Numerical Implementation of Stochastic Differential Equations

Mustapha Bousakla

Exercice 1

Consider the stochastic differential equation $\dot{x} = -ax + \sqrt{D}\xi(t)$ where $\xi(t)$ is a Gaussian white noise of zero mean and correlation $\langle \xi(t)\xi(t')\rangle = \delta(t-t')$.. We will consider a = 2 and D = 0.05.

a) Trajectories: Integrate numerically the equation using a time step $h = 0.001$ writing data at time intervals $t = 0.02$. Plot 10 trajectories in the time interval [0, 5] starting from the initial condition $x(0) = 1$ together with the deterministic trajectory. Identify an initial dynamical regime dominated by deterministic dynamics and a stationary regime in which the system fluctuates around the steady state.

Let's briefly present the Milshtein algorithm for a general stochastic differential equation and Gaussian noise of the form:

$$\dot{x}(t) = q(x(t)) + g(x(t))\xi(t) \quad ; \quad \langle \xi(t)\rangle = 0 \quad ; \quad \langle \xi(t)\xi(t')\rangle = \delta(t-t')$$

Since in our case $g(x) = \sqrt{D}$ is a constant, at each time step the integrated equation becomes:

$$x(t_{i+1}) = x(t_i) + hq(x(t_i) + g(x(t_i)))h^{1/2}u_i \tag{1}$$

being u_i a random Gaussian number with 0 mean and variance 1. Ten different trajectories with initial condition $x(0) = 1$ are depicted in Figure 1 along with the deterministic solution $x(t) = x(0)e^{-at}$, the one coming from the ordinary differential equation (that is, without the stochastic term $g(x)\xi(t)$). Since we are considering $h = 0.001$ and we are taking measures every $\Delta t = 0.02$, we must take these measures every $0.02/0.001 = 20$ steps. Moreover, we need 5000 steps to reach $t = 5$ and hence we take $5000/20 = 250$ measures to plot each trajectory (that is what I call *nsteps* in the code).

It can be noticed that in the interval [0,2] the fluctuations are not very big and hence the deterministic solution seems to dominate, whereas in the interval [2,5] fluctuations around the steady state $x = 0$ are larger.

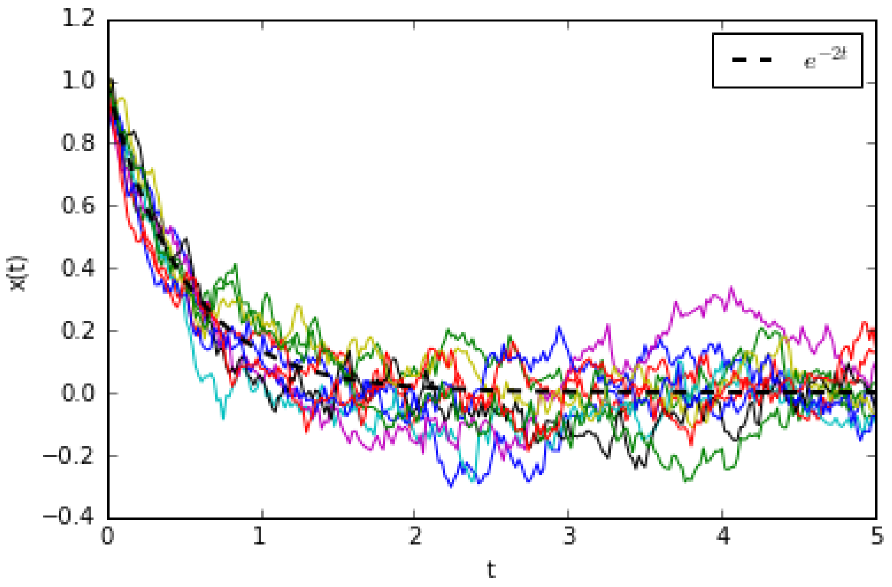

Figure 1: Ten different integrations of $\dot{x} = -ax + \sqrt{D}\xi(t)$ in the interval $t \in [0, 5]$, with a = 2 and D = 0.05 using Milshtein algorithm. The time step is $h = 0.001$, the initial condition x(0) = 1 and data are sampled every $t = 0.02$

b) First moment: Consider $\langle x(t)) \rangle$ where $\langle ... \rangle$ stands for averages over trajectories. Using $x(0) = 1$ and $h = 0.001$, evaluate the first moment on the inverval [0, 5] sampling at time invervals t = 0.05. Perform the averages over 10, 100 and 1000 trajectories. Plot the results and discuss the influence of the number of trajectories.

The averages plotted in Figure 2 are simply computed by averaging at every time t the position $x_i(t)$ of each trajectory i. In this case measures are taken every $\Delta t/h = 50$ steps and to reach $t = 5$ we need $5000/50 = 100$ stored positions ($nstep$). From Figure 2 we infer that the more trajectories are considered in the averages, the more the first moment approaches the deterministic solution (the mean of the fluctuations is 0).

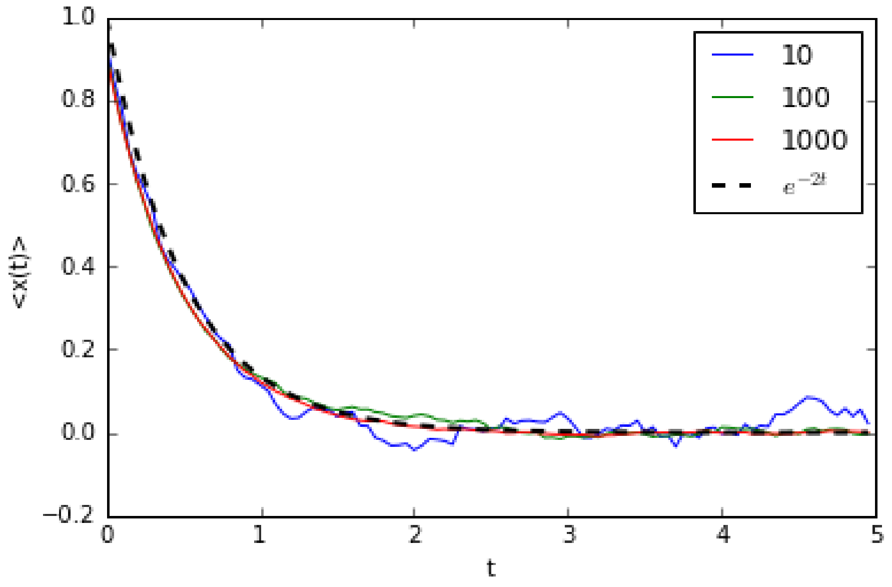

Figure 2: First moment $\langle x(t) \rangle$in the interval $[0, 5]$ evaluated for 10,100 and 1000 trajectories. The time step is $h = 0.001$ and the measures are taken every $t = 0.05$

c) Second moment: Using $x(0) = 1$ and $h = 0.001$, evaluate the second moment $\langle x(t)^2 \rangle$ in the interval [0, 5] sampling at time intervals $\Delta t = 0.05$ and averaging over 1000 trajectories. Plot the result together with the result for $x(t)^2$ for the deterministic trajectory. Discuss the results. Is the second moment equal to the deterministic result? Why?

Unlike the first moment shown in Figure 2, in the second moment fluctuations along the deterministic trajectory are smaller and consequently the number of trajectories is not as influential as before. Mathematically this has a simple answer: the difference between the deterministic $x(t)$ and the average over trajectories $\langle x(t) \rangle_{\text{traj}}$ is a small number and its square is even smaller, and that is why this difference is expected to decrease for larger moments. However, as can be seen from the dashed line in Figure 3, the second moment does not exactly lie on the deterministic solution $x(t)^2$ and the answer comes again from the fluctuations around the deterministic solution (the mean of this fluctuations make the second moment slightly larger than the steady state $x = 0$).

