https://www.halvorsen.blog

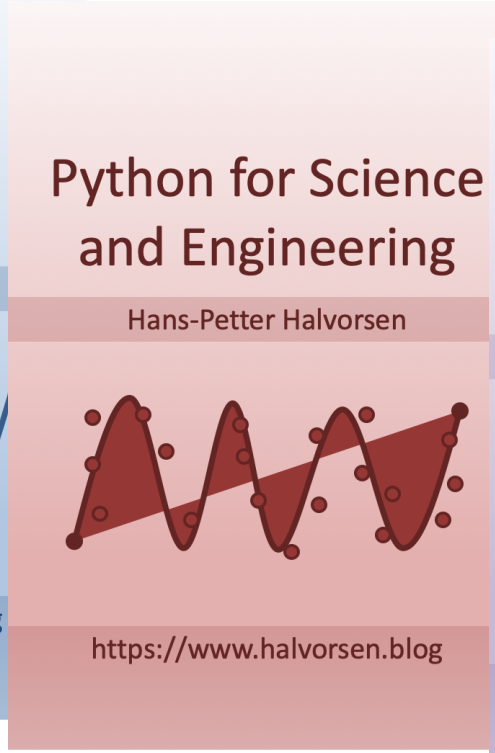
Linear Algebra in Python

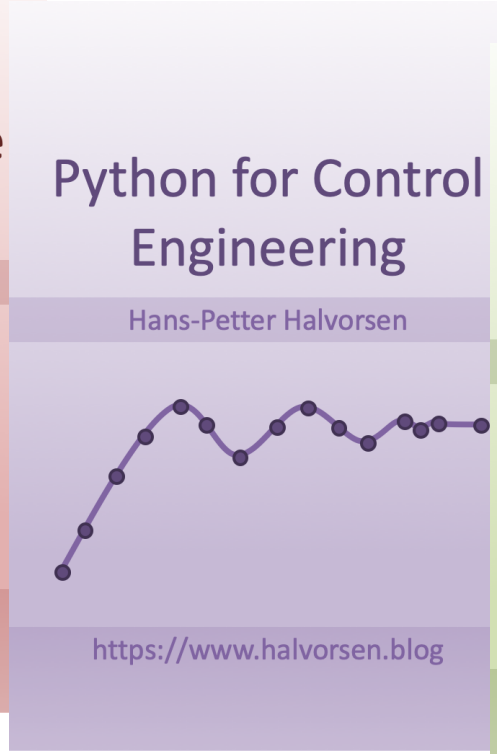
Hans-Petter Halvorsen

Free Textbook with lots of Practical Examples

https://www.halvorsen.blog/documents/programming/python/

Additional Python Resources





https://www.halvorsen.blog/documents/programming/python/

Contents

- Linear Algebra
- Matrices and Vectors
- The NumPy Library
- Solving Linear Equations
- Python Examples

Linear Algebra

- Linear Algebra is central and important in almost all areas of mathematics
- Linear Algebra is the “Mathematics of Data“
- Foundation: Vectors and Matrices
- Linear Equations
- Modern statistics and data analysis depends on Linear Algebra
- Linear Algebra plays an important role in advanced Control Engineering
- Linear Algebra plays an important role in Machine Learning

Matrices

A matrix is a two-dimensional data structure where numbers are arranged into rows and columns.

A general matrix is given by:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix}$$

Where n is number of rows and m is number of columns.

Matrices are very important data structures for many mathematical and scientific calculations.

Examples

Example of a 3 x 3 matrix:

$$A = \begin{bmatrix} 1 & 5 & 3 \\ 4 & 6 & 6 \\ 3 & 8 & 9 \end{bmatrix}$$

Example of a 4 x 2 matrix:

$$A = \begin{bmatrix} 1 & 5 \\ 4 & 5 \\ 3 & 2 \\ 7 & 8 \end{bmatrix}$$

Example of a 3 x 4 matrix:

$$A = \begin{bmatrix} 1 & 5 & 3 & 4 \\ 4 & 5 & 7 & 8 \\ 7 & 8 & 9 & 3 \end{bmatrix}$$

Vectors

A vector is a one-dimensional data structure where numbers are arranged into rows or columns.

A general vector is given by:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Where n is number of rows and m is number of columns.

Matrices and vectors are very important data structures for many mathematical and scientific calculations.

Vectors

Assume the vector x:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

The **Transpose** of vector x is

$$x^T = [x_1 \quad x_2 \quad \cdots \quad x_n]$$

The **Length** of vector x is given by:

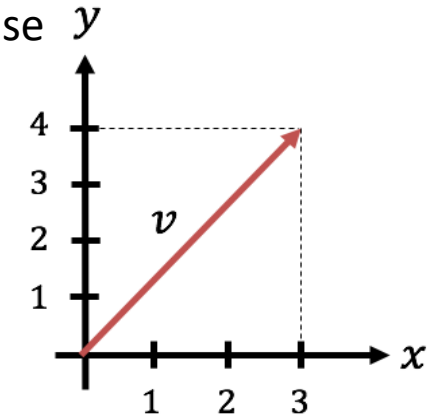
$$\|x\| = \sqrt{x^T x} = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

The length of a vector most makes sense for 2 or 3 dimensional vectors.

It can be visualized like this:

Example:

$$v = [3, 4]$$



In order to find the length of v we use Pythagoras like this:

$$|v| = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

Standard Python

Python doesn't have a built-in type for matrices. However, we can treat list of a list as a matrix.

Example of a 3 x 4 matrix:

$$A = \begin{bmatrix} 1 & 3 & 7 & 2 \\ 5 & 8 & -9 & 0 \\ 6 & -7 & 11 & 12 \end{bmatrix}$$

Example of vector:

$$a = \begin{bmatrix} 1 \\ 3 \\ 7 \\ 2 \end{bmatrix}$$

```
a = [1, 3, 7, 2]

print("a =", a)

A = [[1, 3, 7, 2],
     [5, 8, -9, 0],
     [6, -7, 11, 12]]

print("A =", A)
```

So we can define vectors and matrices with standard Python, but standard Python has no support for manipulation and calculation of them.

But fortunately we can use the **NumPy** package for creating matrices and for matrix manipulation.

Linear algebra (numpy.linalg)

- The **NumPy** library has a submodule for Linear Algebra, namely numpy.linalg

https://numpy.org/doc/stable/reference/routines.linalg.html

- The **SciPy** library also contains a linalg submodule, and there is overlap in the functionality provided by the SciPy and NumPy submodules.

https://docs.scipy.org/doc/scipy/reference/linalg.html#module-scipy.linalg

numpy.array

Let's use the **Array** feature in the **NumPy** Library:

Example of a 3 x 4 matrix:

$$A = \begin{bmatrix} 1 & 3 & 7 & 2 \\ 5 & 8 & -9 & 0 \\ 6 & -7 & 11 & 12 \end{bmatrix}$$

Example of vector:

$$a = \begin{bmatrix} 1 \\ 3 \\ 7 \\ 2 \end{bmatrix}$$

```
import numpy as np

a = np.array([[1],
              [3],
              [7],
              [2]])

print("a =", a)


A = np.array([[1, 3, 7, 2],
              [5, 8, -9, 0],
              [6, -7, 11, 12]])

print("A =", A)
```

numpy.matrix

Let's use the **Matrix** feature in the **NumPy** Library:

Example of a 3 x 4 matrix:

$$A = \begin{bmatrix} 1 & 3 & 7 & 2 \\ 5 & 8 & -9 & 0 \\ 6 & -7 & 11 & 12 \end{bmatrix}$$

Example of vector:

$$a = \begin{bmatrix} 1 \\ 3 \\ 7 \\ 2 \end{bmatrix}$$

```
import numpy as np

a = np.array([[1],
              [3],
              [7],
              [2]])

print("a =", a)


A = np.matrix([[1, 3, 7, 2],
               [5, 8, -9, 0],
               [6, -7, 11, 12]])

print("A =", A)
```

numpy.matrix

Note!
It is no longer recommended to use this class, even for linear algebra. Instead use regular arrays. The class may be removed in the future. See NumPy documentation for details.

More Information:

https://numpy.org/doc/stable/reference/generated/numpy.matrix.html

Matrix Addition, Subtraction, Multiplication

Given the following: $A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$ $B = \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix}$

Matrix **Addition**:

$$A + \mathrm{B} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} = \begin{bmatrix} 0+1 & 1+0 \\ -2+3 & -3+(-2) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -5 \end{bmatrix}$$

Matrix **Subtraction**:

$$A - \mathrm{B} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} = \begin{bmatrix} 0-1 & 1-0 \\ -2-3 & -3-(-2) \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ -5 & -1 \end{bmatrix}$$

Matrix **Multiplication**:

$$A\mathrm{B} = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} = = \begin{bmatrix} 0 \cdot 1 + 1 \cdot 3 & 0 \cdot 0 + 1 \cdot (-2) \\ -2 \cdot 1 - 3 \cdot 3 & -2 \cdot 0 - 3 \cdot (-2) \end{bmatrix} = \begin{bmatrix} 3 & -2 \\ -11 & 6 \end{bmatrix}$$

Matrix Addition, Subtraction, Multiplication

Matrix Addition, Subtraction, Multiplication

In this example we use **numpy.matrix**

Given the following matrices:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix}$$

Note!
It is no longer recommended to use this class, even for linear algebra. Instead use regular arrays. The class may be removed in the future. See NumPy documentation.

```
A+B =
[[ 1  1]
 [ 1 -5]]
A-B =
[[-1  1]
 [-5 -1]]
A*B =
[[3  -2]
 [-11 6]]
```

```
import numpy as np

A = np.matrix([[0, 1],
               [-2, -3]])

B = np.matrix([[1, 0],
               [3, -2]])

C = A + B
print("A+B =", C)

C = A - B
print("A-B =", C)

C = A * B
print("A*B =", C)
```

Cont.

Matrix Addition, Subtraction, Multiplication

In this example we use **numpy.array**

Given the following matrices:

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix}$$

```
A+B = [[ 1  1]
 [ 1 -5]]
A-B = [[-1  1]
 [-5 -1]]

A*B = [[ 0  0]
 [-6  6]]

A*B = [[  3  -2]
 [-11   6]]
A*B = [[  3  -2]
 [-11   6]]
A*B = [[  3  -2]
 [-11   6]]
A*B = [[  3  -2]
 [-11   6]]
```

```
import numpy as np

A = np.array([[0, 1],
              [-2, -3]])

B = np.array([[1, 0],
              [3, -2]])

C = A + B
print("A+B =", C)

C = A - B
print("A-B =", C)

C = A * B
print("A*B =", C) # Not Working!, only elementwise
Multiplication!

#Working Alternative 1
C = A.dot(B)
print("A*B =", C)

#Working Alternative 2
C = np.dot(A,B)
print("A*B =", C)

#Working Alternative 3
C =  np.mat(A) * np.mat(B)
print("A*B =", C)

#Working Alternative 4
C =  np.matmul(A,B)
print("A*B =", C)
```

Matrix Multiplication

- In matrix multiplication the matrices don't need to be quadratic, but the inner dimensions need to be the same.
- The size of the resulting matrix will be the outer dimensions.

$$[A] \times [B] = [C]$$

$$(n \times m) \qquad (m \times p) \qquad (n \times p)$$

Inner dimensions need to be the same

The resulting matrix will be the **outer** dimensions

Examples

$$A = \begin{bmatrix} 1 & 5 & 3 \\ 4 & 6 & 6 \\ 3 & 8 & 9 \end{bmatrix} \quad (3\times3)$$

$$B = \begin{bmatrix} 1 & 5 & 3 & 4 \\ 4 & 5 & 7 & 8 \\ 7 & 8 & 9 & 3 \end{bmatrix} \quad (3\times4)$$

$$C = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad (3\times1)$$

$$D = \begin{bmatrix} 1 & 5 & 3 \end{bmatrix} \quad (1\times3)$$

$A\times B = (3\times3)\times(3\times4) = (3\times4)$

$B\times A = (3\times4)\times(3\times3) =$ Not Legal

$A\times C = (3\times3)\times(3\times1) = (3\times1)$

$C\times A = (3\times1)\times(3\times3) =$ Not Legal

$C\times D = (3\times1)\times(1\times3) = (3\times3)$

$D\times C = (1\times3)\times(3\times1) = (1\times1)$ etc.

```
import numpy as np

A = np.array([[1, 5, 3],
              [4, 6, 6],
              [3, 8, 9]])

B = np.array([[1, 5, 3, 4],
              [4, 5, 7, 8],
              [7, 8, 9, 3]])

C = np.array([[1],
              [4],
              [3]])

D = np.array([[1, 5, 3]])

M =  np.matmul(A,B)
print("M=", M)

#M =  np.matmul(B,A) # Not Working!

M =  np.matmul(A,C)
print("M=", M)

#M =  np.matmul(C,A)  # Not Working!

M =  np.matmul(C,D)
print("M=", M)

M =  np.matmul(D,C)
print("M=", M)
```

Transpose of a Matrix

A general matrix is given by:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix}$$

Where n is number of rows and m is number of columns

$(n \times m)$

The transpose of matrix A is then:

$$A^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{n1} \\ a_{12} & a_{22} & \cdots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1m} & a_{2m} & \cdots & a_{nm} \end{bmatrix} \quad (m \times n)$$

Transpose of a Matrix - Examples

$$A = \begin{bmatrix} 1 & 3 & 7 & 2 \\ 5 & 8 & -9 & 0 \\ 6 & -7 & 11 & 12 \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} 1 & 5 & 6 \\ 3 & 8 & -7 \\ 7 & -9 & 11 \\ 2 & 0 & 12 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 5 \\ 4 & 5 \\ 3 & 2 \\ 7 & 8 \end{bmatrix} \Rightarrow B^T = \begin{bmatrix} 1 & 4 & 3 & 7 \\ 5 & 5 & 2 & 8 \end{bmatrix}$$

Transpose of a Matrix

$$A = \begin{bmatrix} 1 & 3 & 7 & 2 \\ 5 & 8 & -9 & 0 \\ 6 & -7 & 11 & 12 \end{bmatrix}$$

$A^T = ?$

$$B = \begin{bmatrix} 1 & 5 \\ 4 & 5 \\ 3 & 2 \\ 7 & 8 \end{bmatrix}$$

$B^T = ?$

```
Transpose of A=
[[ 1  5  6]
 [ 3  8 -7]
 [ 7 -9 11]
 [ 2  0 12]]
```

```
Transpose of B=
[[1 4 3 7]
 [5 5 2 8]]
```

```
import numpy as np

A = np.array([[1, 3, 7, 2],
              [5, 8, -9, 0],
              [6, -7, 11, 12]])
print("A="); print(A)

Atr = np.transpose(A)

print("Transpose of A="); print(Atr)

B = np.array([[1, 5],
              [4, 5],
              [3, 2],
              [7, 8]])
print("B="); print(B)

Btr = np.transpose(B)

print("Transpose of B="); print(Btr)
```

Determinant of a Matrix

- The Determinant of a matrix is a special number that can be calculated from square matrices

What is the Determinant used for?

- The determinant helps us find the inverse matrix (which we will cover later)
- The Determinant will give us useful information when dealing with Systems of Linear Equations (which we will cover later)
- Used in advanced Control Engineering theory
- Etc.

Determinant of a Matrix

Given a matrix A the Determinant is given by:

$$det(A) = |A|$$

For a $2x2$ matrix we have:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \Rightarrow \quad det(A) = |A| = a_{11}\, a_{22} - a_{21}\, a_{12}$$

Example:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \Rightarrow \quad det(A) = |A| = 1 \cdot 4 - 3 \cdot 2 = 4 - 6 = \underline{\underline{-2}}$$

Determinant of a Matrix

For a $3x3$ matrix we have:

We develop the determinant along a row or a column.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$\Rightarrow \quad det(A) = |A| = a_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{21} \begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} + a_{31} \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix}$$

Here we have developed the determinant along the first column

We see that the determinant of a higher order system can be expressed as a sum of lower order determinants

It will be a little more complicated and time-consuming work for systems with larger order, so we need a programming language like python to handle this.

Determinant of a Matrix

Example:

$$A = \begin{bmatrix} -1 & 3 & 0 \\ 2 & 1 & -5 \\ 1 & 4 & -2 \end{bmatrix} \quad \Rightarrow \quad det(A) = (-1)\begin{vmatrix} 1 & -5 \\ 4 & -2 \end{vmatrix} - 2\begin{vmatrix} 3 & 0 \\ 4 & -2 \end{vmatrix} + 1\begin{vmatrix} 3 & 0 \\ 1 & -5 \end{vmatrix}$$

This gives:

$$\begin{vmatrix} 1 & -5 \\ 4 & -2 \end{vmatrix} = -2 - (-20) = 18$$

$$\begin{vmatrix} 3 & 0 \\ 4 & -2 \end{vmatrix} = -6 - 0 = -6$$

$$\begin{vmatrix} 3 & 0 \\ 1 & -5 \end{vmatrix} = -15 - 0 = -15$$

Finally:

$$det(A) = -18 + 12 - 15 = \underline{\underline{-21}}$$

Determinant of a Matrix

Given the following Matrices:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$det(A) = -2$$

$$B = \begin{bmatrix} -1 & 3 & 0 \\ 2 & 1 & -5 \\ 1 & 4 & -2 \end{bmatrix}$$

$$det(B) = -21$$

Python Solution:

```
-2.0000000000000004
-21.000000000000001
```

```
import numpy as np
import numpy.linalg as la

A = np.array([[1, 2],
              [3, 4]])

Adet = la.det(A)

print(Adet)

B = np.array([[-1, 3, 0],
              [2, 1, -5],
              [1, 4, -2]])

Bdet = la.det(B)

print(Bdet)
```

Inverse Matrices

The **inverse** of a quadratic matrix A is defined by: A^{-1} Note: $AA^{-1} = A^{-1}A = I$

For a $2x2$ matrix we have:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

The inverse A^{-1} is then given by

$$A^{-1} = \frac{1}{det(A)} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}$$

Example:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \Rightarrow A^{-1} = \frac{1}{det(A)} \begin{bmatrix} 4 & -2 \\ -3 & 1 \end{bmatrix}$$

Where:

$$det(A) = |A| = 1 \cdot 4 - 3 \cdot 2 = 4 - 6 = -2$$

This gives: $A^{-1} = \frac{1}{-2} \begin{bmatrix} 4 & -2 \\ -3 & 1 \end{bmatrix} = \begin{bmatrix} -2 & 1 \\ 1.5 & -0.5 \end{bmatrix}$

It will be more complicated for systems with larger order, so we need a programming language like python to handle this.

Inverse Matrices

Given the following Matrices:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$B = \begin{bmatrix} -1 & 3 & 0 \\ 2 & 1 & -5 \\ 1 & 4 & -2 \end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} -2 & 1 \\ 1.5 & -0.5 \end{bmatrix}$$

$$B^{-1} = ?$$

Python Solution:

```
[[-2.   1. ]
 [ 1.5 -0.5]]

[[-0.85714286 -0.28571429  0.71428571]
 [ 0.04761905 -0.0952381   0.23809524]
 [-0.33333333 -0.33333333  0.33333333]]
```

```
import numpy as np
import numpy.linalg as la

A = np.array([[1, 2],
              [3, 4]])

Ainv = la.inv(A)

print(Adet)

B = np.array([[-1, 3, 0],
              [2, 1, -5],
              [1, 4, -2]])

Binv = la.inv(B)

print(Binv)
```

Eigenvalues

The Eigenvalues for a given matrix A is: $\det(\lambda I - A) = 0$

Example: $A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$

$$det(A) = |A| = a_{11}\, a_{22} - a_{21}\, a_{12}$$

$$\lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} = \begin{bmatrix} \lambda & -1 \\ 2 & \lambda + 3 \end{bmatrix}$$

$$\det(\lambda I - A) = \lambda(\lambda + 3) - (-1)(2) = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\Rightarrow \lambda^2 + 3\lambda + 2 = 0 \Rightarrow \lambda = \frac{-3 \pm \sqrt{3^2 - 4 \cdot 1 \cdot (2)}}{2 \cdot 1} = \frac{-3 \pm 1}{2} \Rightarrow \lambda_1 = -1, \lambda_2 = -2$$

In Python we use the function `eig()`. For more details:

https://numpy.org/doc/stable/reference/generated/numpy.linalg.eig.html

Eigenvalues

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}$$

$$eig(A) = \begin{bmatrix} -1 \\ -2 \end{bmatrix}$$

$$\lambda_1 = -1, \lambda_2 = -2$$

```
import numpy as np
import numpy.linalg as la

A = np.array([[0, 1],
              [-2, -3]])

Aeig, v = la.eig(A)

print(Aeig)
```

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

Matrices Rules

Some important Matrices Rules:

$$AB \neq BA$$
$$A(BC) = (AB)C$$
$$(A + B)C = AC + BC$$
$$C(A + B) = CA + CB$$
$$\det(AB) = \det(A)\det(B)$$
$$\det(A^T) = \det(A)$$
$$AA^{-1} = A^{-1}A = I$$

