Introduction to Computer Programming

Week 9.2: Curve Fitting

It can be useful to define a relationship between two variables, x and y.

We often want to 'fit' a function to a set of data points (e.g. experimental data).

Python has several tools (e.g. Numpy and Scipy packages) for finding relationships in a set of data.

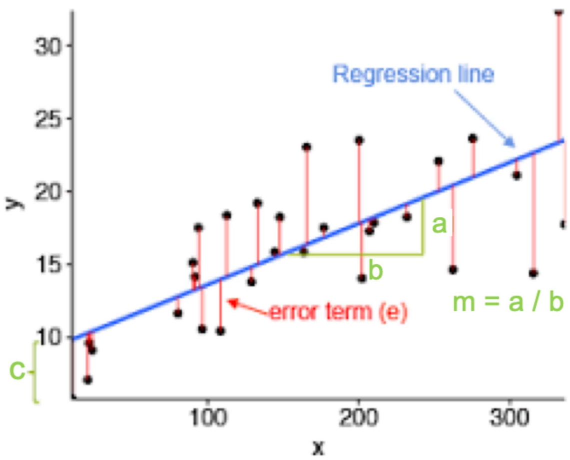
In [2]:

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

Linear Regression

> ***Linear function:***
> *Has form*
> $$f(x) = mx + c$$
> *where m and c are constants.*

Linear regression calculates a **linear function** that minimizes the combined error between the fitted line and the data points.



Fitting a polynomial function

> ***Polynomial function:*** *a function involving only non-negative integer powers of x.*

1st degree polynomial $y = \mathrm{a}x^1 + \mathrm{b}x^0$
(linear function)

2nd degree polynomial $y = \mathrm{c}x^2 + \mathrm{d}x^1 + \mathrm{e}x^0$

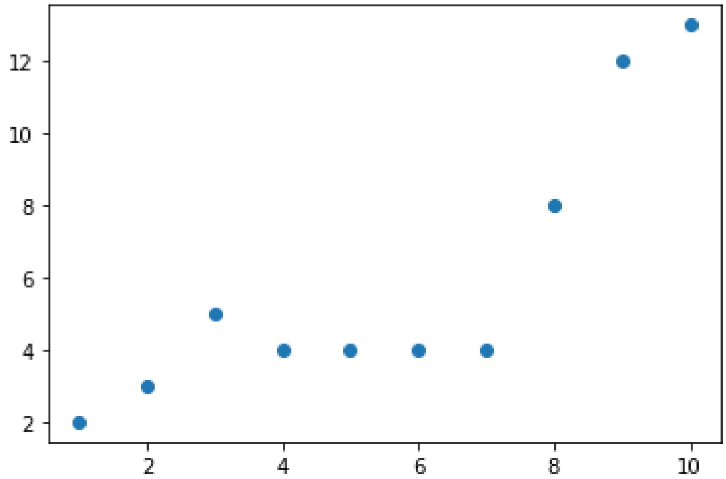
3rd degree polynomial $y = \mathrm{f}x^3 + \mathrm{g}x^2 + \mathrm{h}x^1 + \mathrm{i}x^0$

In [3]:

```
x = np.array([1, 6, 3, 4, 10, 2, 7, 8, 9, 5])
y = np.array([2, 4, 5, 4, 13, 3, 4, 8, 12, 4])
plt.plot(x,y,'o')
```

Out[3]:

```
[<matplotlib.lines.Line2D at 0x1e7c05e47f0>]
```



Fitted function

A polynomial function can be fitted using the `numpy.polyfit` function.

Inputs:

- independent variables
- dependent variables
- degree of the polynomial

Returns:

- coefficients of each term of the polynomial.

In [1]:

```
# coefficients of fited function
coeffs_1 = np.polyfit(x, y, 1) # 1st degree poly

coeffs_2 = np.polyfit(x, y, 2) # 2nd degree poly

print(coeffs_1)
print(coeffs_2[0], coeffs_2[1], coeffs_2[2])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-ff22c2088f27> in <module>
      1 # coefficients of fited function
----> 2 coeffs_1 = np.polyfit(x, y, 1) # 1st degree poly
      3 
      4 coeffs_2 = np.polyfit(x, y, 2) # 2nd degree poly
      5 

NameError: name 'np' is not defined
```

Fitted data

`numpy.poly1d` : generates fitted y data allowing the fitted function to be plotted.

Inputs:

- coefficients of the fitted polynomial function
- x data, monotonically sorted (for plottting)

Returns:

- fitted y data

x values from original data (sorted monotonically for line plot)

In [66]:

```
x_new = np.array(sorted(x))  # x values, sorted monotonically
```

Fitted y values

In [67]:

```
# 1st order polynomial
yfit1 = np.poly1d(coeffs_1)(x_new)

# 2nd order polynomial
yfit2 = np.poly1d(coeffs_2)(x_new)
```

Essentially `poly1d` does the following pure Python operation:

In [68]:

```
yfit1 = coeffs_1[0]*x_new + coeffs_1[1]

yfit2 = coeffs_2[0]*x_new**2 + coeffs_2[1]*x_new + coeffs_2[2]
```

Plotting fitted data

In [2]:

```
# write eqns as strings using coefficients
eqn_1 = f'y={round(coeffs_1[0],2)}*x + {round(coeffs_1[1],2)}'
eqn_2 = f'y={round(coeffs_2[0],2)}*x**2 + {round(coeffs_2[1],2)}*x**1 + {round(coeffs_2[2],


# plot data
plt.plot(x, y, 'o', label='raw data') # raw data
plt.plot(x_new, yfit1, label=eqn_1);  # fitted 1st degree poly
plt.plot(x_new, yfit2, label=eqn_2);  # fitted 2nd degree poly


# 'label' used to display the equation of the line as the legend
plt.legend()


# label the axes
plt.xlabel('x')
plt.ylabel('y')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-5a1179b7a47f> in <module>
      1 # write eqns as strings using coefficients
----> 2 eqn_1 = f'y={round(coeffs_1[0],2)}*x + {round(coeffs_1[1],2)}'
      3 eqn_2 = f'y={round(coeffs_2[0],2)}*x**2 + {round(coeffs_2[1],2)}*x**
1 + {round(coeffs_2[2],2)}'
      4
      5

NameError: name 'coeffs_1' is not defined
```

Fitting an Arbitrary Function

Curve fitting is not limited to polynomial functions.

We can fit any function with unknown constants to the data using the function `curve_fit` from the `scipy` package.

Fitted function

Choose a function to fit e.g.

$$y = ae^{bx}$$

Define the function in the following format:

In [53]:

```
def exponential(x, a, b): # input arguments are independent variable, then unknown constant
    y = a * np.exp(b*x)
    return y
```

Use `curve_fit` to find the constants that best fit the function to the data.
Inputs:

- the function to fit (in the format above)
- the independent variable
- the dependent variable

Returns:

- constants of fitted function
- the covariance of the parameters (a statistical measure of accuracy)

In [54]:

```
from scipy.optimize import curve_fit

# constants of fitted function
c, cov = curve_fit(exponential, x, y)
```

Fitted data

Generate fitted data by runnnig the function we defined (`exponential`), on:

- x data (sorted monotonically if plotting)
- fitted constants (`*` allows `c` to be a data structure of any length)

In [55]:

```
# input  to function to get fitted data
# use monotonically sorted x data
yfit = exponential(x_new, *c)
```

Plotting fitted data

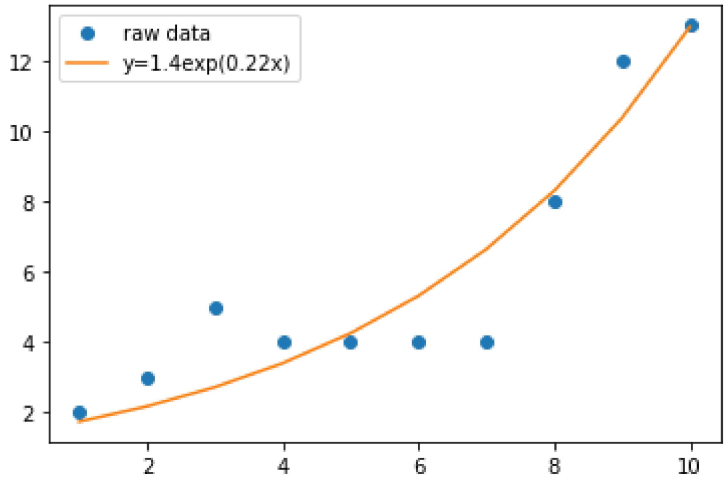
In [56]:

```
# write eqn as string using constants
eqn = f'y={round(c[0],2)}exp({round(c[1],2)}x)'

# plot data
plt.plot(x, y, 'o', label='raw data') # raw data
plt.plot(x_new, yfit, label=eqn);     # fitted function
plt.legend()
```

Out[56]:

```
<matplotlib.legend.Legend at 0x1e7c1d4fc10>
```



How does `polyfit` / `curve_fit` determine which coefficients/constants give the best fit?

