1 SDE approximation by Euler scheme

We simulate a one dimensional Ornstein-Uhlenbeck process by using an Euler approximation of the solution to the corresponding stochastic differential equation.

In [9]: # Euler-Maryuama Approximation for solution of SDE  \[ dX_t = -\gamma X_t \, dt + \sigma \, dB_t \]

import numpy as np
# makes numpy routines and data types available as np.[name our routine or data type]

import matplotlib.pyplot as plt
# makes plotting command available as plt.[name of command]

x0 = 5.
# initial value at time t=0
sigma = 2.
# volatility
gamma = .5
# friction coefficient

tmax = 5.
# simulation of process from time 0 to tmax
stepslist = [10,100,1000,10000]
# produce simulations with step numbers chosen from steplist

for steps in stepslist:
    h = tmax/steps
    # stepsize for time discretization
    std = np.sqrt(h)
    # standard deviation for the distribution of each step

    k = 100
    # number of samples that will be simulated

    noise = np.random.randn(steps,k)*std
    # create a steps times k dimensional matrix of normal random numbers
    # with variance h
sde = np.ones((steps,k))
sde = sde*x0
# initialize the array sde with the initial condition x0

for n in range(steps-1):

t = np.arange(0,steps,1)*h
# creates vector of time points

plt.figure(figsize=(10,6), dpi=80)
# sets size of plot
plt.plot(t, sde, linewidth=0.3)
# produces a plot of the sample solutions in the components of sde
# versus t with thin lines
plt.show()
# output of plot
After some time the process seems to approach an equilibrium distribution.

2 Solution flow of Ornstein-Uhlenbeck SDE

We now simulate simultaneously the solutions starting from different initial conditions where the same noise random variables are applied in each case.

In $\theta$:

```python
# Euler approximation for flow of SDE $dX_t = -\gamma X_t \, dt + \sigma \, dB_t$

import numpy as np
# makes numpy routines and data types available as np.[name ouf routine or data type]

import matplotlib.pyplot as plt
# makes plotting command available as plt.[name of command]

sigma = 2.
# volatility
gamma = .5
# friction coefficient

tmax = 5.
# simulation of process from time 0 to tmax
stepslist = [10, 100, 1000, 10000]
# produce simulations with step numbers chosen from steplist

for steps in stepslist:
```
\[ h = \frac{t_{\text{max}}}{\text{steps}} \]\# steps size for time discretization
\[ \text{std} = \text{np.sqrt}(h) \]\# standard deviation for the distribution of each step

\[ k = 10 \]\# number of initial values to be considered
\[ x0 = \text{np.linspace}(-5., 5., k) \]\# simulate solutions for this list of initial values

\[ \text{noise} = \text{np.random.randn(steps)}*\text{std} \]\# create a steps dimensional vector of normal random numbers with variance \( h \)

\[ \text{sde} = \text{np.zeros}((\text{steps}, k)) \]
\[ \text{sde}[0] = x0 \]\# initialize the array \( \text{sde} \) with the initial conditions in \( x0 \)

\text{for} \ n \ \text{in} \ \text{range}(\text{steps}-1):
\[ \text{sde}[n+1] = \text{sde}[n] - \gamma h \text{sde}[n] + \sigma \text{noise}[n] \]

\[ t = \text{np.arange}(0, \text{steps}, 1)*h \]\# creates vector of time points

\[ \text{plt.figure}(\text{figsize}=(10,6), \text{dpi}=400) \]\# sets size of plot
\[ \text{plt.plot}(t, \text{sde}, \text{linewidth}=0.3) \]\# produces a plot of the sample solutions in the components of \( \text{sde} \) versus \( t \) with thin lines
\[ \text{plt.show()} \]\# output of plot
We observe that the solutions for different initial values approach each other ("loss of memory of initial condition").

In [ ]: