```
################################################################################
###Plots the true solution and the approximation on n (and 2*n) time-points for the
###Ornstein-Uhlenbeck process
###Euler-Maruyama, Milstein (optional: Wagner-Platen)
###SDE_ApproximationOU.py
###Python 2.7
################################################################################

import numpy as np
import matplotlib.pylab as plt
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.
n = 32
#Wiener process w, discretization of [0,T] t
w = wiener(n)
t = timegrid(n)
#Finer versions(using refinmenent)
w2 = refineWiener(w)
t2 = timegrid(2*n)
#################################################
###############Numerical solutions###############
#################################################
#Numerical solution: Euler-scheme
Yt_euler = sde_euler(x0,a,b,w)
Yt2_euler = sde_euler(x0,a,b,w2)
#Numerical solution: Milstein-scheme
Yt_milstein = sde_milstein(x0, a, b, b_dv, w)
Yt2_milstein = sde_milstein(x0, a, b, b_dv, w2)
#Numerical solution: Wagner-Platen-scheme
Yt_wagnerplaten = sde_wagnerplaten(x0, a, b, a_dv, b_dv, a_dvdv, b_dvdv, w)
Yt2_wagnerplaten = sde_wagnerplaten(x0, a, b, a_dv, b_dv, a_dvdv, b_dvdv, w2)

#Analytical solution (Ornstein-Uhlenbeck process) (adapt this for other SDEs.)
temp = np.zeros(n+1)
Xt=np.zeros(n+1)
stochInt = np.zeros(n+1)

```

```
temp2 = np.zeros(2*n+1)
Xt2=np.zeros(2*n+1)
stochInt2 = np.zeros(2*n+1)

Xt[0] = x0
Xt2[0] = x0
stochInt[0] = 0
stochInt2[0] = 0
temp[0] = 0
temp2[0] = 0
for k in range(0,n):
    for j in range(0,k):
        temp[j+1] =(w[j+1]-w[j])*np.exp((-beta)*((t[k+1])-t[j]))
    stochInt[k+1] = np.sum(temp)

    Xt[k+1] = Xt[0]*np.exp((-beta)*(t[k+1])) + sigma*stochInt[k+1]




for k in range(0,2*n):
    #Xt2[k+1] = Xt[0]*np.exp((mu-sigma**2/2)*(t2[k+1]) + sigma*w2[k+1])
    for j in range(0,k):
        temp2[j+1] =(w2[j+1]-w2[j])*np.exp((-beta)*((t2[k+1])-t2[j]))
    stochInt2[k+1] = np.sum(temp2)

    Xt2[k+1] = Xt2[0]*np.exp((-beta)*(t2[k+1])) + sigma*stochInt2[k+1]
#plot
plt.figure(1)
plt.scatter(t, Yt_euler, 1, c='b', label="Euler-method")
plt.scatter(t, Xt, 1, c='k', label="Analytical solution")
plt.fill_between(t, Xt, Yt_euler, color='b',alpha=.1, interpolate=False)
plt.legend(loc='upper left')
ax = plt.gca()
plt.text(0.025, 0.80,'n= ' + str(n), bbox=dict(facecolor='white'), transform = ax.transAxes)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)

plt.figure(2)
plt.scatter(t, Yt_milstein, 1, c='m', label="Milstein-method")
plt.scatter(t, Xt, 1, c='k', label="Analytical solution")
plt.fill_between(t, Xt, Yt_milstein, color='m',alpha=.1, interpolate=False)
plt.legend(loc='upper left')
ax = plt.gca()
plt.text(0.025, 0.80,'n= ' + str(n), bbox=dict(facecolor='white'), transform = ax.transAxes)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)

##plt.figure(3)
##plt.scatter(t, Yt_wagnerplaten, 1, c='forestgreen', label="Wagner-Platen-method")
##plt.scatter(t, Xt, 1, c='k', label="Analytical solution")
##plt.fill_between(t, Xt, Yt_wagnerplaten, color='forestgreen',alpha=.1, interpolate=False)
##plt.legend(loc='upper left')
##ax = plt.gca()
##plt.text(0.025, 0.80,'n= ' + str(n), bbox=dict(facecolor='white'), transform = ax.transAxes)
##plt.xlabel('t', fontsize=16)
##plt.ylabel('x', fontsize=16)

plt.figure(4)
plt.scatter(t2, Yt2_euler, 1, c='b', label="Euler-method")
plt.scatter(t2, Xt2, 1, c='k', label="Analytical solution")
plt.fill_between(t2, Xt2, Yt2_euler, color='b',alpha=.1, interpolate=False)
plt.legend(loc='upper left')
```

```
ax = plt.gca()
plt.text(0.025, 0.80,'n= ' + str(2*n), bbox=dict(facecolor='white'), transform = ax.transAxes)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)

plt.figure(5)
plt.scatter(t2, Yt2_milstein, 1, c='m', label="Milstein-method")
plt.scatter(t2, Xt2, 1, c='k', label="Analytical solution")
plt.fill_between(t2, Xt2, Yt2_milstein, color='m',alpha=.1, interpolate=False)
plt.legend(loc='upper left')
ax = plt.gca()
plt.text(0.025, 0.80,'n= ' + str(2*n), bbox=dict(facecolor='white'), transform = ax.transAxes)
plt.xlabel('t', fontsize=16)
plt.ylabel('x', fontsize=16)

##plt.figure(6)
##plt.scatter(t2, Yt2_wagnerplaten, 1, c='forestgreen', label="Wagner-Platen-method")
##plt.scatter(t2, Xt2, 1, c='k', label="Analytical solution")
##plt.fill_between(t2, Xt2, Yt2_wagnerplaten, color='forestgreen',alpha=.1, interpolate=False)
##plt.legend(loc='upper left')
#ax = plt.gca()
#plt.text(0.025, 0.80,'n= ' + str(2*n), bbox=dict(facecolor='white'), transform = ax.transAxes)
##plt.xlabel('t', fontsize=16)
##plt.ylabel('x', fontsize=16)

plt.show()
```