Deshpande_Charudatta_Linear_Algebra

January 21, 2018

1 Linear algebra

This is a voluntary problem set that helps you to learn linear algebra (and the related numpy commands.) Consult Greene "Econometric Analysis" Appendix A when solving these problems.

If handed in (and solved correctly ;-), it gives you 5 credits.

In [61]:
```
######################################################################
### Student Name - Charudatta Deshpande                            ###
###                                                                ###
### Collaborators - Ram Ganesan, Charles Hemstreet, Mehdi Muntazir ###
###                                                                ###
######################################################################
### Import required libraries.                                       #
######################################################################
###
import numpy as np
import scipy as sp
from scipy import linalg
import pandas as pd
import math as mt
from numpy.linalg import matrix_rank
```

1.1 1 Matrix multiplication

- create the following matrices: a = [1 2 3 4], b = [5 6 7 8]′ (note: b includes the transposition sign)
- compute the following matrix products $a \cdot b$ and $a' \cdot b'$

In [62]:
```
#
#Input the matrices.
#
a=np.matrix([[1,2,3,4]])
b=np.matrix([[5],[6],[7],[8]])
print(a)
print(b)
#
# Calculate the dot product ab
```

```
#
c = np.dot(a,b)
print('ab is equal to : ', c)
#
# Calculate the dot product ab
#
a_dash = a.transpose()
b_dash = b.transpose()
d = np.dot(a_dash,b_dash)
print('ab is equal to : \n', d)
```

```
[[1 2 3 4]]
[[5]
 [6]
 [7]
 [8]]
ab is equal to :   [[70]]
ab is equal to :
 [[ 5  6  7  8]
 [10 12 14 16]
 [15 18 21 24]
 [20 24 28 32]]
```

1.2 2 Linear (in)dependence

Consider three vectors: a = [1 2 3 4], b = [5 6 7 8] and c = [9 10 11 12]

- are these vectors linearly independent? Calculate the rank, a related determinant, and show how they are related/unrelated

```
In [63]: #
# Input the matrices.
#
# This is about determinant.
# If the matrices can be described in terms of each other, they are dependent.
# else they are independent.
# Create a single matrix out of 3 vectors. Calculate rank.
# If rank is greater than 3 (number of rows), these vectors are independent.
#
a = np.matrix([[1,2,3,4]])
b = np.matrix([[5,6,7,8]])
c = np.matrix([[9,10,11,12]])
X = np.row_stack([a, b, c])
print(X)
#
# Method 1 to calculate rank
#
print('Matrix X rank is: ', matrix_rank(X))
```

```
#
# Method 2 to calculate rank
#
TOLERANCE = 1e-14
U, s, V = np.linalg.svd(X)
print(s)
print('Matrix X rank using second methos is: ', np.sum(s > TOLERANCE))
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
Matrix X rank is:  2
[  2.54368356e+01   1.72261225e+00   4.20733283e-16]
Matrix X rank using second methos is:  2
```

In [64]:
```
## Explanation of Q 2.
## The rank is 2 and number of rows are 3.
## The number of rows is greater than the rank, so these vectors are
## not independent (are dependent).
## This means that one vector can be defined as a linear combination of the
## other two vectors.
```

1.3 3 Find the inverses of the following matrices:

A = [1 2 3 4 5 6 7 8 -1 10 11 12 13 14 15 17],
B = [1 0 0 0 4 0 0 0 16]
and
C = [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 17]

- Check the results by left- and right multiplication of the inverse. Explain.

