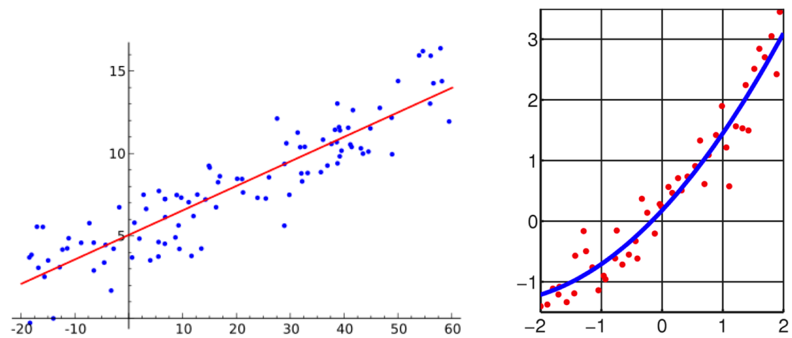
Week 6 - Least-squares fitting and roots

In [1]:
```
from IPython.display import Image
Image(filename='LS_fit.png',width="400") `
```

Out[1]:



=> The method of least squares is most important application in `data-fitting`

=> It is a mathematical procedure for finding the $\mathbf{best-fitting}$ curve to a given set of points by minimizing the sum of the squares of the offsets (`"the residuals"`) of the points from the curve.

Let $\mathbf{(x_i, y_i)}$ is the given data containing a significant amount of random noise casued by measurement errors.

=> $\mathbf{(x_i, y_i)}$ with (n+1) data points where i = 0,1,2,...,n

Let $\mathbf{f(x) = f(x; a_0, a_1, \ldots, a_m)}$ is the function to be fotted to $\mathbf{(x_i, y_i)}$

$a_0, a_1, \ldots, a_m$ are (m+1) variable parameters

Note: $m < n$ (if $m = n$, it's not fitting but interpolation)

In [ ]:

$\underline{\mathbf{best-fit}}$: The best-fit in the Least-Squares sense minimizes the sum of `squared residuals`

i.e, minimizing the function,

$$\mathbf{S(a_0, a_1, \ldots, a_m) = \sum_{i=0}^{n} \ [y_i - f(x_i)]^2 \quad \rightarrow (1)}$$

=> $r_i = [y_i - f(x_i)]$ is the $\mathbf{residual}$, which is the difference between data value and fitted value

=> The optimal values of varaible parmameters $\mathbf{a_j}$ can be obtained from

$$\partial \mathbf{S} = \mathbf{0} \quad \mathbf{i} = \mathbf{0}, \mathbf{1}, \ldots \mathbf{m} \rightarrow \mathbf{(2)}$$

In [ ]:

=> **Standard deviation**: is the spread of the data about the fitting curve $f(x)$

$$\sigma = \sqrt{\frac{\mathbf{S}}{\mathbf{n-m}}} \quad \rightarrow \mathbf{(3)}$$

In [ ]:

Fitting Linear Forms

Consider the least-squares fit of the $linear\ form$,

$f(x) = a_0 f_0(x) + a_1 f_1(x) + \ldots + a_m f_m(x) = \sum_{j=0}^{m} a_j f_j(x)$

where each $f_j(x)$ is a predetermined function of x, called a $basis\ function$.

In [6]:

```
from IPython.display import Image
Image(filename='expl.png',width="520")
```

Out[6]:

In minimizing the function $f(x)$, the equation(1) yields,

$$S = \sum_{i=0}^{n} \left[y_i - \sum_{j=0}^{m} a_j f_j(x_i) \right]^2$$

the equation(2) yields,

$$\frac{\partial S}{\partial a_k} = -2 \left\{ \sum_{i=0}^{n} \left[y_i - \sum_{j=0}^{m} a_j f_j(x_i) \right] f_k(x_i) \right\} = 0, \quad k = 0, 1, \ldots, m$$

Dropping the constant (–2) and interchanging the order of summation, we get

$$\sum_{j=0}^{m} \left[\sum_{i=0}^{n} f_j(x_i) f_k(x_i) \right] a_j = \sum_{i=0}^{n} f_k(x_i) y_i, \quad k = 0, 1, \ldots, m$$

In matrix notation these equations are

Aa = b

where

$$A_{kj} = \sum_{i=0}^{n} f_j(x_i) f_k(x_i) \qquad b_k = \sum_{i=0}^{n} f_k(x_i) y_i$$

- Equation **Aa = b**, known as the $normal\ equations$ of the least-squares fit
- It can be solved with the Gauss Elimination method.
- Note that the coefficient matrix is symmetric(i.e.,$\mathbf{A_{kj}} = \mathbf{A_{jk}}$).

In [1]:

```
import os
import sys

sys.path
```

Out[1]:

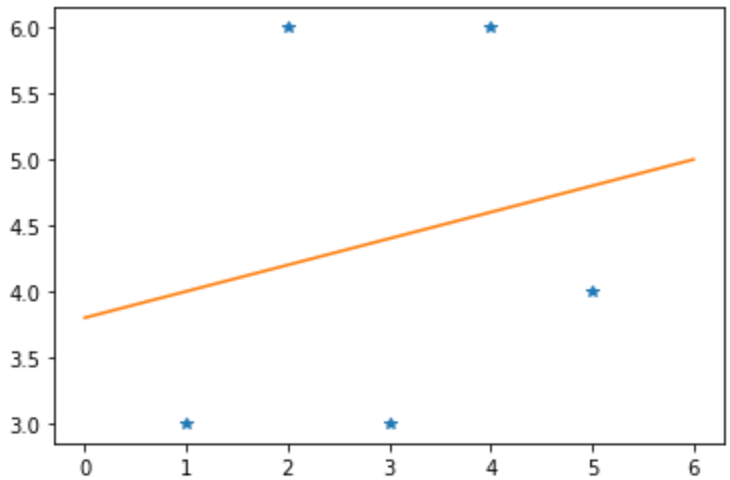
```
['',
 '/Users/srivanij/anaconda3/lib/python36.zip',
 '/Users/srivanij/anaconda3/lib/python3.6',
 '/Users/srivanij/anaconda3/lib/python3.6/lib-dynload',
 '/Users/srivanij/anaconda3/lib/python3.6/site-packages',
 '/Users/srivanij/anaconda3/lib/python3.6/site-packages/aeosa',
 '/Users/srivanij/anaconda3/lib/python3.6/site-packages/IPython/extensions',
 '/Users/srivanij/.ipython']
```

In [16]:

```
import sys
sys.path.append('../myModules/')

from polyFit import *
from gaussPivot import *
from polyFit import *
from plotPoly import *
```

In [17]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import *

x = np.array([1,2,3,4,5])
y = np.array([3,6,3,6,4])

# 다항식 피팅
pfit = np.polyfit(x,y,1)  # n, n-1, ..., 0
line = np.poly1d(pfit)

xx = np.linspace(0,6,50)
yy = line(xx)

plt.plot(x,y,'*')   # original data
plt.plot(xx,yy)     # linear fit
plt.show()

# curve_fit(f,xdata,ydata)
from scipy.optimize import curve_fit

def f(x,m,b):
    return m*x + b

parm, cvar = curve_fit(f,x,y)
yy1 = f(xx,parm[0],parm[1])
```



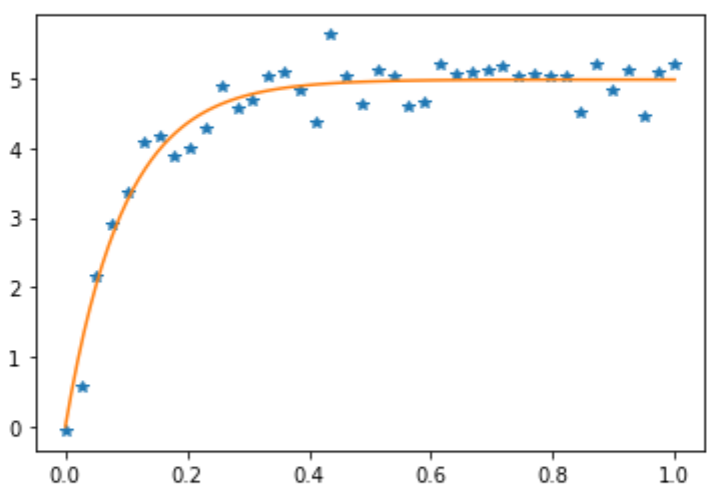
In [8]:

