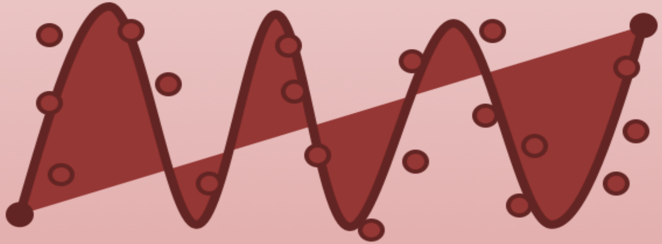
Curve Fitting in Python

Hans-Petter Halvorsen

Free Textbook with lots of Practical Examples

Python for Science and Engineering

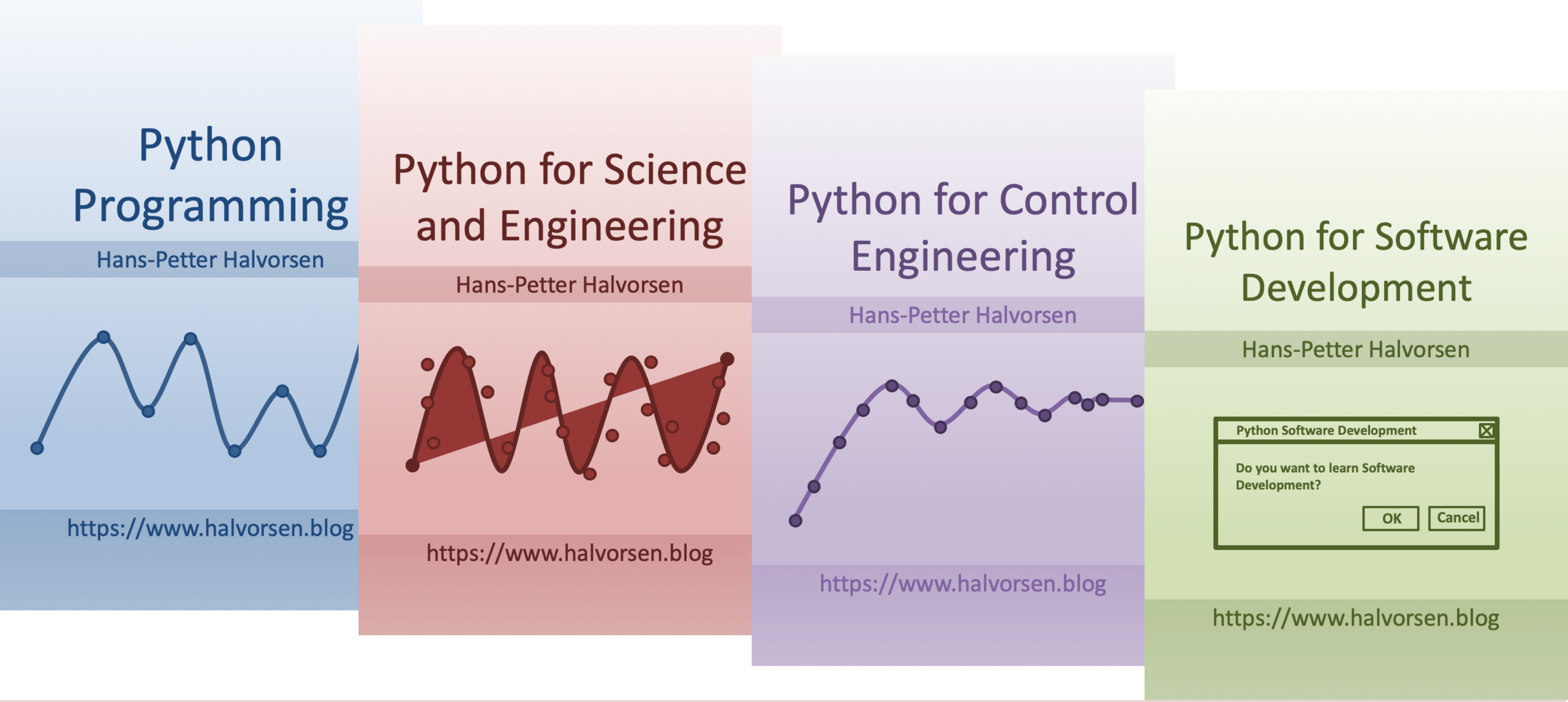
Hans-Petter Halvorsen



https://www.halvorsen.blog

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

Additional Python Resources



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

Contents

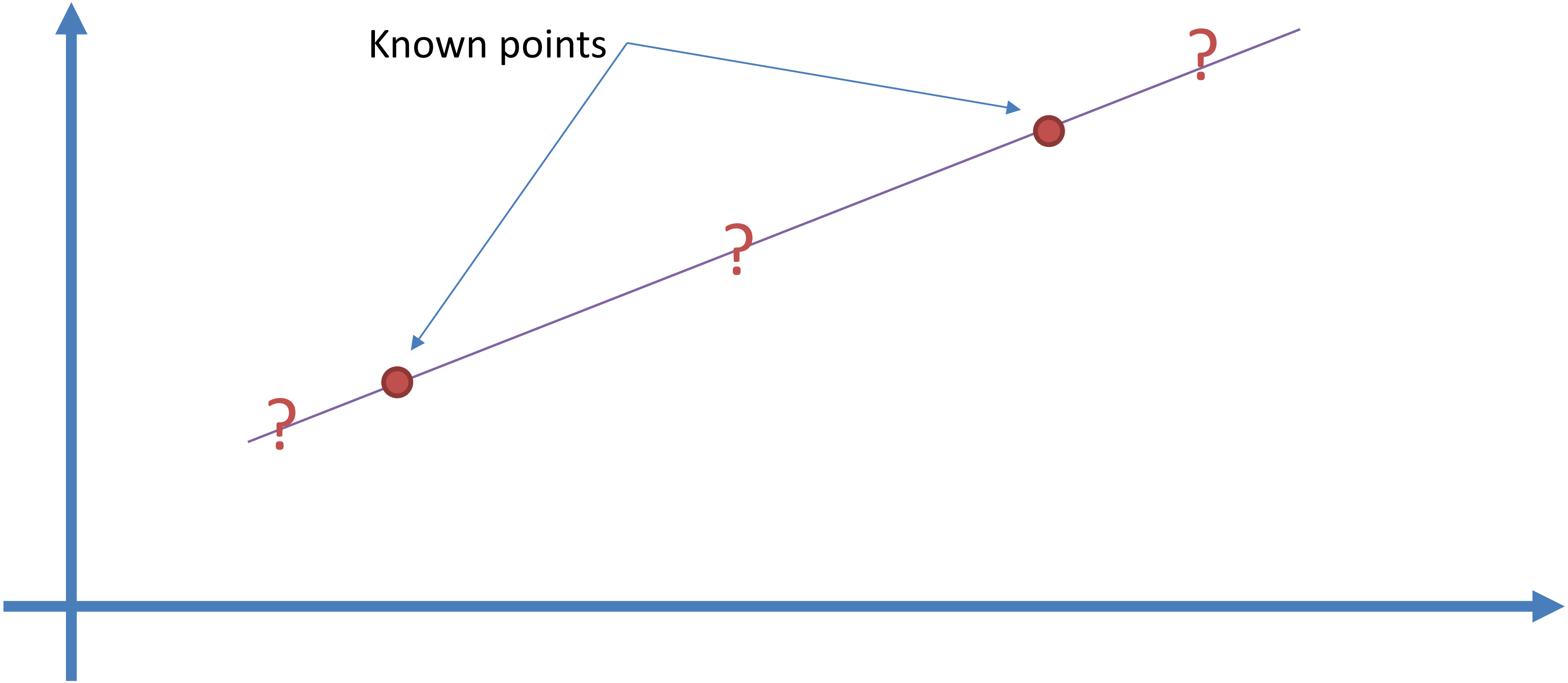
- Curve Fitting
- Linear Regression
- Polynomial Regression
- NumPy and SciPy
- Python Examples

Curve Fitting

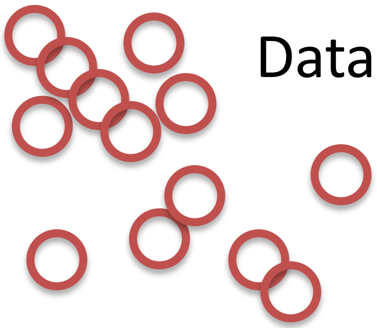
- In a previous example/video we found interpolated points, i.e., we found values between the measured points using the **interpolation** technique.
- It would be more convenient to model the data as mathematical function $y = f(x)$.
- Then we could easily calculate any data we want based on this model.

Interpolation

Interpolation is used to estimate data points between two known points



Curve Fitting



Mathematical Model

$$y = f(x)$$

Curve Fitting is all about fitting data to a Mathematical Model

Curve Fitting in Python

- Python has curve fitting functions that allows us to create empiric data model.
- It is important to have in mind that these models are good only in the region we have collected data.
- Here are some of the functions available in Python used for curve fitting:
  - `polyfit()`, `polyval()`, `curve_fit()`, …
- Some of these techniques use a polynomial of degree N that fits the data Y best in a least-squares sense.

SciPy

- SciPy is a free and open-source Python library used for scientific computing and engineering
- SciPy contains modules for optimization, linear algebra, interpolation, image processing, ODE solvers, etc.
- SciPy is included in the Anaconda distribution

Polynomials

A polynomial is expressed as:

$$p(x) = p_1 x^n + p_2 x^{n-1} + \cdots + p_n x + \ p_{n+1}$$

where $p_1, p_2, p_3, \ldots$ are the coefficients of the polynomial.

