https://www.halvorsen.blog

Solving Linear Equations with Python

Hans-Petter Halvorsen

Free Textbook with lots of Practical Examples

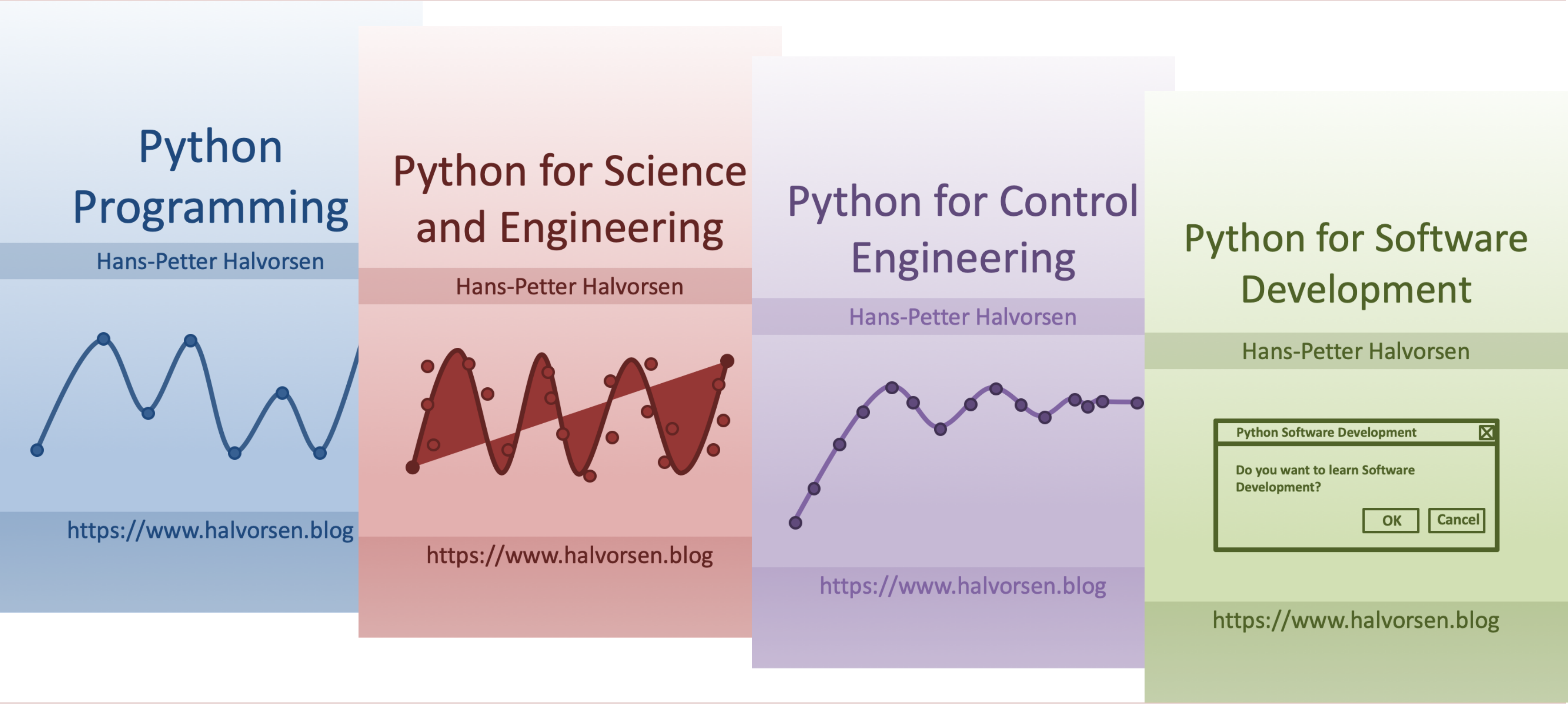
Python for Science and Engineering

Hans-Petter Halvorsen

https://www.halvorsen.blog

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

Additional Python Resources



Python Programming

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Science and Engineering

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Control Engineering

Hans-Petter Halvorsen

https://www.halvorsen.blog

Python for Software Development

Hans-Petter Halvorsen

Python Software Development

Do you want to learn Software Development?

OK Cancel

https://www.halvorsen.blog

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

Contents

Linear Equations in Python

- Python can be used to solve a large amount of linear equations using built-in functions
- Typically, you will use the **NumPy** library

NumPy

- The Python Standard Library consists basic Math functions, for more advanced Math functions, you typically want to use the NumPy Library
- If you don’t have Python yet and want the simplest way to get started, you can use the **Anaconda Distribution** - it includes Python, NumPy, and other commonly used packages for scientific computing and data science.
- Or use “pip install numpy“

https://numpy.org

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:

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

Let's solve these equations manually

From eq(1) we get:

$$x_1 = 5 - 2x_2$$

We put this in eq(2):

$$3(5 - 2x_2) + 4x_2 = 6$$

This gives:

$$15 - 6x_2 + 4x_2 = 6$$
$$15 - 2x_2 = 6$$
$$15 - 6 = 2x_2$$
$$2x_2 = 9$$
$$x_2 = \frac{9}{2} = 4.5$$

$$x_1 = 5 - 9 = -4$$

Final solution:

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

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.

Non-Quadratic Example

Given the following linear equations:

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

We have 3 equations and 2 unknows (x_1, x_2)

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

$$Ax = b$$

Note! The A matrix is not square, i.e. the inverse matrix does not to exists!

This gives:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 7 & 8 \end{bmatrix} \qquad b = \begin{bmatrix} 5 \\ 6 \\ 9 \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Non-Quadratic Example

Given the following linear equations:

$$
\begin{aligned}
x_1 + 2x_2 &= 5 \\
3x_1 + 4x_2 &= 6 \\
7x_1 + 8x_2 &= 9
\end{aligned}
$$

```
import numpy as np

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

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

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

print(x)
```

The Python code examples gives the following error:
LinAlgError: Last 2 dimensions of the array must be square

Non-Quadratic Example - Python

The previous Python code examples gives the following error:
LinAlgError: Last 2 dimensions of the array must be square

This is because the A matrix is not square, i.e. the inverse matrix does not to exists!

In many cases we cannot find the inverse matrix, e.g., when the matrix is not quadratic.
Finding the inverse matrix for large matrices is also time-consuming.

The numpy.linalg module has different functions that can handle this.

https://docs.scipy.org/doc/numpy/reference/routines.linalg.html

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)
```

The results becomes:

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

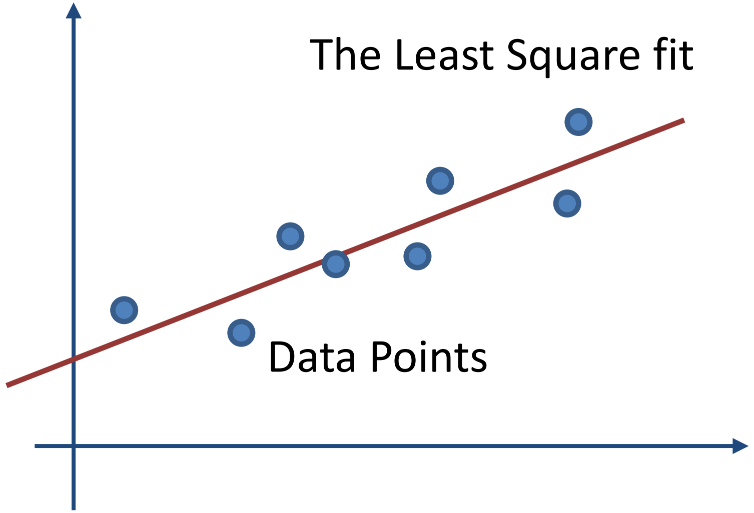
Least Square Method

Given:

$$Ax = b$$

The Least Square Method is given by:

$$x_{LS} = (A^T A)^{-1} A^T b$$



The Least Square Method works for Non-Quadratic matrices as well.

Least Square Method - Python

Implementing Least Square Method from scratch:

```
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)

x_ls = np.linalg.inv(A.transpose() * np.mat(A)) * A.transpose() * b
print(x_ls)
```

Compare built-in LSM and LMS from scratch

Comparing Different Methods

Given the following:

$$\begin{aligned} 4x + 3y + 2z &= 25 \\ -2x + 2y + 3z &= -10 \\ 3x - 5y + 2z &= -4 \end{aligned}$$

All 4 methods gives:

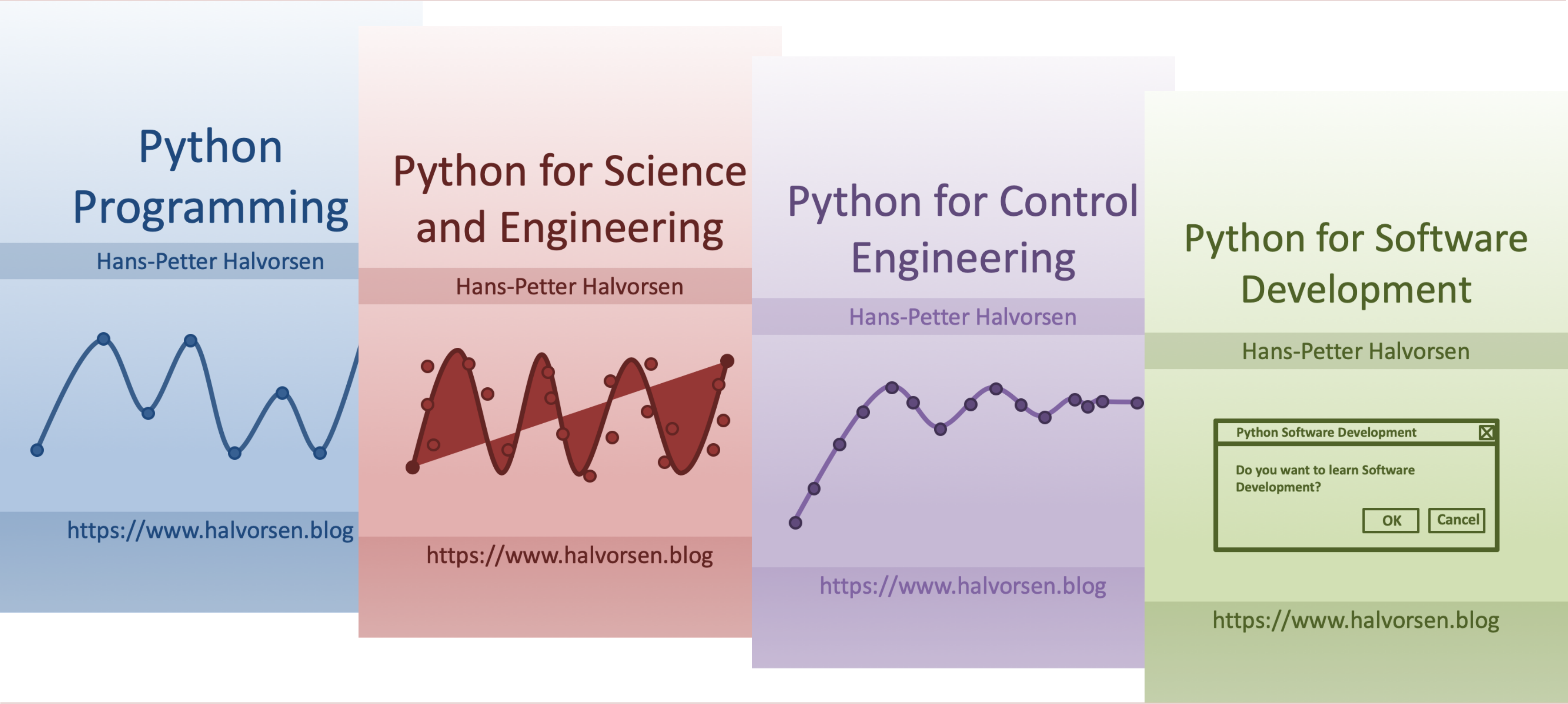
```
[[ 5.]
 [ 3.]
 [-2.]]
```

Meaning:

$x = 5, y = 3, z = -2$

```
import numpy as np

A = np.array([[4, 3, 2],
              [-2, 2, 3],
              [3, -5, 2]])

b = np.array([[25],
              [-10],
              [-4]])

x_alt1 = np.linalg.inv(A).dot(b)
print(x_alt1)

x_alt2 = np.linalg.solve(A, b)
print(x_alt2)

x_alt3 = np.linalg.lstsq(A, b, rcond=None)[0]
print(x_alt3)

x_alt4 = np.linalg.inv(A.transpose() * np.mat(A)) * A.transpose() * b
print(x_alt4)
```

Additional Python Resources



Python Programming
Hans-Petter Halvorsen
https://www.halvorsen.blog

Python for Science and Engineering
Hans-Petter Halvorsen
https://www.halvorsen.blog

Python for Control Engineering
Hans-Petter Halvorsen
https://www.halvorsen.blog

Python for Software Development
Hans-Petter Halvorsen
Python Software Development
Do you want to learn Software Development?
OK Cancel
https://www.halvorsen.blog

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