How can we measure 'goodness' of fit e.g. to choose order of polynomial for best fit line?

Root Mean Square Error (RMSE)

A widely used measure of the error between fitted values and raw data.

Error/residual, ε:
The difference between the raw value $y(x)$ and the fitted value $a(x)$.

$$\varepsilon = a(x) - y(x)$$

Sum of the squared errors (so that negative and positive errors do not cancel) for N data points:

$$S = \sum_{i=1}^{N} \epsilon_i^2$$

RMSE:

$$RMSE = \sqrt{\frac{1}{N} S} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \epsilon_i^2}$$

Smaller RMSE indictes smaller error (i.e. a better fit between raw and fitted data).

We can optimise the fitted function by minimising the RMSE (used by `curve_fit`).

Also referred to as the *least squares* approach.

We can also use RMSE to compare fitted functions and determine statistically which is a better fit.

In [35]:

```
def RMSE(x, y, yfit):
    "Returns the RMSE of a polynomial of specified order fitted to x-y data"
    # error
    e = (yfit - y)

    # RMSE
    return np.sqrt(np.sum(e**2)/ len(x))
```

Let's compare the RMSE of each polynomial we fitted to the x,y data earlier

In [4]:

```
for order in range(1, 3):
    coeffs = np.polyfit(x, y, degree) # coefficients of fitted polynomial
    yfit = np.poly1d(coeffs)(x)      # no need to sort x monotonically, not plotting line
    rmse = RMSE(x,y,yfit)            # goodness of fit
    print(f'polynomial order {degree}, RMSE = {rmse}')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-9bef6b97e6d8> in <module>
      1 for order in range(1, 3):
----> 2     coeffs = np.polyfit(x, y, degree) # coefficients of fitted polyn
omial
      3     yfit = np.poly1d(coeffs)(x)      # no need to sort x monotonical
ly, not plotting line
      4     rmse = RMSE(x,y,yfit)            # goodness of fit
      5     print(f'polynomial order {degree}, RMSE = {rmse}')

NameError: name 'np' is not defined
```

The second order polynomial gives a better fit.

What about the exponential function?

In [63]:

```
c, cov = curve_fit(exponential, x, y)  # constants of fitted function
yfit = exponential(x, *c)              # no need to sort x monotonically, not plotting line
rmse = RMSE(x,y,yfit)                  # goodness of fit
print(f'RMSE = {rmse}')
print(f'{eqn}, RMSE = {rmse}')
```

```
RMSE = 1.3338248760975377
y=1.4exp(0.22x), RMSE = 1.3338248760975377
```

Of the three functions tested, the second order polynomial gives a better fit, statitically.

Summary

1. Find constants of fitted function

    - **Polynomial functions:** Find coefficients of polynomial by running `polyfit` on data and specifying order of polynomial.
    - **Arbitrary functions:** Find constants of arbitrary function by defining function to fit and running `curve_fit` on raw data and function to fit.

2. Generate fitted data (arrange x data monontonically if plotting as graph):

    - **Polynomial functions:** Use `poly1D` to generate the fitted data using fitted coefficients for given input range.
    - **Arbitrary functions:** Call function defined in step 1 using a range of x data and fitted coefficents as inputs.

3. Test goodness of fit: RMSE or other otimisation method.

In-class Demos

Example 1:

Fit third deree polynomial function the x,y data given.

Find the root mean square error for the fitted data

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\varepsilon_i^2}$$

In [ ]:

```
x = np.array([1, 6, 3, 4, 10, 2, 7, 8, 9, 5])
y = np.array([2, 4, 5, 4, 13, 3, 4, 8, 12, 4])
```

In [71]:

```
# Find coefficients of polynomial


# Generate fitted data


# RMSE
```

Example 2:

Import data in from `sample_data/signal_data.csv` .

Fit a function of the form $y = a \sin(x + b)$ to the data.
(i.e. find constants a and b).

Plot the raw and fitted data on the same graph.

In [1]:

```
# IMPORT DATA



# FIT FUNCTION




# GENERATE FITTED DATA



# PLOT
```

In [ ]:

```

```