```
# Ball free falling in oil,  data = np.genfromtxt('oildata.dat',delimiter=','

data0 = np.array([0.    , 0.0256, 0.0513, 0.0769, 0.1026, 0.1282, 0.1538, 0.1
       0.2051, 0.2308, 0.2564, 0.2821, 0.3077, 0.3333, 0.359 , 0.3846,
       0.4103, 0.4359, 0.4615, 0.4872, 0.5128, 0.5385, 0.5641, 0.5897,
       0.6154, 0.641 , 0.6667, 0.6923, 0.7179, 0.7436, 0.7692, 0.7949,
       0.8205, 0.8462, 0.8718, 0.8974, 0.9231, 0.9487, 0.9744, 1.    ])

data1 = np.array([-0.0582,  0.5609,  2.1524,  2.8921,  3.3555,  4.092 ,  4.15
        3.8705,  3.9884,  4.2759,  4.8735,  4.5771,  4.6779,  5.0256,
        5.0751,  4.8384,  4.366 ,  5.6249,  5.0227,  4.633 ,  5.1175,
        5.0317,  4.6023,  4.6559,  5.2119,  5.0483,  5.075 ,  5.1081,
        5.1749,  5.0215,  5.0547,  5.0363,  5.0381,  4.4984,  5.2146,
        4.8216,  5.128 ,  4.4662,  5.0975,  5.1963])

plt.plot( data0,data1,'*')

def veloc(t,v0,tau):
    return v0*(1 - np.exp(-t/tau))

# We can find the optimum parameter set using curve_fit

parm, cvar = curve_fit(veloc, data0, data1)
print ('Params ', parm)

param0 = parm[0]  #5
param1 = parm[1]  #0.1

xx = np.linspace(0,1,100)
yy = veloc(xx, param0, param1)
plt.plot(xx,yy,'-')
```

```
Params  [4.98137121 0.09643936]
```

Out[8]:

```
[<matplotlib.lines.Line2D at 0x7ff7486ef7f0>]
```



Example 1:

In [27]:

```
from IPython.display import Image
Image(filename='ex2.png',width="500")
```

Out[27]: Use linear regression to find the line that fits the data

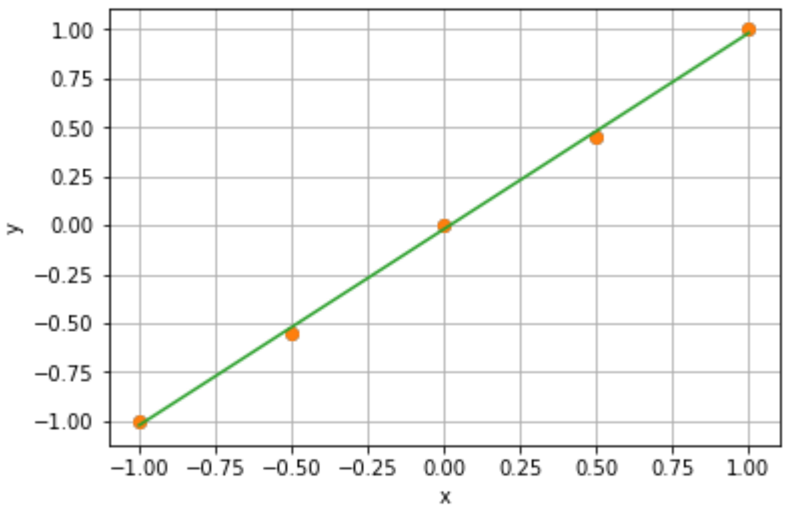
x	-1.0	-0.5	0	0.5	1.0
y	-1.00	-0.55	0.00	0.45	1.00

and determine the standard deviation.

In [5]:

```
import numpy as np
import math
from gaussPivot import *
from polyFit import *
from plotPoly import *
import matplotlib.pyplot as plt


xData = np.array([-1.0,-0.5,0,0.5,1.0])
yData = np.array([-1.00,-0.55,0.00,0.45,1.00])

# Find the coefficients of the line
m = 1
coeff = polyFit(xData,yData,m)
print("Coefficients are:\n",coeff)

# Find the standard deviation
print("Std. deviation =",stdDev(coeff,xData,yData))

# Plot the data and it's line fit
plt.plot(xData,yData,'o',label ='Data')
plotPoly(xData,yData,coeff,xlab='x',ylab='y')
plt.show()
```

```
Coefficients are:
 [-0.02  1.  ]
Std. deviation = 0.031622776601683805
```



Example 2:

In [6]:

```
from IPython.display import Image
Image(filename='ex12.png',width="600")
```

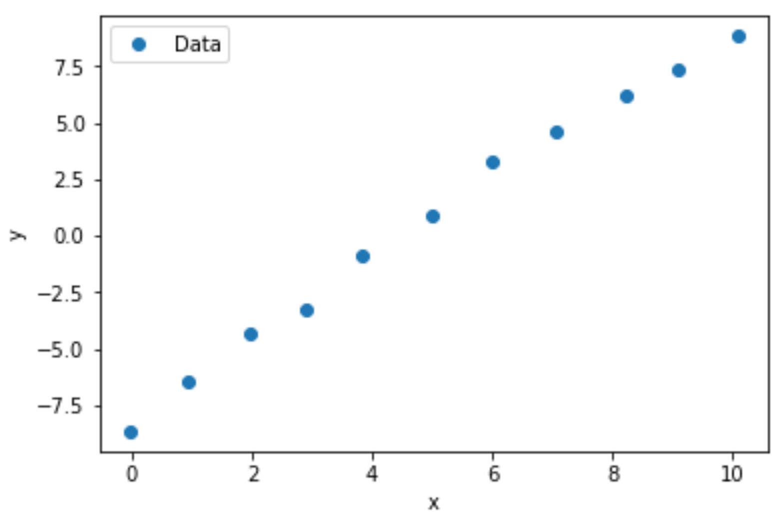
Out[6]:

Write a program that fits a polynomial of arbitrary degree m to the data points shown in the following table. Use the program to determine m that best fits this data in the least-squares sense.

x	-0.04	0.93	1.95	2.90	3.83	5.00
y	-8.66	-6.44	-4.36	-3.27	-0.88	0.87
x	5.98	7.05	8.21	9.08	10.09	
y	3.31	4.63	6.19	7.40	8.85	

In [8]:

```
import numpy as np
import math
from gaussPivot import *
from polyFit import *
from plotPoly import *
import matplotlib.pyplot as plt

xData = np.array([-0.04,0.93,1.95,2.90,3.83,5.0, \
                  5.98,7.05,8.21,9.08,10.09])
yData = np.array([-8.66,-6.44,-4.36,-3.27,-0.88,0.87, \
                  3.31,4.63,6.19,7.4,8.85])

## Plot the Given data

plt.plot(xData,yData,'o',label ='Data')
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



In [9]:

```
# Find the best fit by knowing the standard deviation

while True:
    try:
        m = eval(input("\nDegree of polynomial ==> "))
        coeff = polyFit(xData,yData,m)
        print("Coefficients are:\n",coeff)
        print("Std. deviation =",stdDev(coeff,xData,yData))
    except SyntaxError: break
input("Finished. Press return to exit")
```

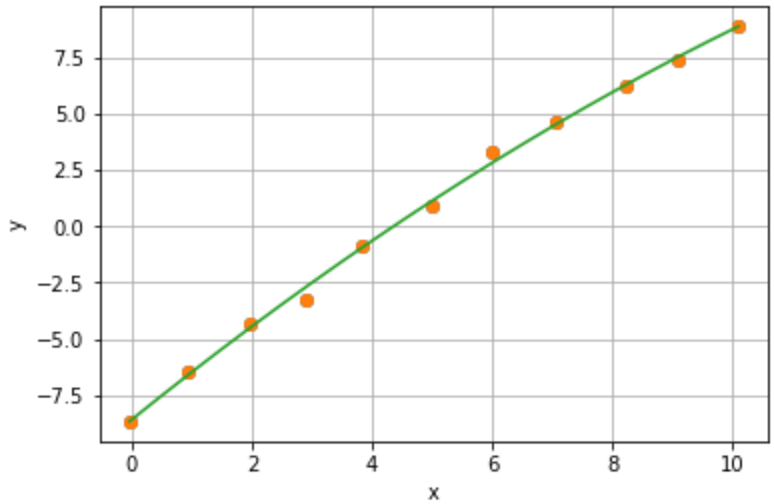
```
Degree of polynomial ==> 1
Coefficients are:
 [-7.94533287  1.72860425]
Std. deviation = 0.5112788367370911

Degree of polynomial ==> 2
Coefficients are:
 [-8.57005662  2.15121691 -0.04197119]
Std. deviation = 0.3109920728551074

Degree of polynomial ==> 3
Coefficients are:
 [-8.46603423e+00  1.98104441e+00  2.88447008e-03 -2.98524686e-03]
Std. deviation = 0.31948179156753187

Degree of polynomial ==>
Finished. Press return to exit
```

Out[9]: ''

Because the quadratic $f(x) = -8.5700 + 2.1512x - 0.041971x^2$ produces the smallest standard deviation, it can be considered as the “best” fit to the data.

In [10]:

```
# Plot the data and it's 'best' polynomial fit

coeff1 = [-7.94533287,1.72860425]
coeff2 = [-8.57005662,2.15121691,-0.04197119]
coeff3 = [-8.46603423e+00 , 1.98104441e+00 , 2.88447008e-03, -2.98524686e-03]

import matplotlib.pyplot as plt

plt.plot(xData,yData,'o',label ='Data')
#plotPoly(xData,yData,coeff1,xlab='x',ylab='y')
plotPoly(xData,yData,coeff2,xlab='x',ylab='y')
#plotPoly(xData,yData,coeff3,xlab='x',ylab='y')
plt.show()
```



Example 3:

In [20]:

```
from IPython.display import Image
Image(filename='ex3.png',width="600")
```

Out[20]:

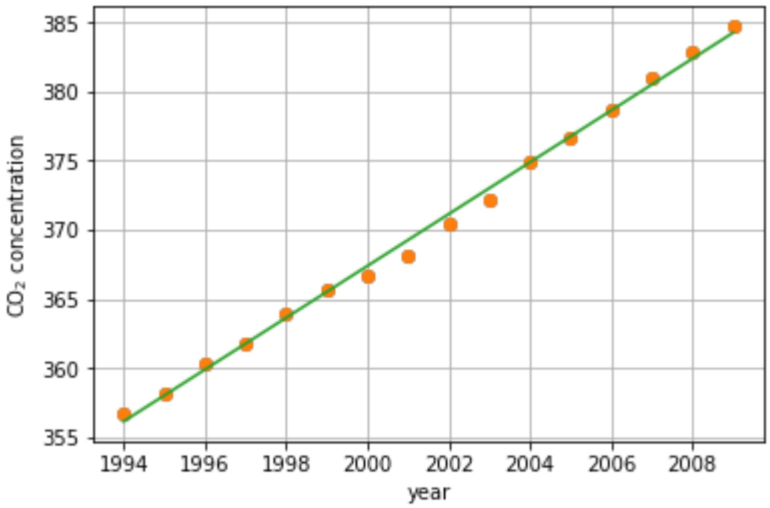
■ The following table shows the annual atmospheric CO_2 concentration (in parts per million) in Antarctica. Fit a straight line to the data and determine the average increase of the concentration per year.

Year	1994	1995	1996	1997	1998	1999	2000	2001
ppm	356.8	358.2	360.3	361.8	364.0	365.7	366.7	368.2
Year	2002	2003	2004	2005	2006	2007	2008	2009
ppm	370.5	372.2	374.9	376.7	378.7	381.0	382.9	384.7

In [36]:

```
xData = np.linspace(1994,2009,16)
yData = np.array([356.8,358.2,360.3,361.8,364.0,365.7,366.7,368.2,\
                  370.5,372.2,374.9,376.7,378.7,381.0,382.9,384.7])

# Find the coefficients of the line

m = 1
coeff = polyFit(xData,yData,m)
print("Coefficients are:\n",coeff)

# Find the standard deviation
print("Std. deviation =",stdDev(coeff,xData,yData))

# Plot the data and it's line fit
plt.plot(xData,yData,'o',label ='Data')
plotPoly(xData,yData,coeff,xlab='year',ylab='CO$_2$ concentration')
plt.show()
```

```
Coefficients are:
 [-3.37701382e+03  1.87220588e+00]
Std. deviation = 0.5462025126946465
```



The line fit of the given data is f(x) = -3.37701382e+03 + 1.87220588*x, where x is 'year'.

The average increase in the concentration of CO_2 per year is 1.87220588.

Example 4:

In [10]:

```
from IPython.display import Image
Image(filename='ex4.png',width="600")
```

Out[10]:

■ The kinematic viscosity μ_k of water varies with temperature T as shown in the following table. Determine the cubic that best fits the data, and use it to compute μ_k at $T = 10°, 30°, 60°$, and $90°$C.

T (°C)	0	21.1	37.8	54.4	71.1	87.8	100
μ_k (10^{-3} m^2/s)	1.79	1.13	0.696	0.519	0.338	0.321	0.296

Example 5:

In [23]:

```
from IPython.display import Image
Image(filename='ex5.png',width="600")
```

Out[23]: ■ The intensity of radiation of a radioactive substance was measured at half-year intervals. The results were

t (years)	0	0.5	1	1.5	2	2.5
γ	1.000	0.994	0.990	0.985	0.979	0.977
t (years)	3	3.5	4	4.5	5	5.5
γ	0.972	0.969	0.967	0.960	0.956	0.952

where γ is the relative intensity of radiation. Knowing that radioactivity decays exponentially with time, $\gamma(t) = ae^{-bt}$, estimate the radioactive half-life of the substance.

Exponential decay equation of readio-activity is $N(t) = N_0 e^{-\lambda t}$

λ is called 'decay constant', t is time.

Half-life of a radio-active substance is, $t_{\frac{1}{2}} = \dfrac{ln(2)}{\lambda}$

Solution:

It is clear that, we need to fit the given data of time t and γ with the exponential decay equation $\gamma(t) = ae^{-bt}$

We have (t,γ) data as follows:

t	0	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5	5.5
γ	1.000	0.994	0.990	0.985	0.979	0.977	0.972	0.969	0.967	0.960	0.956	0.952

We can find,

t	0	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5	5.5
$ln\gamma(t)$	0	-0.006	-0.01	-0.015	-0.021	-0.023	-0.028	-0.031	-0.033	-0.040	-0.044	-0.049

To transform our problem to linear regression problem as follows:

$ln(\gamma(t))$= $ln(ae^{-bt})$ = $ln(a) - bt$

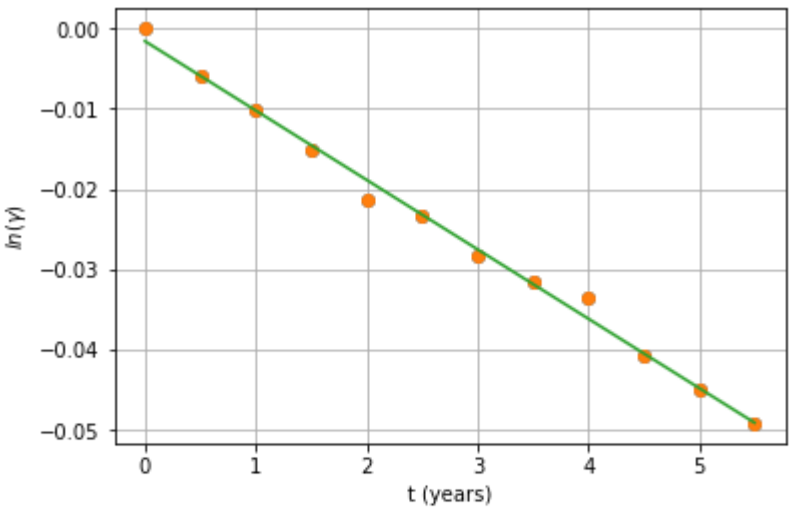
In [93]:
```
yData = np.array([1.000,0.994,0.990,0.985,0.979,0.977,\
                  0.972,0.969,0.967,0.960,0.956,0.952])

np.log(yData)
```

Out[93]:
```
array([ 0.        , -0.00601807, -0.01005034, -0.01511364, -0.02122364,
       -0.02326863, -0.02839947, -0.03149067, -0.03355678, -0.04082199,
```

In [94]:

```
xData = np.linspace(0,5.5,12)
yData = np.array([1.000,0.994,0.990,0.985,0.979,0.977,\
                  0.972,0.969,0.967,0.960,0.956,0.952])
lnyData = np.log(yData)

# Find the coefficients of the line

m = 1
coeff = polyFit(xData,lnyData,m)
print("Coefficients are:\n",coeff)

# Find the standard deviation
print("Std. deviation =",stdDev(coeff,xData,lnyData))

# Plot the data and it's line fit
plt.plot(xData,lnyData,'o',label ='ln')
plotPoly(xData,lnyData,coeff,xlab='t (years)',ylab='$ln(\gamma)$')
plt.show()
```

```
Coefficients are:
 [-0.00158547 -0.00863955]
Std. deviation = 0.0012743449541175837
```



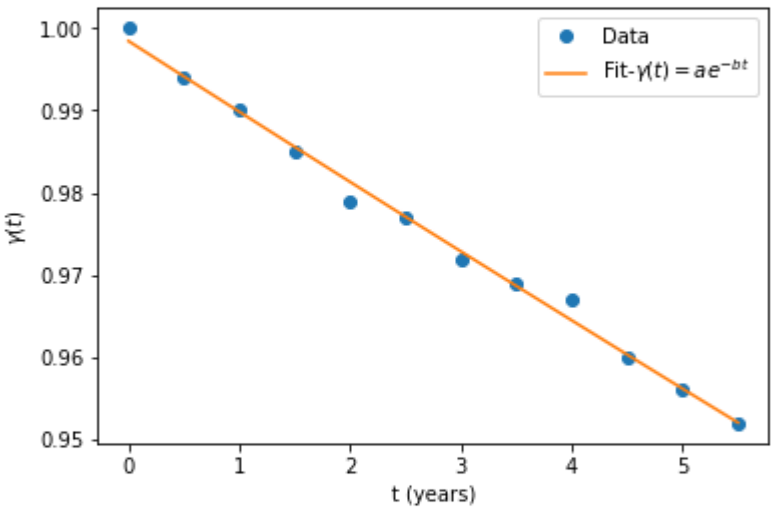
In [108…

```
# The coefficients

lna = coeff[0]
a = np.exp(lna)

b = coeff[1]
a,b

print('The coefficient b is the decay constant:',-b)
print('The half-life of the given radio-active substance is',np.log(2)/-b,'ye
```

```
The coefficient b is the decay constant: 0.008639549701453631
The half-life of the given radio-active substance is 80.22954951498468 years
```

In [110…

```
def fx(a,b,x):
    pl = a*np.exp(b*x)
    return pl

plt.plot(xData,yData,'o',label ='Data')
plt.plot(xData,fx(a,b,xData),'-',label ='Fit-$\gamma(t) = ae^{-bt}$')
plt.xlabel('t (years)')
plt.ylabel('$\gamma(t)$')
plt.legend()
plt.show()
```

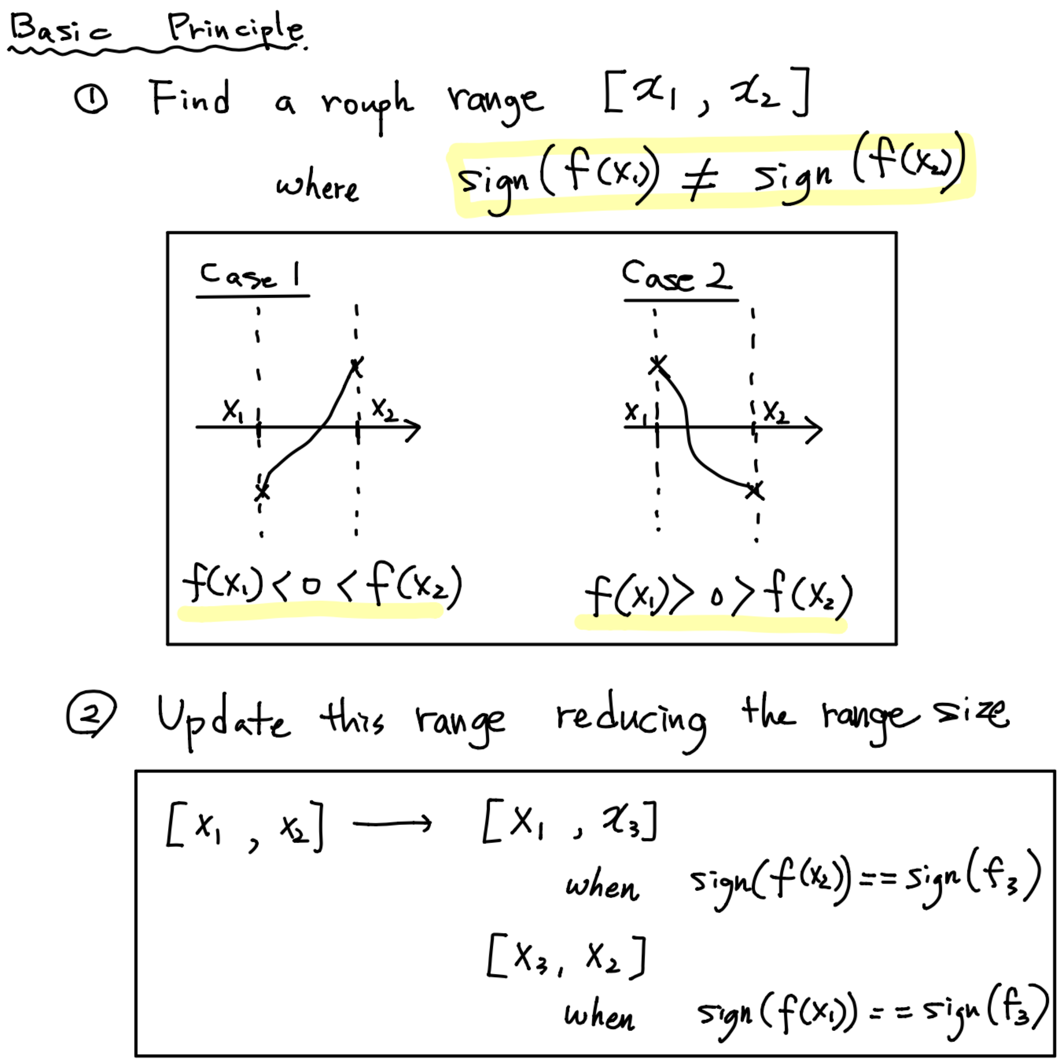


Chap 4 Roots of Equations

In [67]:

```
from IPython.display import Image
Image(filename='BasicPrinciple.png',width="550")
```

Out[67]:



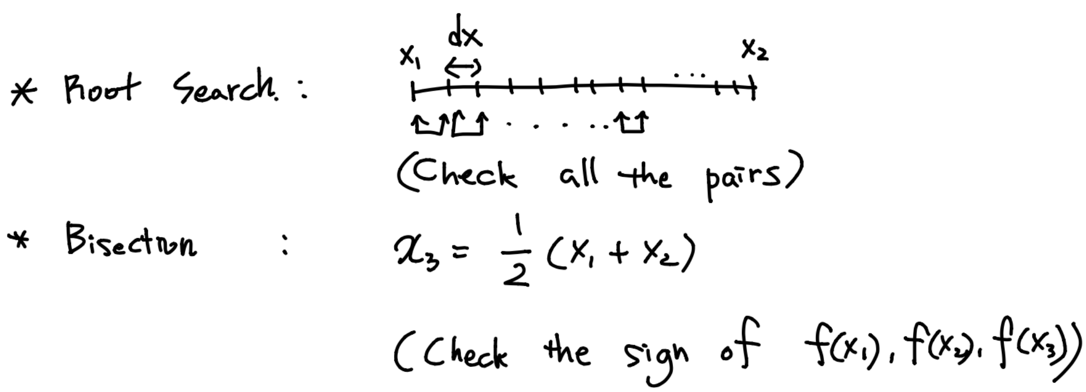
Physical Probelms

1. Rootsearch method & Bisection method

In [71]:

```
from IPython.display import Image
Image(filename='Theory_rootsearch_bisection.png',width="550")
```

Out[71]:



In [2]:
```
from IPython.display import Image
Image(filename='PhysicsProblem1.png',width="750")
```

Out[2]:

■ The speed v of a Saturn V rocket in vertical flight near the surface of earth can be approximated by

$$v = u \ln \frac{M_0}{M_0 - \dot{m}t} - gt$$

where

$$u = 2\,510 \text{ m/s} = \text{velocity of exhaust relative to the rocket}$$

$$M_0 = 2.8 \times 10^6 \text{ kg} = \text{mass of rocket at liftoff}$$

$$\dot{m} = 13.3 \times 10^3 \text{ kg/s} = \text{rate of fuel consumption}$$

$$g = 9.81 \text{ m/s}^2 = \text{gravitational acceleration}$$

$$t = \text{time measured from liftoff}$$

Determine the time when the rocket reaches the speed of sound (335 m/s).

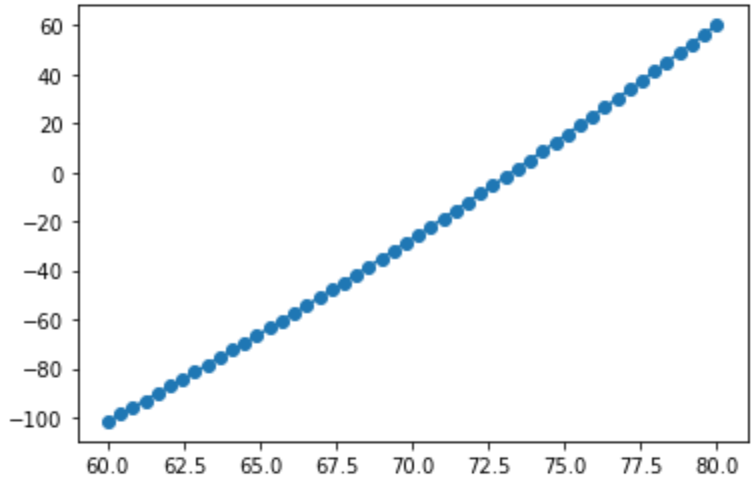
In [18]:
```
#   f(x) = 0  is the goal.
#   g(x) = 2   ->  f(x) =  g(x) - 2 = 0

#   v    = u*log(M0/(M0 - mt*t)) - gt  = 335
#   f(t) = u*log(M0/(M0 - mt*t)) - gt  - 335

u  = 2510
M0 = 2.8*1e6
mt = 13.3*1e3
g  = 9.81

def f(t):
    f = u*log(M0/(M0 - mt*t)) - gt  - 335
    return f
```

In [19]:
```
from rootsearch import *
from bisection import *

from math import log


def f(t):
    u = 2510
    M0 = 2.8*10**6
    mdot = 13.3*10**3
    g = 9.81

    return u*log(M0/(M0-mdot*t))-g*t - 355
```

In [20]:
```
xxp = np.linspace(60,80,10)
len(xxp)
```

Out[20]:
```
10
```

In [21]:
```
import matplotlib.pyplot as plt
import numpy as np
xx = np.linspace(60,80)
yy = [f(xx0) for xx0 in xx]
plt.plot(xx,yy,'-o')
plt.show()
```



In [6]:
```
print("t =",bisection(f,60,80),' sec')

rootrange = rootsearch(f,60,80,0.01)
print("t =",(rootrange[0]+rootrange[1])/2,' sec')
#input("Press return to exit")
```

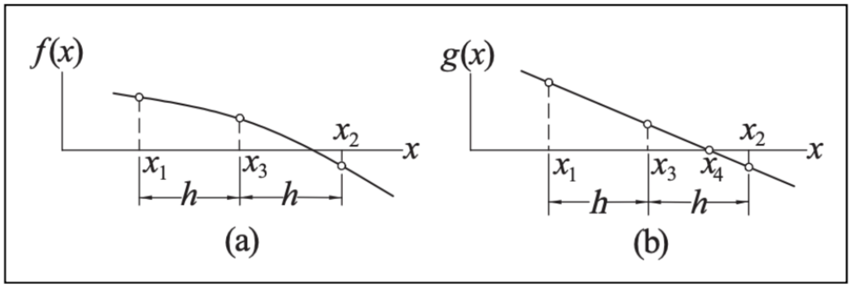
```
t = 73.28175809438108  sec
t = 73.28500000000395  sec
```

2. Ridder's method

In [69]:
```
from IPython.display import Image
Image(filename='Theory_Ridder.png',width="450")
```

Out[69]:

$$g(x) = f(x)e^{(x-x_1)Q}$$

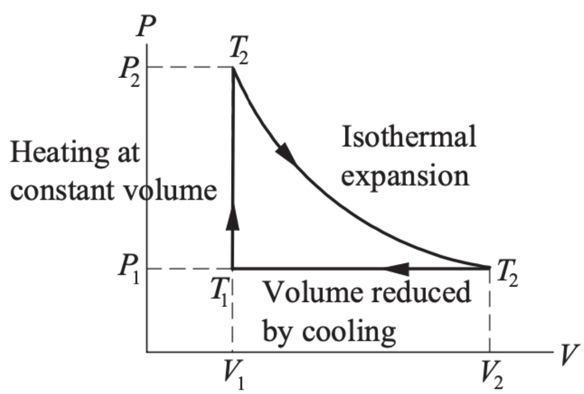


$$x_3 = \frac{1}{2}(x_1 + x_2)$$

$$x_4 = x_3 \pm (x_3 - x_1)\frac{f_3}{\sqrt{f_3^2 - f_1 f_2}}$$

In [9]:
```
from IPython.display import Image
Image(filename='PhysicsProblem4.png',width="750")
```

Out[9]:



The figure shows the thermodynamic cycle of an engine. The efficiency of this engine for monatomic gas is

$$\eta = \frac{\ln(T_2/T_1) - (1 - T_1/T_2)}{\ln(T_2/T_1) + (1 - T_1/T_2)/(\gamma - 1)}$$

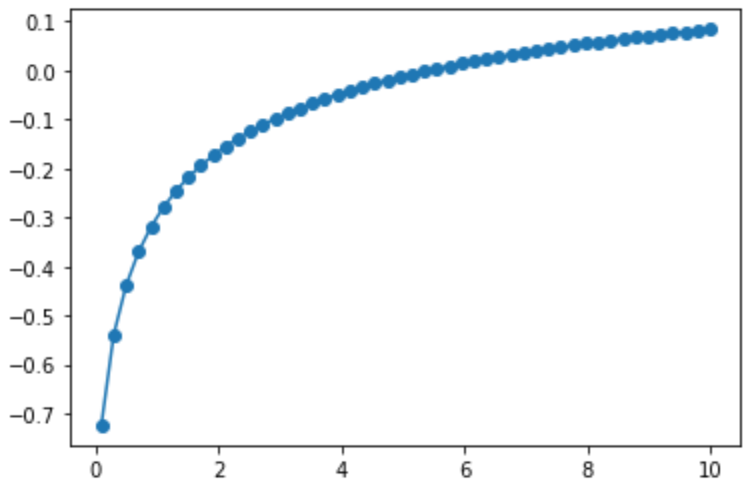
where T is the absolute temperature and $\gamma = 5/3$. Find T_2/T_1 that results in 30% efficiency ($\eta = 0.3$).

In [59]:

```
from ridder import *
def f(t2t1):
    gamma = 5/3
    from math import log
    fac1 = log(t2t1) - (1-1/t2t1)
    fac2 = log(t2t1) + (1-1/t2t1)/(gamma-1)
    return fac1/fac2 - 0.3
```

In [60]:

```
import matplotlib.pyplot as plt
import numpy as np
xx = np.linspace(0.1,10)
yy = [f(xx0) for xx0 in np.linspace(0.1,10)]
plt.plot(xx,yy,'-o')
plt.show()
```



In [62]:

```
print("T2/T1 =",ridder(f,0.1,10))
```

```
T2/T1 = 5.412548241399094
```

In [14]:

```
## Examples using scipy

from scipy.optimize import root_scalar
# scipy.optimize.root_scalar(f, args=(), method=None, bracket=None)
import numpy as np


def f(x):
    return x/2 - np.sin(x)

root_scalar(f, bracket=[0,1], method='ridder')
```

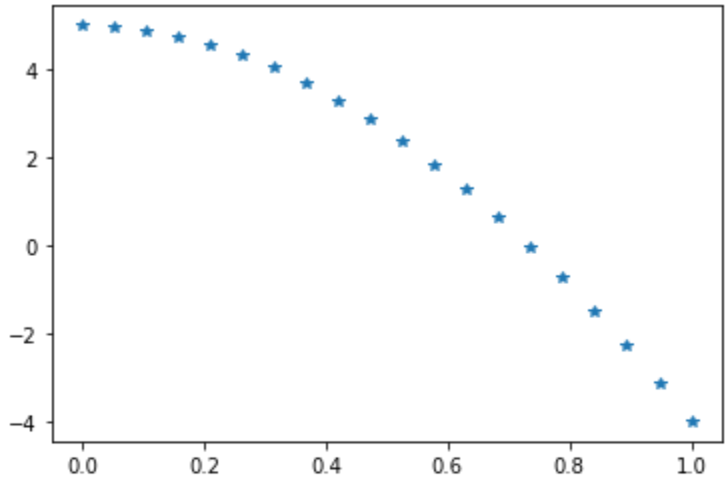
Out[14]:

```
      converged: True
           flag: 'converged'
 function_calls: 2
     iterations: 32759
           root: 0.0
```

In [22]:

```
def f(x):
    return x**3.0 - 10.0*x**2.0 + 5

x = np.linspace(0,1,20)
plt.plot(x,f(x),'*')

import time

start = time.time()
x1, x2 = rootsearch(f,0,1,0.1)
end = time.time(); print('Time elapsed ',end-start)
```

```
Time elapsed  0.00010395050048828125
```



In [24]:

```
import time

start = time.time()
x1, x2 = rootsearch(f,0,1,0.1)
end = time.time(); print('Time elapsed ',end-start)


start = time.time()
ans = root_scalar(f,bracket=[0,1],xtol=1e-8)
end = time.time(); print('Time elapsed ',end-start)
print('ans ',ans)
```

```
Time elapsed  8.416175842285156e-05
Time elapsed  9.894371032714844e-05
ans        converged: True
           flag: 'converged'
 function_calls: 8
     iterations: 7
           root: 0.7346035077893034
```

In [25]:

```
start = time.time()
x1 = 0; x2 = 1
for i in range(8):
    dx = ( x2 - x1 )/10.0
    x1, x2 = rootsearch(f,x1,x2,dx)


end = time.time(); print('Time elapsed ',end-start)
sol = (x1 + x2)*0.5
print('Sol ',sol)
```

```
Time elapsed  0.00022792816162109375
Sol  0.7346035050000002
```

Bisection method, 이분법 연습.

$\epsilon = \Delta x/2^n$ -> n 을 풀면 $n = \ln(\Delta x/\epsilon)/\ln(2)$

For $\epsilon = 10^{-4}$ we can find the number of iterations $n \sim \ln(\Delta x/\epsilon)$ given a bracketing interval

In [28]:

```
from bisection import bisection
x1 = 0; x2 = 1

start = time.time()
bisection(f,x1,x2,switch=1,tol=1.0e-9)
end = time.time(); print('Time elapsed ',end-start)



def f(x):
    return x - np.tan(x)

xx = np.linspace(0,20,400)
#plt.plot(xx,f(xx))
plt.plot(xx,np.tan(xx))

rootsearch(np.tan,1,2,0.001)

sol = bisection(np.tan,1,2,switch=0,tol=1.0e-9)
print(sol)
```

```
Time elapsed  9.512901306152344e-05
1.570796327199787
```

In [29]:

```
rootsearch(np.tan,1,2,0.001)

sol = bisection(np.tan,1,2,switch=0,tol=1.0e-9)
print(sol)
```

```
1.570796327199787
```

In [30]:

```
# 0 에서 20까지 tan(x)의 모든 루트를 찾으시오

def f(x):
    return np.tan(x)

a, b, dx = (0, 30, 0.001)

while True:
    x1, x2 = rootsearch(f,a,b,dx)
    if x1 != None:
        a = x2
        rootval = bisection(f,x1,x2,1)
        if rootval != None:
            print('Root ',rootval)
    else:
        break
```

```
Root  0
Root  3.141592653751138
Root  6.283185307026343
Root  9.424777960300661
Root  12.566370614527127
Root  15.707963267799922
Root  18.84955592107778
Root  21.991148575309836
Root  25.132741228588202
Root  28.274333881866575
```

Find all roots

$x \sin x + 3 \cos x - x = 0$ in (-6, 6)

In [ ]:

```

```

3. Newton-Raphson's method

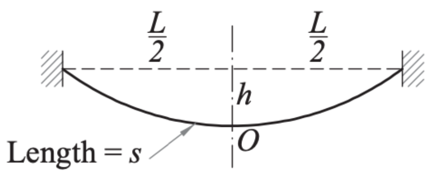
Speeds up the root search knowing the derivative

Let x be an estimate of the root of $f(x) = 0$.
Do until $|\Delta x| < \varepsilon$:
Compute $\Delta x = -f(x)/f'(x)$.

3.1 The one-equation system

In [28]:
```
from IPython.display import Image
Image(filename='PhysicsProblem3.png',width="750")
```

Out[28]:



A cable is suspended as shown in the figure. Its length s and the sag h are related to the span L by

$$s = \frac{2}{\lambda}\sinh\frac{\lambda L}{2} \qquad h = \frac{1}{\lambda}\left(\cosh\frac{\lambda L}{2} - 1\right)$$

where

$$\lambda = w_0/T_0$$
$$w_0 = \text{weight of cable per unit length}$$
$$T_0 = \text{cable tension at } O$$

Compute s for $L = 160$ m and $h = 15$ m.

In [19]:

```
import numpy as np
from math import sinh
from newtonRaphson import *

L = 160 # m


# h(lam)  = (1/lam)*(cosh(lam*L/2)-1) = 15
# f(lam)  = (1/lam)*(cosh(lam*L/2)-1) - 15

def f(lam):
    L = 160 # m
    from math import cosh
    return (cosh(lam*L/2)-1)/lam - 15

def ft(lam):
    L = 160 # m
    from math import sinh
    return (L/2)*sinh(lam*L/2)/lam
```

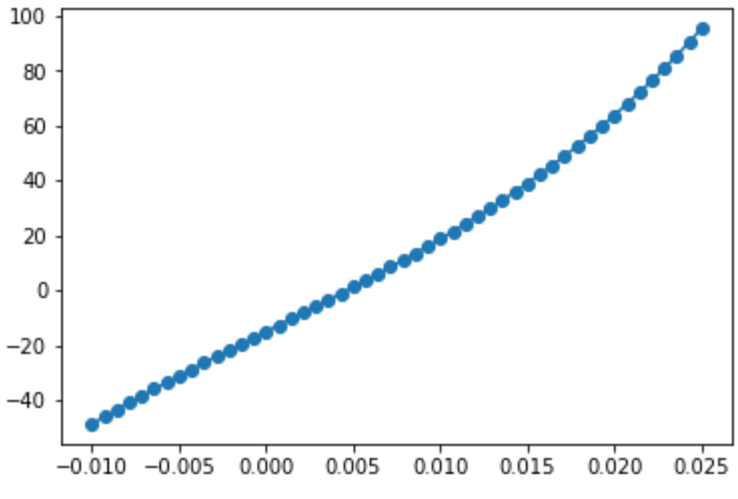
In [26]:

```
a = -0.01
b = 0.025
root = newtonRaphson(f,ft,a,b,tol=1.0e-11)
print('root, f(root) ',root,f(root))
```

```
root, f(root)  0.004634177953423561 2.6242290118716483e-08
```

In [17]:

```
import matplotlib.pyplot as plt
import numpy as np
xx = np.linspace(-0.01,0.025)
yy = [f(xx0) for xx0 in xx]
plt.plot(xx,yy,'-o')
plt.show()
```



In [87]:
```
x = np.array([0.1])
#mylam = newtonRaphson(f,x)

root = newtonRaphson(f,df,a,b,tol=1.0e-9).
mys = 2/mylam*sinh(mylam*L/2)

print('s = ',mys[0],'/m')
#input("\n Press return to exit")
```

```
s =  163.69044030096188 /m
```

3.2. Systems of equations

In [74]:
```
from IPython.display import Image
Image(filename='Theory_Systems_of_Equations.png',width="550")
```

Out[74]:

Estimate the solution vector $\mathbf{x}$.

Do until $|\Delta\mathbf{x}| < \varepsilon$:

  Compute the matrix $\mathbf{J}(\mathbf{x})$

  Solve $\mathbf{J}(\mathbf{x})\,\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x})$ for $\Delta\mathbf{x}$.

  Let $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}$.

$$\mathbf{f}(\mathbf{x}+\Delta\mathbf{x}) \approx \mathbf{f}(\mathbf{x}) + \mathbf{J}(\mathbf{x})\,\Delta\mathbf{x} \;= \mathbf{0}$$

where $\mathbf{J}(\mathbf{x})$ is the *Jacobian matrix* $J_{ij} = \dfrac{\partial f_i}{\partial x_j} \approx \dfrac{f_i(\mathbf{x}+\mathbf{e}_j h) - f_i(\mathbf{x})}{h}$

In [3]:
```
'''
soln = newtonRaphson2(f,x,tol=1.0e-9).
Solves the simultaneous equations f(x) = 0 by
the Newton-Raphson method using {x} as the initial guess.
Note that {f} and {x} are vectors.
'''

from newtonRaphson2 import *
import numpy as np

x0 = np.array([1.0,1.0,1.0])


def fv(xx) :
    # We declare a vector array
    #fv = np.array([0,0,0])
    #fv = np.zeros(3)
    fv = np.zeros(len(xx))

# We assign the vector elements to x,y,z variables.
    x = xx[0]
    y = xx[1]
    z = xx[2]

# We define the different function elements
    fv[0] = np.sin(x) + y**2.0 + np.log(z) - 7.0
    fv[1] = 3.0*x  + 2.0**y - z**3.0 + 1.0
    fv[2] = x + y + z - 5.0

    return fv

soln = newtonRaphson2(fv,x0,tol=1.0e-9)
```

In [4]:
```
soln
```

Out[4]:
```
array([0.59905376, 2.3959314 , 2.00501484])
```

In [75]:
```
from IPython.display import Image
Image(filename='Example4_8.png',width="650")
```

Out[75]:

EXAMPLE 4.8

Determine the points of intersection between the circle $x^2 + y^2 = 3$ and the hyperbola $xy = 1$.

Solution. The equations to be solved are

$$f_1(x, y) = x^2 + y^2 - 3 = 0 \tag{a}$$

$$f_2(x, y) = xy - 1 = 0 \tag{b}$$

The Jacobian matrix is

$$\mathbf{J}(x, y) = \begin{bmatrix} \partial f_1/\partial x & \partial f_1/\partial y \\ \partial f_2/\partial x & \partial f_2/\partial y \end{bmatrix} = \begin{bmatrix} 2x & 2y \\ y & x \end{bmatrix}$$

Thus the linear equations $\mathbf{J}(\mathbf{x})\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x})$ associated with the Newton-Raphson method are

$$\begin{bmatrix} 2x & 2y \\ y & x \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} -x^2 - y^2 + 3 \\ -xy + 1 \end{bmatrix} \tag{c}$$

In [9]:

```
from newtonRaphson2 import *
import numpy as np

x0 = np.array([3.0,0.1])
x0 = np.array([1.0,3.0])
x0 = np.array([-1.0,-3.0])


def fv(xx) :
    # We declare a vector array
    #fv = np.array([0,0,0])
    #fv = np.zeros(3)
    fv = np.zeros(len(xx))

# We assign the vector elements to x,y,z variables.
    x = xx[0]
    y = xx[1]
#     z = xx[2]

# We define the different function elements
    fv[0] = x**2.0 + y**2.0 - 3.0
    fv[1] = x*y - 1.0
#     fv[2] = x + y + z - 5.0

    return fv

soln = newtonRaphson2(fv,x0,tol=1.0e-9)

print('Solution ',soln)
soln[0]**2 + soln[1]**2
```

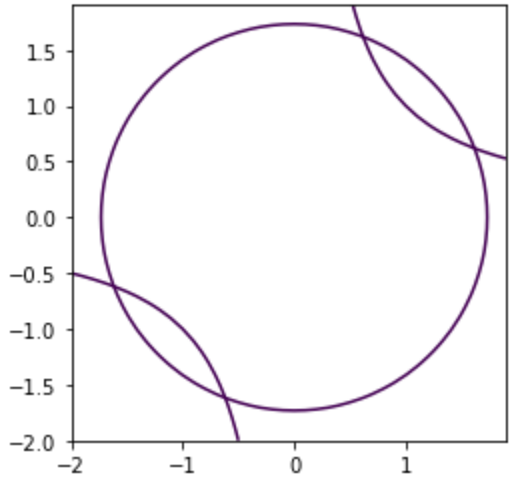
```
Solution  [-0.61803399 -1.61803399]
```

Out[9]: 2.999999999999664

In [9]:

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

# x^2 + y^2 - 3 =0
# x*y - 1 = 0

# f(x,y)
lim = 2.0
delta = 0.1
xdata = np.arange(-lim, lim, delta)
ydata = np.arange(-lim, lim, delta)

X, Y = np.meshgrid(xdata,ydata)

def f1(x,y):
    return x**2.0 + y**2.0 - 3

def f2(x,y):
    return x*y - 1

F1 = f1(X,Y)
F2 = f2(X,Y)

fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_aspect('equal')

plt.contour(X,Y,F1,[0])
plt.contour(X,Y,F2,[0])
```

Out[9]: <matplotlib.contour.QuadContourSet at 0x7ff7481f3a90>



In [10]:

```
# root(func,xini). Build in scipy library for root finding of multi-equation

def f1(x):
    return x[0]**2.0 + x[1]**2.0 - 3

def f2(x):
    return x[0]*x[1] - 1

def func(x):
    return [f1(x),f2(x)]

x0 = [1,0]
from scipy.optimize import root

sol = root(func,x0)
sol.x
```

Out[10]:

```
array([1.61803399, 0.61803399])
```

Another similar problem

$$x + (x - y)^3/2 - 1 = 0$$

$$(y - x)^3/2 + y = 0$$

In [11]:

```
# Use scipy library
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.root.ht

def fun(x):
    return [x[0]  + 0.5 * (x[0] - x[1])**3 - 1.0,
            0.5 * (x[1] - x[0])**3 + x[1]]



# We can define the Jacobian if it is known

def jac(x):
    return np.array([[1 + 1.5 * (x[0] - x[1])**2,
                      -1.5 * (x[0] - x[1])**2],
                     [-1.5 * (x[1] - x[0])**2,
                      1 + 1.5 * (x[1] - x[0])**2]])


from scipy import optimize
sol = optimize.root(fun, [0, 0], jac=jac) #, method='hybr')
sol.x
```

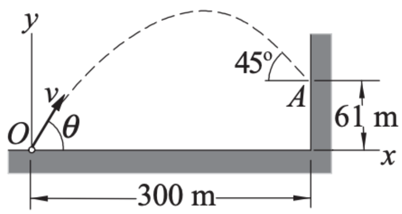
Out[11]:

```
array([0.8411639, 0.1588361])
```

In [72]:

```
from IPython.display import Image
Image(filename='PhysicsProblem2.png',width="650")
```

Out[72]: ■



A projectile is launched at O with the velocity v at the angle θ to the horizontal. The parametric equations of the trajectory are

$$x = (v\cos\theta)t$$
$$y = -\frac{1}{2}gt^2 + (v\sin\theta)t$$

where t is the time measured from the instant of launch, and $g = 9.81$ m/s^2 represents the gravitational acceleration. If the projectile is to hit the target A at the 45° angle shown in the figure, determine v, θ, and the time of flight.

In [49]:

```
import numpy as np
from math import cos, sin, pi
from newtonRaphson2 import *

def f(x):
    f = np.zeros(len(x))
    g = 9.81
    v = x[0]; theta = x[1]; t = x[2];

    f[0] = v*cos(theta)*t - 300
    f[1] = -g*t**2/2 + v*sin(theta)*t -61
    f[2] = v*cos(theta) + (-g*t + v*sin(theta)) # |v_x| = |v_y| at the target

    return f

x =np.array([60, 54*pi/180, 8])

myv, mytheta, myt = newtonRaphson2(f,x)
print('v = ',myv,'m/s,\n theta = ', mytheta*180/pi,'deg,\n t = ', myt,' sec.'
```

```
v =  60.35334598173611 m/s,
 theta =  54.59096092208302 deg,
 t =  8.578949178728887  sec.
```

4. Zeros of Polynomials

4.1 Horner's deflation

In [88]:

```
from IPython.display import Image
Image(filename='Theory_Horner_Deflation.png',width="750")
```

Out[88]:

$$\boxed{P_n(x) = (x - r)\,P_{n-1}(x)}$$

If we let

$$P_{n-1}(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$$

then Eq. (4.12) becomes

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n$$
$$= (x - r)(b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1})$$

Equating the coefficients of like powers of x, we obtain

$$b_{n-1} = a_n \qquad b_{n-2} = a_{n-1} + r b_{n-1} \qquad \cdots \qquad b_0 = a_1 + r b_1$$

which leads to *Horner's deflation algorithm:*

```
b[n-1] = a[n]
for i in range(n-2,-1,-1):
    b[i] = a[i+1] + r*b[i+1]
```

4.2 Laguerre's method

In [89]:

```
from IPython.display import Image
Image(filename='Theory_Laguerre.png',width="750")
```

Out[89]:

Let x be a guess for the root of $P_n(x) = 0$ (any value will do).
Do until $|P_n(x)| < \varepsilon$ or $|x - r| < \varepsilon$ (ε is the error tolerance):
Evaluate $P_n(x)$, $P_n'(x)$ and $P_n''(x)$ using `evalPoly`.
Compute $G(x)$ and $H(x)$ from Eqs. (4.14).
Determine the improved root r from Eq. (4.16) choosing the sign that results in the *larger magnitude of the denominator.*
Let $x \leftarrow r$.

$$P_n(x) = (x-r)(x-q)^{n-1} \quad = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

$$P_n'(x) = (x-q)^{n-1} + (n-1)(x-r)(x-q)^{n-2}$$
$$= P_n(x)\left(\frac{1}{x-r} + \frac{n-1}{x-q}\right)$$

$$G(x) = \frac{P_n'(x)}{P_n(x)} \quad = \frac{1}{x-r} + \frac{n-1}{x-q} \longrightarrow x - q = \frac{n-1}{G(x) - \frac{1}{x-r}}$$

Substitute

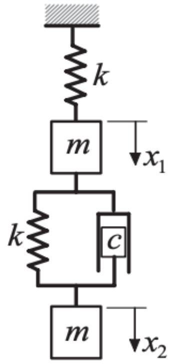
$$H(x) = \frac{P_n''(x)}{P_n(x)} - \left[\frac{P_n'(x)}{P_n(x)}\right]^2 = G^2(x) - \frac{P_n''(x)}{P_n(x)} \quad = -\frac{1}{(x-r)^2} - \frac{n-1}{(x-q)^2}$$

$$x - r = \frac{n}{G(x) \pm \sqrt{(n-1)\left[nH(x) - G^2(x)\right]}}$$

In [90]:

```
from IPython.display import Image
Image(filename='PhysicsProblem5.png',width="750")
```

Out[90]: ■



The two blocks of mass m each are connected by springs and a dashpot. The stiffness of each spring is k, and c is the coefficient of damping of the dashpot. When the system is displaced and released, the displacement of each block during the ensuing motion has the form

$$x_k(t) = A_k e^{\omega_r t} \cos(\omega_i t + \phi_k),\ k = 1, 2$$

where A_k and ϕ_k are constants, and $\omega = \omega_r \pm i\omega_i$ are the roots of

$$\omega^4 + 2\frac{c}{m}\omega^3 + 3\frac{k}{m}\omega^2 + \frac{c}{m}\frac{k}{m}\omega + \left(\frac{k}{m}\right)^2 = 0$$

Determine the two possible combinations of ω_r and ω_i if $c/m = 12\ \mathrm{s}^{-1}$ and $k/m = 1\,500\ \mathrm{s}^{-2}$.

In [92]:

```
from polyRoots import *
import numpy as np

cm = 12; km=1500;
c = np.array([km**2,km*cm,3*km,2*cm,1])
root = polyRoots(c)

print('(omega_real, omega_imag) = ',[root[0].real,root[0].imag])
print('(omega_real, omega_imag) = ',[root[2].real,root[2].imag])
```

```
(omega_real, omega_imag) =  [-0.6230196283078494, -24.03024141494682]
(omega_real, omega_imag) =  [-11.37698037169215, -61.35447280658925]
```