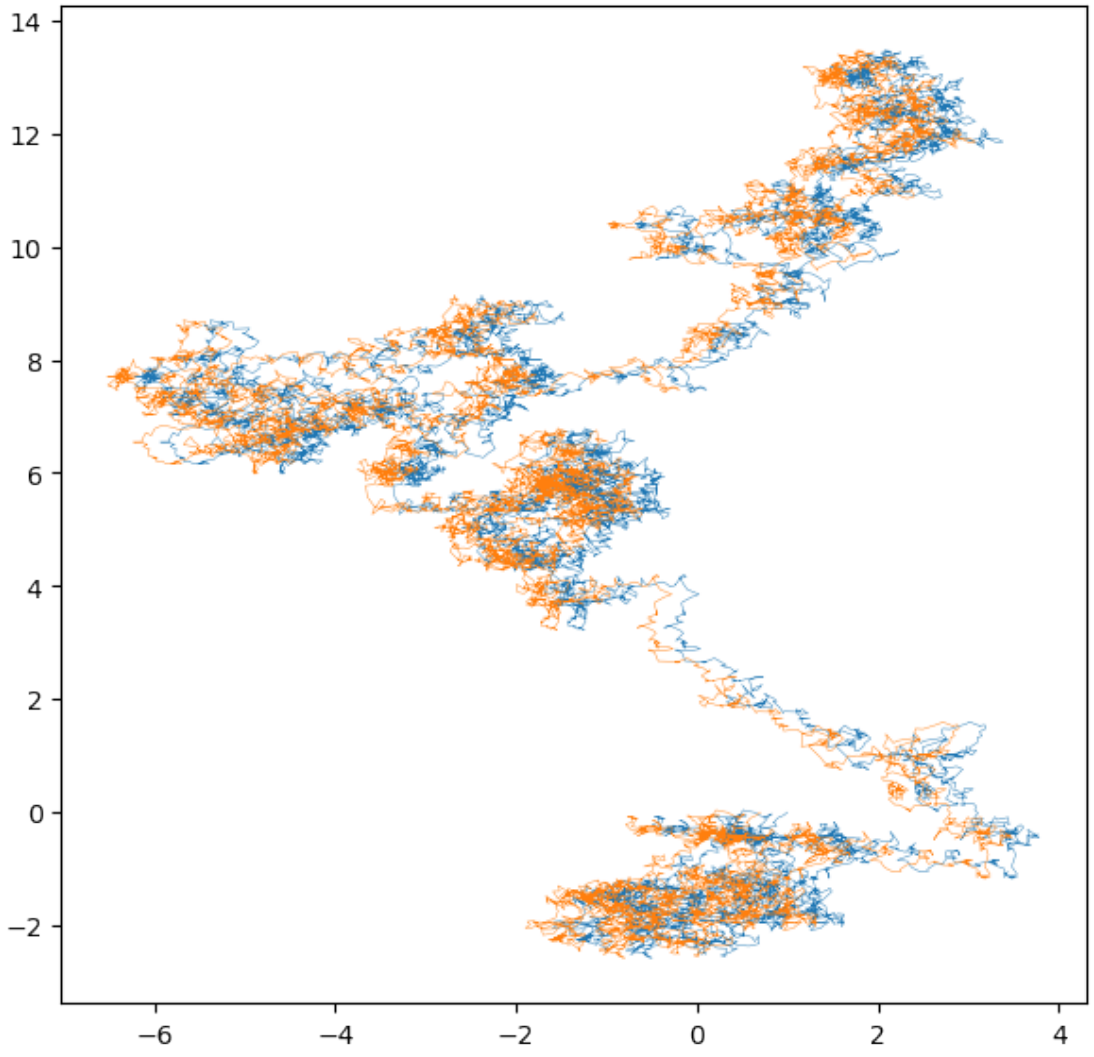
SDE2D Flow Examples for linear SDE with additive noise with output

