

```

1 #####
2 ###Plots the true solution and the approximation on n (and 2*n) time-points for the
3 ###Ornstein-Uhlenbeck process
4 ###Euler-Maruyama, Milstein (optional: Wagner-Platen)
5 ###SDE_ApproximationOU.py
6 ###Python 2.7
7 #####
8
9 import numpy as np
10 import matplotlib.pyplot as plt
11 from NumericalSDE import *
12
13 #####
14 ### Ornstein-Uhlenbeck process (OU)
15 ###  $dX_t = a(X_t)dt + b(X_t)dW_t$ 
16 ###  $X_0 = x_0$ 
17 ###  $a(x) = -\text{beta} \cdot x$ ,  $b(x) = \text{sigma}$ 
18 ### beta, sigma positive constants
19 ### True solution:
20 ###
21 #####
22 #Parameter
23 sigma = 1.5
24 beta = 1.0
25 #functions a, b
26 def a(x):
27     return -beta*x
28 def b(x):
29     return sigma
30 #derivatives of a, b
31 def a_dv(x):
32     return -beta
33 def b_dv(x):
34     return 0
35 def a_dv dv(x):
36     return 0
37 def b_dv dv(x):
38     return 0
39 #starting value x0
40 x0 = 1
41 #####
42 # Number of steps.
43 n = 32
44 #Wiener process w, discretization of [0,T] t
45 w = wiener(n)
46 t = timegrid(n)
47 #Finer versions(using refinment)
48 w2 = refineWiener(w)
49 t2 = timegrid(2*n)
50 #####
51 #####Numerical solutions#####
52 #####
53 #Numerical solution: Euler-scheme
54 Yt_euler = sde_euler(x0,a,b,w)
55 Yt2_euler = sde_euler(x0,a,b,w2)
56 #Numerical solution: Milstein-scheme
57 Yt_milstein = sde_milstein(x0, a, b, b_dv, w)
58 Yt2_milstein = sde_milstein(x0, a, b, b_dv, w2)
59 #Numerical solution: Wagner-Platen-scheme
60 Yt_wagnerplaten = sde_wagnerplaten(x0, a, b, a_dv, b_dv, a_dv dv, b_dv dv, w)
61 Yt2_wagnerplaten = sde_wagnerplaten(x0, a, b, a_dv, b_dv, a_dv dv, b_dv dv, w2)
62
63 #Analytical solution (Ornstein-Uhlenbeck process) (adapt this for other SDEs.)
64 temp = np.zeros(n+1)
65 Xt=np.zeros(n+1)
66 stochInt = np.zeros(n+1)
67

```

```

68 temp2 = np.zeros(2*n+1)
69 Xt2=np.zeros(2*n+1)
70 stochInt2 = np.zeros(2*n+1)
71
72 Xt[0] = x0
73 Xt2[0] = x0
74 stochInt[0] = 0
75 stochInt2[0] = 0
76 temp[0] = 0
77 temp2[0] = 0
78 for k in range(0,n):
79     for j in range(0,k):
80         temp[j+1] =(w[j+1]-w[j])*np.exp((-beta)*((t[k+1])-t[j]))
81         stochInt[k+1] = np.sum(temp)
82
83     Xt[k+1] = Xt[0]*np.exp((-beta)*(t[k+1])) + sigma*stochInt[k+1]
84
85
86
87
88 for k in range(0,2*n):
89     #Xt2[k+1] = Xt[0]*np.exp((mu-sigma**2/2)*(t2[k+1]) + sigma*w2[k+1])
90     for j in range(0,k):
91         temp2[j+1] =(w2[j+1]-w2[j])*np.exp((-beta)*((t2[k+1])-t2[j]))
92         stochInt2[k+1] = np.sum(temp2)
93
94     Xt2[k+1] = Xt2[0]*np.exp((-beta)*(t2[k+1])) + sigma*stochInt2[k+1]
95 #plot
96 plt.figure(1)
97 plt.scatter(t, Yt_euler, 1, c='b', label="Euler-method")
98 plt.scatter(t, Xt, 1, c='k', label="Analytical solution")
99 plt.fill_between(t, Xt, Yt_euler, color='b',alpha=.1, interpolate=False)
100 plt.legend(loc='upper left')
101 ax = plt.gca()
102 plt.text(0.025, 0.80,'n= ' + str(n), bbox=dict(facecolor='white'), transform =
ax.transAxes)
103 plt.xlabel('t', fontsize=16)
104 plt.ylabel('x', fontsize=16)
105
106 plt.figure(2)
107 plt.scatter(t, Yt_milstein, 1, c='m', label="Milstein-method")
108 plt.scatter(t, Xt, 1, c='k', label="Analytical solution")
109 plt.fill_between(t, Xt, Yt_milstein, color='m',alpha=.1, interpolate=False)
110 plt.legend(loc='upper left')
111 ax = plt.gca()
112 plt.text(0.025, 0.80,'n= ' + str(n), bbox=dict(facecolor='white'), transform =
ax.transAxes)
113 plt.xlabel('t', fontsize=16)
114 plt.ylabel('x', fontsize=16)
115
116 ##plt.figure(3)
117 ##plt.scatter(t, Yt_wagnerplatten, 1, c='forestgreen', label="Wagner-Platen-method")
118 ##plt.scatter(t, Xt, 1, c='k', label="Analytical solution")
119 ##plt.fill_between(t, Xt, Yt_wagnerplatten, color='forestgreen',alpha=.1,
interpolate=False)
120 ##plt.legend(loc='upper left')
121 ##ax = plt.gca()
122 ##plt.text(0.025, 0.80,'n= ' + str(n), bbox=dict(facecolor='white'), transform =
ax.transAxes)
123 ##plt.xlabel('t', fontsize=16)
124 ##plt.ylabel('x', fontsize=16)
125
126 plt.figure(4)
127 plt.scatter(t2, Yt2_euler, 1, c='b', label="Euler-method")
128 plt.scatter(t2, Xt2, 1, c='k', label="Analytical solution")
129 plt.fill_between(t2, Xt2, Yt2_euler, color='b',alpha=.1, interpolate=False)
130 plt.legend(loc='upper left')

```

```

131 ax = plt.gca()
132 plt.text(0.025, 0.80, 'n= ' + str(2*n), bbox=dict(facecolor='white'), transform =
    ax.transAxes)
133 plt.xlabel('t', fontsize=16)
134 plt.ylabel('x', fontsize=16)
135
136 plt.figure(5)
137 plt.scatter(t2, Yt2_milstein, 1, c='m', label="Milstein-method")
138 plt.scatter(t2, Xt2, 1, c='k', label="Analytical solution")
139 plt.fill_between(t2, Xt2, Yt2_milstein, color='m', alpha=.1, interpolate=False)
140 plt.legend(loc='upper left')
141 ax = plt.gca()
142 plt.text(0.025, 0.80, 'n= ' + str(2*n), bbox=dict(facecolor='white'), transform =
    ax.transAxes)
143 plt.xlabel('t', fontsize=16)
144 plt.ylabel('x', fontsize=16)
145
146 ##plt.figure(6)
147 ##plt.scatter(t2, Yt2_wagnerplatten, 1, c='forestgreen', label="Wagner-Platen-method")
148 ##plt.scatter(t2, Xt2, 1, c='k', label="Analytical solution")
149 ##plt.fill_between(t2, Xt2, Yt2_wagnerplatten, color='forestgreen', alpha=.1,
    interpolate=False)
150 ##plt.legend(loc='upper left')
151 #ax = plt.gca()
152 #plt.text(0.025, 0.80, 'n= ' + str(2*n), bbox=dict(facecolor='white'), transform =
    ax.transAxes)
153 ##plt.xlabel('t', fontsize=16)
154 ##plt.ylabel('x', fontsize=16)
155
156 plt.show()
157

```