In [1]:
```
# Example 1: optimisation
# Fitting data to normal distribution, with known variance, by MLE

# We use generated data for this example
# Scipy has built in function for normal distribution
from scipy.stats import norm
# There are many more in package"stats",
# such as pearson, gamma, log-normal distributions.

# norm.rvs() generates random normal variables,
# default is standard normal distribution.
# 'data' is N(5,1) distributed
data=norm.rvs(size=50,loc=5)
data[0:4]
```

Out[1]:
```
array([ 2.75107875,  3.97744349,  4.31571681,  4.2916725 ])
```

In [2]:
```
# Here we want to minimise the negative log-likelihood function
import numpy as np
def nllh(mu):
    z=norm.pdf(data,scale=1,loc=mu).prod()
    return(-np.log(z))
```

In [3]:
```
# import minimisation function
from scipy.optimize import minimize
```

In [4]:
```
# choose a starting point, 9
# choose an optimisation method, Nelder-Mead
minimize(nllh,9,method='Nelder-Mead')
```

Out[4]:
```
 final_simplex: (array([[ 4.94769287],
       [ 4.9477478 ]]), array([ 71.5976041 ,  71.59760413]))
           fun: 71.59760409601401
       message: 'Optimization terminated successfully.'
          nfev: 38
           nit: 19
        status: 0
       success: True
             x: array([ 4.94769287])
```

In [5]:
```
# x is the final value of mu
# fun is final negative log-likelihood value
```

In [6]:
```
# Example 2: Area of a semi-circle
# y=sqrt(1-x^2)
def y(x):
    z=(1-x**2)**(0.5)
    return(z)
```

In [7]:
```
# import integrate package
import scipy.integrate as integrate
```

In [8]:
```
# quad() calculates definite integral
integrate.quad(y, -1, 1)
# output is (estimate,upper bound of error)
```

Out[8]:
```
(1.5707963267948983, 1.0002354500215915e-09)
```

In [9]:
```
# the result is 1.57, half pi
```