6

Linear Algebra
Least Squares

Anahita Zarei, Ph.D.

Overview

- Dot Product
- Distance
- Unit Vector
- Orthogonality
- Least squares solution to inconsistent systems
- Reading: Chapter 6 with emphasis on sections 6.1, 6.5, 6.6

Section 6.1

INNER PRODUCT, LENGTH, AND ORTHOGONALITY

INNER PRODUCT

- If **u** and **v** are vectors in $\mathbb{R}^n$, then we regard **u** and **v** as $n \times 1$ matrices.

$$\text{u} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} \qquad \text{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

- The inner product (dot product) of **u** and **v** is

$$u_1v_1 + u_2v_2 + \cdots + u_nv_n.$$

INNER PRODUCT

- Can you express the dot product in terms of matrix multiplication of u and v?

- Hint: The transpose $\mathbf{u}^T$ is a 1xn matrix, and the matrix product $\mathbf{u}^T\mathbf{v}$ is a 1x1 matrix, which we write as a single real number (a scalar) without brackets.

INNER PRODUCT

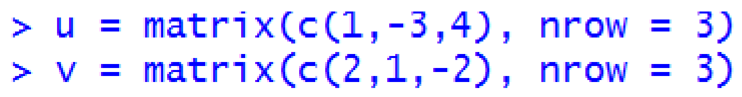
- If u= $\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}$ and v = $\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$,

  then the inner product of **u** and **v** is

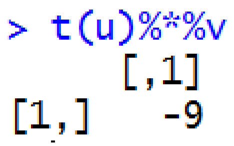
$$\begin{bmatrix} u_1 & u_2 & \cdots & u_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n \quad .$$

Dot Product in R - Example

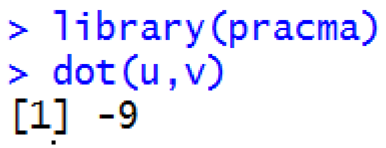
- $u = \begin{bmatrix} 1 \\ -3 \\ 4 \end{bmatrix}$ $v = \begin{bmatrix} 2 \\ 1 \\ -2 \end{bmatrix}$
- Find the dot product of u and v in R.



```
> u = matrix(c(1,-3,4), nrow = 3)
> v = matrix(c(2,1,-2), nrow = 3)
```



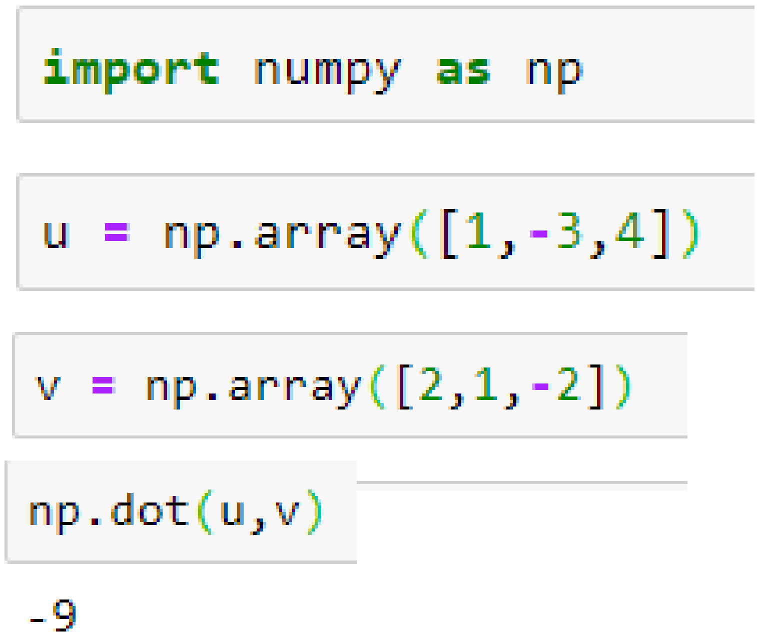
```
> t(u)%*%v
     [,1]
[1,]   -9
```



```
> library(pracma)
> dot(u,v)
[1] -9
```

Dot Product in Python - Example

- $u = \begin{bmatrix} 1 \\ -3 \\ 4 \end{bmatrix}$ $v = \begin{bmatrix} 2 \\ 1 \\ -2 \end{bmatrix}$
- Find the dot product of u and v in Python.



```
import numpy as np
```

```
u = np.array([1,-3,4])
```

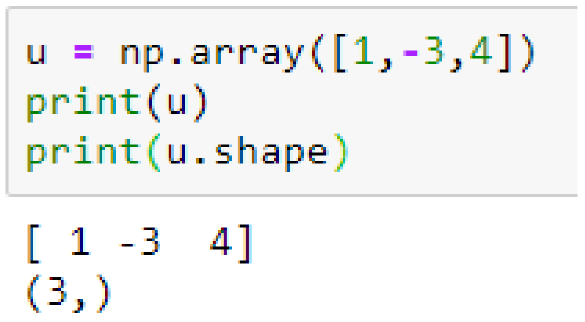
```
v = np.array([2,1,-2])
```

```
np.dot(u,v)
```

```
-9
```

Arrays in Python

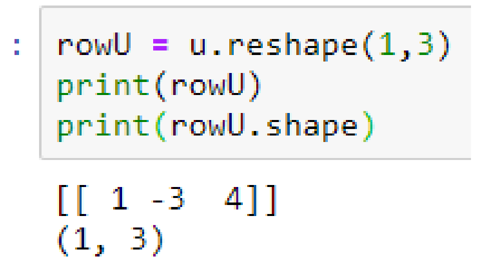
- Note that when you enter array U in Python as follows it'll be a 1D array by default.



```
u = np.array([1,-3,4])
print(u)
print(u.shape)
```

```
[ 1 -3  4]
(3,)
```

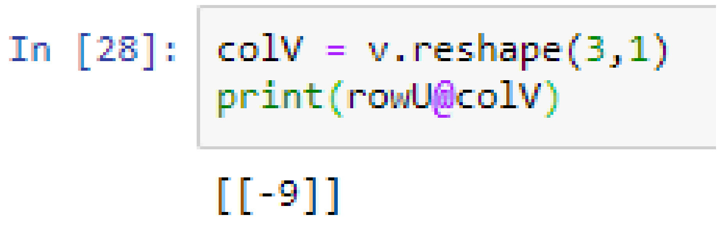
- You can make it a row or a column vector by using the reshape command.



```
: rowU = u.reshape(1,3)
  print(rowU)
  print(rowU.shape)
```

```
[[ 1 -3  4]]
(1, 3)
```

- You can also perform the dot product using matrix multiplication.