In [65]:
```
#
# Input the matrices.
#
A=np.matrix([[1,2,3,4],[5,6,7,8],[-1,10,11,12],[13,14,15,17]])
B=np.matrix([[1,0,0],[0,4,0],[0,0,16]])
C=np.matrix([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,17]])
#
# Calculate inverse matrices
#
A_inverse = np.linalg.inv(A)
B_inverse = np.linalg.inv(B)
C_inverse = np.linalg.inv(C)
print('A inverse is equal to : \n', A_inverse)
print('B inverse is equal to : \n', B_inverse)
print('C inverse is equal to : \n', C_inverse)
#
# Calculate dot products of original and inverse matrices
```

```python
#
print('A.A_inverse is equal to : \n', np.dot(A, A_inverse))
print('A_inverse.A is equal to : \n', np.dot(A_inverse, A))
print('B.B_inverse is equal to : \n', np.dot(B, B_inverse))
print('B_inverse.B is equal to : \n', np.dot(B_inverse, B))
print('C.C_inverse is equal to : \n', np.dot(C, C_inverse))
print('C_inverse.C is equal to : \n', np.dot(C_inverse, C))
```

```
A inverse is equal to :
 [[ -1.00000000e-01   2.00000000e-01  -1.00000000e-01   2.73285668e-16]
 [  4.50000000e-01  -2.65000000e+00   2.00000000e-01   1.00000000e+00]
 [ -2.60000000e+00   5.70000000e+00  -1.00000000e-01  -2.00000000e+00]
 [  2.00000000e+00  -3.00000000e+00  -7.40148683e-17   1.00000000e+00]]
B inverse is equal to :
 [[ 1.      0.      0.    ]
 [ 0.      0.25    0.    ]
 [ 0.      0.      0.0625]]
C inverse is equal to :
 [[ -1.54070514e+15   3.08141027e+15  -1.54070514e+15  -4.70000000e-01]
 [  3.08141027e+15  -6.16282054e+15   3.08141027e+15   1.94000000e+00]
 [ -1.54070514e+15   3.08141027e+15  -1.54070514e+15  -2.47000000e+00]
 [  1.15789474e+00  -1.31578947e+00  -8.42105263e-01   1.00000000e+00]]
A.A_inverse is equal to :
 [[  1.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   1.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   7.10542736e-15   1.00000000e+00  -3.55271368e-15]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   1.00000000e+00]]
A_inverse.A is equal to :
 [[  1.00000000e+00   1.77635684e-15   1.77635684e-15   2.22044605e-15]
 [ -1.77635684e-15   1.00000000e+00  -1.77635684e-15  -2.66453526e-15]
 [  1.33226763e-15   8.88178420e-16   1.00000000e+00   1.77635684e-15]
 [  4.44089210e-16   1.77635684e-15   1.77635684e-15   1.00000000e+00]]
B.B_inverse is equal to :
 [[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
B_inverse.B is equal to :
 [[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
C.C_inverse is equal to :
 [[  1.   0.   1.   0.]
 [ -4.   0.  -4.   0.]
 [ -4.   8.   0.   0.]
 [  0.  16.   0.   1.]]
C_inverse.C is equal to :
 [[  8.00000000e+00   8.00000000e+00   8.00000000e+00   1.20000000e+01]
 [ -8.00000000e+00  -1.60000000e+01  -8.00000000e+00  -1.60000000e+01]
```

```
[  4.00000000e+00   4.00000000e+00   8.00000000e+00   8.00000000e+00]
[ -8.88178420e-16  -1.77635684e-15  -1.77635684e-15   1.00000000e+00]]
```

```
In [66]: ## Explanation of Q 3.
         ## Dot product of a matrix and its inverse should be an identity matrix irrespective
         ## of the order of the multiplication.
         ## For A, A.A_inverse and A_inverse.A are very close to being an identity matrix.
         ## However, the identity matrices are not perfect.
         ## For B, B.B_inverse and B_inverse.B are perfect identity matrices.
         ## For C, C.C_inverse and C_inverse.C are not identity matrices.
         ## This indicates that C is not an invertible matrix.
```

1.4 4 Characteristic roots

- Find the roots (eigenvalues) of the matrices A, B, C above.
- Calculate the condition numbers of these matrices in two ways: the default numpy way, and in this way as it is explained in Greene (2003, page 829)
- Explain the results

```
In [67]: #
         #Calculate eigenvalues for A, B and C.
         #
         eigvals_A = np.linalg.eigvals(A)
         eigvals_B = np.linalg.eigvals(B)
         eigvals_C = np.linalg.eigvals(C)
         eigvals_A = np.absolute(eigvals_A)
         eigvals_B = np.absolute(eigvals_B)
         eigvals_C = np.absolute(eigvals_C)
         #
         print('eigenvalues for A: \n', eigvals_A)
         print('eigenvalues for B: \n', eigvals_B)
         print('eigenvalues for C: \n', eigvals_C)
         #
         #Calculate condition numbers for A, B and C - the default numpy way.
         #
         print('condition number for A using Numpy method: \n', np.linalg.cond(A))
         print('condition number for B using Numpy method: \n', np.linalg.cond(B))
         print('condition number for C using Numpy method: \n', np.linalg.cond(C))
         #
         #Calculate condition numbers for A, B and C - as explained in Greene (2003, page 829)
         #
         condition_number_A = np.sqrt(eigvals_A.max() / eigvals_A.min())
         print('condition number for A using Greene method: \n', condition_number_A)
         condition_number_B = np.sqrt(eigvals_B.max() / eigvals_B.min())
         print('condition number for A using Greene method: \n', condition_number_B)
         condition_number_C = np.sqrt(eigvals_C.max() / eigvals_C.min())
         print('condition number for A using Greene method: \n', condition_number_C)
```

```
eigenvalues for A:
 [ 35.83449756   1.34371891   1.6853005    0.49291597]
eigenvalues for B:
 [  1.   4.  16.]
eigenvalues for C:
 [  3.66727818e+01   2.00000000e+00   2.08470091e-15   3.27218155e-01]
condition number for A using Numpy method:
 299.050204082
condition number for B using Numpy method:
 16.0
condition number for C using Numpy method:
 7.60469682786e+16
condition number for A using Greene method:
 8.5263707646
condition number for A using Greene method:
 4.0
condition number for A using Greene method:
 132632528.659
```

In [68]:
```
# Explanation of Q 4 -
# The condition numbers using Numpy method and Greene method
# do not match. This could be mostly due to an error I made in the
# calculation of Eigenvalues, or use of an incorrect option in
# calculation of condition numbers using Greene method.
# I understood how the Greene method is used, however I was getting
# negative eigenvalues, and using those in square root function was
# returning an invalid calculation. So I had to use absolute values
# and I believe I missed something there that is giving me wrong
# condition numbers using Greene method.
```