Matrices Rules

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 1 \\ 1 & -5 \end{bmatrix}$$

Some important Matrices Rules:

$$AB \neq BA$$
$$A(BC) = (AB)C$$
$$(A + B)C = AC + BC$$
$$C(A + B) = CA + CB$$
$$\det(AB) = \det(A)\det(B)$$
$$\det(A^T) = \det(A)$$
$$AA^{-1} = A^{-1}A = I$$

```
import numpy as np

A = np.array([[0, 1],
              [-2, -3]])

B = np.array([[1, 0],
              [3, -2]])

C = np.array([[1, 1],
              [1, -5]])

print("AB Not Equal BA")
L = np.matmul(A,B)
print("L=");print(L)

R = np.matmul(B,A)
print("R=");print(R)

print("A(BC) = (AB)C")
L = np.matmul(A,np.matmul(B,C))
print("L=");print(L)

R = np.matmul(np.matmul(A,B),C)
print("R=");print(R)
```

Matrices Rules

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 \\ 3 & -2 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 1 \\ 1 & -5 \end{bmatrix}$$

Some important Matrices Rules:

$$AB \neq BA$$
$$A(BC) = (AB)C$$
$$(A+B)C = AC + BC$$
$$C(A+B) = CA + CB$$
$$\det(AB) = \det(A)\det(B)$$
$$\det(A^T) = \det(A)$$
$$AA^{-1} = A^{-1}A = I$$

```
import numpy as np

A = np.array([[0, 1],
              [-2, -3]])

B = np.array([[1, 0],
              [3, -2]])

C = np.array([[1, 1],
              [1, -5]])

print("(A+B)C = AC + BC")
L = np.matmul(A+B,C)
print("L=");print(L)

R = np.matmul(A,C) + np.matmul(B,C)
print("R=");print(R)

print("AA^(-1) = A^(-1)A = I")
L = np.matmul(A, la.inv(A))
print("L=");print(L)

R = np.matmul(la.inv(A),A)
print("R=");print(R)
```

Solving Linear Equations

Hans-Petter Halvorsen

Linear Equations

Given the following linear equations:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots = b_1$$
$$a_{21}x_1 + a_{21}x_2 + a_{23}x_3 + \cdots = b_2$$
$$\cdots$$

These equations can be set on the following general form:

$$Ax = b$$

Where A is a matrix, x is a vector with the unknowns and b is a vector of constants

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Solution:

$$x = A^{-1}b$$

(assuming A^{-1} is possible)

Example

Given the following linear equations:

$$x_1 + 2x_2 = 5$$
$$3x_1 + 4x_2 = 6$$

We want to put the equations on the following general form:

$$Ax = b$$

This gives:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad b = \begin{bmatrix} 5 \\ 6 \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

The solution is given by:

$$x = A^{-1}b$$

Example - Python

Python code:

```
import numpy as np
import numpy.linalg as la

A = np.array([[1, 2],
              [3, 4]])

b = np.array([[5],
              [6]])

Ainv = la.inv(A)

x = Ainv.dot(b)

print(x)
```

This gives the following solution:

```
[[-4. ]
 [ 4.5]]
```

This means:

$$x_1 = -4$$
$$x_2 = 4.5$$

Which is the same as the solution we got from our manual calculations

Note! The A matrix must be square and of full-rank, i.e. the inverse matrix needs to exists.

Example – Python (Alt2)

We can also use the **`linalg.solve()`** function

```
x = np.linalg.solve(A, b)
```

Python code:

```
import numpy as np

A = np.array([[1, 2],
              [3, 4]])

b = np.array([[5],
              [6]])

x = np.linalg.solve(A, b)

print(x)
```

This gives the following solution:

```
[[-4. ]
 [ 4.5]]
```

This means:

$$x_1 = -4$$
$$x_2 = 4.5$$

Which is the same as the solutions we got from the other methods

Note! The A matrix must be square and of full-rank, i.e. the inverse matrix needs to exists.