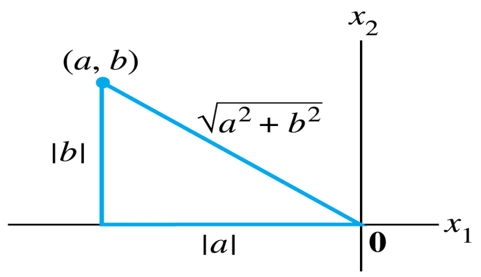


```
In [28]: colV = v.reshape(3,1)
         print(rowU@colV)
```

```
[[-9]]
```

THE LENGTH OF A VECTOR

- If we identify $\mathrm{v} = \begin{bmatrix} a \\ b \end{bmatrix}$ with a geometric point in the plane, then $\|\mathrm{v}\|$ coincides with the standard notion of the length of the line segment from the origin to **v**.
- This follows from the Pythagorean Theorem applied to a triangle such as the one shown in the following figure.



Interpretation of $\|\mathbf{v}\|$ as length.

THE LENGTH OF A VECTOR

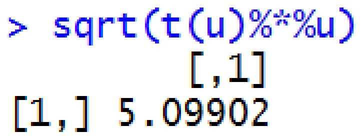
- **Definition:** The **length** (or **norm**) of **v** is the nonnegative scalar $\|v\|$ defined by

$$\|v\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

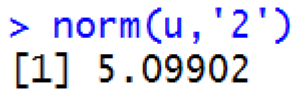
- Can you express norm of v in terms of a dot product?
- $||v|| = \sqrt{v \cdot v}$

THE LENGTH OF A VECTOR in R

- Find length of $u = \begin{bmatrix} 1 \\ -3 \\ 4 \end{bmatrix}$ in R.



```
> sqrt(t(u)%*%u)
         [,1]
[1,] 5.09902
```

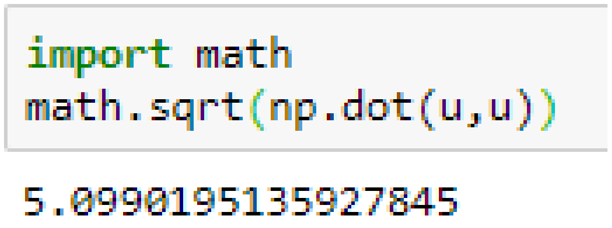


```
> norm(u,'2')
[1] 5.09902
```

- Note that R by default finds L1 norm, unless you specify what type of norm needs to be calculated.

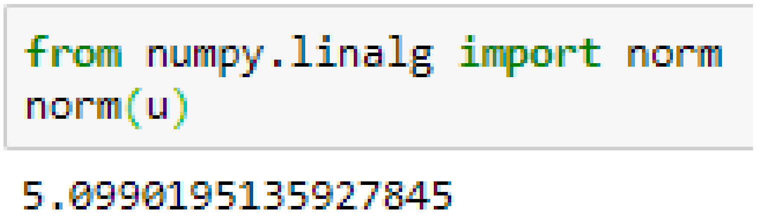
THE LENGTH OF A VECTOR in Python

- Find length of $u = \begin{bmatrix} 1 \\ -3 \\ 4 \end{bmatrix}$ in Python.



```
import math
math.sqrt(np.dot(u,u))
```

```
5.0990195135927845
```



```
from numpy.linalg import norm
norm(u)
```

```
5.0990195135927845
```

Alternative method for finding the dot product

- If **u** and **v** are nonzero vectors in either $\mathbb{R}^2$ or $\mathbb{R}^3$, then there is a nice connection between their inner product and the angle between the two line segments from the origin to the points identified with **u** and **v**.
- The formula is $u \cdot v = \|u\|\|v\|\cos\vartheta$ (proof on the next slide)
- Two vectors **u** and **v** are **orthogonal** (to each other) if $u \cdot v = 0$.

Proof

Unit Vector

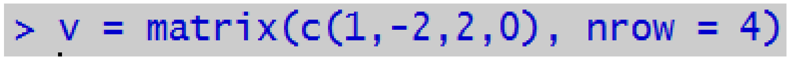
- A vector whose length is 1 is called a **unit vector**.

- If we *divide* a nonzero vector **v** by its length—that is, multiply by $1/\|\mathbf{v}\|$—we obtain a unit vector **u** because the length of **u** is. $(1/\|\mathbf{v}\|)\|\mathbf{v}\|$

- The process of creating **u** from **v** is sometimes called **normalizing v**, and we say that **u** is *in the same direction* as **v**.

Finding Unit Vector – Example R

- **Example 2:** Let $\mathrm{v} = (1, -2, 2, 0)$. (Assume v is a column vector). Find a unit vector **u** in the same direction as **v**.

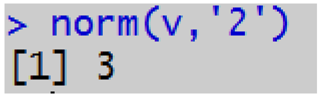


```
> v = matrix(c(1,-2,2,0), nrow = 4)
```

- **Solution:** First, compute the length of **v**:

$$\|\mathrm{v}\|^2 = \mathrm{v} \cdot \mathrm{v} = (1)^2 + (-2)^2 + (2)^2 + (0)^2 = 9$$

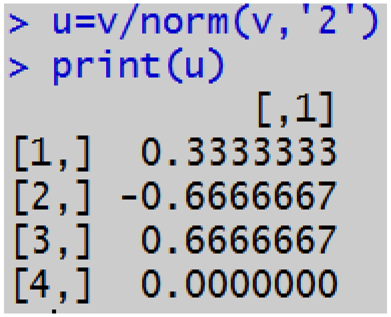
$$\|\mathrm{v}\| = \sqrt{9} = 3$$



```
> norm(v,'2')
[1] 3
```

- Then, multiply **v** by $1/\|\mathrm{v}\|$ to obtain

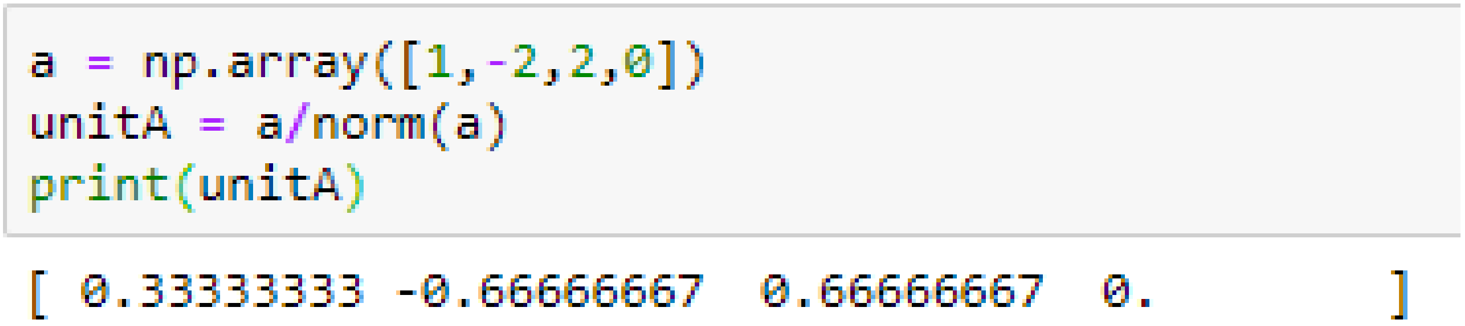
$$\mathrm{u} = \frac{1}{\|\mathrm{v}\|}\mathrm{v} = \frac{1}{3}\mathrm{v} = \frac{1}{3}\begin{bmatrix} 1 \\ -2 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/3 \\ -2/3 \\ 2/3 \\ 0 \end{bmatrix}$$



```
> u=v/norm(v,'2')
> print(u)
           [,1]
[1,]  0.3333333
[2,] -0.6666667
[3,]  0.6666667
[4,]  0.0000000
```

Finding Unit Vector – Example Python

- **Example 2:** Let $\mathrm{v} = (1, -2, 2, 0)$. (Assume v is a column vector). Find a unit vector **u** in the same direction as **v**.



```
a = np.array([1,-2,2,0])
unitA = a/norm(a)
print(unitA)
```

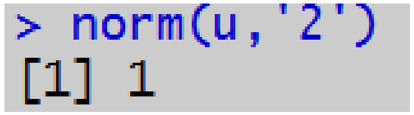
```
[ 0.33333333 -0.66666667  0.66666667  0.        ]
```

Verify your answer

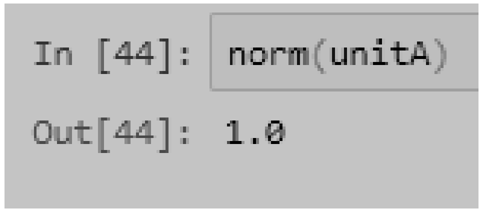
- To check that $\|u\| = 1$, it suffices to show that $\|u\|^2 = 1$
- .

$$\|u\|^2 = u□u = \left(\frac{1}{3}\right)^2 + \left(-\frac{2}{3}\right) + \left(\frac{2}{3}\right)^2 + (0)^2$$

$$= \frac{1}{9} + \frac{4}{9} + \frac{4}{9} + 0 = 1$$



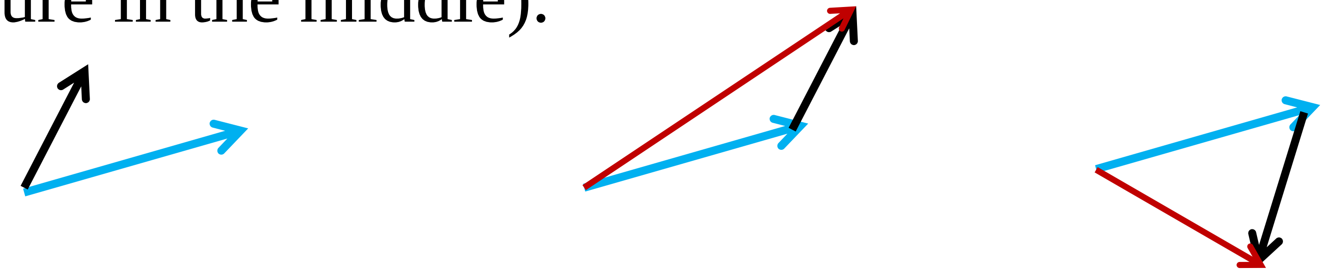
```
> norm(u,'2')
[1] 1
```



```
In [44]: norm(unitA)

Out[44]: 1.0
```

DISTANCE IN $\mathbb{R}^n$

- **Definition:** For **u** and **v** in $\mathbb{R}^n$, the **distance between u and v**, written as dist (**u**, **v**), is the length of the vector $u - v$. That is, $\text{dist}\,(u,v) = \|u - v\|$
- Recall that we can geometrically add two vectors by moving the tail of one vector to the head of the other (figure in the middle).



- Subtraction can be done similarly by adding the negative of the vector. (figure on the right)

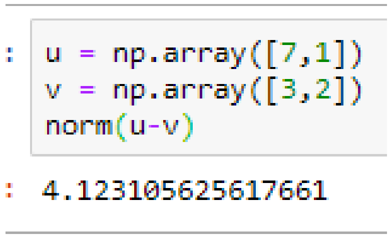
Example

- **Example 4:** Compute the distance between the vectors $\mathrm{u} = (7,1)$ and $\mathrm{v} = (3,2)$

Solution

.

- **Solution:** Calculate



```
: u = np.array([7,1])
  v = np.array([3,2])
  norm(u-v)

: 4.123105625617661
```

$$\mathrm{u}-\mathrm{v}=\begin{bmatrix}7\\1\end{bmatrix}-\begin{bmatrix}3\\2\end{bmatrix}=\begin{bmatrix}4\\-1\end{bmatrix}$$

$$\|\mathrm{u}-\mathrm{v}\|=\sqrt{4^2+(-1)^2}=\sqrt{17}$$

Section 6.5

Least Squares Problems

Background

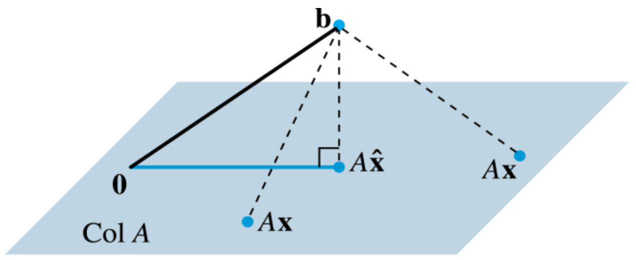
- Consider Ax=b where the systems is inconsistent. i.e. there doesn’t exist a solution x that satisfies this matrix equation.
- When a solution is demanded and none exists, the best one can do is to find an x that makes **Ax as close as possible to b.**
- Think of Ax as an approximation to b. The smaller the distance between b and Ax given by ||b-Ax||, the better the approximation.

LEAST-SQUARES PROBLEMS

- **Definition:** If A is $m \times n$ and **b** is in $\mathbb{R}^m$, a **least-squares solution** of $A\mathrm{x} = \mathrm{b}$ is an $\hat{\mathrm{x}}$ in $\mathbb{R}^n$ such that
$$\|\mathrm{b} - A\hat{\mathrm{x}}\| \leq \|\mathrm{b} - A\mathrm{x}\|$$
for all **x** in $\mathbb{R}^n$.
- Note that vector Ax is a linear combinations of columns of A. (Recall the 2nd version of matrix-vector multiplication definition).
- This implies that no matter what x we select, the vector Ax will be on the plane spanned by columns of A (column space of A)

Shortest Distance

- So we seek an x that makes Ax the closest point in Col A to b.
- Recall that the shortest distance is the orthogonal distance.



The vector $\mathbf{b}$ is closer to $A\hat{\mathbf{x}}$ than to $A\mathbf{x}$ for other $\mathbf{x}$.

- Therefore, the dotted line that is perpendicular to the plane will result in smallest distance between b and Ax.
- What’s the expression for the dotted line?

Shortest Distance

- If a vector is orthogonal to a plane it'll be orthogonal to any vector on that plane, including columns of matrix A, namely a_1 and a_2.
- So $b - A\hat{x}$ is orthogonal to each column of A.
- If $\mathbf{a}_j$ is any column of A, then $a_j \cdot (b - A\hat{x}) = 0$, and $a_j^T(b - A\hat{x}) = 0$.

SOLUTION OF THE GENREAL LEAST-SQUARES PROBLEM

- Since each a_j^T is a row of A^T,

$$A^T(\mathrm{b} - A\hat{\mathrm{x}}) = 0 \qquad (2)$$

- Thus

$$A^T\mathrm{b} - A^T A\hat{\mathrm{x}} = 0$$

$$A^T A\hat{\mathrm{x}} = A^T\mathrm{b}$$

- These calculations show that each least-squares solution of $A\mathrm{x} = \mathrm{b}$ satisfies the equation

$$A^T A\mathrm{x} = A^T\mathrm{b} \qquad (3)$$

- The matrix equation (3) represents a system of equations called the **normal equations** for $A\mathrm{x} = \mathrm{b}$.
- A solution of (3) is often denoted by $\hat{\mathrm{x}}$.

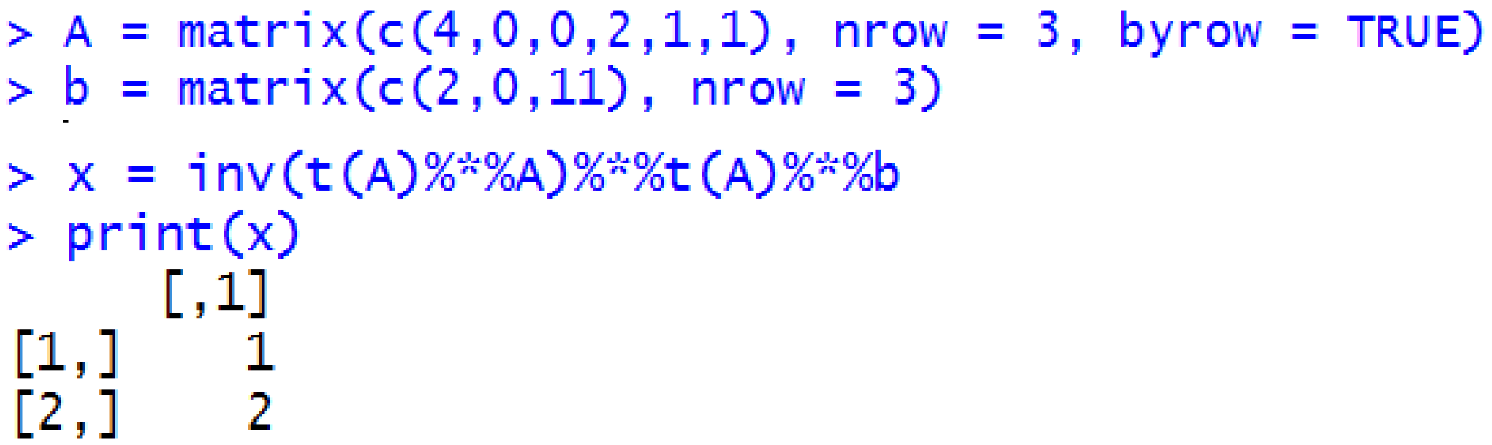
$$\hat{\mathrm{x}} = (A^T A)^{-1} A^T\mathrm{b}$$

SOLUTION OF THE GENREAL LEAST-SQUARES PROBLEM

- **Example 1:** Find a least-squares solution of the inconsistent system $A\mathrm{x} = \mathrm{b}$ for

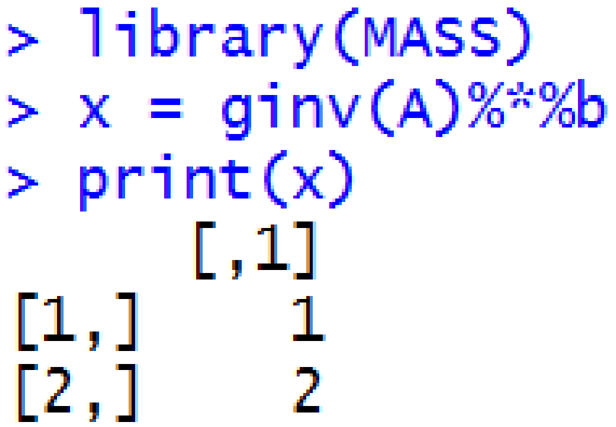
$$A = \begin{bmatrix} 4 & 0 \\ 0 & 2 \\ 1 & 1 \end{bmatrix}, \mathrm{b} = \begin{bmatrix} 2 \\ 0 \\ 11 \end{bmatrix}$$

Solution in R



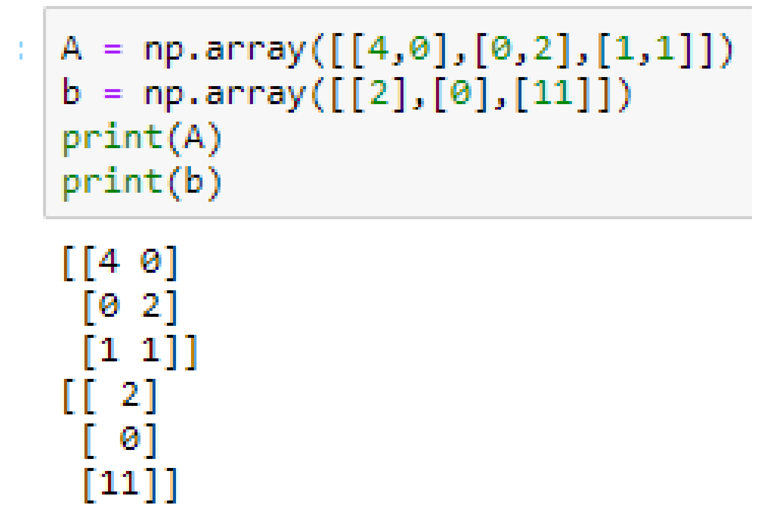
```
> A = matrix(c(4,0,0,2,1,1), nrow = 3, byrow = TRUE)
> b = matrix(c(2,0,11), nrow = 3)

> x = inv(t(A)%*%A)%*%t(A)%*%b
> print(x)
     [,1]
[1,]    1
[2,]    2
```

OR



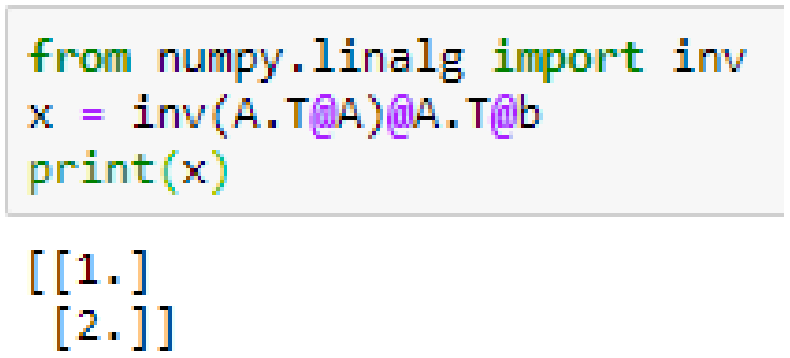
```
> library(MASS)
> x = ginv(A)%*%b
> print(x)
     [,1]
[1,]    1
[2,]    2
```

Solution in Python



```
A = np.array([[4,0],[0,2],[1,1]])
b = np.array([[2],[0],[11]])
print(A)
print(b)
```

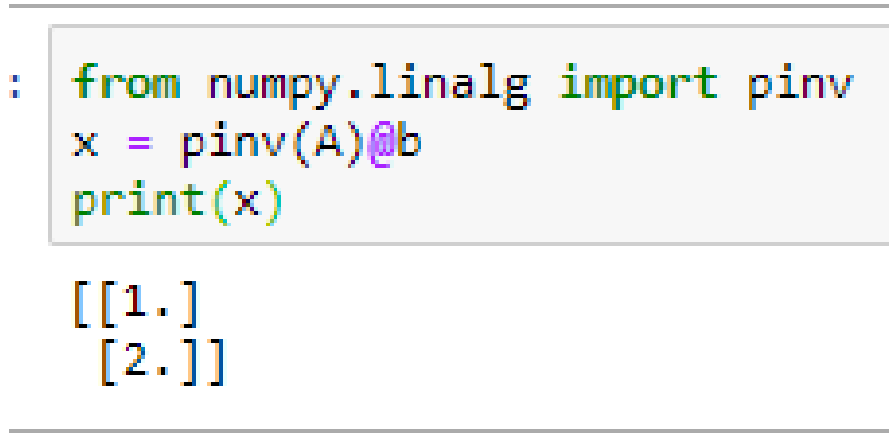
```
[[4 0]
 [0 2]
 [1 1]]
[[ 2]
 [ 0]
 [11]]
```



```
from numpy.linalg import inv
x = inv(A.T@A)@A.T@b
print(x)
```

```
[[1.]
 [2.]]
```

OR

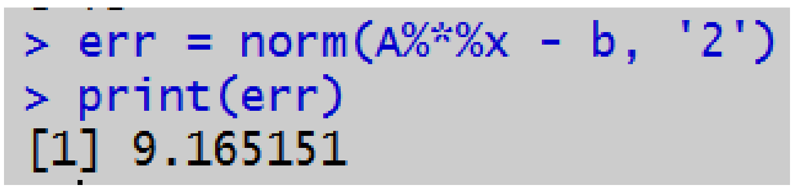


```
from numpy.linalg import pinv
x = pinv(A)@b
print(x)
```

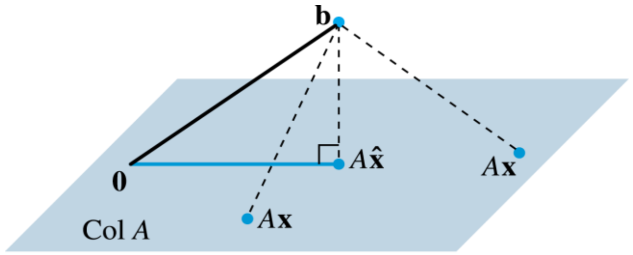
```
[[1.]
 [2.]]
```

Least Squares Error in R and Python

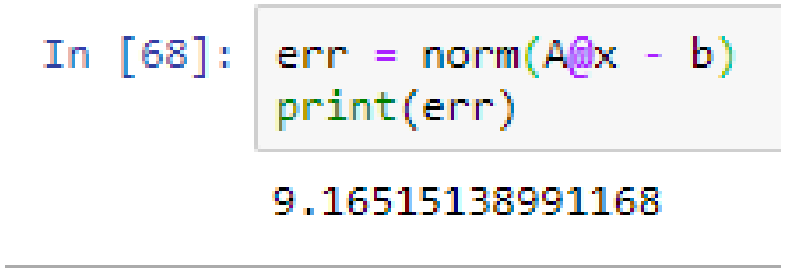
- Determine the least-squares error in previous example.



```
> err = norm(A%*%x - b, '2')
> print(err)
[1] 9.165151
```



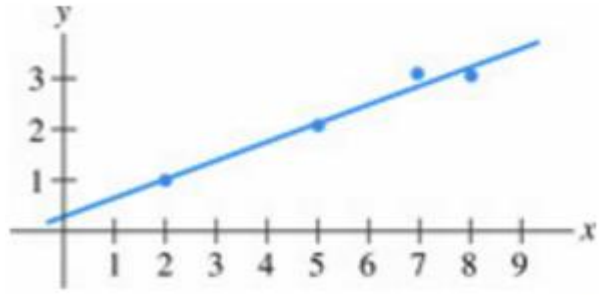
The vector $\mathbf{b}$ is closer to $A\hat{\mathbf{x}}$ than to $A\mathbf{x}$ for other $\mathbf{x}$.



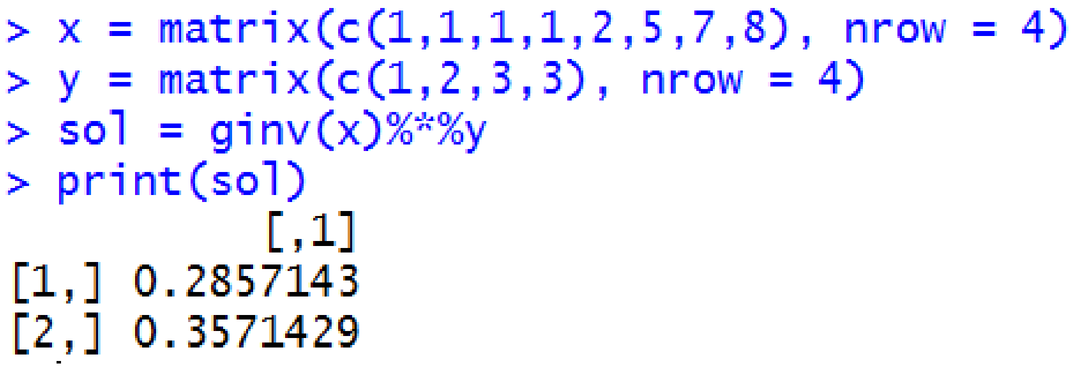
```
In [68]: err = norm(A@x - b)
         print(err)

         9.16515138991168
```