We have **Linear Regression** and **Polynomial Regression**

Polynomials in Python

Given the following polynomial:

$$p(x) = -5.45x^4 + 3.2x^2 + 8x + 5.6$$

We need to rewrite it like this in Python:

$$p(x) = 5.6 + 8x + 3.2x^2 + 0x^3 - 5.45x^4$$

```
import numpy.polynomial.polynomial as poly

p = [5.6, 8, 3.2, 0, -5.45]

r = poly.polyroots(p)
print(r)
```

$p(x) = 0 \rightarrow x = ?$

Linear Regression

Linear Regression

- Linear Regression is a special case of Polynomial Regression
- Linear Regression is a 1.order Polynomial ($n = 1$)

$$p(x) = p_1x + p_2$$

Or:

$$y(x) = ax + b$$

Linear Regression - Example

Assume the Data:

x	y
0	15
1	10
2	9
3	6
4	2
5	0

```
from scipy.optimize import curve_fit

def linear_model(x, a, b):
    return a * x + b

x = [0, 1, 2, 3, 4, 5]
y = [15, 10, 9, 6, 2, 0]

popt, pcov = curve_fit(linear_model, x, y)

print(popt)
```

We want to find a linear model $y(x) = ax + b$ that fits the data points

Linear Regression - Example

Assume the Data:

x	y
0	15
1	10
2	9
3	6
4	2
5	0

From the Python code we get the following results:

```
[-2.91428571 14.28571429]
```

This means $a \approx -2.91$ and $b \approx 14.29$

Or:

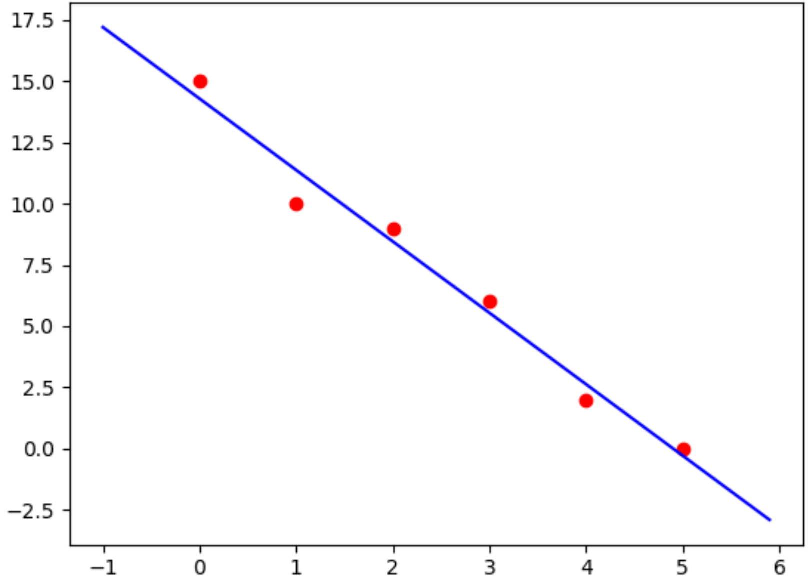
$$y = -2.91x + 14.29$$

The `curve_fit()` function returns two items, which we call popt and pcov. The popt argument are the best-fit parameters (p optimal) for a and b. The pcov variable contains the covariance matrix, which indicates the uncertainties and correlations between parameters.

Example - Improved

Next, it is also a good idea to plot the actual data in the same plot as the model for comparison.

We extend the code as follows:




```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

def linear_model(x, a, b):
    return a * x + b

x = [0, 1, 2, 3, 4, 5]
y = [15, 10, 9, 6, 2, 0]

popt, pcov = curve_fit(linear_model, x, y)

print(popt)

plt.plot(x,y, 'or')

xstart = -1
xstop = 6
increment = 0.1
xmodel = np.arange(xstart,xstop,increment)

a = popt[0]
b = popt[1]

ymodel = a*xmodel + b

plt.plot(xmodel,ymodel, 'b')
```

Polynomial Regression

Hans-Petter Halvorsen

Polynomial Regression

- In the previous section we used linear regression which is a 1. order polynomial.
- In this section we will study higher order polynomials.
- In polynomial regression we will find the following model:

$$y(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n$$

Polynomial Regression - Example

Given the following Data:

x	y
0	15
1	10
2	9
3	6
4	2
5	0

We will use the Python to find and compare the models using different orders of the polynomial.

We will investigate models of 2.order, 3.order, 4.order and 5.order.

We have only 6 data points, so a model with order higher than 5 will make no sense.

Typically we have much more data, but this is just an example to demonstrate the principle of curve fitting.

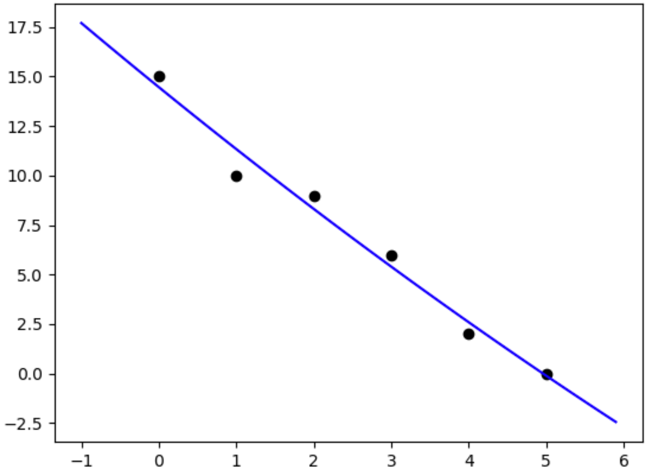
We want to find models on the form:

$$y(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_{n-1} x + a_n$$

Polynomial Regression - Example

We start with a 2.order model:

$$y(x) = ax^2 + bx + c$$



[0.05357143 -3.18214286 14.46428571]

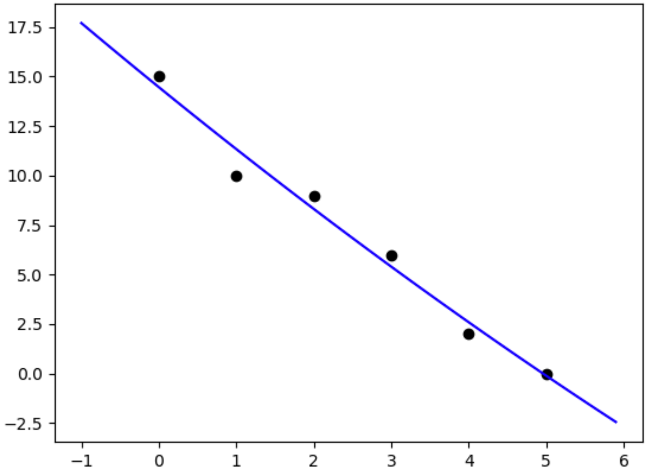
$$y(x) = 0.05x^2 - 3.18x + 14.46$$

```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4, 5]
y = [15, 10, 9, 6, 2, 0]

def model(x, a, b, c):
    y = a * x ** 2 + b * x + c
    return y

popt, pcov = curve_fit(model, x, y)
print(popt)

plt.plot(x,y, 'ok')

xstart = -1
xstop = 6
increment = 0.1
xmodel = np.arange(xstart,xstop,increment)

a = popt[0]
b = popt[1]
c = popt[2]

ymodel = model(xmodel, a, b, c)

plt.plot(xmodel,ymodel, 'b')
```

Example – Improved Solution

We start with a 2.order model:

$$y(x) = ax^2 + bx + c$$



[0.05357143 -3.18214286 14.46428571]

$$y(x) = 0.05x^2 - 3.18x + 14.46$$

```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4, 5]
y = [15, 10, 9, 6, 2, 0]

def model(x, a, b, c):
    y = a * x ** 2 + b * x + c
    return y

popt, pcov = curve_fit(model, x, y)

print(popt)

plt.plot(x,y, 'ok')

xstart = -1
xstop = 6
increment = 0.1
xmodel = np.arange(xstart,xstop,increment)

ymodel = model(xmodel, *popt)

plt.plot(xmodel,ymodel, 'b')
```

Example cont.

1.order model:

$$y(x) = ax + b$$

2.order model:

$$y(x) = ax^2 + bx + c$$

3.order model:

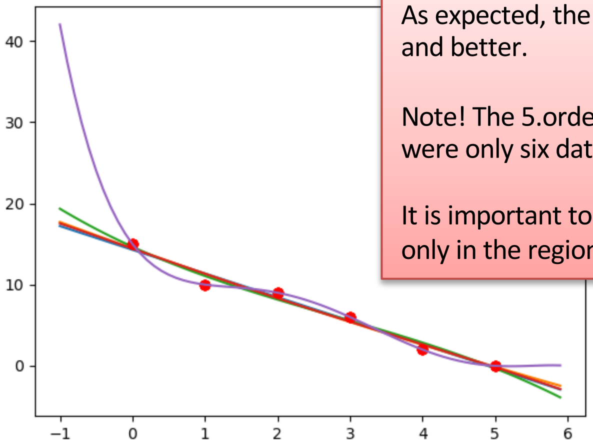
$$y(x) = ax^3 + bx^2 + cx + d$$

4.order model:

$$y(x) = ax^4 + bx^3 + cx^2 + dx + e$$

5.order model:

$$y(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$$



As expected, the higher order models match the data better and better.

Note! The 5.order model matches exactly because there were only six data points available.

It is important to have in mind that these models are good only in the region we have collected data.

```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4, 5]
y = [15, 10, 9, 6, 2, 0]

def model1(x, a, b):
    y = a * x + b
    return y

def model2(x, a, b, c):
    y = a * x ** 2 + b * x + c
    return y

def model3(x, a, b, c, d):
    y = a * x**3 + b * x**2 + c * x + d
    return y

def model4(x, a, b, c, d, e):
    y = a * x**4 + b * x**3 + c * x**3 + d * x + e
    return y

def model5(x, a, b, c, d, e, f):
    y = a * x**5 + b * x**4 + c * x**3 + d * x**2 + e * x + f
    return y

popt, pcov = curve_fit(model5, x, y)

print(popt)

plt.plot(x,y, 'or')

xstart = -1
xstop = 6
increment = 0.1
xmodel = np.arange(xstart,xstop,increment)

#ymodel = model1(xmodel, *popt)
#ymodel = model2(xmodel, *popt)
#ymodel = model3(xmodel, *popt)
#ymodel = model4(xmodel, *popt)
ymodel = model5(xmodel, *popt)

plt.plot(xmodel,ymodel, 'b')
```

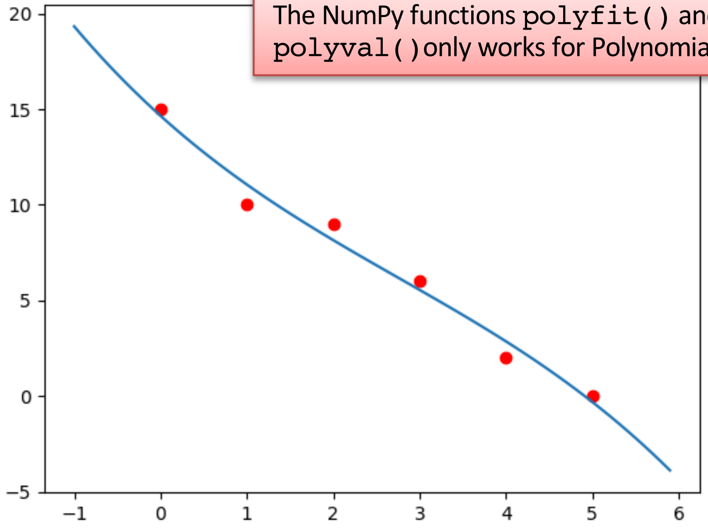
polyfit() and polyval()

polyfit() and polyval()

In this example we will use the NumPy functions `polyfit()` and `polyval()`.

We start with a 3.order model:

$$y(x) = ax^3 + bx^2 + cx + d$$



The NumPy functions `polyfit()` and `polyval()` only works for Polynomials

```python
import numpy as np
import matplotlib.pyplot as plt

# Original Data
x = [0, 1, 2, 3, 4, 5]
y = [15, 10, 9, 6, 2, 0]
plt.plot(x,y, 'or')

# Set Model order
model_order = 3

# Find Model
p = np.polyfit(x, y, model_order)
print(p)

# Plot the Model
xstart = -1
xstop = 6
increment = 0.1
xmodeldata = np.arange(xstart,xstop,increment)

ymodel = np.polyval(p, xmodeldata)
plt.plot(xmodeldata,ymodel)
```

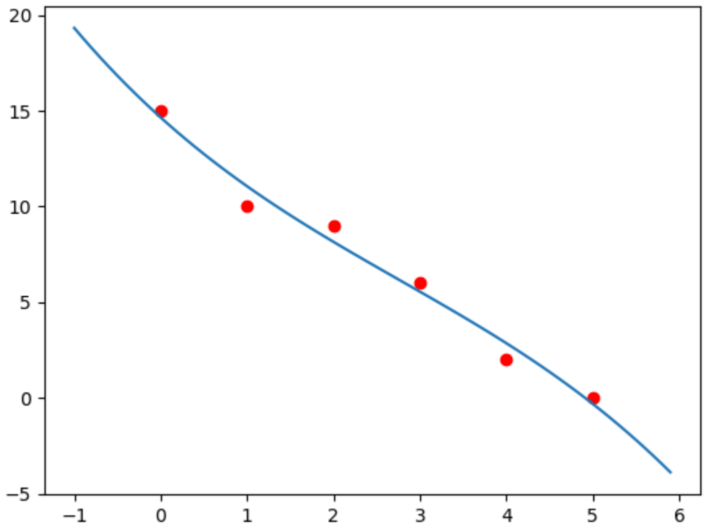
polyfit() and polyval()

In this example we will use the NumPy functions `polyfit()` and `polyval()`.

We start with a 3.order model:

$$y(x) = ax^3 + bx^2 + cx + d$$

x	y
0	15
1	10
2	9
3	6
4	2
5	0



We get the following results:

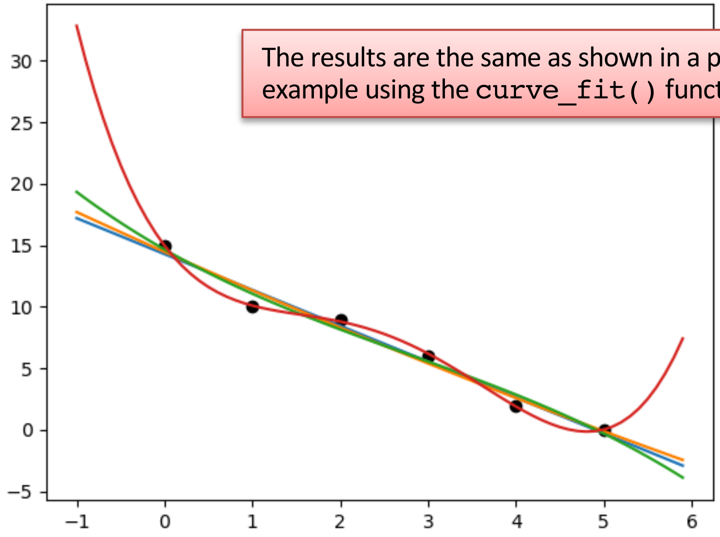
```
[-0.06481481  0.53968254 -4.07010582 14.65873016]
```

This means the following 3.order model:

$$y(x) = -0.06x^3 + 0.54x^2 - 4.1x + 14.7$$

Example modified

Let's extend the code by creating different models with different orders. For easy comparison of different models in the same program we can use a **For loop** as shown in the code example.



The results are the same as shown in a previous example using the `curve_fit()` function

```
import numpy as np
import matplotlib.pyplot as plt

# Original Data
x = [0, 1, 2, 3, 4, 5]
y = [15, 10, 9, 6, 2, 0]

plt.plot(x,y, 'ok')

# x values for model
xstart = -1
xstop = 6
increment = 0.1
xmodel = np.arange(xstart,xstop,increment)


startorder = 1
endorder = 5

for model_order in range(startorder, endorder, 1):

    # Finding the Model
    p = np.polyfit(x, y, model_order)

    print(p)

    # Plot the Model
    ymodel = np.polyval(p, xmodel)

    plt.plot(xmodel,ymodel)
```

Other Examples

Curve Fitting

We have now used the `curve_fit()` function for finding a linear model ($y = ax + b$) and find Polynomial models of different orders ($y(x) = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n$)
But we can adjust a given data set to all kinds of models that we specify in our Python function

```
..

def model(x, ..):
    y = ..
    return y

x = [..]
y = [..]

popt, pcov = curve_fit(model, x, y)
```

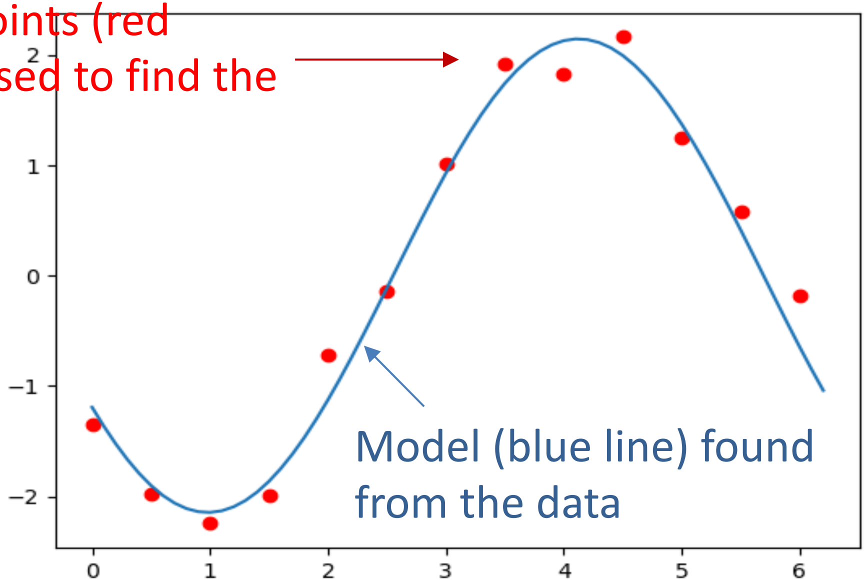
Assume we want to fit some data to a sin() function, a logarithmic function, an exponential function, etc.

Curve Fitting

Assume we want to fit some given data to the following model:

$$y(x) = \mathrm{a} \cdot \sin(x + b)$$

Data Points (red dots) used to find the Model



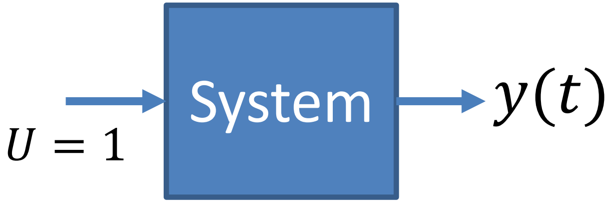
Model (blue line) found from the data

```python
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

start = 0
stop = 2*np.pi
increment = 0.5
x = np.arange(start,stop,increment)

a = 2
b = 10
np.random.seed()
y_noise = 0.2 * np.random.normal(size=x.size)
y = a * np.sin(x + b)
y = y + y_noise

plt.plot(x,y, 'or')


def model(x, a, b):
    y = a * np.sin(x + b)
    return y

popt, pcov = curve_fit(model, x, y)

increment = 0.1
xmodeldata = np.arange(start,stop,increment)

ymodel = model(xmodeldata, *popt)

plt.plot(xmodeldata,ymodel)
```

Dynamic System

Dynamic System - Example

We have a set of logged data which We have logged data from a “real system“.



We want to fit the data to the following model:

$$y(t) = KU(1 - e^{-\frac{t}{T}})$$

t	$y(t)$
0	…
1	…
2	…
3	…
…	…
…	…

Where K and T are Model Parameters we need to find

The equation above is actually the solution for the differential equation given below:

$$\dot{y} = \frac{1}{T}(-x + Ku)$$

We apply a step (u=U=1) in the input signal and log the output signal

Python Code

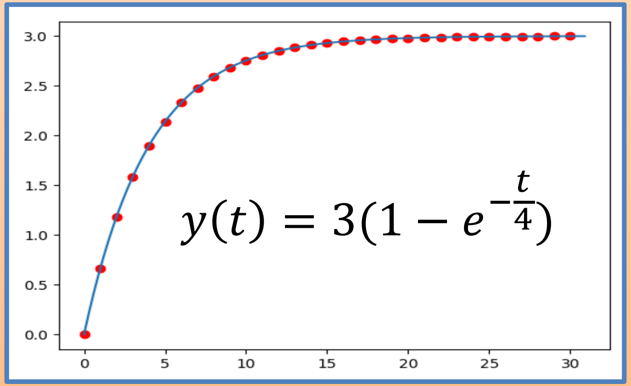
```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

t = [0, 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
   16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]
y = [0, 0.66, 1.18, 1.58, 1.89, 2.14, 2.33, 2.47, 2.59, 2.68, 2.75, 2.80, 2.85, 2.88, 2.90, 2.92,
   2.94, 2.95, 2.96667301, 2.97, 2.97, 2.98, 2.98, 2.99, 2.99, 2.99, 2.99, 2.99, 2.99, 2.99, 2.99]

def model(t, K, T):
    y = K * (1-np.exp(-t/T))
    return y

popt, pcov = curve_fit(model, t, y)
print(popt)
plt.plot(x,y, 'or')

start = 0
stop = 31
increment = 0.1
xmodeldata = np.arange(start,stop,increment)
ymodel = model(xmodeldata, *popt)
plt.plot(xmodeldata, ymodel)
```

$$y(t) = K(1 - e^{-\frac{t}{T}})$$

The Python code gives the following results:

```
[3.  4.]
```

This means $K = 3$ and $T = 4$



Simulated Data

In the example I have simulated a 1. order dynamic system

$$u(t) \longrightarrow \boxed{H(s) = \frac{K}{Ts+1}} \longrightarrow y(t)$$

Where K is the Gain and T is the Time constant

Differential Equation:

$$\dot{y} = \frac{1}{T}(-y + Ku)$$

In the time domain we get the following solution (using Inverse Laplace):

$$y(t) = KU(1 - e^{-\frac{t}{T}})$$

```
import matplotlib.pyplot as plt
import control

s = control.TransferFunction.s

K = 3
T = 4
H  = K/(T*s + 1)
print ('H(s) =', H)

start = 0
stop = 31
increment = 1
t = np.arange(start,stop,increment)

t, y = control.step_response(H, t)

plt.plot(t,y)

print(t)
print(y)
```

Least Square Method

Hans-Petter Halvorsen

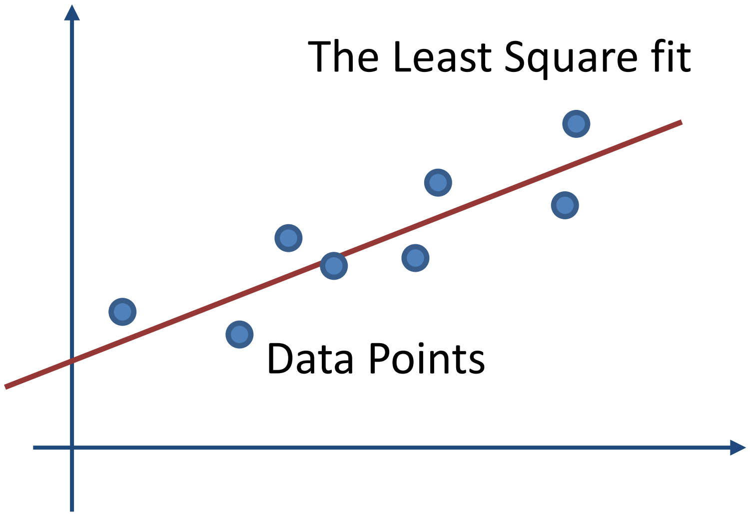
Least Square Method (LSM)

The least squares method requires the model to be set up in the following form based on input-output data :

$$Y = \Phi\theta$$

The Least Square Method is given by:

$$\theta_{LS} = (\Phi^T\Phi)^{-1}\Phi^T Y$$



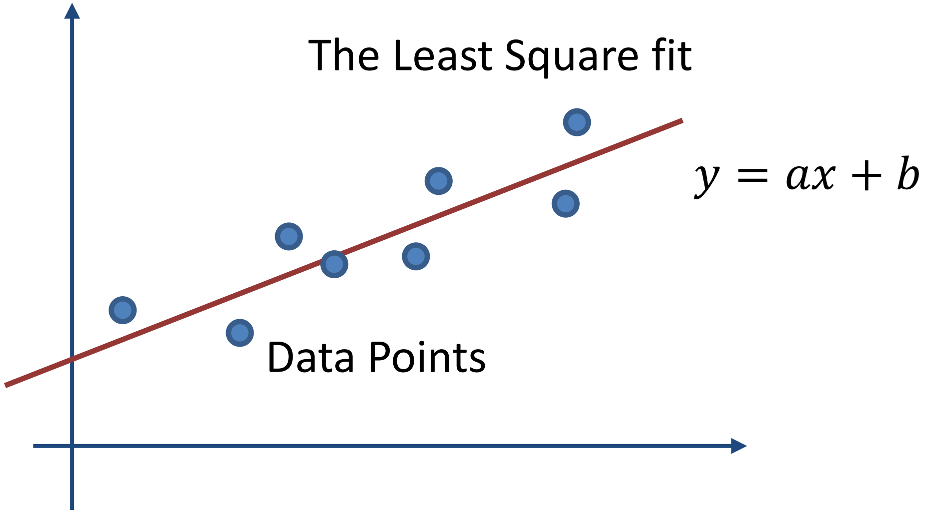
LSM Example

Given the following Data:

x	y
0	15
1	10
2	9
3	6
4	2
5	0

$$y = ax + b$$

We need to find a and b



$$
\begin{aligned}
15 &= a \cdot 0 + b \\
10 &= a \cdot 1 + b \\
9 &= a \cdot 2 + b \\
6 &= a \cdot 3 + b \\
2 &= a \cdot 4 + b \\
0 &= a \cdot 5 + b
\end{aligned}
$$

$$Y = \Phi\theta$$

$$\begin{bmatrix} 15 \\ 10 \\ 9 \\ 6 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \\ 3 & 1 \\ 4 & 1 \\ 5 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

Python Code

```
import numpy as np

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

Y = np.array([[15],[10],[9],[6],[2],[0]])

theta_ls = np.linalg.lstsq(Phi, Y, rcond=None)[0]
print(theta_ls)

theta_ls = np.linalg.inv(Phi.transpose() * np.mat(Phi)) * Phi.transpose() * Y
print(theta_ls)
```

Compare built-in LSM and LMS from scratch

From the Python code we get the following results:

[-2.91428571 14.28571429]

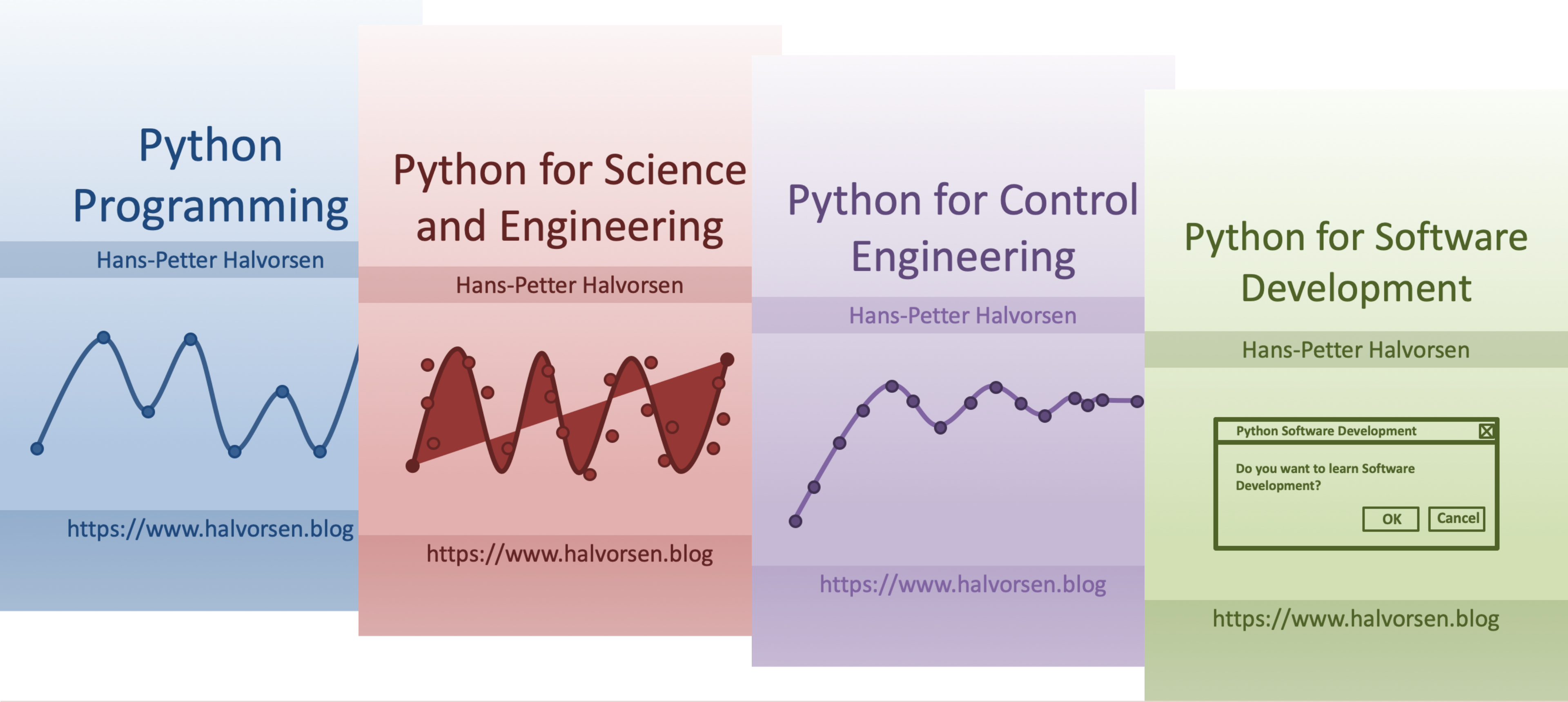
This means $a = -2.91$ and $b = 14.29$

Or:

$$y = -2.91x + 14.29$$

Which is the same results as shown in previous examples

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