```
# --------------------------------------------------------------------------
#                        POLYNOMIALS with NUMPY
# --------------------------------------------------------------------------

# We can find the roots of a polynomial using NUMPY

# Suppose we have the polynomial
#      x^2 + 5x + 6 =0
# We can check by hand that the roots are
#      -2 and -3
# With numpy, we do
import numpy.polynomial.polynomial as poly
# This module provides a number of objects (mostly functions)
# useful for dealing with Polynomial series, including a
# Polynomial class that encapsulates the usual arithmetic operations.
# https://docs.scipy.org/doc/numpy-1.13.0/reference/
 routines.polynomials.polynomial.html
# https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/
 numpy.polynomial.polynomial.Polynomial.html
p = poly.Polynomial([6, 5, 1])
# NOTICE that the number "6" comes FIRST.
#
# To get the roots, we can write
print(p.roots())
# or
solu = poly.Polynomial.roots(p)
print(solu)

pe = poly.Polynomial([2, 3, 6, 5, 1])
# This is the polynomial x^4 + 5x^3 + 6x^2 + 3x + 2
solu = poly.Polynomial.roots(pe)
print(solu)



# --------------------------------------------------------------------------
#                     NONLINEAR EQUATIONS with SCIPY
# --------------------------------------------------------------------------

# Nonlinear equations are harder to solve than linear equations.
# Even solving a single nonlinear equation may be challenging.
# We can use SCIPY for it

# ----------------------------
# Just ONE nonlinear equation
# ----------------------------
# x = - 2cos(x)
# We need to write it in the form of a function:
# f(x) = x + 2cos(x)
# and find its roots (zeros)
```

```python
# f(x) = 0
import math
from scipy.optimize import fsolve
def func(x):
    return x + 2*math.cos(x)
solu = fsolve(func,0.3)
# 0.3 above is a first guess around which the root will be searched
print(solu)


# SEE MORE ALTERNATIVES IN
# https://docs.scipy.org/doc/scipy/reference/optimize.html


# --------------------------------
# A system of nonlinear equations
# --------------------------------
#
#  x^2 + y^2 = 20  ( this is circle of radius sqrt(20) )
#  y = x^2  (this is a parabola)
#
# Let us first have a look at these two equations
# in a plot

# ----------
# Exercise
# ----------
# Make a plot of x vs y
# For the circle, use polar coordinates
# Number of points 300
# Show the legend for the two equations
# Range for the plot: -7<=x<=7, -7<=y<=7


# Now let us turn to SCIPY
# We need to deal with vectors (in the case above, a vector of dimension 2)
# We want to find the ZEROS (ROOTS) of both functions at the same time,
# function F0 = x^2 + y^2 - 20
# and
# function F1 = y - x^2
import numpy as np
from scipy.optimize import fsolve

def func(valIni):
    x=valIni[0]
    y=valIni[1]
#    FF = np.empty( (2) )
    FF = np.zeros(2, float)
    FF[0] = x**2 + y**2 -20
    FF[1] = y - x**2
    return FF
```

```
valGuess = np.array([1,1])
solu = fsolve(func,valGuess)
print(solu)


# ----------------------------------------------------------------------
#                  NONLINEAR EQUATIONS: NUMERICAL METHODS
# ----------------------------------------------------------------------

# But it is important to know what are the different
# existing numerical methods.
# Let us start with a simple method known as...

# ------------------
# RELAXATION METHOD
# ------------------
# We need to write the equation in the form
#     x = f(x)

# Suppose we want to solve
#       x = 2 - exp(-x)
# There is no analytic method to solve it, so we turn to computational
# methods. A simple method, that works in many cases, is to iterate
# the equation. It works as follows:
# 1) We guess an initial value,
# 2) Plug it on the right-side and get a new value x',
# 3) Repeat the process until the value converges to a FIXED POINT,
# that is, it stops changing.

# For the equation above, let us start with x=1
import math
x = 1.0
for n in range(20):
    x = 2 - math.exp(-x)
    print(n,x)
# We can see that x is converging to 1.8414...



# ------------------
# NOTE NOTE NOTE
# ------------------
#
# Let us consider the case
#     ln(x) + x^2 - 1 = 0
# We need to write it in the form x=f(x)
# which we can do by writing
#     ln(x) = 1 - x^2
# and taking the exponential of both sides
#     x = exp(-x^2 +1)
#
```

```
# We can immediately see that the SOLUTION is x=1
#
# BUT,
# if we try the RELAXATION METHOD starting with x=1/2
# we see that it does NOT converge
import math
x = 0.5
for n in range(20):
    x = math.exp(1 - x**2)
    print(n,x)




# ------------------
# TRICK TRICK TRICK
# ------------------
# In cases like this, we can try to rewrite the equation,
# for example, by applying "ln" to isolate "x" from the right side
# ln(x) = 1 - x^2
# x^2 = 1 - ln(x)
# x = sqrt[ 1 - ln(x) ]
#
# Starting with this equation and x=1/2, we now see convergence
import math
x = 0.5
for n in range(20):
    x = math.sqrt(1 - math.log(x))
    print(n,x)
# We can see that x is converging to 1.


# ---------------------
# Why the Method works
# ---------------------
#
# Taylor expansion of f(x) around the root x*
# f(x) = f(x*) + f'(x*) (x - x*) + ...
# but we also have that the new estimate x' comes from
# x' = f(x)
# so we can approximate
# x' = f(x*) + f'(x*) (x - x*)
# We also now by definition that f(x*) = x*, so
# x' = x* + f'(x*) (x - x*)
# (x' - x*) = f'(x*) (x - x*)
# This means that if f'(x*)<1, at each iteration,
# (x' - x*) becomes smaller than (x - x*) and so
# x' converges to x*

# If there is no convergence, it is because f'(x*)>1.
# This can be fixed by changing from
```

```
# x = f(x) to f^(-1)(x) = x
# Convergence is guaranteed if the derivative of
# the inverse function f^(-1)(x) at x* is smaller than 1.
# This will hold, because
# Derivative[f^(-1)(x)] = 1/Derivative[f(x)]
# How do we know this?
# Call u = f^(-1)(x), so
# the derivative of f^(-1)(x) is
#     du/dx
# From above, we also have that
#     f(u) = x, so
# the derivative of f(x) is
#     dx/du which is the reciprocal of du/dx.



# ------------------
# PROBLEM
# ------------------
# But sometimes we cannot find the inverse f^{-1}(x) to
# guarantee convergence. In this case,
# we need to resort to another method to solve the
# nonlinear equation


# ------------------
# ERROR ERROR ERROR
# ------------------

# The error between the estimate x
# and the next estimate x'

# 1) To find an expression for the error, let us start with
# the Taylor expansion of the function f(x) around the
# correct solution x*
# f(x) = f(x*) + f'(x=x*) (x-x*) + ...
# but we also know that the new value x' in the iteration is
# x' = f(x)
# so
# x' = f(x*) + f'(x=x*) (x-x*) + ...
# We also know, by definition, that x* = f(x*), so up to 1st order
# we have
# x' = x* + f'(x=x*) (x-x*)
# then
# x'-x* = (x-x*) f'(x*)
#
# 2) Suppose that our current estimate of the solution is x and the
# next estimate, after the iteration, is x'
# Let us call "e" the error between x* and x
# x* = x + e
# and e' the error between x* and x'
```

```
# x* = x' + e'
#
# 3) We can then rewrite
# x'-x* = (x-x*) f'(x*)
# as
# -e' = -e f'(x*)  so   e = e'/f'(x*)
#
# 4) Coming back to x* and using e from above,
# x* = x+e = x + e'/f'(x*)
# We also have that
# x* = x'+e'
# so
# x'+e' = x + e'/f'(x*)
# x-x' = e' (1 - 1/f'(x*))
#
# 5) From the steps above and
# making the approximation f'(x*) ~ f'(x), we find that
# the expression for the error (e') on the
# new estimate x' is
#   error ~ (x - x')/[1 - 1/f'(x) ]
#
# We can then repeat the iteration until the magnitude
# of the estimated error falls below some target value
#
# For the example above

import sympy as syp
x = syp.Symbol('x')
ff = syp.sqrt(1 - syp.log(x))
gg = syp.diff(ff,x)
print(gg)

der = syp.lambdify(x,gg)

import math
x = 0.5
for n in range(20):
    xold = x
    x = math.sqrt(1 - math.log(x))
    xnew = x
    error = (xold - xnew)/(1 - 1/der(xold))
    print(n,xnew,error)


# -----------------------------------
# Suppose we wanted an error <10^(-6)
# -----------------------------------
import sympy as syp
x = syp.Symbol('x')
ff = syp.sqrt(1 - syp.log(x))
gg = syp.diff(ff,x)
```

```
#print(gg)
der = syp.lambdify(x,gg)

import math
x = 0.5
accu=1.e-6
error=1.
howmany=0
while abs(error)>accu:
    xold = x
    x = math.sqrt(1 - math.log(x))
    xnew = x
    error = (xold - xnew)/(1 - 1/der(xold))
    howmany = howmany + 1
    print(howmany,xnew,error)



# ------------------------------
# EXERCISE 6.3:  FERROMAGNETISM
# ------------------------------

# In the mean-field theory of ferromagnetism, the strength M of magnetization
# of a ferromagnet material like iron depends on the temperature T according to
# the formula
#                M = mu tanh(JM/kT)
# where mu is the magnetic moment, J is a coupling constant, and k is
 Boltzmann's
# constant. To simplify a little, let us make the substitution m = M/mu and
# C = mu J/k, so that
#                m = tanh(Cm/T).

# This equation always has a solution at m = 0, which implies a material that
# is not magnetized.


import numpy as np
import matplotlib.pyplot as plt
# Looking at a plot of tanh
tot=300
xvalues = np.linspace(-5,5,tot)
yvalues=[]
for n in range(tot):
    yvalues.append( math.tanh(xvalues[n]) )
plt.plot(xvalues,yvalues)
plt.show()

# But are there solutions for m = tanh(Cm/T) for m != 0?
# There is no known method for solving this equation exactly,
# but we can do it numerically.
#
```

```
# Let us assume that C=1 and look for solution as a function of T accurate to
# within 10^(-6) of the true answer.
#
# Note: tanh and cosh are functions from "math".
#
# For each value of the temperature between T=0.01 and the maximum value
# Tmax=2, start with m=1 and iterate the equation until the magnitude of the
# error falls below the target value 10^(-6).
# Study 1000 values between T=0.1 and T=2.
#
# Make a plot of m vs T
# You should see that m becomes abruptly = 0 for T>1.
# At this point, we have a PHASE TRANSITION.
# T=1 is called the CRITICAL TEMPERATURE of the magnet.



# ------------------------------------------------------------------------

#            See PDF notes and/or the chapter 6 of the book for
#                     the methods below

# ------------------------------------------------------------------------


# -----------------
# BINARY SEARCH
# -----------------
# To solve nonlinear equations with a SINGLE variable x.



# -------------------
# REGULA FALSI METHOD
# -------------------
# A better way to decrease the interval around the root
# than the bisection method


# -----------------
# NEWTON'S METHOD
# -----------------
# This method converges faster to the solution than the
# relaxation method or the binary search.

# ----------------------------------------
# EXERCISE 6.4:  INVERSE HYPERBOLIC TANGENT
# ----------------------------------------
# a) Let us use Newton's method to calculate the inverse (or arc)
# hyperbolic tangent of a number u, such that
#       u = tanh(x)
```

```
# This is equivalent to saying that  x is a root of the equation
#        tanh(x) - u = 0

# Start from an initial guess x=0
# Choose as accuracy 10^(-12)

# Remember that the derivative of tanh(x) is 1/cosh^2(x), so that
# equation for the new guess x' is
#        x' = x - (tanh(x) - u) cosh^2(x)

# b) Make a plot of arctanh(u) for 100 values of u
#   between -0.99 and 0.99



# ------------------
# SECANT METHOD
# ------------------




# -----------
# EXERCISES
# -----------

# Using the Bisection Method and SCIPY, solve
# (i) x Exp[x] =1
# (ii) Cos[x] = x

# Using the Method of False Position and SCIPY, solve
# (i) Tan[x] = 1/(1+x^2)     0 <= x < Pi/2
# (ii) Cos[x] = x
# [comparing with item (ii) above for the bisection method,
# which method works faster for this case?]

# Using Newton's Method and SCIPY find the real zero of:
# (i) ArcTan[x] =1    start with x=1
# (ii)  Log[x] = 3    start with x=10



# ------------------------------------------
# NEWTON'S METHOD for TWO or MORE VARIABLES
# ------------------------------------------


# -----------
# EXERCISE
# -----------
```

```
# Using Newton's Method find the solutions for
# f(x,y) = exp(3x)+4y
# g(x,y) = 3y^3 - 2 ln(x) + 7.31 x^2
# Use as an initial guess xo=1 and yo=2
# Stop when |f| and |g| are smaller than 10^(-5)




# ------------------
# MAXIMA and MINIMA
# ------------------

# To find either the local or global minima or maxima of a function,
# we differentiate it and set it equal to zero. We then just have to
# find the roots of the differentiated function.
# If the function has many variables, we do the partial derivative of each
# and solve the set of equations.


# ---------------------
# GOLDEN RATIO SEARCH
# ---------------------

# An alternative method to find a minimum or maximum of a
# function of a SINGLE variable.
# But it will not tell us whether it is a global or local
# minimum (maximum).
```