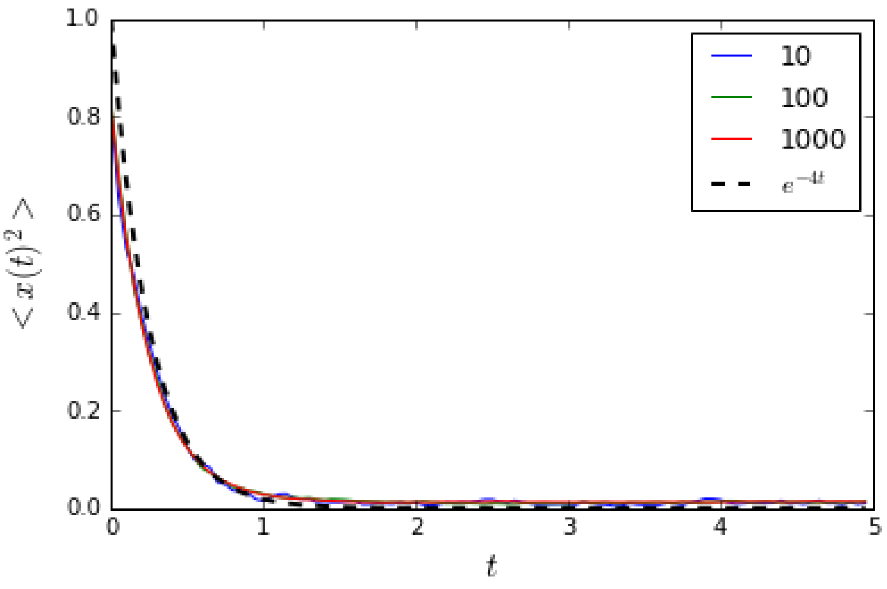

Figure 3: Second moment $\langle x(t)^2 \rangle$ in the interval $[0, 5]$ evaluated for 10,100 and 1000 trajectories.

d) Influence of the time step: Taking $x(0) = 1$ as initial condition and performing the averages over 1000 trajectories, evaluate the first and second moments at time $t = 1$, $\langle x(1) \rangle$ and $\langle x(1)^2 \rangle$, using different time steps: h = 0.5, h = 0.1, h = 0.01, h = 0.001 and h = 0.0001. Plot the results as function of $log(h)$. Discuss the results.

It is clear that very small time steps will considerably increase the number of measures for each trajectory and thus the stochastic trajectory will resemble more the deterministic solution. Thus, the average $\langle x(1) \rangle$ is closest to the deterministic solution $e^{-2a} = 0.135$ for the smallest h as represented in Figure 4. On the other hand, the less points you computed before reaching $x(t) = 1$, the more influential will be the stochastic Gaussian number in the Milhstein algorithm (equation 1) and, of course, the terms containing h will be very influential too.

Regarding the second moment, we can see that all results are larger than the deterministic solution $x(1) = e^{-4} = 0.018$ and that is because if we were to draw the second moment like in Figure 3, we would find that it is slightly above the deterministic trajectory due to fluctuations.

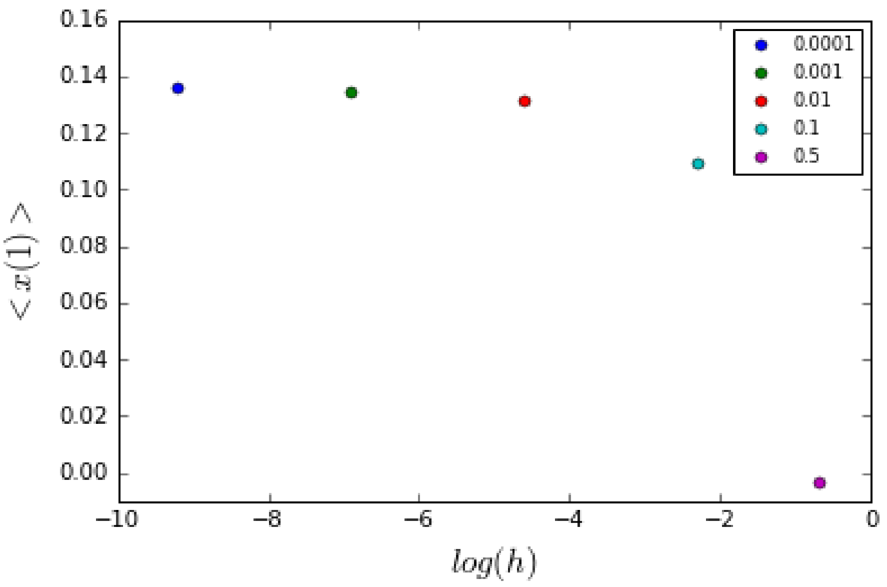

Figure 4: First moment $\langle x(1) \rangle$ computed over 1000 trajectories for 5 values of h. Different colors correspond to different h.

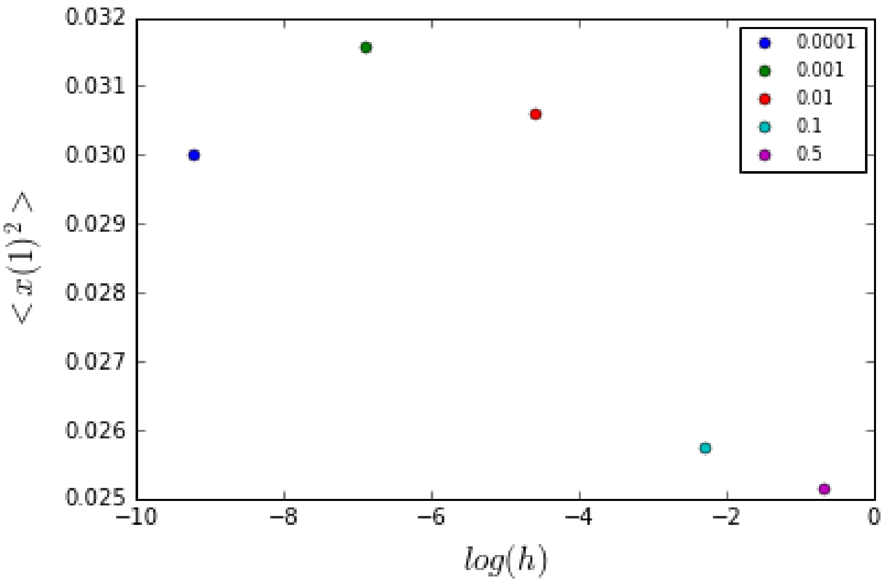

Figure 5: Second moment $\langle x(1)^2 \rangle$ computed over 1000 trajectories for 5 values of h.Different colors correspond to different h.

e) Correlation function from ensamble averages: Consider the correlation function $C(t, s) = \langle x(t)x(t+s) \rangle$ obtained averaging over 1000 trajectories. Take $x(0) = 1$ and $h = 0.001$. Evaluate $C(t, s)$ as function of s for $s \in [0, 5]$ sampling at intervals $\Delta s = 0.02$ for $t = 0.5$, $t = 1$, $t = 2$ and $t = 5$ and plot the results. Identify the stationary regime in which $C(t, s) = C(s)$, namely the correlation function does not depend on t. In the stationary regime, identify the correlation time, namely the value of s at which the correlation decays in $1/e$.

For each time, I plotted $5/\Delta s = 250$ values of $C(t,s)$ and the stationary regime for each t begins at $5-t$: for instance, at time $t=3$ the positions $x_i(3)$ are not correlated to values more than $\Delta t = \Delta s = 2$ ahead. More specifically, that is true if the 1000 trajectories are defined in the interval [0,5] as I did, but even defining them in [0,10] would not make a difference because the correlation function quickly decays for larger times and there exists a stationary regime regardless of t. Ignoring some small fluctuations, the stationary regime lies in $[3,5]$. The largest correlation time is for the correlation function $C(s,t=0.5)$ and it is 0.55. That means that for values of s larger than this correlation time the correlation function will have decayed to less than $C(0,t)/e$ for any t (for higher t the decay is faster).

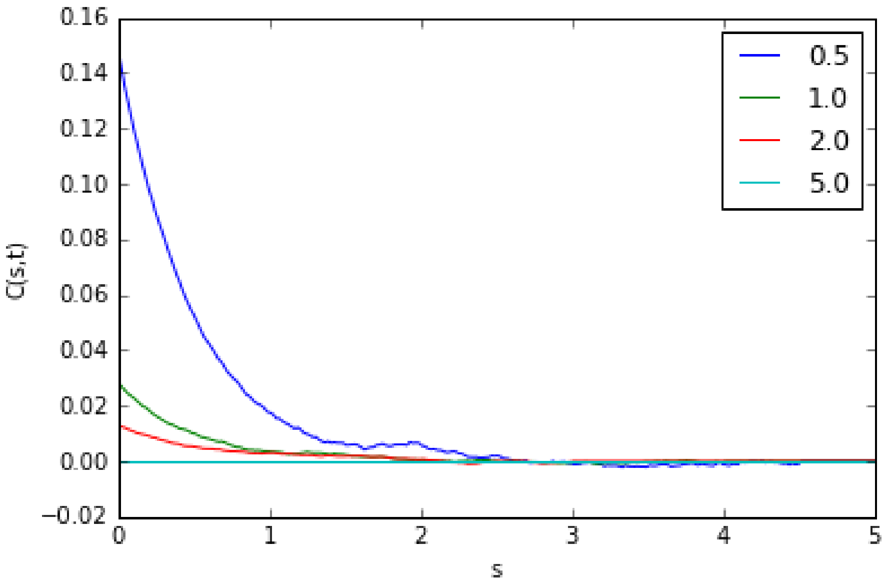

Figure 6: Correlation function $C(s,t)$ evaluated over 1000 trajectories taking $h=0.001$ and $\Delta s = 0.02$. Different colors correspond to different times t (it is not that easy to write t in the legend). The stationary regime for all t seems to be the interval $[3,5]$ (overlooking small fluctuations).

f) Correlation function from time averages Generate a very long trajectory starting from x = 0 to avoid the transient and using $h = 0.001$ and storing data at time intervals $\Delta t = 0.02$. Evaluate $C(s) = \langle x(t)x(t+s)\rangle$ where $\langle ...\rangle$ stands for averaging over time in a single (long) trajectory. Note that to have enough statistics the trajectory must be at least 1000 times longer than the typical correlation time evaluated in the previous point. Plot $\tilde{C}(s)$ and compare with $C(s)$.

I took $nsteps = 50000$ and since data is stored every $\Delta t = 0.02$, the trajectory lies in the time interval [0,1000] (1800 times bigger than the correlation time computed before). The average now is over time, not over trajectories as done above. The plot from Figure 7 seems to be very similar to the curve $t=0.5$ in Figure 6.

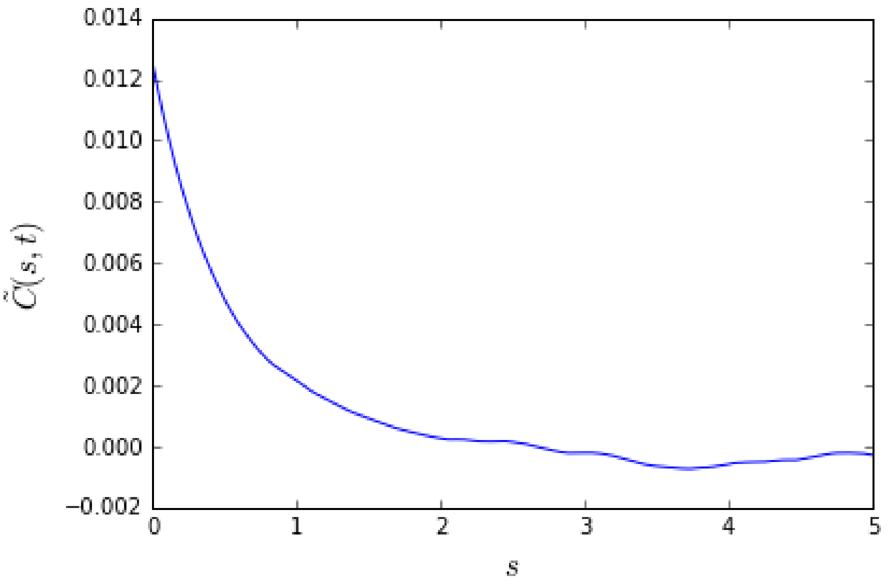

Figure 7: Correlation function $C(s,t)$ evaluated over a long trajectory in the interval [0,1000] taking $h = 0.001$ and $\Delta s = 0.02$.

g) Include listing of the programs:

```
#EX a)
import numpy as np
import math
import random
from matplotlib import pyplot as plt
import time


a = 2
g = np.sqrt(0.05)
h = 0.001
h_sqrt=np.sqrt(h)
nsteps = 250
x = 1
t = 0

def q(x):
    return -a*x


# MILSHTEIN ALGORITHM
traj=np.zeros((10,nsteps))
for i in range(10):
    x=1
    t=0
    time=np.zeros(nsteps)
    for j in range(nsteps):
        for k in range(20):
            uh=h_sqrt*np.random.normal()
```

```
                x = x + h * q(x) + g * uh
                t=t+h
            traj[i][j]=x
            time[j]=t
        plt.plot(time,traj[i])

time2=np.linspace(0,5,100)
plt.plot(time2,np.exp(-a*time2), 'k--', linewidth=2)
plt.xlim([0,5])
plt.ylabel('x(t)')
plt.xlabel('t')
plt.show()

#EX b)
#por fin primer momento

time_start = time.clock()
a = 2
g = np.sqrt(0.05)
h = 0.001
h_sqrt=np.sqrt(h)
nsteps = 100
x = 1
t = 0

def q(x):
    return -a*x
t=0
time2=np.zeros(nsteps)
for i in range(nsteps):
    time2[i]=h*i*50


for k in range(1,4):
    suma=np.zeros(nsteps)
    num=10**k
    traj=np.zeros((num,nsteps))
    for i in range(num):
        x=1
        t=0
        for j in range(nsteps):
            for k in range(50):
                uh=h_sqrt*np.random.normal()
                x = x + h * q(x) + g * uh
                t=t+h
            traj[i][j]=x

    for l in range(nsteps):
        for m in range(num):
            suma[l]+=traj[m][l]
    plt.plot(time2,suma/num, label=num)

plt.plot(time2,np.exp(-a*time2), 'k--', linewidth=2, label=r'$e^{-2t}$')
plt.xlim([0,5])
plt.ylabel('<x(t)>')
plt.xlabel('t')
plt.legend()
plt.show()


time_elapsed = (time.clock() - time_start)
print(time_elapsed)

#Ex c
#second moment
time_start = time.clock()
a = 2
g = np.sqrt(0.05)
h =0.001
h_sqrt=np.sqrt(h)
nsteps = 100
x = 1
t = 0

def q(x):
    return -a*x
t=0
time2=np.zeros(nsteps)
for i in range(nsteps):
    time2[i]=h*i*50


for k in range(1,4):
    suma=np.zeros(nsteps)
    num=10**k
    traj=np.zeros((num,nsteps))
    for i in range(num):
        x=1
        t=0
        for j in range(nsteps):
            for k in range(50):
                uh=h_sqrt*np.random.normal()
                x = x + h * q(x) + g * uh
                t=t+h
            traj[i][j]=x

    for l in range(nsteps):
        for m in range(num):
                suma[l]+=traj[m][l]**2


    plt.plot(time2,suma/num, label=num)

plt.plot(time2,np.exp(-2*a*time2), 'k--', linewidth=2, label=r'$e^{-4t}$')
plt.xlim([0,5])
plt.ylabel(r'$<x(t)^2>$', fontsize=15)
plt.xlabel(r'$t$', fontsize=15)
plt.legend()
plt.show()


time_elapsed = (time.clock() - time_start)
print(time_elapsed)

#Ex d
#influence of time step
time_start=time.clock()

a = 2
g = np.sqrt(0.05)
h = [0.0001,0.001,0.01,0.1,0.5]
h_sqrt=np.sqrt(h)
nsteps = [10000,1000,100,10,2]
x = 1
t = 0
numtraj=1000

def q(x):
    return -a*x


for k in range(5):
    suma=np.zeros(nsteps[k])
    suma2=np.zeros(nsteps[k])
    traj=np.zeros((numtraj,nsteps[k]))
    time2=np.zeros(nsteps[k])
    for i in range(numtraj):
        x=1
        t=0
        for j in range(nsteps[k]):
            uh=h_sqrt[k]*np.random.normal()
            x = x + h[k] * q(x) + g * uh
            t=t+h[k]
            traj[i][j]=x
            time2[j]=t

    for l in range(nsteps[k]):
        for m in range(numtraj):
                suma[l]+=traj[m][l]


    plt.plot(np.log(h[k]),suma[nsteps[k]-1]/numtraj,'o', markersize=5,
    label=h[k])


plt.ylim([-0.01,0.16])
plt.ylabel(r'$<x(1)>$', fontsize=15)
plt.xlabel(r'$log(h)$', fontsize=15)
plt.legend(numpoints=1, fontsize=8)

plt.show()


time_elapsed = (time.clock() - time_start)
print(time_elapsed)

#influence of time step second moment
time_start = time.clock()
a = 2
g = np.sqrt(0.05)
h = [0.0001,0.001,0.01,0.1,0.5]
h_sqrt=np.sqrt(h)
nsteps = [10000,1000,100,10,2]
x = 1
t = 0
numtraj=1000

def q(x):
    return -a*x


for k in range(5):
    suma=np.zeros(nsteps[k])
    suma2=np.zeros(nsteps[k])
    traj=np.zeros((numtraj,nsteps[k]))
    time2=np.zeros(nsteps[k])
    for i in range(numtraj):
        x=1
        t=0
        for j in range(nsteps[k]):
            uh=h_sqrt[k]*np.random.normal()
            x = x + h[k] * q(x) + g * uh
            t=t+h[k]
            traj[i][j]=x
            time2[j]=t

    for l in range(nsteps[k]):
        for m in range(numtraj):
            suma2[l]+=traj[m][l]**2


    plt.plot(np.log(h[k]),suma2[nsteps[k]-1]/numtraj,'o', markersize=5,
    label=h[k])


plt.ylabel(r'$<x(1)^2>$', fontsize=15)
plt.xlabel(r'$log(h)$', fontsize=15)
plt.legend(numpoints=1, fontsize=8)
plt.show()


time_elapsed = (time.clock() - time_start)
print(time_elapsed)

#Ex e
a = 2
g = np.sqrt(0.05)
h = 0.001
h_sqrt=np.sqrt(h)
nsteps = 1000
x = 1
t = 0
numtraj=1000
time2=np.zeros(nsteps)
for i in range(nsteps):
    time2[i]=h*i*5


def q(x):
    return -a*x




traj=np.zeros((numtraj,nsteps))
tstep= [100, 200, 400, 1000] #correspond to times 0.5,1,2,5
s=np.linspace(0,5,250)

for i in range(numtraj):
    x=1
    t=0
    for j in range(nsteps):
        for k in range(5):
            uh=h_sqrt*np.random.normal()
            x = x + h * q(x) + g * uh
            t=t+h
        traj[i][j]=x
```

```
for l in range(4):
    Corr=np.zeros(250)
    for m in range(250):
        suma=0
        for p in range(numtraj):
            if tstep[l]+4*m <nsteps:
                suma+=traj[p][tstep[l]-1]*traj[p][tstep[l]+4*m] #increase
                #of 0.02 in s is increase of 4 steps
        Corr[m]=suma/numtraj
    plt.plot(s,Corr, label=tstep[l]*0.005)
    plt.plot(Corr,s,label=tstep[l]*0.005)
    print(Corr[0]*1/np.e)



plt.ylabel('C(s,t)')
plt.xlabel('s')
plt.legend()
plt.show()

time_elapsed = (time.clock() - time_start)
print(time_elapsed)

#EX f
a = 2
g = np.sqrt(0.05)
h = 0.001
h_sqrt=np.sqrt(h)
nsteps = 50000
x = 1
t = 0
traj=np.zeros(nsteps)

time2=np.zeros(nsteps)
for i in range(nsteps):
    time2[i]=h*i*20



def q(x):
    return -a*x

s=np.linspace(0,5,250)

for i in range(nsteps):
    for k in range(20):
        uh=h_sqrt*np.random.normal()
        x = x + h * q(x) + g * uh
        t=t+h
    traj[i]=x
```

```
for m in range(250):
    suma=0
    for p in range(nsteps):
        if p+m <nsteps:
            suma+=traj[p]*traj[p+m] #increase of 0.02 in s is increase of
            #1 step
    Corr[m]=np.float(suma)/nsteps

plt.plot(s,Corr)
plt.ylabel(r'$\tilde{C}(s,t)$', fontsize=15)
plt.xlabel(r'$s$', fontsize=15)
plt.show()
```

Exercice 2

Consider $\dot{x} = ax - bx^3 + \sqrt{D}\xi(t)$ where $\xi(t)$ is a Gaussian white noise of zero mean and correlation $\langle\xi(t)\xi(t') = \delta(t-t')\rangle$. Take $a = 4$, $b = 1$, $D = 0.01$, $x(0) = 0$ as initial condition and use $h = 0.001$ as integration time step.

a) Trajectories: Integrate numerically 20 trajectories until time $t = 4$ storing data every $\Delta t = 0.01$ and plot them. Discuss the results.

From the figure below it can be deduced that there are 3 fixed points: one unstable at $x = 0$ and two stable at $x = \pm 2$. $x = 0$ is unstable because depending on the fluctuations the trajectory will be directed either to $x = 2$ or $x = -2$ and, once there, it will remain there regardless of the fluctuations, which somehow act like small perturbations.

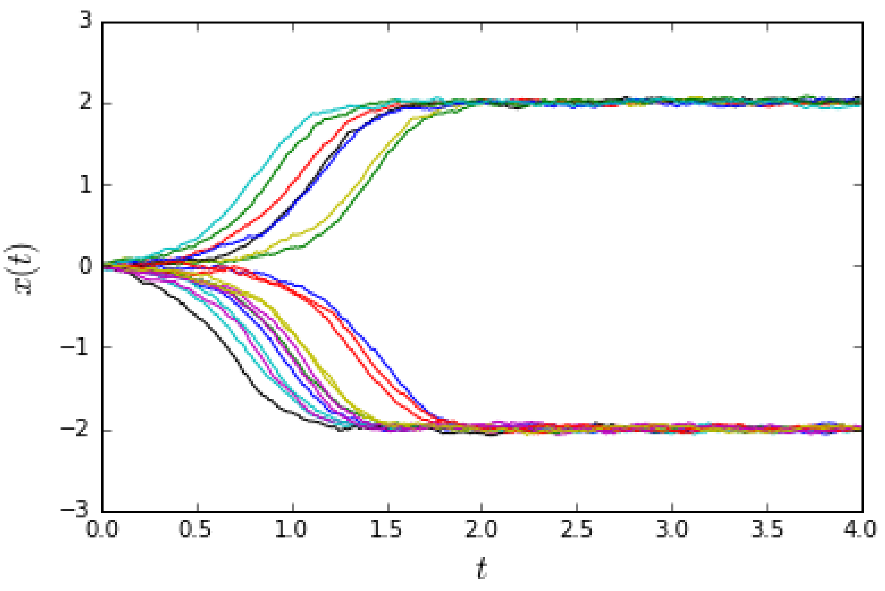

Figure 8: Twenty different integrations of $\dot{x} = ax - bx^3 + \sqrt{D}\xi(t)$ in the interval $t \in [0, 5]$, with a = 4 and D = 0.01 using Milshtein algorithm. The time step is $h = 0.001$, the initial condition x(0) = 1 and data are sampled every $t = 0.01$.

b) Transient anomalous fluctuations: Evaluate $\langle x(t)\rangle$, $\langle x(t)^2\rangle$ and $\langle x(t)^4\rangle$, where $\langle ...\rangle$ stands for averages over trajectories, sampling at intervals $\Delta t = 0.01$. Use 1000 trajectories for the averages. Plot the results and also the variance of x^2, namely $\sigma^2_{x^2} = \langle x(t)^4 \rangle - \langle x(t)^2 \rangle^2$ and show that it goes through a maximum at an intermediate time. Discuss the results.

We can interpret the fact that $\langle x(t) \rangle = 0$ for 1000 trajectories as follows: the mean value of $x(t)$ is 0 for sufficiently large number of trajectories because the stable points $x = \pm 2$ have the same probability and on average we will get half of the trajectories on $x = +2$ and the other half on $x = -2$, being thus the mean position 0. This is generally true for odd moments $\langle x(t)^3 \rangle$, $\langle x(t)^5 \rangle$, etc. However, even moments will keep increasing as plotted in the figure below because the signs of the trajectories $x = \pm 2$ will not longer cancel each other.

Regarding the variance of x^2 in Figure 10, the maximum is placed at $t \approx 1.3$ and this is the same time at which $\langle x(t)^2 \rangle$ has a maximum slope and hence maximum growth. At the tails (at the fixed points), on the other hand, $\langle x(t)^2 \rangle$ is constant and consequently the variance is 0.

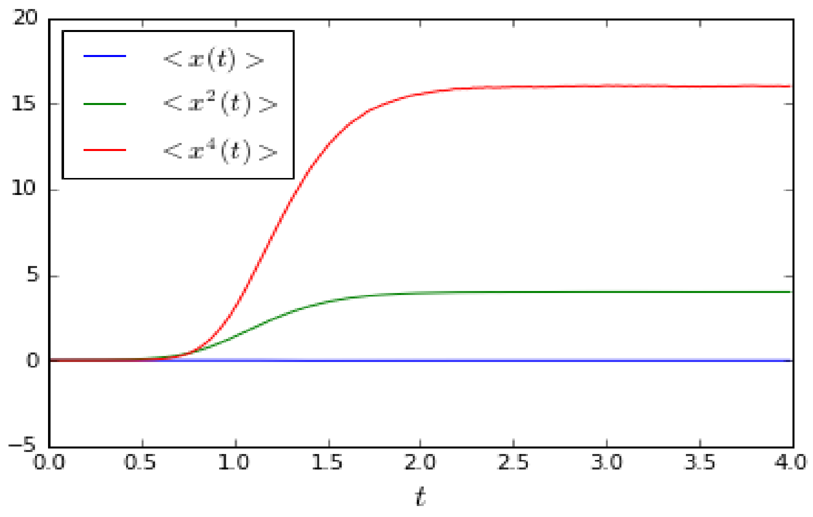

Figure 9: First, second and fourth moment over 1000 trajectories in the interval $[0, 4]$.

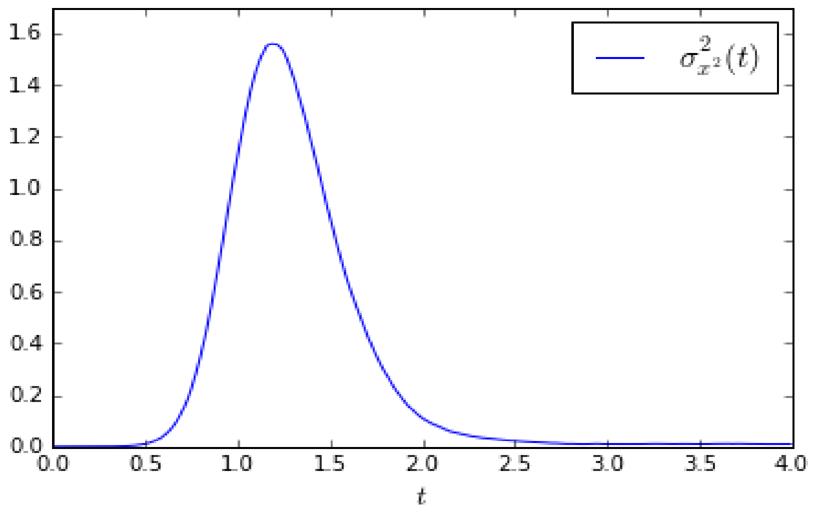

Figure 10: Variance $\sigma^2_{x^2}$ of the second moment evaluated for 1000 trajectories. The maximum is at $t \approx 1.3$

c) Probability density function: Integrate numerically 5000 trajectories and evaluate an histogram for the distribution of x at times $t = 0.6$, $t = 0.9$, $t = 1.2$, $t = 1.5$, $t = 1.8$, $t = 3$ and $t = 4$. Normalize the histogram such that the total area is 1 and so that it corresponds to the probability distribution of x at these times and plot the results. Discuss the changes in shape.

The histograms are coherent with the trajectories plotted in Figure 8. For small times most trajectories are gathered symmetrically around the initial state $x = 0$ resembling a Gaussian distribution, and as time increases the distribution of positions gets flatter with no mean position (this corresponds to the splitting of the trajectories towards $x = \pm 2$). Eventually, for times near $t = 4$, we get two symmetric peaks at the stable points $x = \pm 2$ with no trajectories in between. At $t = 4$ the peak at $x = +2$ is slightly higher because there seems to be more trajectories on the upper branch of Figure 8.

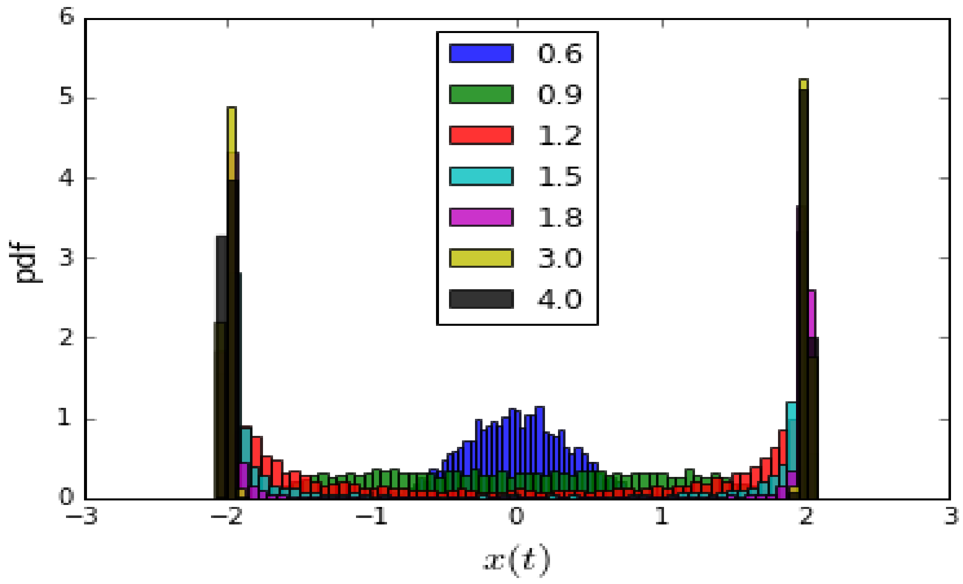

Figure 11: Histogram of $x(t)$ at four different times given by the color in the legend. At each time $x(t)$ comes from different 5000 trajectories.

d) First passage time distribution: Integrate the equations numerically to generate 5000 trajectories until the trajectory reaches $|x(t)| = xb = 0.5$, recording the time at which the trajectory reaches that value. Plot a histogram of the distribution of the times normalized such that the total area is 1, this is known as the first passage time distribution. Is it symmetrical? Discuss the results.

There is an important numerical condition I had to impose: since I take measures every $\Delta t = 0.01$, I stored the times at which the trajectories reached $x_b = 0.5 \pm \epsilon$ in a time interval of length Δt so that I ensure that I store one time per trajectory (and not more!). If I rather stored the times at which the trajectories reach exactly $x_b = 0.5$ I would have got an almost empty histogram. I could have also solved this by taking measures every time step h, but this would have increased the correlations.

The histogram shown right below is not exactly symmetrical and we have to refer again to Figure 8: the splitting of the trajectories to steady states $x = \pm 2$ is not exactly symmetrical regarding the number of trajectories on each branch and regarding as well the times at which each trajectory reaches $|x| = 0.5$. We could draw two horizontal lines at $x = \pm 0.5$ and count the intersections and their times to strengthen our faith on this very true fact.

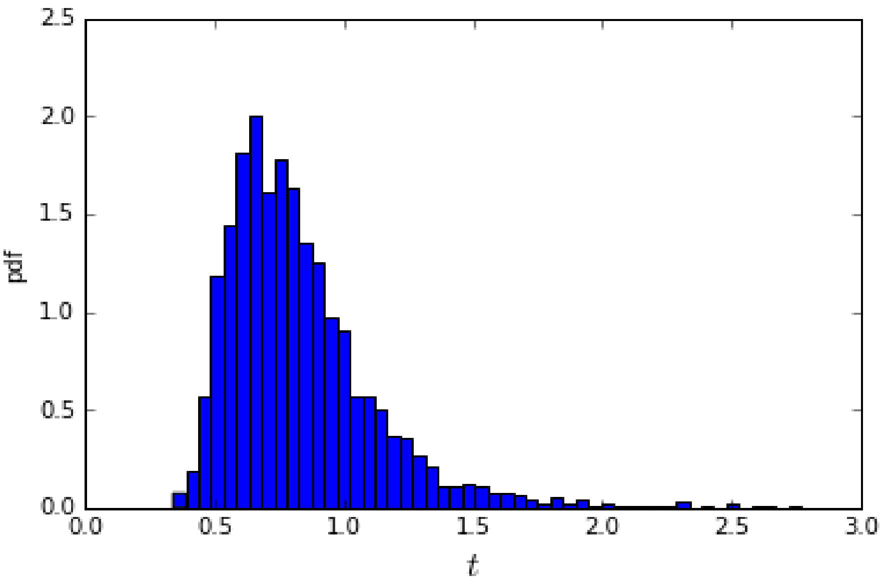

Figure 12: Histogram of $|x(t)| = 0.5$ for 5000 trajectories. Most trajectories reach $|x(t)| = 0.5$ at times between 0.65 and 0.8.

e) Mean first passage time: Generate 5000 trajectories as before evaluating the average of the time at which trajectories reach $|x(t)| = xb = 0.5$ (mean first passage time) for $D = 0.1$, $D = 0.01$, $D = 0.001$ and $D = 0.0001$. Plot the results as function of lnD. Compare with $(1/2a)lnD$.

My experimental points do not lie on $-1/(2a)ln(D)$; they are rather placed in parallel above this line. In other words, the slope is the same but the intersection with the y axis is different. To reconcile between both results we simply add ≈ 0.25 to the theoretical results. Finally, we note that for higher values of D the trajectories reach x_b at earlier times and that can be anticipated from the Milhstein algorithm (equation 1): the higher $g(x) = \sqrt{D}$, the higher the difference $x(t_{i+1}) - x(t_i)$.

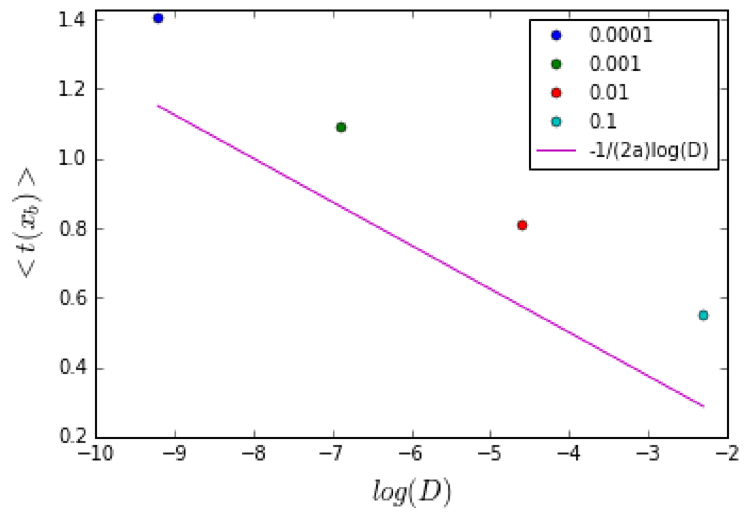

Figure 13: Mean times at which 5000 trajectories reach x_b for different D's represented in colored legend.

f) Include listing of the programs.

```
#trajectories

a = 4
D = 0.01
deltat = 0.01
tmax=4
h = 0.001
h_sqrt=np.sqrt(h)
g = np.sqrt(D)
nsteps = np.int(tmax/deltat)
def q(x):
    return a*x-x**3


time2=np.zeros(nsteps)
for i in range(nsteps):
    time2[i]=h*i*10

for i in range(20):
    x=0
    t=0
    traj=np.zeros(nsteps)
    for j in range(nsteps):
        for k in range(10):
            uh=h_sqrt*np.random.normal()
            x = x + h * q(x) + g * uh
            t=t+h
        traj[j]=x
    plt.plot(time2, traj)



plt.xlim([0,4])
plt.ylabel(r'$x(t)$', fontsize=15)
plt.xlabel(r'$t$', fontsize=15)
plt.show()

#Ex b
a = 4
D = 0.01
deltat = 0.01
tmax=4
h = 0.001
h_sqrt=np.sqrt(h)
g = np.sqrt(D)
nsteps = np.int(tmax/deltat)
numtraj=1000
```

```
time2=np.zeros(nsteps)
for i in range(nsteps):
    time2[i]=h*i*10

def q(x):
    return a*x-x**3



traj=np.zeros((numtraj,nsteps))

for i in range(numtraj):
    x=0
    t=0
    for j in range(nsteps):
        for k in range(10):
            uh=h_sqrt*np.random.normal()
            x = x + h * q(x) + g * uh
            t=t+h
        traj[i][j]=x



suma1=np.zeros(nsteps)
suma2=np.zeros(nsteps)
suma4=np.zeros(nsteps)

for l in range(nsteps):
    for m in range(numtraj):
        suma1[l]+=traj[m][l]
        suma2[l]+=traj[m][l]**2
        suma4[l]+=traj[m][l]**4


plt.plot(time2,suma1/numtraj, label=r'$<x(t)>$')
plt.plot(time2,suma2/numtraj, label=r'$<x^2(t)>$')
plt.plot(time2,suma4/numtraj, label=r'$<x^4(t)>$')
plt.xlabel(r'$t$', fontsize=15)
plt.legend(loc=2, fontsize=13)
plt.show()

plt.plot(time2,suma4/numtraj-(suma2/numtraj)**2, label=r'$\sigma^2_{x^2}(t)
plt.ylim([0,1.7])
plt.legend(fontsize=15)
plt.xlabel(r'$t$', fontsize=13)
plt.show()

#Ex c
a = 4
```

```
D = 0.01
deltat = 0.01
tmax=4
h = 0.001
h_sqrt=np.sqrt(h)
g = np.sqrt(D)
nsteps = np.int(tmax/deltat)
numtraj=5000


time2=np.zeros(nsteps)
for i in range(nsteps):
    time2[i]=h*i*10

def q(x):
    return a*x-x**3



traj=np.zeros((numtraj,nsteps))

for i in range(numtraj):
    x=0
    t=0
    for j in range(nsteps):
        for k in range(10):
            uh=h_sqrt*np.random.normal()
            x = x + h * q(x) + g * uh
            t=t+h
        traj[i][j]=x

times=[0.6,0.9,1.2,1.5,1.8,3,4]


timestep=[ 60,  90, 120, 150, 180, 300, 400]

for i in range(np.size(timestep)):
    times2=np.empty(np.size(times))
    for k in range(np.size(times)):
        times2[k]=times[i]

    histog=np.zeros(numtraj)
    temps=timestep[i]
    for j in range(numtraj):
        histog[j]=traj[j][temps-1]
    plt.hist(histog, normed=1, bins=60, alpha=0.8, label=times2)

plt.xlabel(r'$x(t)$', fontsize=16)
plt.ylabel('pdf', fontsize=13)
plt.legend(fontsize=12, loc=9)
plt.show()

#Ex d
epsilon=0.005
times05=[]
for i in range(numtraj):
    for j in range(nsteps):
        if 0.5-epsilon <np.abs(traj[i][j])<0.5+epsilon:
            times05.append(j*0.01)



plt.hist(times05, bins=50, normed=1)
plt.xlabel(r'$t$', fontsize=15)
plt.ylabel('pdf')
plt.show()

#EX e
a = 4
D = [0.0001,0.001,0.01,0.1]
deltat = 0.01
tmax=4
h = 0.001
h_sqrt=np.sqrt(h)
g = np.sqrt(D)
nsteps = np.int(tmax/deltat)
numtraj=5000
epsilon=0.01

time2=np.zeros(nsteps)
for i in range(nsteps):
    time2[i]=h*i*10

def q(x):
    return a*x-x**3



for k in range(np.size(D)):
    traj=np.zeros((numtraj,nsteps))
    media=np.zeros(np.size(D))
    for i in range(numtraj):
        x=0
        t=0
        for j in range(nsteps):
            for l in range(10):
                uh=h_sqrt*np.random.normal()
                x = x + h * q(x) + g[k] * uh
                t=t+h
            traj[i][j]=x
    times05=[]
    for i in range(numtraj):
        for j in range(nsteps):
            if 0.5-epsilon <np.abs(traj[i][j])<0.5+epsilon:
                times05.append(j*0.01)

    media[k]=np.mean(times05)
    plt.plot(np.log(D[k]), media[k], 'o', markersize=5, label=D[k])


plt.xlabel(r'$log(D)$', fontsize=15)
plt.ylabel(r'$<t(x_b)>$', fontsize=15)
plt.plot(np.log(D),-0.125*np.log(D), label='-1/(2a)log(D)')
plt.legend(numpoints=1, fontsize=10)
plt.show()
```

Exercice 3

Consider the stochastic differential equation $\dot{x} = -ax + \sqrt{D}\xi_{OU}(T)$ where $\xi_{OU}(t)$ is an Ornstein-Uhlenbeck noise of zero mean and correlation $\langle \xi(t)\xi(t')\rangle = (1/2\tau)e^{-|t-t'|/\tau}$. We will consider $a = 2$ and $D = 0.05$.

a) Program: Implement a program to integrate this equation using the Heun method with exact generation of the process g_h.

According to Heun's method, the integration of a general SDE equation $\dot{x}(t) = q(x(t)) + g(x(t))\xi(t)$ at each time is:

$$x(t_{i+1}) = x(t_i) + \frac{h}{2}[q(x(t_i)) + q(x(t_i) + k)] + \frac{1}{2}g_h(t_i)[g(x(t_i)) + g(x(t_i) + k)]$$

which holds if $g(x)$ and $q(x)$ are time independent like in our case:

$$g(x) = \sqrt{D} \quad , \quad q(x) = -ax$$

$g_h(t_i)$ is the integrarion of the Ornstein-Uhlenbeck noise:

$$g_h(t_i) = \int_{t_i}^{t_{i+2}} \xi_{OU}(s)ds$$

The exact generation of $g_h(t_i)$ can be done recursively:

$$g_h(t_i) = p(g_h(t_i) - a_i) + a_{i+1} - b_i + b_{i+1}$$

Being $p = e^{-\frac{h}{\tau}}$, $a_i = \int_{t_i}^{t_{i+1}} \xi(s)ds$ and $b_i = -pe^{-t_i/\tau}\int_{t_i}^{t_{i+1}} e^{s/\tau}\xi(s)ds$. I won't enter into much detail about where do all these terms come from because I took all this from the theory slides, but let's close these set of equations by expressing a_i and b_i in terms of two uncorrelated Gaussian numbers $u_i, v_i \in G(0,1)$:

$$a_i = \gamma u_i = \sqrt{h}u_i$$

$$b_i = \beta u_i + \gamma v_i = \frac{\tau(p-1)}{\sqrt{h}} u_i + v_i \sqrt{\frac{\tau}{2}(1-p^2) - \frac{\tau^2}{2}(1-p)^2}$$

The variables are initialized according to $a_0 = \sqrt{h} u_o$, $b_0 = \beta u_0 + \gamma v_o$ and $g_h(t_0) = \sqrt{\tau/2}(1-p)u_g + a_0 + b_0$, with $u_g \in G(0,1)$. The numerical implementation is shown below.

```
#program with any g(x)
import numpy as np
import math
import itertools
import random
from matplotlib import pyplot as plt
import scipy
import time

def q(x):
    return -2*x

def g(x):
    return x


# variables for gh
tau= 0.005
h= 0.01
nsteps=np.int(5/h)
alpha = h_sqrt = np.sqrt(h)
p = np.exp(-h/tau)
beta = tau * (p - 1) / h_sqrt
gamma = np.sqrt(tau/2 * (1 - p**2) - beta**2)

t=0
x=1
a_old=0
b_old=np.sqrt(0.5*tau)*(p-1)*np.random.normal()
gh_old=0

traj=np.zeros(nsteps)
time=np.zeros(nsteps)
for i in range(nsteps):
    u=np.random.normal()
    a=h_sqrt*u
    b=beta*u+gamma*np.random.normal()
    gh=p*(gh_old-a_old)+a-b_old+b
    a_old=a
    b_old=b
    gh_old=gh
    k=h*q(x)+gh*g(x)
    x=x+0.5*(k+h*q(x+k)+gh*g(x+k))
    t=t+h
```

```
        traj[i]=x
        time[i]=t

plt.plot(time,traj)
plt.show()
```

b) Trajectories: For $g(x) = 1$ and $\tau = 0.5$ plot 10 trajectories in the time interval $[0, 5]$ with initial condition $x(0) = 1.0$, using $h = 0.01$ and writing data every time step. Do the same for $\tau = 2$, $\tau = 0.05$ and $\tau = 0.005$ and discuss the effect of the noise correlation time τ.

It is deduced from Figure 14 that as the noise correlation time is increased the fluctuations tend to decrease significantly. Indeed, for $\tau \to 0$ the Ornstein-Uhlenbeck noise becomes a Gaussian noise as can be proved by taking the limit $\tau \to 0$ and applying Hôpital in $\langle \xi(t)\xi(t')\rangle = (1/2\tau)e^{-|t-t'|/\tau}$. Judging by the large fluctuations, this pseudo-Gaussian noise has an important impact on the trajectories for small values of τ.

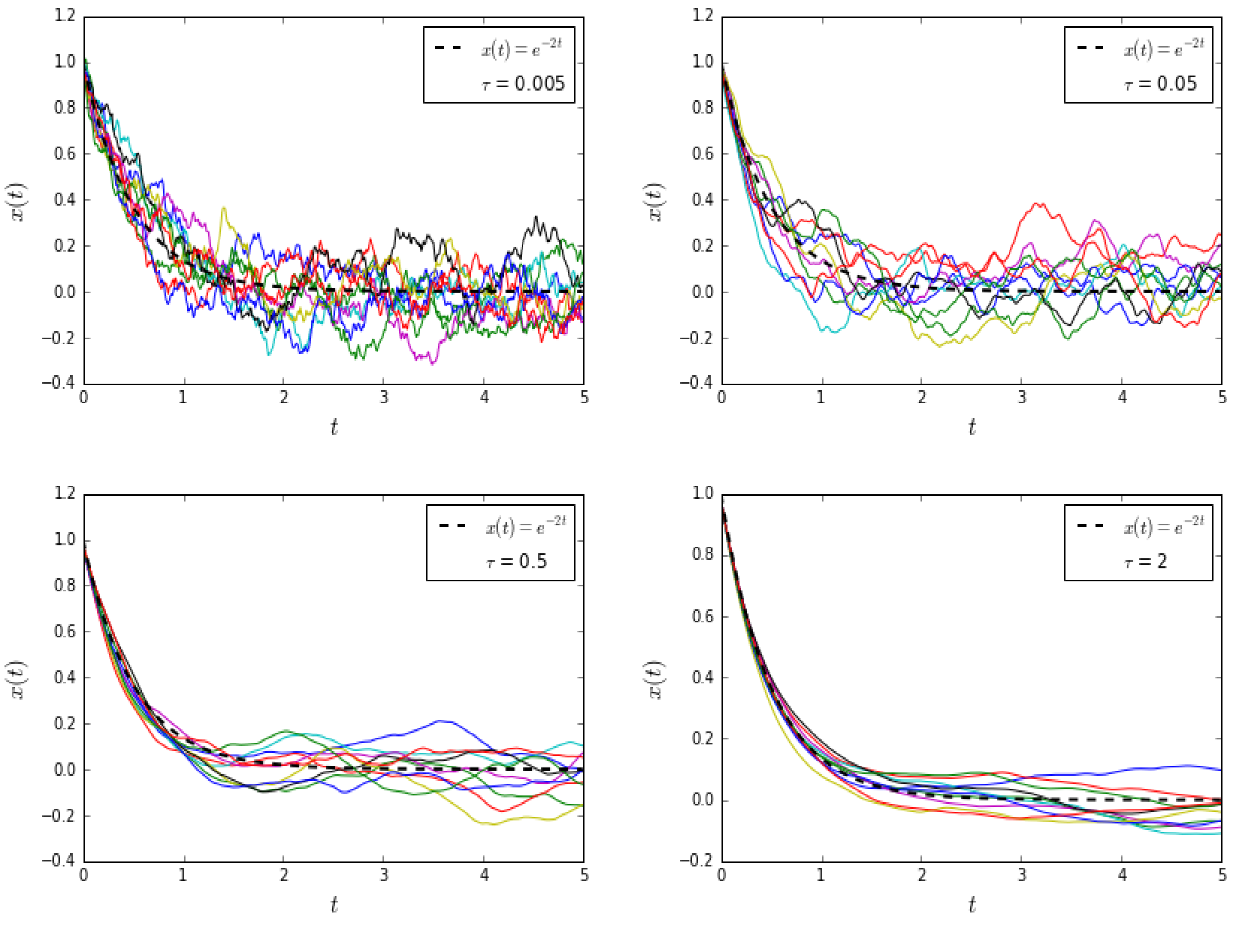

Figure 14: 10 trajectories for 4 values of τ with $g(x) = 1$ and using $h = 0.01$.

c) Correlation: For $g(x) = 1$ evaluate the correlation function $C(t, s) = \langle x(t)x(t + s)\rangle =$ averaging over 1000 trajectories. Take $x(0) = 1$, $h = 0.01$ and $\tau = 0.5$. Evaluate $C(t, s)$ as function of s for $s \in [0, 5]$ sampling at intervals $\Delta s = 0.05$ for $t = 0.5$, $t = 1$, $t = 2$ and $t = 5$ and plot the results. Identify the stationary regime in which $C(t, s) = C(s)$, namely the correlation function does not depend on t. In the stationary regime, identify the correlation time, namely the value of s at which the correlation decays in $1/e$.

The stationary regime happens to begin at approximately $s = 2.5$ since from this value all correlation functions are 0. The first value of s at which $C(s,t) = C(0,t)/e$ is 0.55 and this is then the correlation time for any t (stationary regime).

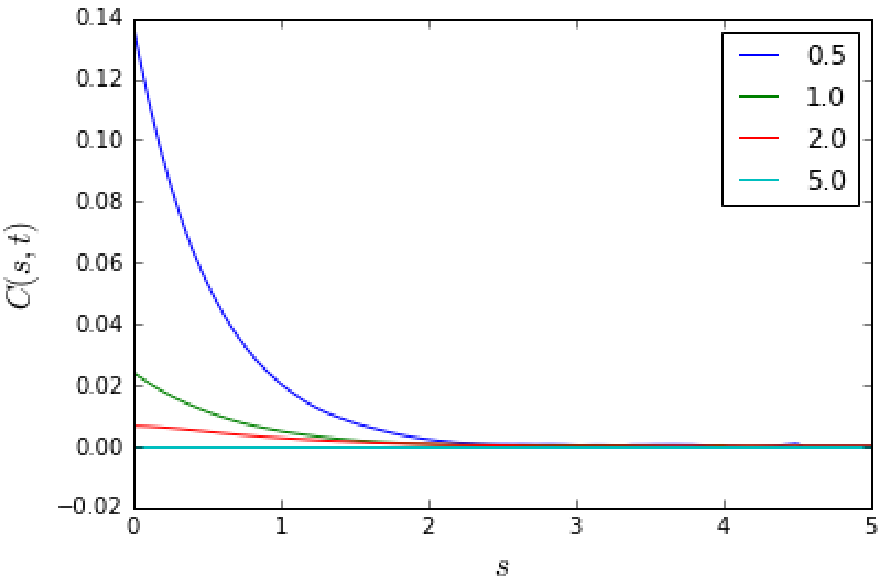

Figure 15: Correlation function $C(s,t)$ for the times depicted in the legend and $\tau = 0.5$.

d) Effect of the noise correlation time: Repeat the previous calculation for $\tau = 0.05$ and $\tau = 2$ and discuss how the correlation function changes with the noise correlation time τ.

The correlation function does not seem to be significantly influenced by a change in τ. If we were to join both graphs below in one plot we would notice that for higher values of τ the correlation $C(s,t)$ is slightly smaller (I didn't decide to join the plots because the 8-labels legend was not very clear).

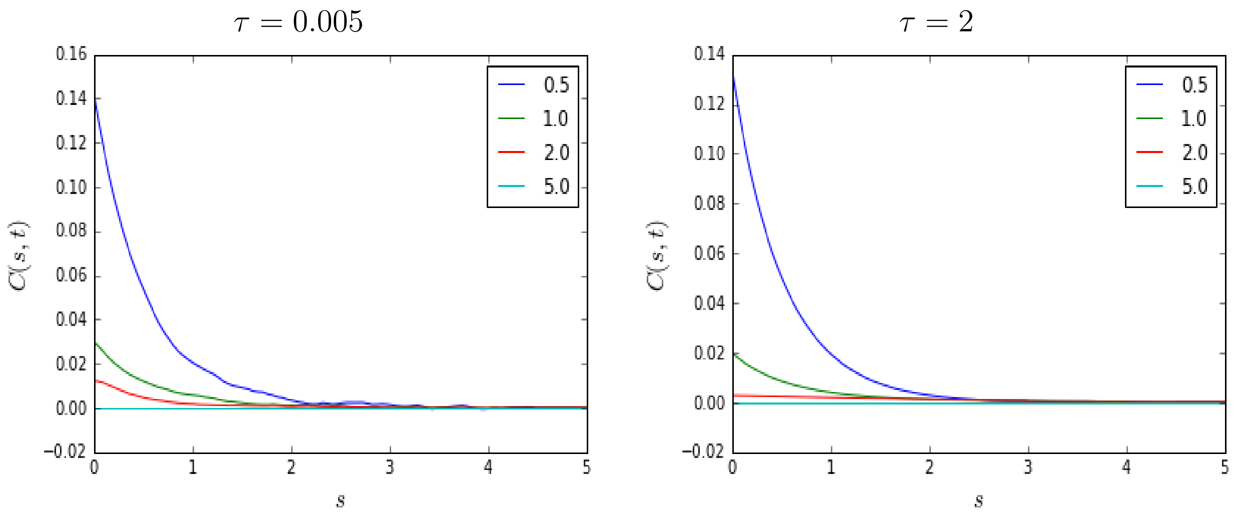

Figure 16: Correlation function $C(s,t)$ for the times depicted in the legend and two values of τ.

e) Multiplicative noise: Consider now $g(x) = x$ and $\tau = 0.5$ plot 10 trajectories in the time interval [0, 5] with initial condition $x(0) = 1.0$, using $h = 0.01$ and writing data every time step. Compare with the results of 3b).

The difference with the trajectories plotted in 3b) for $\tau = 0.05$ is quite notorious: with a multiplicative noise (that is, making the stochastic term x-dependent) we get much less fluctuations and indeed the stochastic trajectories resemble at least in shape the deterministic one.

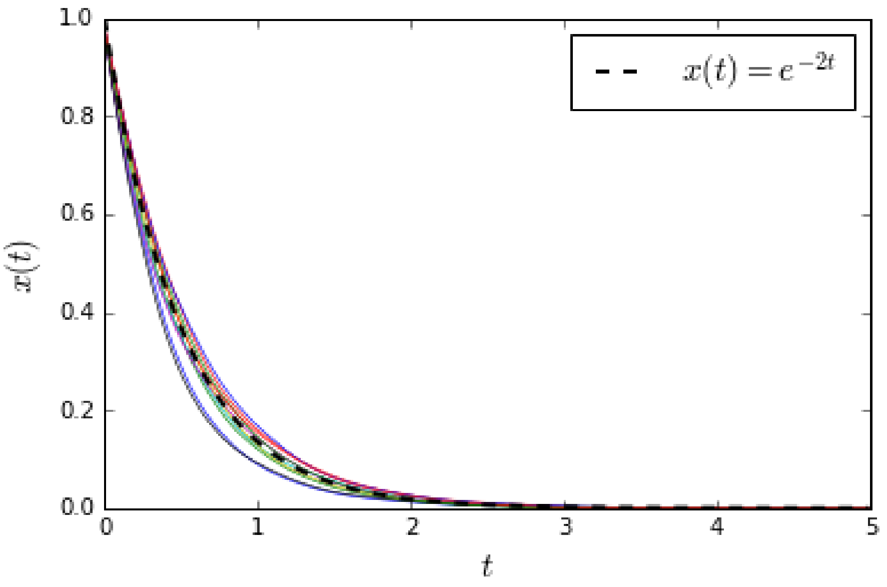

Figure 17: 10 trajectories for $g(x) = x$, $h = 0.01$ and $\tau = 0.5$.

f) Include the listing of the programs.

```
#exercice b
g =np.sqrt(0.05) # sqrt(D)g(x)

# variables for gh
tau= 2
h= 0.01
nsteps=np.int(5/h)
numtraj=10
alpha = h_sqrt = np.sqrt(h)
p = np.exp(-h/tau)
beta = tau * (p - 1) / h_sqrt
gamma = np.sqrt(tau/2 * (1 - p**2) - beta**2)




time=np.zeros(nsteps)
for i in range(nsteps):
    time[i]=h*i

for k in range(numtraj):
    traj=np.zeros(nsteps)
    t=0
    x=1
    a_old=0
    b_old=np.sqrt(0.5*tau)*(p-1)*np.random.normal()
    gh_old=0

    for i in range(nsteps):
        u=np.random.normal()
        a=h_sqrt*u
        b=beta*u+gamma*np.random.normal()
        gh=p*(gh_old-a_old)+a-b_old+b
        a_old=a
        b_old=b
        gh_old=gh
        k=h*q(x)+gh*g
        x=x+0.5*(k+h*q(x+k)+gh*g)
        t=t+h
        traj[i]=x

    plt.plot(time,traj)


plt.xlabel(r'$t$', fontsize=15)
plt.ylabel(r'$x(t)$', fontsize=15)
plt.plot(time, np.exp(-2*time),'k--',linewidth=2, label=r'$x(t)=e^{-2t}$')
plt.legend(fontsize=15)
plt.plot([],[], color='white',label=r'$\tau$ = 2')
plt.legend()
plt.show()


#correlation times and correl function

g =np.sqrt(0.05) # sqrt(D)g(x)

# variables for gh
tau= 0.5
h= 0.01
nsteps=np.int(5/h)
numtraj=1000
alpha = h_sqrt = np.sqrt(h)
p = np.exp(-h/tau)
beta = tau * (p - 1) / h_sqrt
gamma = np.sqrt(tau/2 * (1 - p**2) - beta**2)




time=np.zeros(nsteps)
for i in range(nsteps):
    time[i]=h*i
```

```
traj=np.zeros((numtraj,nsteps))
for i in range(numtraj):
    t=0
    x=1
    a_old=0
    b_old=np.sqrt(0.5*tau)*(p-1)*np.random.normal()
    gh_old=0

    for j in range(nsteps):
        u=np.random.normal()
        a=h_sqrt*u
        b=beta*u+gamma*np.random.normal()
        gh=p*(gh_old-a_old)+a-b_old+b
        a_old=a
        b_old=b
        gh_old=gh
        k=h*q(x)+gh*g
        x=x+0.5*(k+h*q(x+k)+gh*g)
        t=t+h
        traj[i][j]=x


tstep=[50,100,200,500] #correspond to times 0.5,1,2,5
s=np.linspace(0,5,100)
for l in range(4):
    Corr=np.zeros(100)
    for m in range(100):
        suma=0
        for p in range(numtraj):
            if tstep[l]+5*m <nsteps:
                suma+=traj[p][tstep[l]-1]*traj[p][tstep[l]+5*m] #increase c
        Corr[m]=suma/numtraj
    plt.plot(s,Corr, label=tstep[l]*0.01)
    plt.plot(Corr,s)
    print(Corr[0]*1/np.e)



plt.ylabel(r'$C(s,t)$', fontsize=15)
plt.xlabel(r'$s$', fontsize=15)
plt.legend()
plt.show()

#exercice e
def g(x):
    return np.sqrt(0.05)*x

# variables for gh
tau= 0.5
```

```
h= 0.01
nsteps=np.int(5/h)
numtraj=10
alpha = h_sqrt = np.sqrt(h)
p = np.exp(-h/tau)
beta = tau * (p - 1) / h_sqrt
gamma = np.sqrt(tau/2 * (1 - p**2) - beta**2)




time=np.zeros(nsteps)
for i in range(nsteps):
    time[i]=h*i

for k in range(numtraj):
    traj=np.zeros(nsteps)
    t=0
    x=1
    a_old=0
    b_old=np.sqrt(0.5*tau)*(p-1)*np.random.normal()
    gh_old=0

    for i in range(nsteps):
        u=np.random.normal()
        a=h_sqrt*u
        b=beta*u+gamma*np.random.normal()
        gh=p*(gh_old-a_old)+a-b_old+b
        a_old=a
        b_old=b
        gh_old=gh
        k=h*q(x)+gh*g(x)
        x=x+0.5*(k+h*q(x+k)+gh*g(x+k))
        t=t+h
        traj[i]=x

    plt.plot(time,traj, linewidth=0.6)


plt.xlabel(r'$t$', fontsize=15)
plt.ylabel(r'$x(t)$', fontsize=15)
plt.plot(time, np.exp(-2*time),'k--',linewidth=2, label=r'$x(t)=e^{-2t}$')
plt.legend(fontsize=15)
plt.show()
```