------------------------------\n\
Solution Vector x =\n\
{x}\n\
")
```

```
------------------------------------
Solve A x = b Using The Pseudo-Inverse of A
From  x = A^+ b = V Sigma^+ U^T b
------------------------------------
Solution Vector x =
[[0.66666667]
 [0.33333333]]
```

0.5.3 $A^T y = c$ with SVD factorization.

[ ]:
```
U, sigma, VT = svd(A.T)
Sigma = diagsvd(sigma, n, m)
PseudoAT = (VT).T @ pinv(Sigma) @ U.T
x = PseudoAT @  c

print(f"\
------------------------------------\n\
Solve A^T y = C Using The Pseudo-Inverse of A^T\n\
From  y = A^+ c = V Sigma^+ U^T c\n\
------------------------------------\n\
Solution Vector x =\n\
{x}\n\
")
```

```
------------------------------------
```

```
Solve A^T y = C Using The Pseudo-Inverse of A^T
From  y = A^+ c = V Sigma^+ U^T c
------------------------------------
Solution Vector x =
[[ 0.11111111]
 [ 0.55555556]
 [-0.11111111]]
```

0.5.4 Question 7

[2 points] Find the Singular Values of A and the Pseudo-Inverse of A^+

[ ]:
```
U, sigma, VT = svd(A)
print(f"\
The Singular Values of A:\n{sigma}\n\n\
The Pseudo-inverse of A:\n{PseudoA}\n\n\
To illustrate the Pseudo-inverse, we can multiply A^+A and get the identity:
 ↪\n{PseudoA @ A}\n\n\
")
```

```
The Singular Values of A:
[3.3409995  2.19948229]

The Pseudo-inverse of A:
[[ 0.25925926  0.12962963 -0.09259259]
 [-0.14814815  0.42592593 -0.01851852]]

To illustrate the Pseudo-inverse, we can multiply A^+A and get the identity:
[[1.00000000e+00 0.00000000e+00]
 [3.12250226e-17 1.00000000e+00]]
```

[1 point] What is the Rank of A?

[ ]:
```
np.linalg.matrix_rank(A)
```

[ ]: 2

With the function above we can determine the rank to be 2, however since there are only 2 Singular Values, this also indicates that it's only 2 as well.