Python Example – Least Square

Python code:

```
import numpy as np

A = np.array([[1, 2],
              [3, 4],
              [7, 8]])

b = np.array([[5],
              [6],
              [9]])

x = np.linalg.lstsq(A, b, rcond=None)[0]

print(x)
```

Note! In this example the A matrix is not quadratic, and the inverse of A does not exist! We need to use another method, e.g., the Least Square Method (LSM)

Given the following linear equations:

$$x_1 + 2x_2 = 5$$
$$3x_1 + 4x_2 = 6$$
$$7x_1 + 8x_2 = 9$$

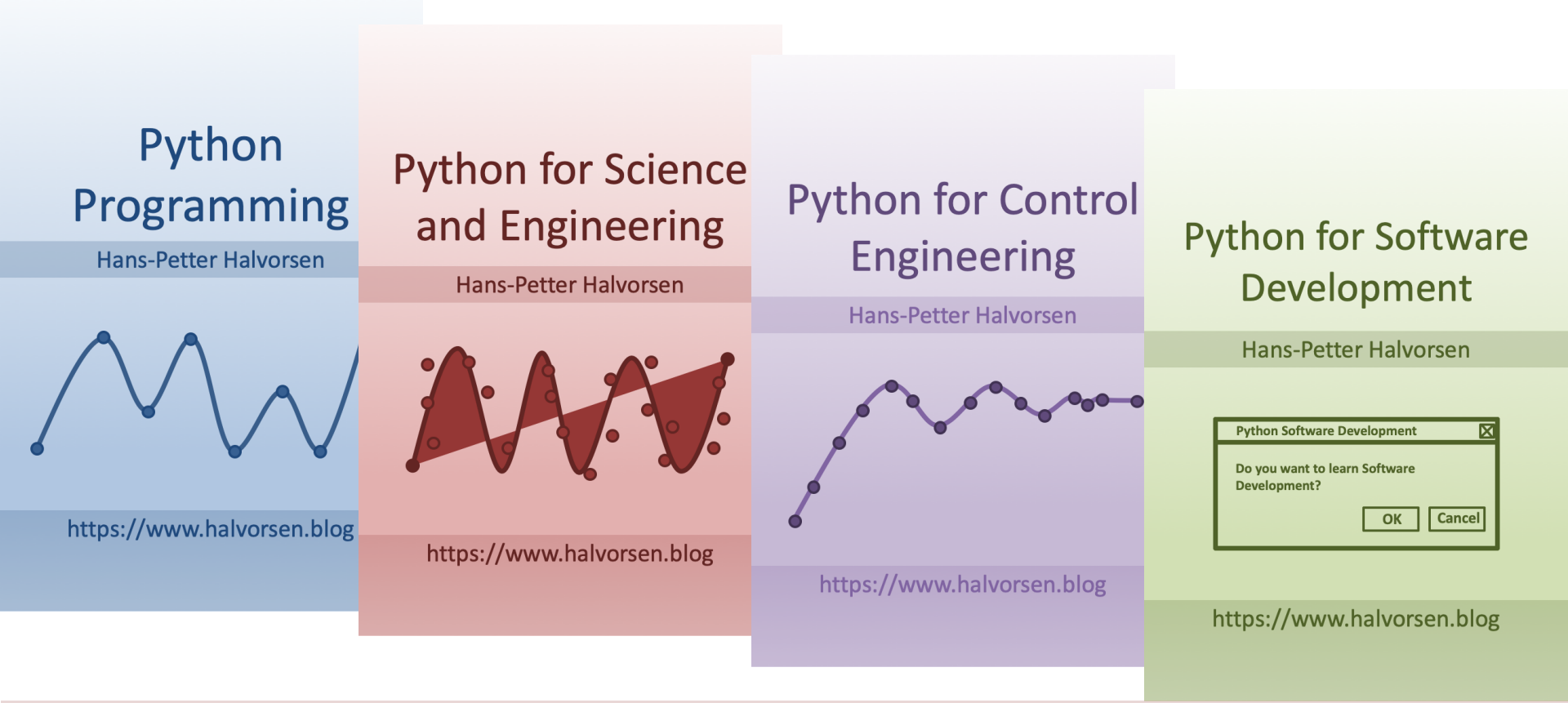
$$Ax = b$$

The results becomes:

```
[[-3.5       ]
 [ 4.17857143]]
```

$$x_1 = 3.5, x_2 = 4.2$$

Additional Python Resources



https://www.halvorsen.blog/documents/programming/python/

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog