```python
#############################################################################
###Code for the Monte-Carlo estimation of the convergence order for the
###Ornstein-Uhlenbeck process
###MC_ErrorAnalysis_ConvergenceOrderOU.py
###Python 2.7
#############################################################################

import numpy as np
import matplotlib.pylab as plt
from scipy.stats import linregress
from NumericalSDE import *

#################################################
### Ornstein-Uhlenbeck process (OU)
### dXt = a(Xt)dt + b(Xt)dWt
### X0 = x0
### a(x) = -beta*x, b(x) = sigma
### beta, sigma positive constants
### True solution:
###
#################################################
#Parameter
sigma = 1.5
beta = 1.0
#functions a, b
def a(x):
    return -beta*x
def b(x):
    return sigma
#derivativs of a, b
#def a_dv(x):
#    return -beta
def b_dv(x):
    return 0
#def a_dvdv(x):
#    return 0
#def b_dvdv(x):
#    return 0
#starting value x0
x0 = 1
############################################################################
#Number of steps. We will do the Monte-Carlo-analysis for the
#step amounts n in nval() starting with n we will then always
#double the amount of steps.
n = 8
nval = np.array([n, n*2, n*4, n*8, n*16, n*32, n*64, n*128])
nsize = np.size(nval)
#Number of MC simulations for each step amount n
nsim = 20000

#############################################################################
#Arrays for the errors for each simulation and corresponding n.
#we are collecting the errors of the Euler, Milstein and Wagner-Platen-scheme.
error_euler = np.zeros((nsim,nsize))
error_milstein = np.zeros((nsim,nsize))
#error_wagnerplaten = np.zeros((nsim,nsize))
#empty arrays
Yt_euler = np.zeros(nsize)
Yt_milstein = np.zeros(nsize)
#Yt_wagnerplaten = np.zeros(nsize)
Xt_T = np.zeros(nsize)
stochIntApprox = 0

##############################################################################
#Monte-Carlo-algorithm
for m in range(0,nsim):
    w = wiener(nval[0])
```

```
    t = timegrid(nval[0])
    for k in range(0,nsize):
        Yt_euler = sde_euler(x0,a,b,w)
        Yt_milstein = sde_milstein(x0,a,b,b_dv,w)
        #Yt_wagnerplaten = sde_wagnerplaten(x0,a,b,a_dv,b_dv,a_dvdv,b_dvdv,w)

        for j in range(0,nval[k]):
            stochIntApprox = stochIntApprox + (w[j+1]-w[j])*np.exp((-beta)*((t[nval[k]])-t[j]))


        Xt_T = x0*np.exp((-beta)*(t[nval[k]])) + sigma*stochIntApprox

        w = refineWiener(w)
        t = timegrid(nval[k]*2)
        stochIntApprox = 0

        #Squared-mean criterion for the error
        error_euler[m,k] = (Yt_euler[nval[k]] - Xt_T)**2
        error_milstein[m,k] = (Yt_milstein[nval[k]] - Xt_T)**2
        #error_wagnerplaten[m,k] = (Yt_wagnerplaten[nval[k]] - Xt_T)**2

    print str(float(m)/nsim*100) + '%'#progress
##############################################################################

#Monte-Carlo estimates of the errors
mc_error_euler=np.zeros(nsize)
mc_error_milstein=np.zeros(nsize)
#mc_error_wagnerplaten=np.zeros(nsize)
for k in range(0,nsize):
    mc_error_euler[k] = sqrt(np.mean(error_euler[:,k]))
    mc_error_milstein[k] = sqrt(np.mean(error_milstein[:,k]))
    #mc_error_wagnerplaten[k] = sqrt(np.mean(error_wagnerplaten[:,k]))

#Regression (Example: a[0] returns the slope and a[1] returns the intersect)
a = linregress(np.log2(nval), np.log2(mc_error_euler))
b = linregress(np.log2(nval), np.log2(mc_error_milstein))
#c = linregress(np.log2(nval), np.log2(mc_error_wagnerplaten))

#log-log-plot: error estimates and step amount n
plt.figure(1)
plt.scatter(np.log2(nval), np.log2(mc_error_euler), 3, c='b', label="Euler-scheme. Slope: " + str("{0:.3f}".format(a[0])))
plt.scatter(np.log2(nval), np.log2(mc_error_milstein), 3, c='m', label="Milstein-scheme. Slope: " + str("{0:.3f}".format(b[0])))
#plt.scatter(np.log2(nval), np.log2(mc_error_wagnerplaten), 2, c='forestgreen', label="Wagner-Platen-scheme. Slope: " + str("{0:.3f}".format(c[0])))
plt.xlabel('$log_2$ n', fontsize=12)
plt.ylabel('$log_2$ error', fontsize=12)
plt.legend(loc='lower left')
#plot regression line
plt.plot(np.log2(nval), np.log2(nval)*a[0]+a[1], c='b',alpha=0.5)
plt.plot(np.log2(nval), np.log2(nval)*b[0]+b[1], c='m',alpha=0.5)
#plt.plot(np.log2(nval), np.log2(nval)*c[0]+c[1], c='forestgreen')

#Second plot: Error analysis
plt.figure(2)
plt.plot(np.log2(nval), mc_error_euler, c='b',alpha=0.5, label="Euler-scheme")
plt.plot(np.log2(nval), mc_error_milstein, c='m',alpha=0.5, label="Milstein-scheme")
#plt.plot(np.log2(nval), mc_error_wagnerplaten, c='forestgreen', label="Wagner-Platen-scheme")
plt.xlabel('$log_2$ n', fontsize=12)
plt.ylabel('error', fontsize=12)
plt.legend(loc='upper right')

plt.show()
```