

Non-Linear models are functions that are non-linear in their parameters

Exponential , power, Gaussian, Fourier etc.. functions, are non-linear because their coefficients are not linear

Exponential $y = y_0 \exp(-k x)$

Power $y = a x^b$

Examples of non-linear models and their applications:

<https://acsess.onlinelibrary.wiley.com/doi/10.2134/agronj2012.0506>

look at table1 for a summary of non-linear models

Non-linear least-squares procedure

For non-linear models, we will use the `scipy.optimize` module, and in particular the `curve_fit()` function, which implements a **non-linear least-squares procedure**.

Import the `curve_fit()` in this way:

```
from scipy.optimize import curve_fit
```

```
popt, pcov = curve_fit(func, xarray, yarray, guess_array)
```

func - the fitting model function, which can be a custom function defined by def

xarray – array-like (1D array, Series) the independent variable

yarray – array-like (1D array, Series) the dependent variable

initial_guess – a list or 1D array containing the initial guess for each parameter

Useful when `curve_fit()` cannot calculate the covariance matrix.

The function returns:

popt array – fitting coefficients – 1D array

Optimal values for the parameters so that the sum of the square residuals is minimized.

pcov matrix – covariance matrix – 2D array $N \times N$ where N is the number of fitting parameters.

Calculate the Residuals for each point.

$$r_i = y_i - y_{ip}$$

```
yp= func(x_data, *popt) #predicted values  
residuals = y_data - yp #the * operator means all the elements  
in variable poprt.
```

Calculate the standard error of the parameters

the square root of the diagonal elements of the covariance matrix gives the standard error for each parameter

```
perr = np.sqrt(np.diag(pcov))
```

Example – Fit a non-linear model with SciPy

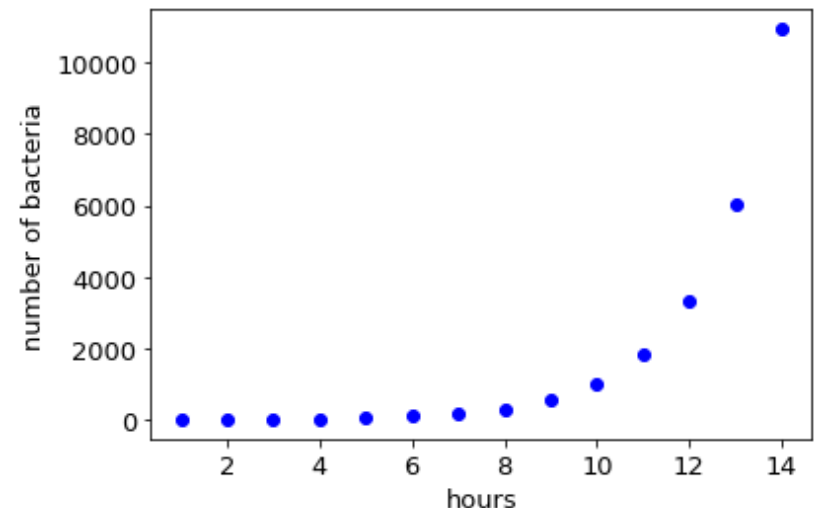
Download the data file `bacteria.csv` from Canvas DATA, which contains data on bacterial growth. The 1st field is the hours since observation and the 2nd field the number of bacteria in the sample. We will fit the data with an exponential growth model.

$$y = a e^{bx}$$

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

data=np.loadtxt('bacteria.csv',delimiter=',')
x=data[:,0] #hours since observation
y=data[:,1] #number of bacteria

#plot row data
plt.rcParams['font.size']=13
fig, ax=plt.subplots()
ax.plot(x, y, 'ob', label='data')
ax.set_xlabel('hours')
ax.set_ylabel('number of bacteria')
```



```
#choose the model, and make the function model
def mymodel(x,a,b): #the independent variable (x) must be listed first
    return a*(np.exp(b*x))
```

```
#fit with curve_fit
popt, pcov = curve_fit(mymodel, x, y)
```

#Sometimes an **initial guess** of the parameters is required for the fitting method to work.

```
#popt, pcov = curve_fit(mymodel, x, y, [2.1, 0.3])) #provide an
initial guess
```

```
print(popt)
[2.49652581 0.59892497] #fitted parameters a and b
```

The order of the parameters in popt and pcov matches the order of the parameters in the mymodel function

```
print(pcov) #covariance matrix
[[ 7.07224351e-05 -2.08093101e-06]
 [-2.08093101e-06  6.14351150e-08]]
```

Example – Fit a non-linear model with SciPy

#Use the function to create an array containing the predicted values.