Example

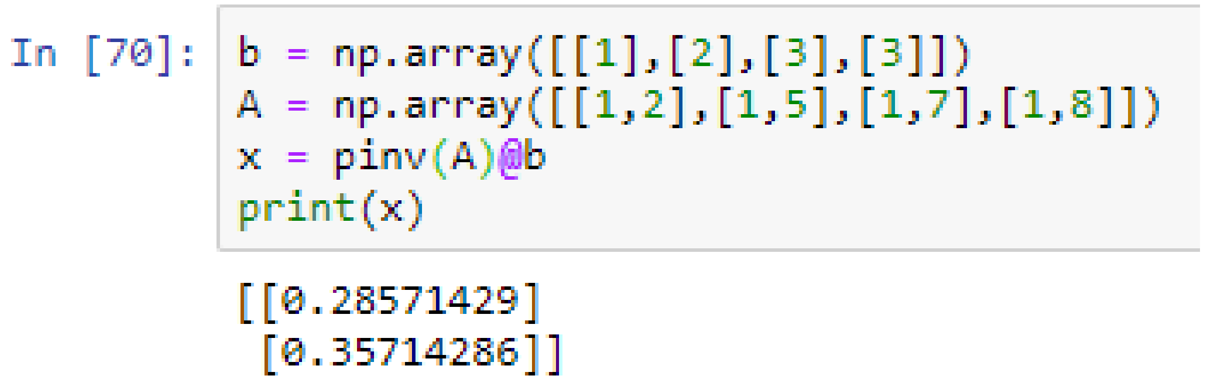
- Find the equation $y = \beta_0 + \beta_1 x$ of the least-squares line that best fits the data points (2 , 1) , (5 , 2) , (7 , 3) , and (8 , 3) .



Solution in R and Python



```
> x = matrix(c(1,1,1,1,2,5,7,8), nrow = 4)
> y = matrix(c(1,2,3,3), nrow = 4)
> sol = ginv(x)%*%y
> print(sol)
          [,1]
[1,] 0.2857143
[2,] 0.3571429
```



```
In [70]: b = np.array([[1],[2],[3],[3]])
         A = np.array([[1,2],[1,5],[1,7],[1,8]])
         x = pinv(A)@b
         print(x)

         [[0.28571429]
          [0.35714286]]
```

How to read a csv file into R

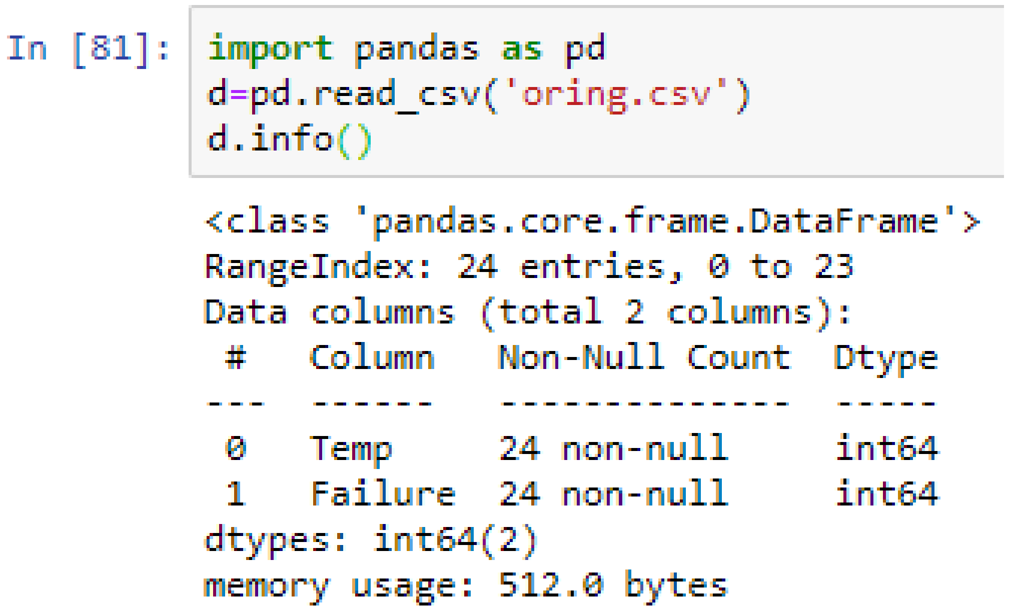
```
myData = read.csv("U://MyDocuments/DataAnalytics/Stat2/Assignments/hw4_logistics/Oring.csv")
str(myData)
```

```
## 'data.frame':    24 obs. of  2 variables:
##  $ Temp   : int  53 56 57 63 66 67 67 67 68 69 ...
##  $ Failure: int  1 1 1 0 0 0 0 0 0 0 ...
```

```
summary(myData)
```

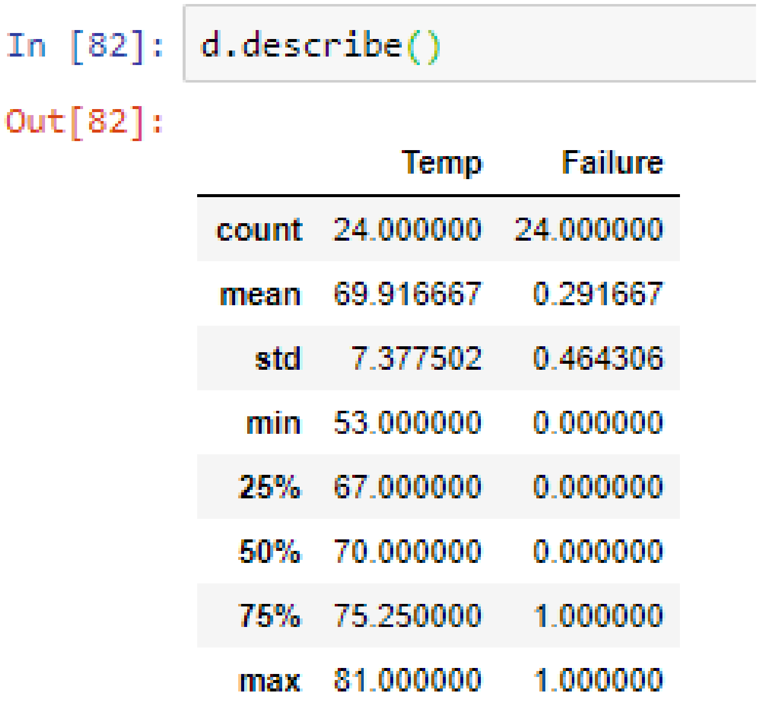
```
##       Temp          Failure      
##  Min.   :53.00   Min.   :0.0000  
##  1st Qu.:67.00   1st Qu.:0.0000  
##  Median :70.00   Median :0.0000  
##  Mean   :69.92   Mean   :0.2917  
##  3rd Qu.:75.25   3rd Qu.:1.0000  
##  Max.   :81.00   Max.   :1.0000  
```

How to read a csv file into Python



```
In [81]: import pandas as pd
         d=pd.read_csv('oring.csv')
         d.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24 entries, 0 to 23
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Temp     24 non-null     int64
 1   Failure  24 non-null     int64
dtypes: int64(2)
memory usage: 512.0 bytes
```



```
In [82]: d.describe()
```

Out[82]:

	Temp	Failure
count	24.000000	24.000000
mean	69.916667	0.291667
std	7.377502	0.464306
min	53.000000	0.000000
25%	67.000000	0.000000
50%	70.000000	0.000000
75%	75.250000	1.000000
max	81.000000	1.000000