```
xsmall=np.arange(np.min(x),np.max(x)+0.1,0.5) #make more x values  
to smooth the model function
```

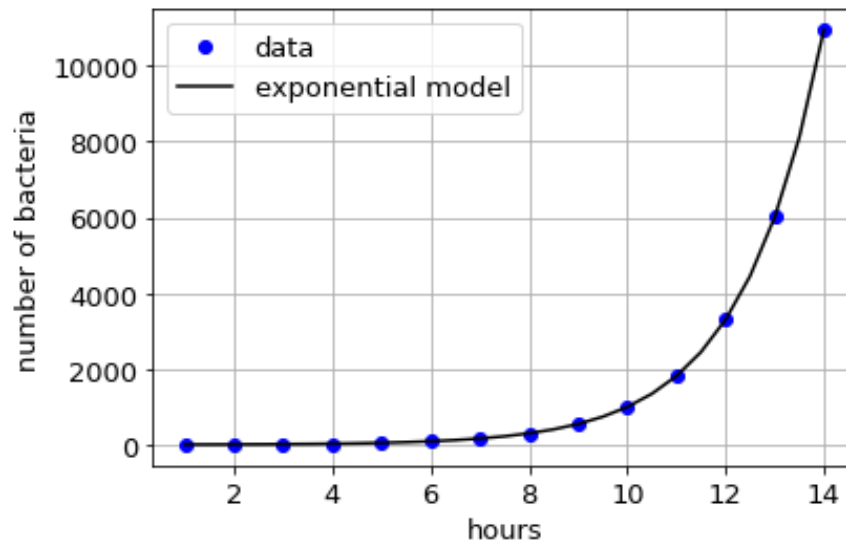
```
ypsmall=mymodel(xsmall,*popt) #the * operator means all the  
elements in variable pop.
```

```
#plot the fitted model.
```

```
ax.plot(xsmall,ypsmall,'k-',label='exponential model')
```

```
ax.grid()
```

```
ax.legend()
```



Non-linear models - Evaluating the fit

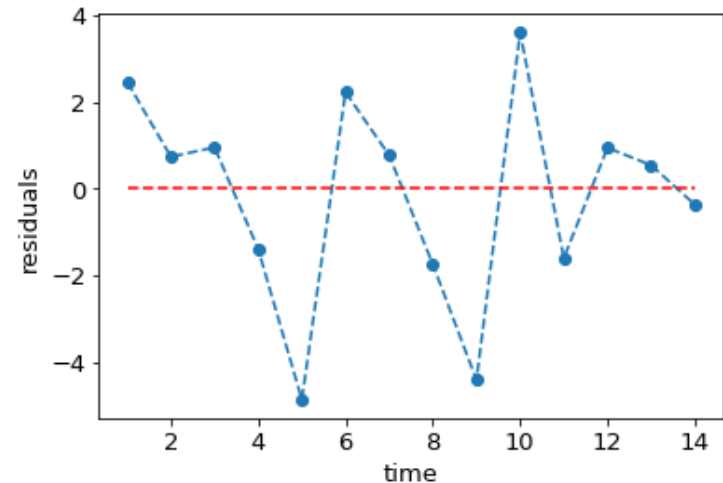
You can calculate the Residuals and make diagnostic plots.

```
#calculate and plot the residuals
yp=mymodel(x,*popt)
residuals = y - yp

fig1, ax1=plt.subplots()
ax1.plot(x, residuals, 'o--')

#make and plot a line y=0
ax1.plot(x,np.zeros(x.size), '--r')
ax1.set_xlabel('time')
ax1.set_ylabel('residuals')
```

“good” fit: values are distributed randomly around 0, and there is no cluster of values all positive or all negative.



Non-linear models - Calculating standard errors of the parameters

The covariance matrix is:

```
print(pcov)
[[ 7.07224351e-05 -2.08093101e-06]
 [-2.08093101e-06  6.14351150e-08]]
```

To calculate the standard error for each parameter, we calculate the square root of the diagonal elements of the covariance matrix.

```
perr = np.sqrt(np.diag(pcov))
print(perr)
[0.00840966  0.00024786]

print("a=%.4f std_err=%.4f" % (popt[0], perr[0]))
print("b=%.4f std_err=%.4f" % (popt[1], perr[1]))
a=2.4965 std_err=0.0084
b=0.5989 std_err=0.0002
```

The estimated parameters are:

a=2.4965±0.0084
b=0.5989±0.0002

Make predictions

We can calculate how many bacteria are expected after 15 hours.

```
xpred=15
```

```
ypred=mymodel(xpred,*popt)
```

```
ax.plot(xpred,ypred,'dg', label='prediction')
```

```
ax.legend()
```

```
plt.show()
```