Generating Random Numbers in R – rnorm and rep

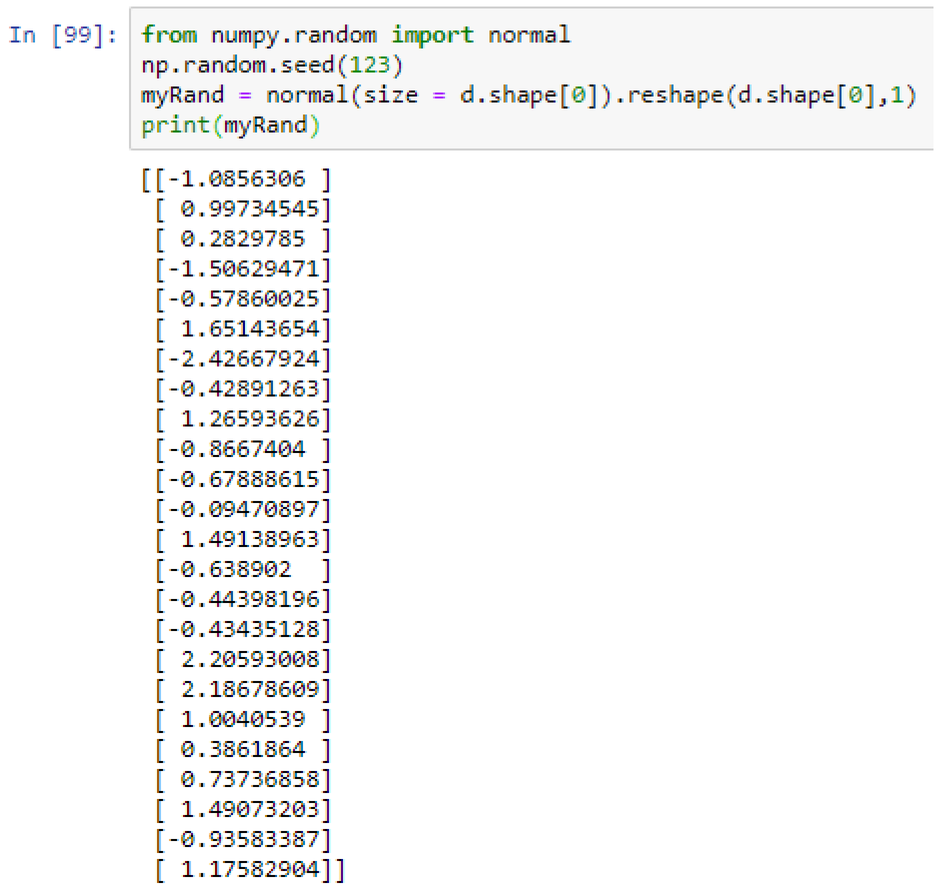
```
myRand = rnorm(n = nrow(myData))
print(myRand)
```

```
##  [1] -1.502733e+00 -6.416589e-01 -5.478076e-01 -4.825966e-01 -2.208273e-01
##  [6] -1.057852e+00 -8.274779e-06 -6.115095e-01 -2.574924e-01 -1.702548e-01
## [11] -2.066829e+00 -1.034635e+00 -3.624287e-01 -2.228764e+00  1.800802e+00
## [16]  1.933479e-01  5.434829e-01 -1.779400e-01 -1.375305e+00 -9.575774e-01
## [21] -3.319702e-01 -3.919302e-01 -4.068536e-01  4.128651e-01
```

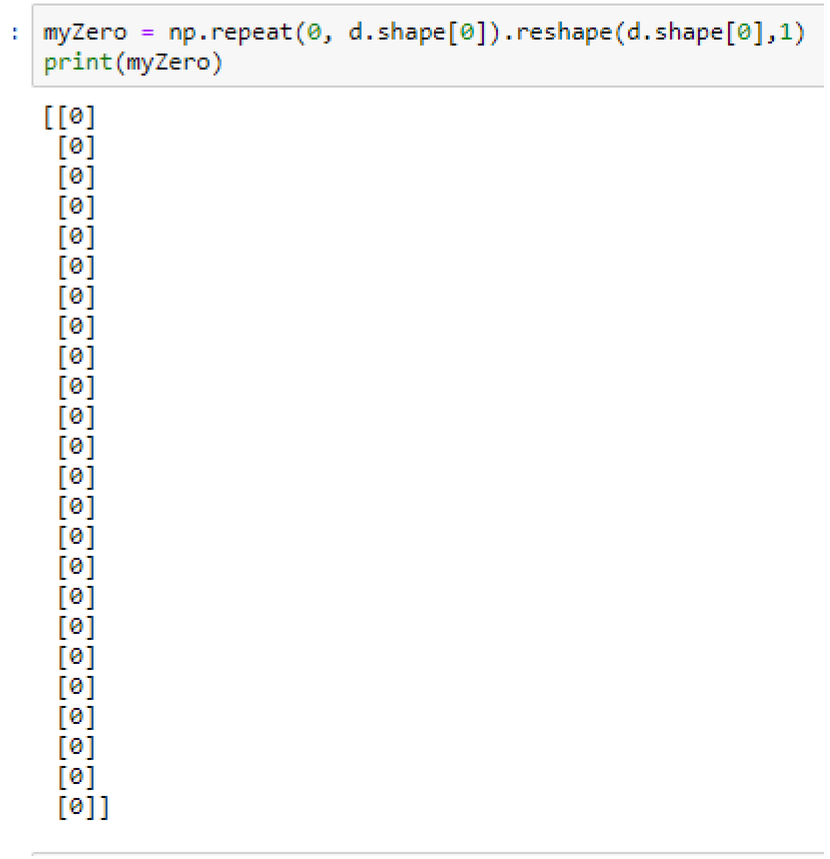
```
myZero = rep(0, nrow(myData))
print(myZero)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Generating Random Numbers in Python – normal and repeat



```
In [99]: from numpy.random import normal
         np.random.seed(123)
         myRand = normal(size = d.shape[0]).reshape(d.shape[0],1)
         print(myRand)

         [[-1.0856306 ]
          [ 0.99734545]
          [ 0.2829785 ]
          [-1.50629471]
          [-0.57860025]
          [ 1.65143654]
          [-2.42667924]
          [-0.42891263]
          [ 1.26593626]
          [-0.8667404 ]
          [-0.67888615]
          [-0.09470897]
          [ 1.49138963]
          [-0.638902  ]
          [-0.44398196]
          [-0.43435128]
          [ 2.20593008]
          [ 2.18678609]
          [ 1.0040539 ]
          [ 0.3861864 ]
          [ 0.73736858]
          [ 1.49073203]
          [-0.93583387]
          [ 1.17582904]]
```



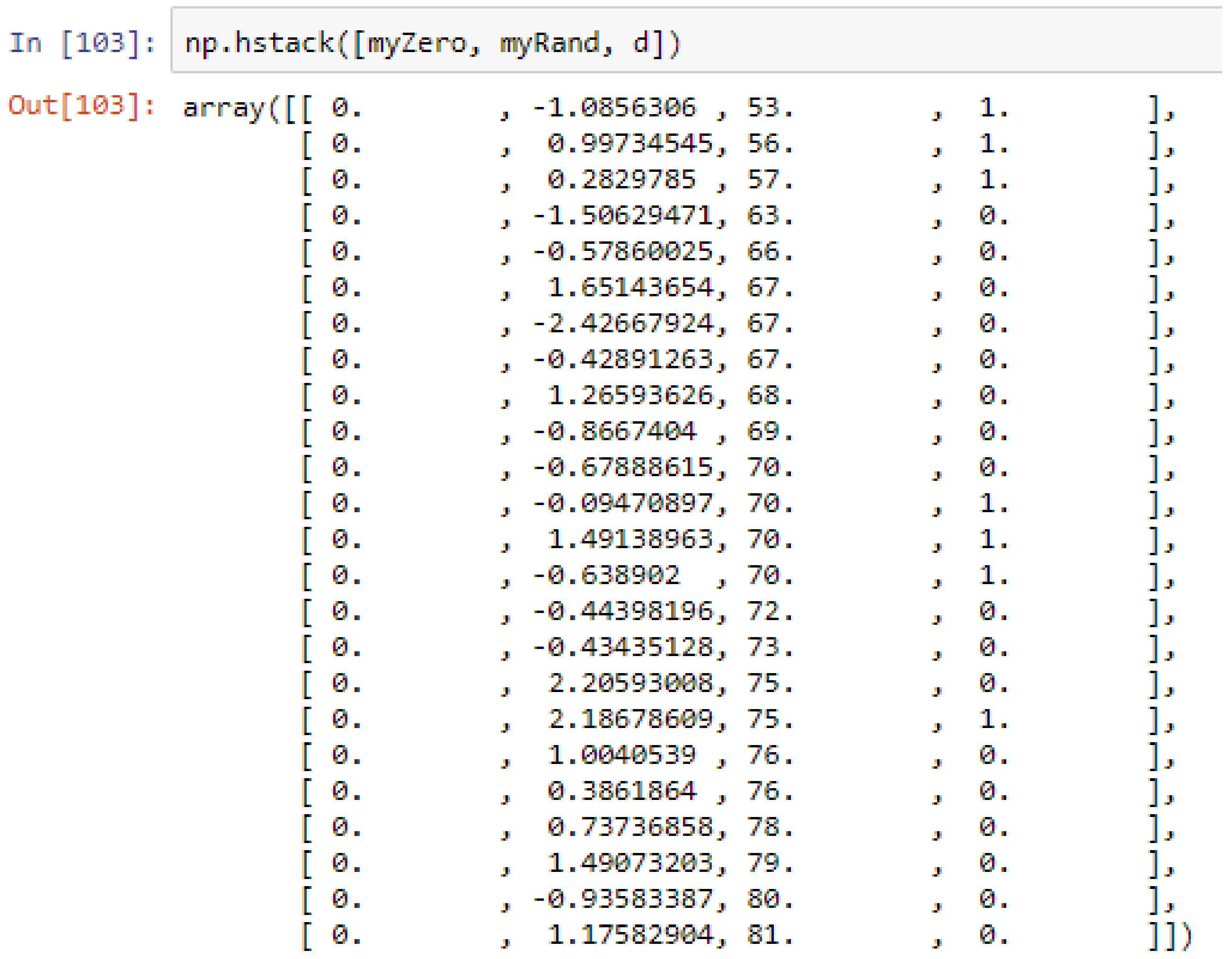
```
: myZero = np.repeat(0, d.shape[0]).reshape(d.shape[0],1)
  print(myZero)

  [[0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]
   [0]]
```

How to combine R arrays by columns (or rows)

```
myData = cbind(myZero, myRand, myData)
print(myData)
```

```
##    myZero        myRand Temp Failure
## 1       0 -1.502733e+00   53       1
## 2       0 -6.416589e-01   56       1
## 3       0 -5.478076e-01   57       1
## 4       0 -4.825966e-01   63       0
## 5       0 -2.208273e-01   66       0
## 6       0 -1.057852e+00   67       0
## 7       0 -8.274779e-06   67       0
## 8       0 -6.115095e-01   67       0
## 9       0 -2.574924e-01   68       0
## 10      0 -1.702548e-01   69       0
## 11      0 -2.066829e+00   70       0
## 12      0 -1.034635e+00   70       1
## 13      0 -3.624287e-01   70       1
## 14      0 -2.228764e+00   70       1
## 15      0  1.800802e+00   72       0
## 16      0  1.933479e-01   73       0
## 17      0  5.434829e-01   75       0
## 18      0 -1.779400e-01   75       1
## 19      0 -1.375305e+00   76       0
## 20      0 -9.575774e-01   76       0
## 21      0 -3.319702e-01   78       0
## 22      0 -3.919302e-01   79       0
## 23      0 -4.068536e-01   80       0
## 24      0  4.128651e-01   81       0
```

How to combine Python arrays by columns (or rows)



```
In [103]: np.hstack([myZero, myRand, d])

Out[103]: array([[ 0.        , -1.0856306 , 53.        ,  1.        ],
                 [ 0.        ,  0.99734545, 56.        ,  1.        ],
                 [ 0.        ,  0.2829785 , 57.        ,  1.        ],
                 [ 0.        , -1.50629471, 63.        ,  0.        ],
                 [ 0.        , -0.57860025, 66.        ,  0.        ],
                 [ 0.        ,  1.65143654, 67.        ,  0.        ],
                 [ 0.        , -2.42667924, 67.        ,  0.        ],
                 [ 0.        , -0.42891263, 67.        ,  0.        ],
                 [ 0.        ,  1.26593626, 68.        ,  0.        ],
                 [ 0.        , -0.8667404 , 69.        ,  0.        ],
                 [ 0.        , -0.67888615, 70.        ,  0.        ],
                 [ 0.        , -0.09470897, 70.        ,  1.        ],
                 [ 0.        ,  1.49138963, 70.        ,  1.        ],
                 [ 0.        , -0.638902  , 70.        ,  1.        ],
                 [ 0.        , -0.44398196, 72.        ,  0.        ],
                 [ 0.        , -0.43435128, 73.        ,  0.        ],
                 [ 0.        ,  2.20593008, 75.        ,  0.        ],
                 [ 0.        ,  2.18678609, 75.        ,  1.        ],
                 [ 0.        ,  1.0040539 , 76.        ,  0.        ],
                 [ 0.        ,  0.3861864 , 76.        ,  0.        ],
                 [ 0.        ,  0.73736858, 78.        ,  0.        ],
                 [ 0.        ,  1.49073203, 79.        ,  0.        ],
                 [ 0.        , -0.93583387, 80.        ,  0.        ],
                 [ 0.        ,  1.17582904, 81.        ,  0.        ]])
```

Reference

- Linear Algebra and Its Applications by David Lay