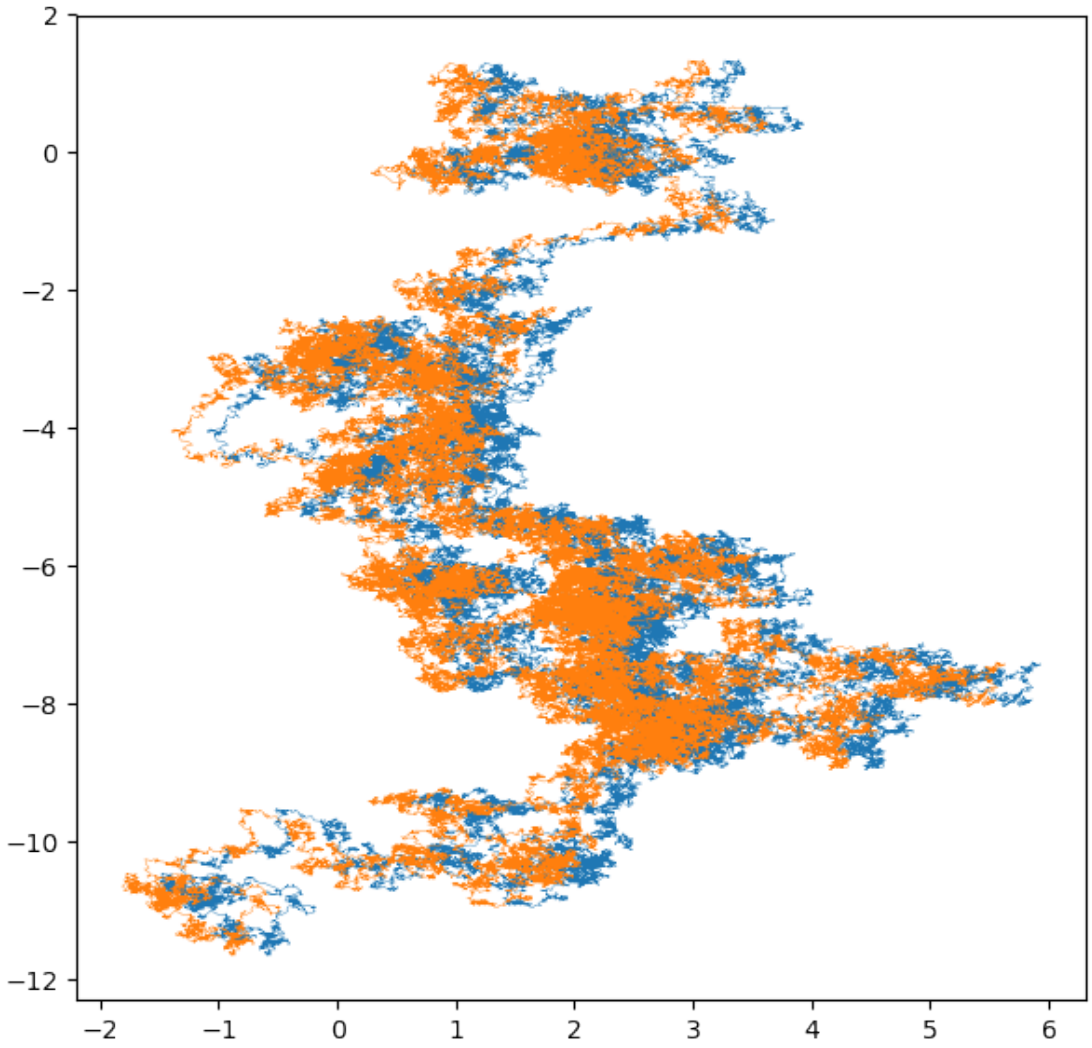
October 30, 2018

a) Two dimensional Brownian motion

In [10]:
```
# Euler-Maryuama Approximation for solution of SDE
# dX_t = A X_t dt + sigma dB_t
# Simulation of flow starting from several initial conditions

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]

k = 2
# number of copies
x0 = np.array([[1.,.7],
       [0.,0.]])
# initial value at time t=0
A = np.array([[0.,0.],[0.,0.]])
# drift matrix for Brownian motion
sigma = 1.
# volatility

tmax = 40.
# simulation from time 0 to tmax
stepslist = [10000,100000]
# number of steps that will be simulated


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

    noise = np.random.randn(2,steps)*std
    # create a 2 times steps dimensional matrix of normal random numbers with varianc
```

```
sde = np.ones((2,steps,k))
sde[:,0,:] = x0


for n in range(steps-1):
    for i in range(k):
        sde[:,n+1,i] = sde[:,n,i]+h*np.matmul(A,sde[:,n,i])+sigma*noise[:,n]


plt.figure(figsize=(7,7), dpi=100)
plt.plot(sde[0],sde[1],linewidth=.3)
plt.show()
```

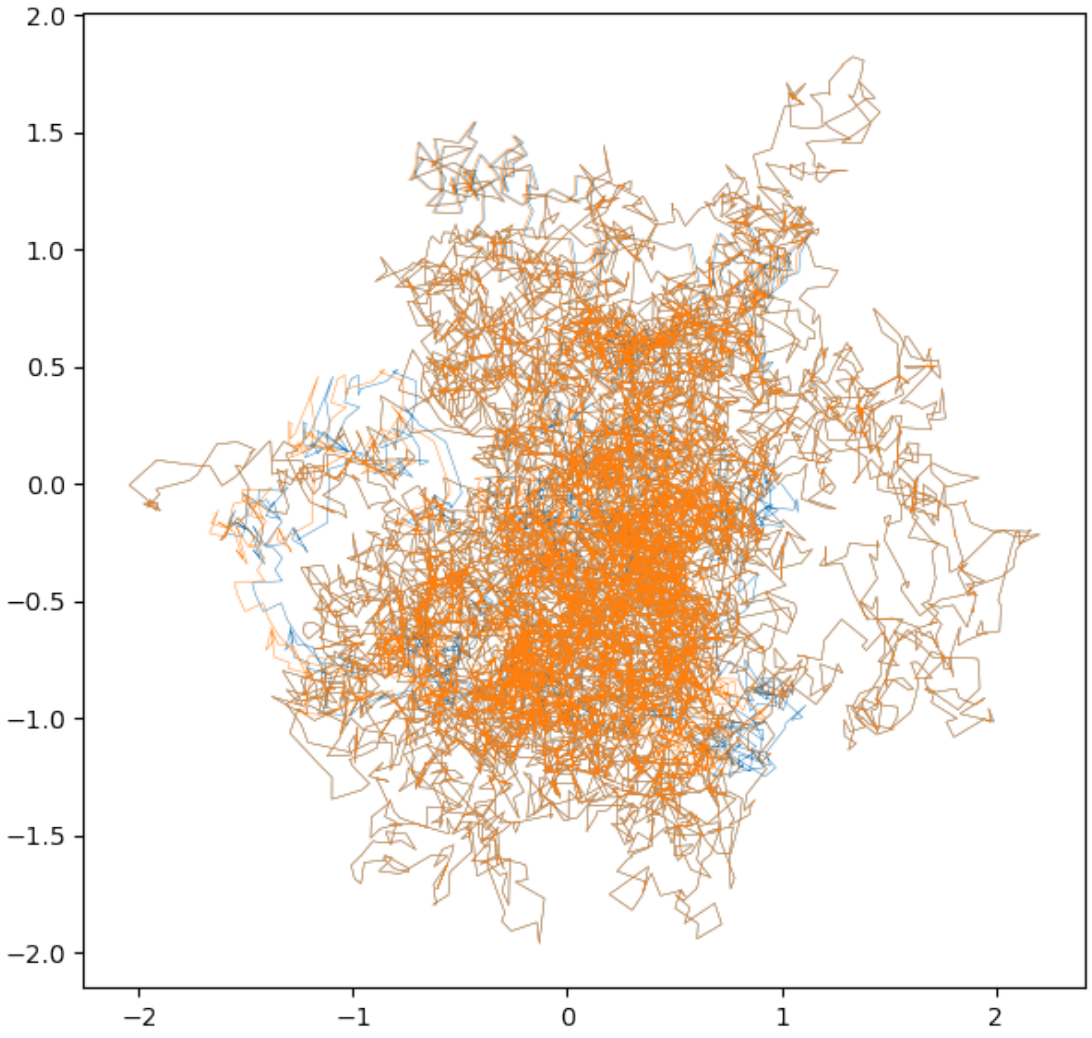


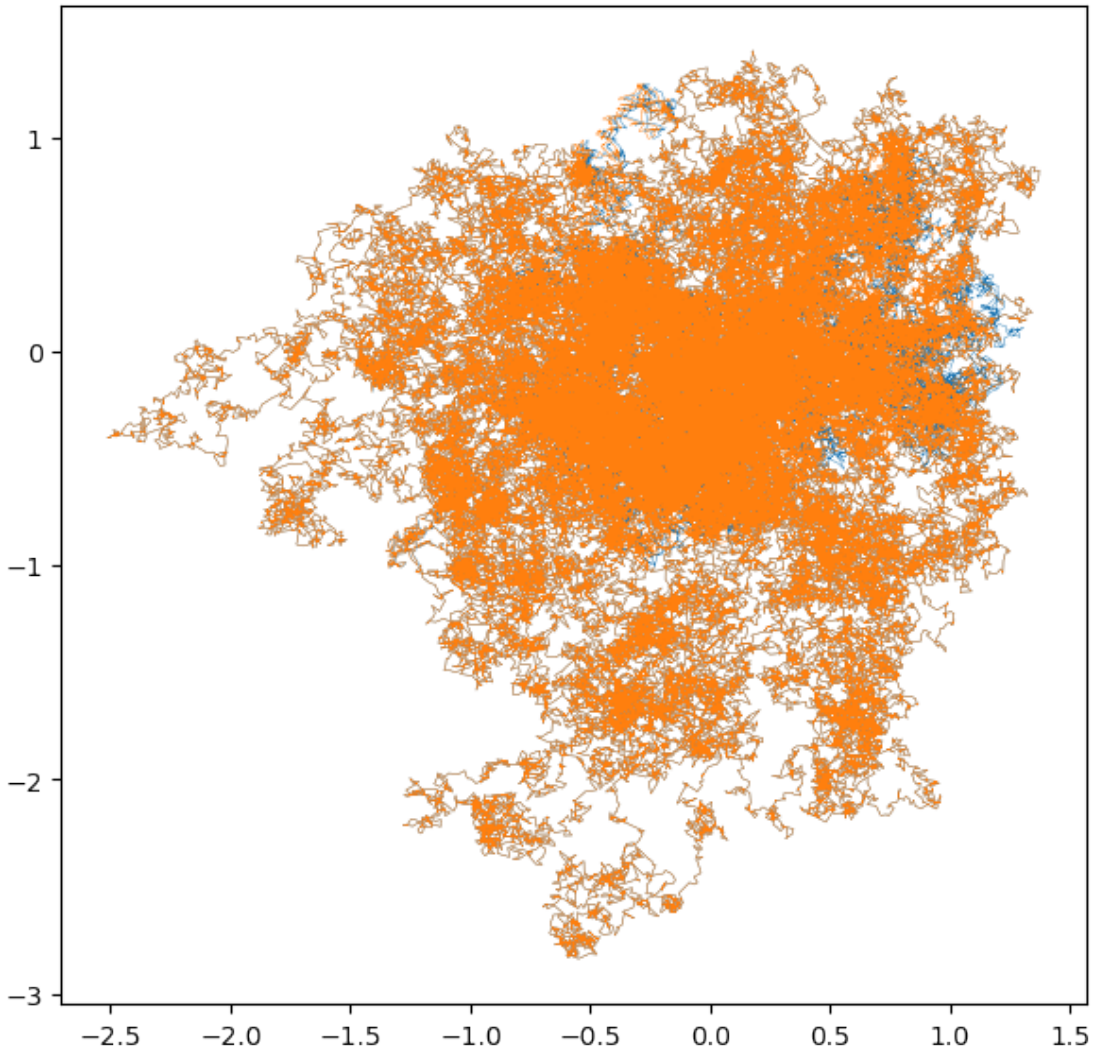


b) Classical Ornstein-Uhlenbeck process in two dimensions

```
In [11]: k = 2
         # number of copies
         x0 = np.array([[1.,.7],
                [0.,0.]])
         # initial value at time t=0
         A = np.array([[-1.,0.],[0.,-1.]])
         # drift matrix for Brownian motion
         sigma = 1.
         # volatility

         tmax = 40.
         # simulation from time 0 to tmax
         stepslist = [10000,100000]
```

```
# number of steps that will be simulated


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

    noise = np.random.randn(2,steps)*std
    # create a 2 times steps dimensional matrix of normal random numbers with varianc

    sde = np.ones((2,steps,k))
    sde[:,0,:] = x0


    for n in range(steps-1):
        for i in range(k):
            sde[:,n+1,i] = sde[:,n,i]+h*np.matmul(A,sde[:,n,i])+sigma*noise[:,n]


    plt.figure(figsize=(7,7), dpi=100)
    plt.plot(sde[0],sde[1],linewidth=.3)
    plt.show()
```

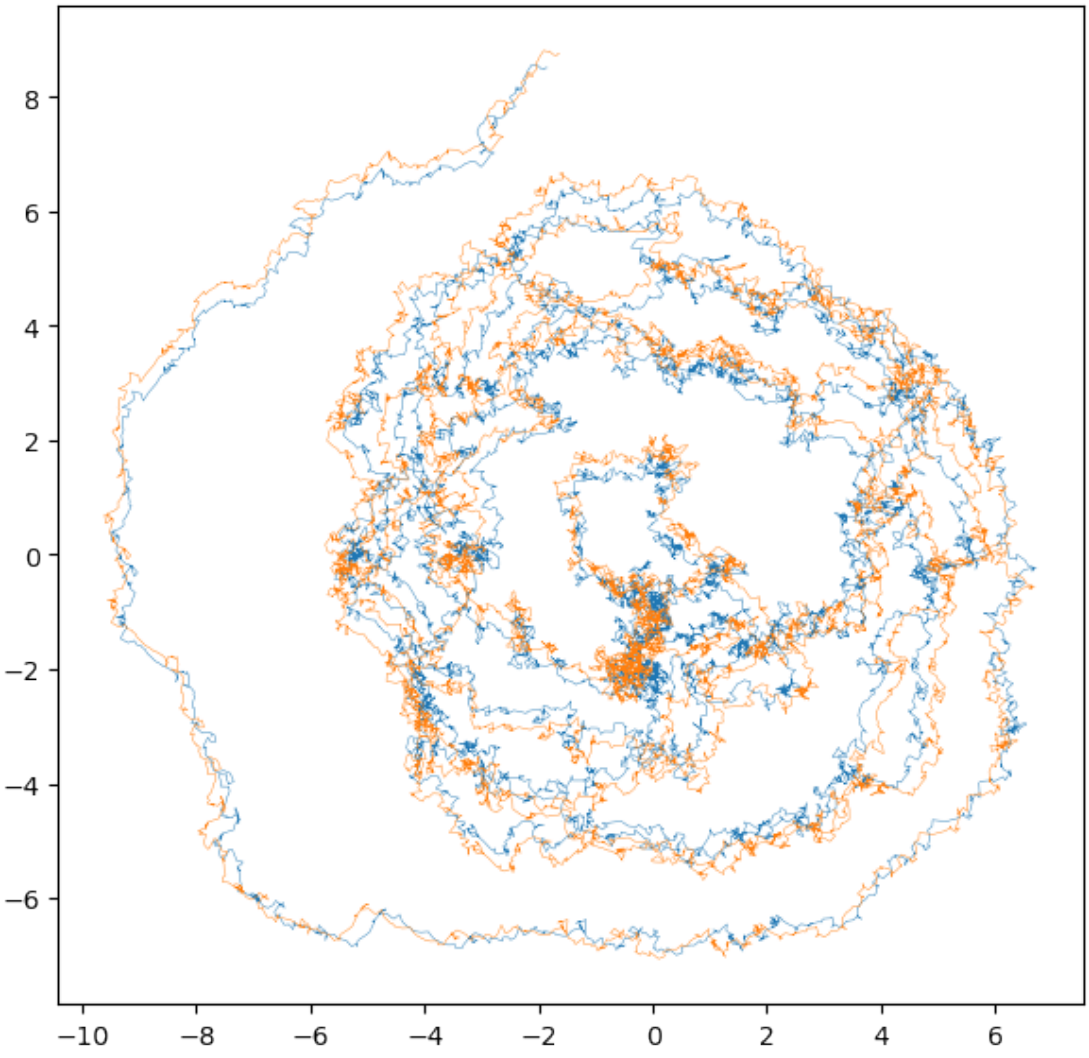

2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2

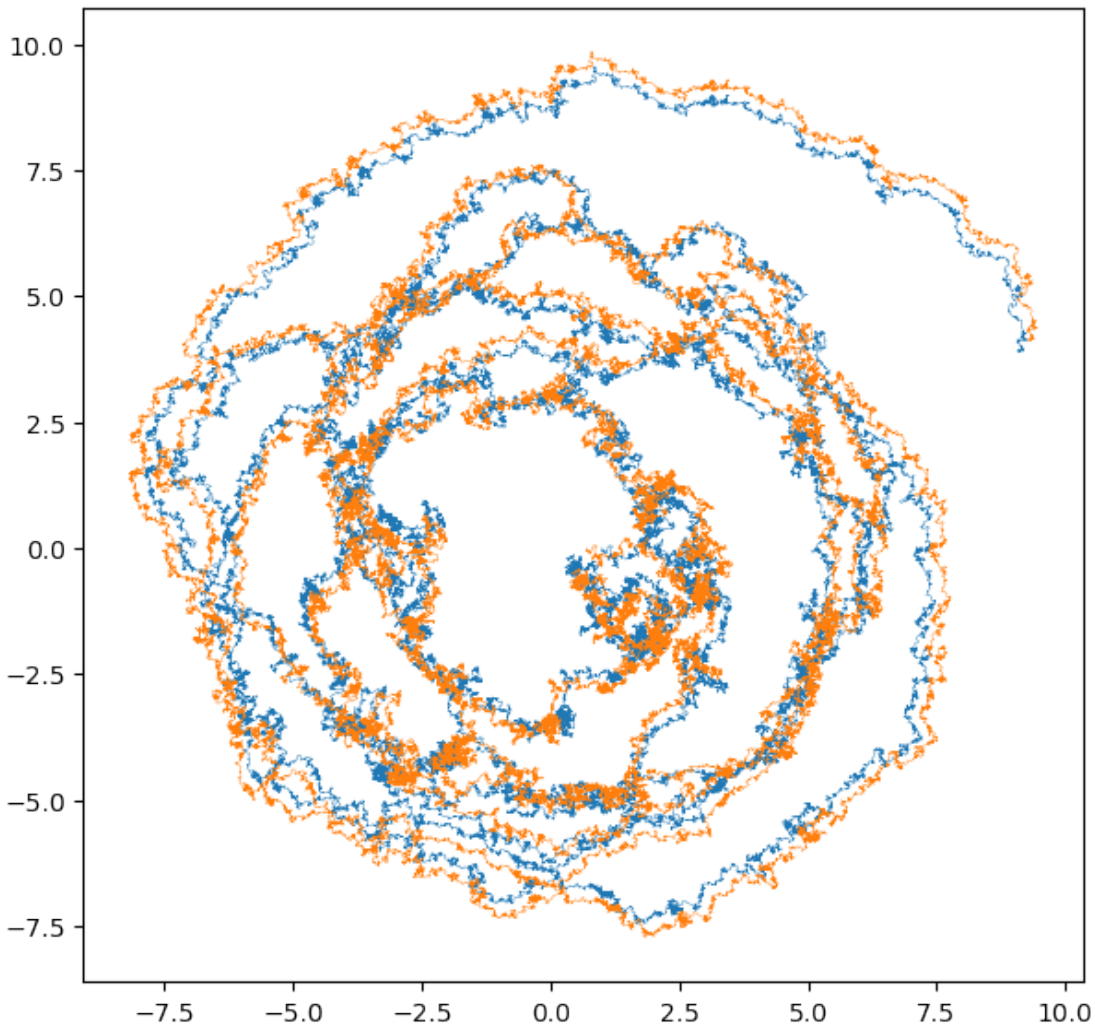


c) Randomly perturbed rotations

```
In [12]: k = 2
         # number of copies
         x0 = np.array([[1.,.7],
                [0.,0.]])
         # initial value at time t=0
         A = np.array([[0.,1.],[-1.,0.]])
         # drift matrix for randomly perturbed rotation
         sigma = 1.
         # volatility

         tmax = 40.
         # simulation from time 0 to tmax
```

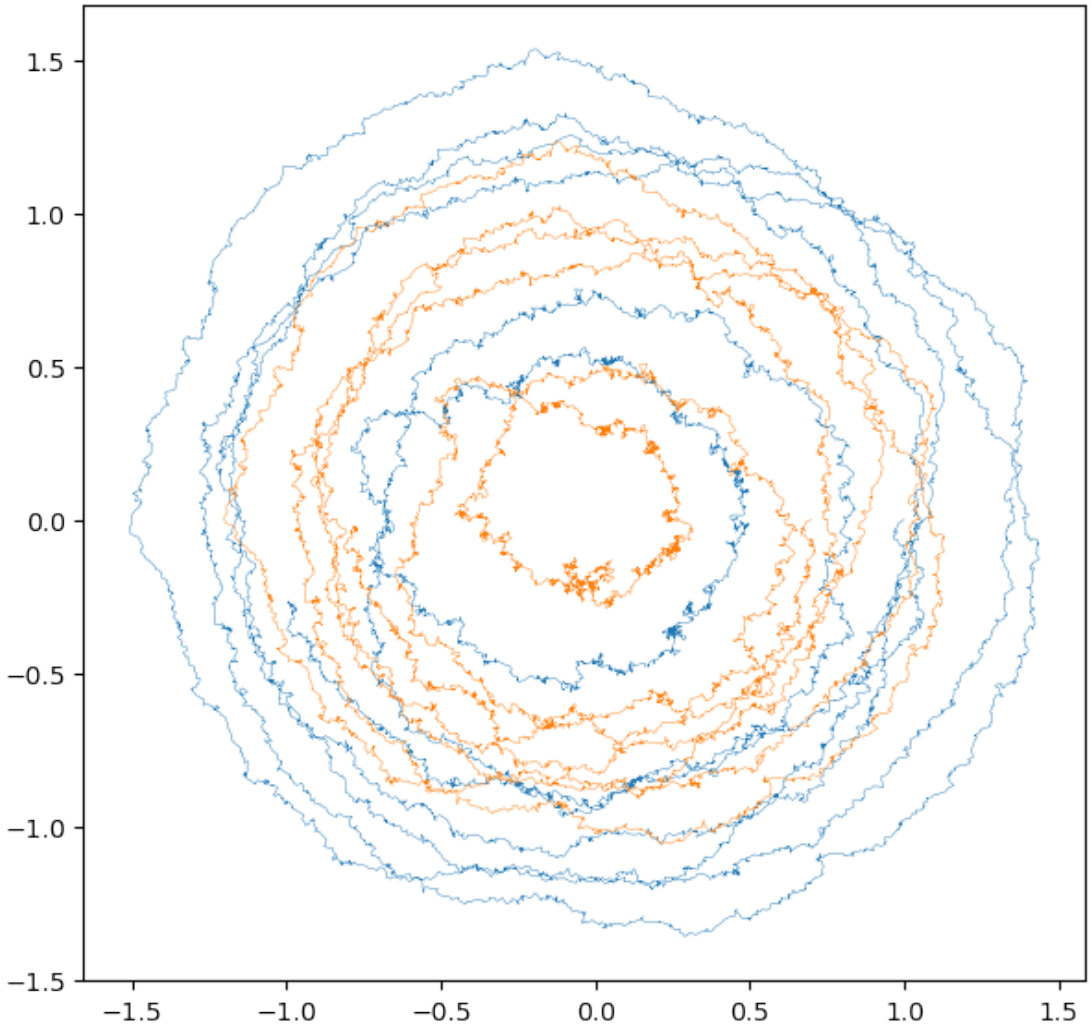
```
stepslist = [10000,100000]
# number of steps that will be simulated


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

    noise = np.random.randn(2,steps)*std
    # create a 2 times steps dimensional matrix of normal random numbers with varianc

    sde = np.ones((2,steps,k))
    sde[:,0,:] = x0


    for n in range(steps-1):
        for i in range(k):
            sde[:,n+1,i] = sde[:,n,i]+h*np.matmul(A,sde[:,n,i])+sigma*noise[:,n]


    plt.figure(figsize=(7,7), dpi=100)
    plt.plot(sde[0],sde[1],linewidth=.3)
    plt.show()
```

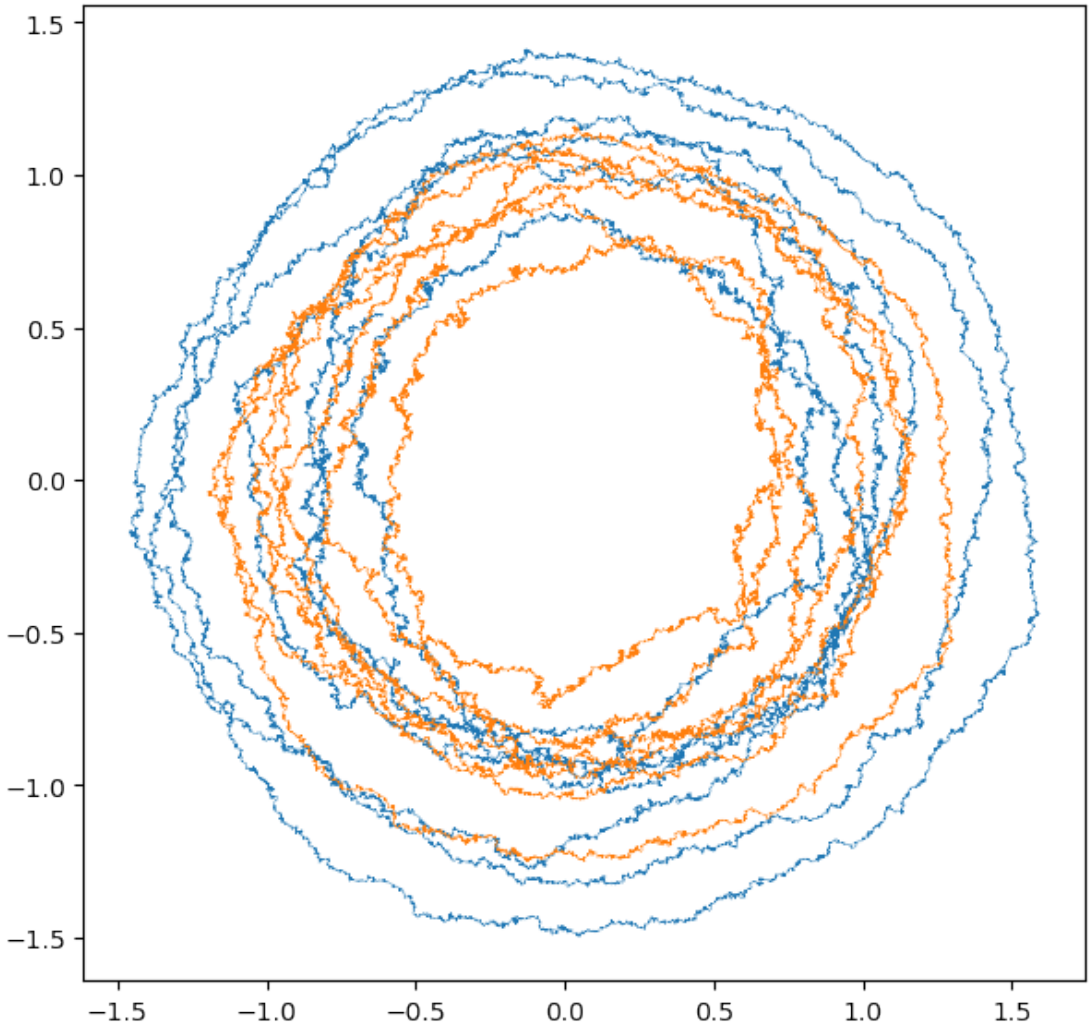

8
6
4
2
0
−2
−4
−6
−10
−8
−6
−4
−2
0
2
4
6



In [13]:
```
k = 2
# number of copies
x0 = np.array([[1.,.7],
       [0.,0.]])
# initial value at time t=0
A = np.array([[0.,1.],[-1.,0.]])
# drift matrix for randomly perturbed rotation
sigma = .1
# small noise coefficient

tmax = 40.
# simulation from time 0 to tmax
stepslist = [10000,100000]
# number of steps that will be simulated
```

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

    noise = np.random.randn(2,steps)*std
    # create a 2 times steps dimensional matrix of normal random numbers with varianc

    sde = np.ones((2,steps,k))
    sde[:,0,:] = x0


    for n in range(steps-1):
        for i in range(k):
            sde[:,n+1,i] = sde[:,n,i]+h*np.matmul(A,sde[:,n,i])+sigma*noise[:,n]


    plt.figure(figsize=(7,7), dpi=100)
    plt.plot(sde[0],sde[1],linewidth=.3)
    plt.show()
```

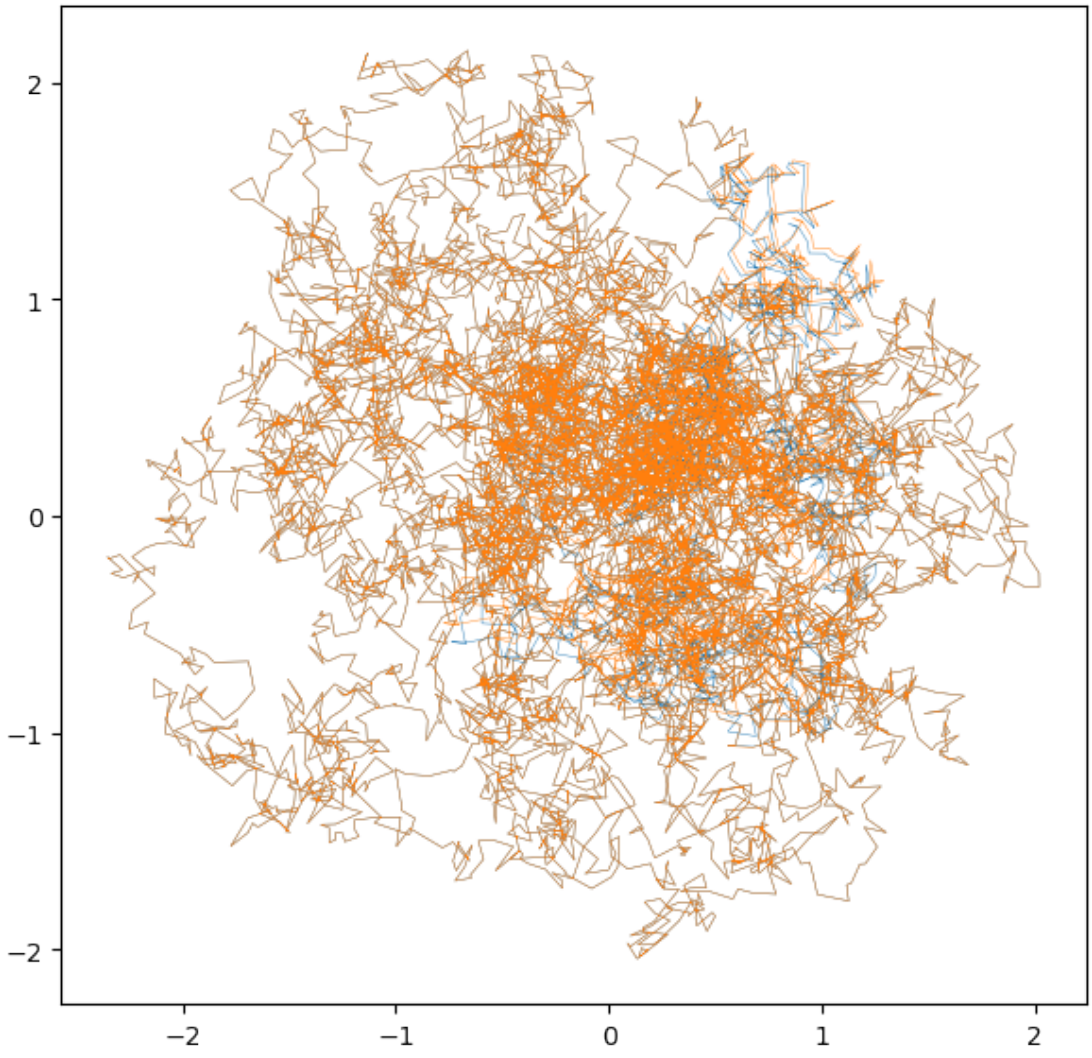

1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

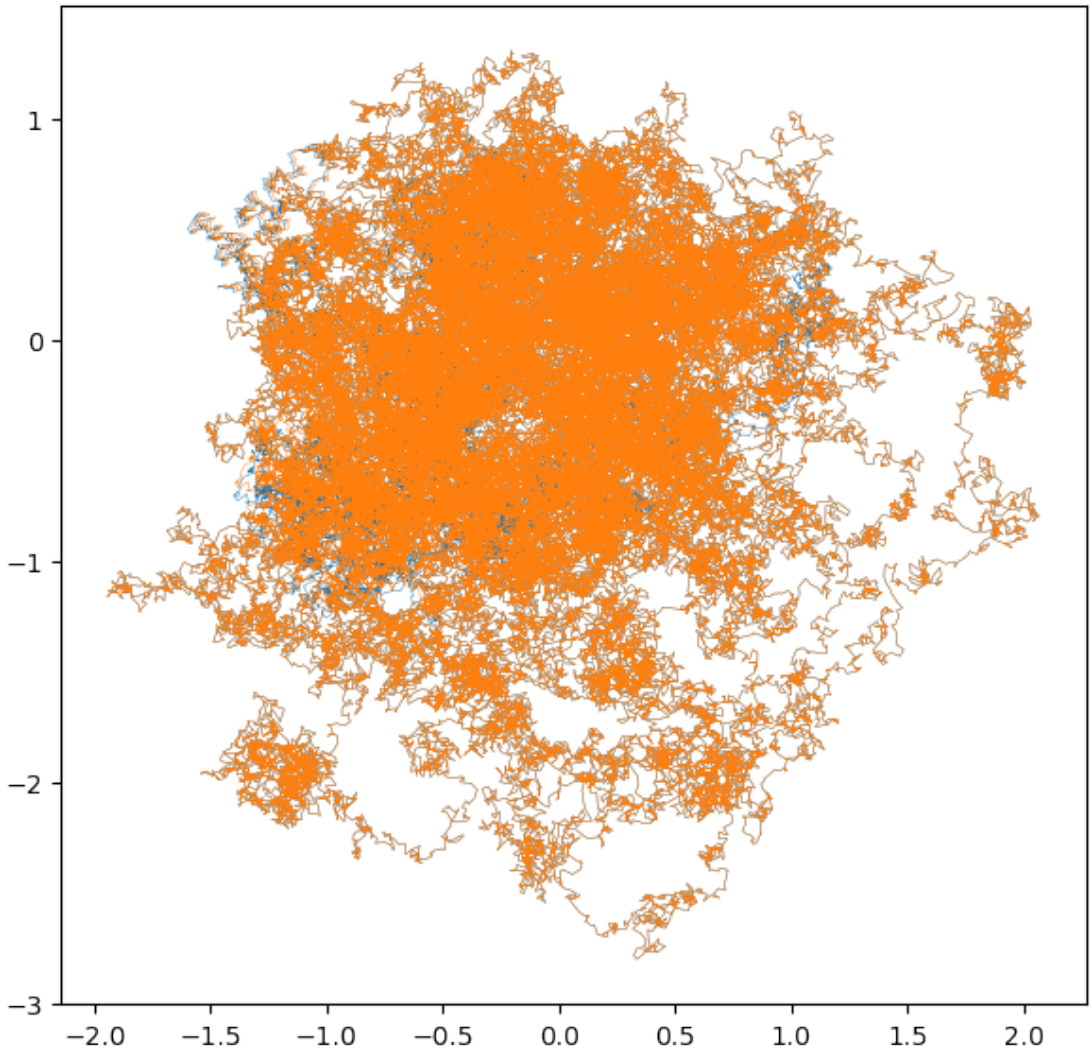


d) Randomly perturbed rotations with damping

```
In [14]: k = 2
         # number of copies
         x0 = np.array([[1.,.7],
                [0.,0.]])
         # initial value at time t=0
         gamma = -1.
         # damping parameter
         A = np.array([[gamma,1.],[-1.,gamma]])
         # drift matrix for randomly perturbed rotation with damping
         sigma = 1.
         # volatility

         tmax = 40.
```

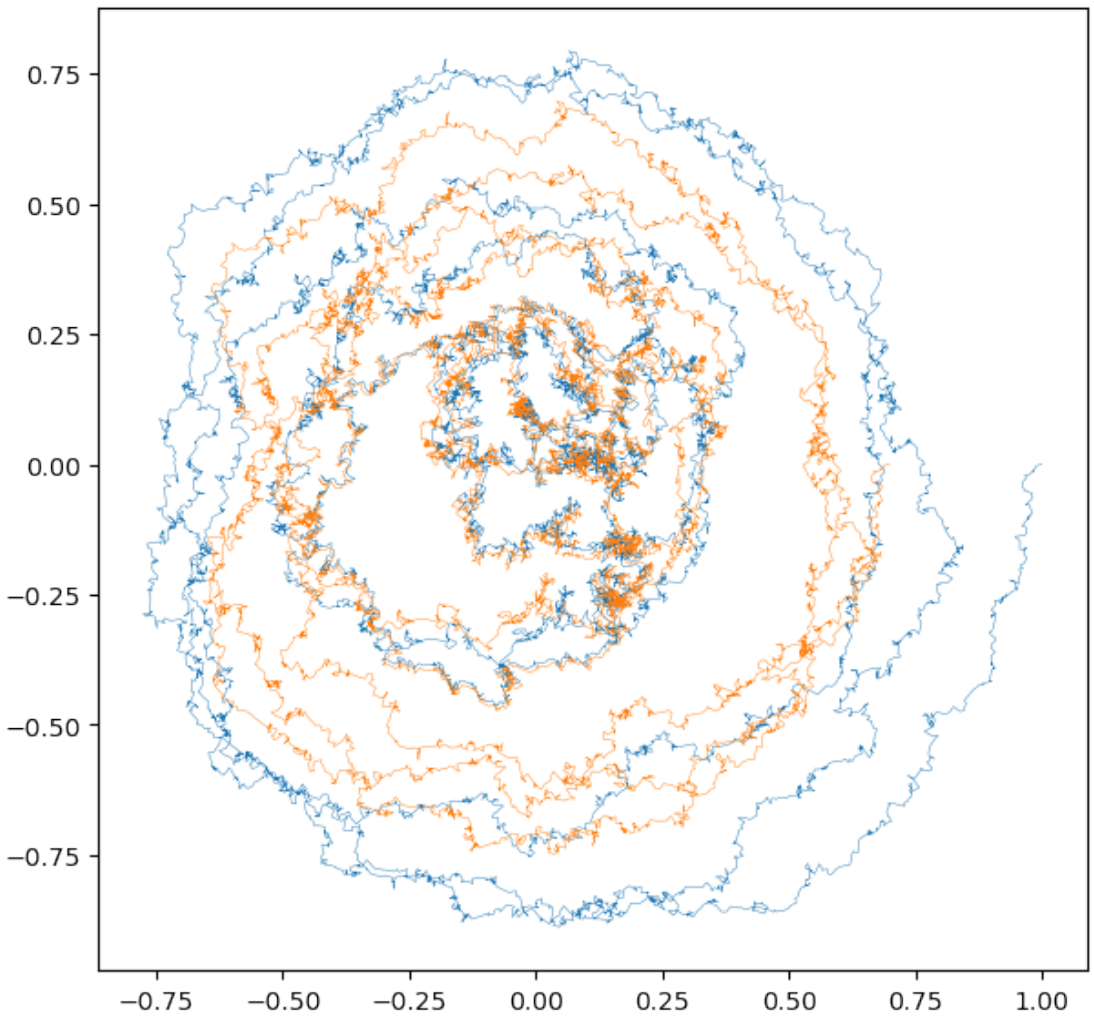
```
# simulation from time 0 to tmax
stepslist = [10000,100000]
# number of steps that will be simulated


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

    noise = np.random.randn(2,steps)*std
    # create a 2 times steps dimensional matrix of normal random numbers with varianc

    sde = np.ones((2,steps,k))
    sde[:,0,:] = x0


    for n in range(steps-1):
        for i in range(k):
            sde[:,n+1,i] = sde[:,n,i]+h*np.matmul(A,sde[:,n,i])+sigma*noise[:,n]


    plt.figure(figsize=(7,7), dpi=100)
    plt.plot(sde[0],sde[1],linewidth=.3)
    plt.show()
```

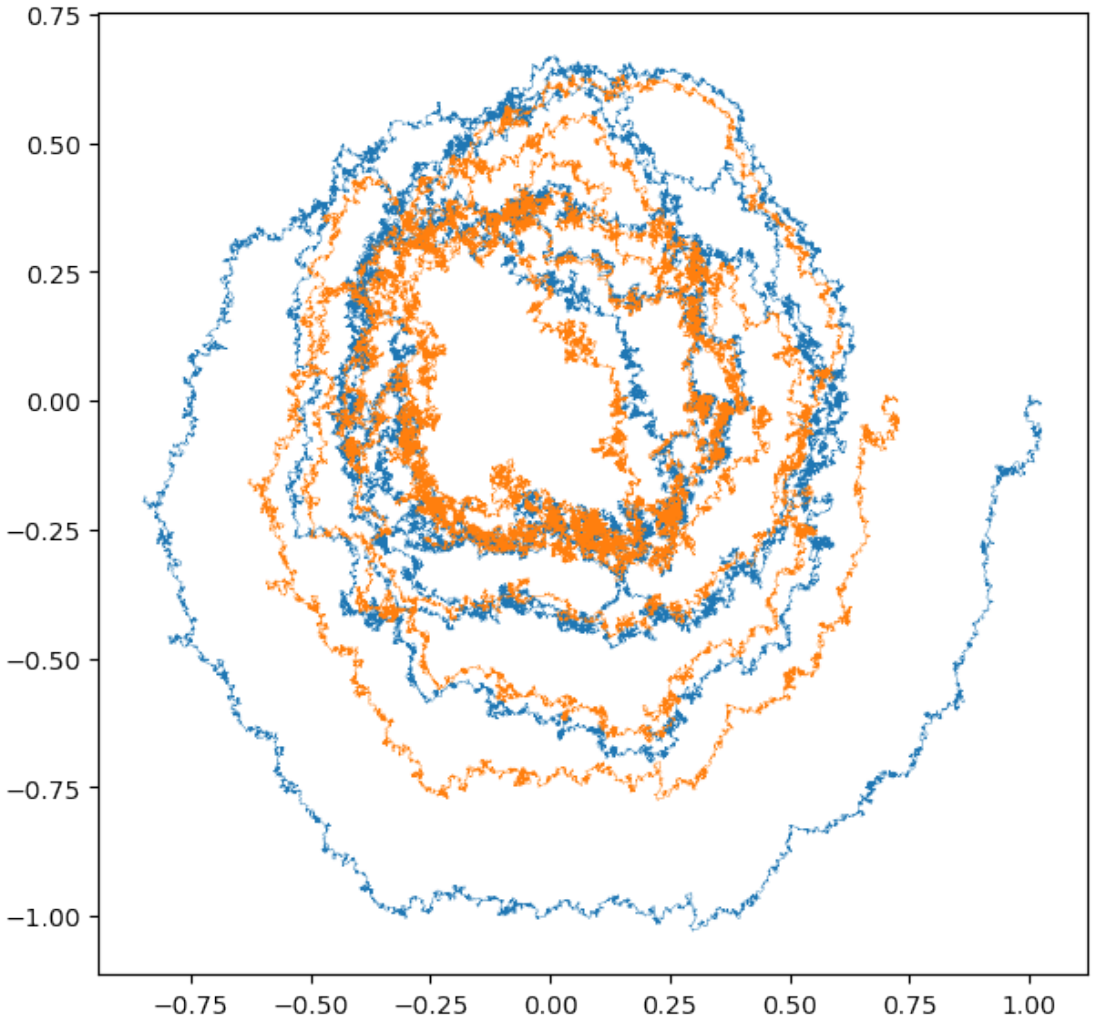

2
1
0
−1
−2
−2
−1
0
1
2




```
In [15]: k = 2
         # number of copies
         x0 = np.array([[1.,.7],
                [0.,0.]])
         # initial value at time t=0
         gamma = -.1
         # small damping parameter
         A = np.array([[gamma,1.],[-1.,gamma]])
         # drift matrix for randomly perturbed rotation with damping
         sigma = .1
         # volatility

         tmax = 40.
         # simulation from time 0 to tmax
```

```
stepslist = [10000,100000]
# number of steps that will be simulated


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

    noise = np.random.randn(2,steps)*std
    # create a 2 times steps dimensional matrix of normal random numbers with varianc

    sde = np.ones((2,steps,k))
    sde[:,0,:] = x0


    for n in range(steps-1):
        for i in range(k):
            sde[:,n+1,i] = sde[:,n,i]+h*np.matmul(A,sde[:,n,i])+sigma*noise[:,n]


    plt.figure(figsize=(7,7), dpi=100)
    plt.plot(sde[0],sde[1],linewidth=.3)
    plt.show()
```

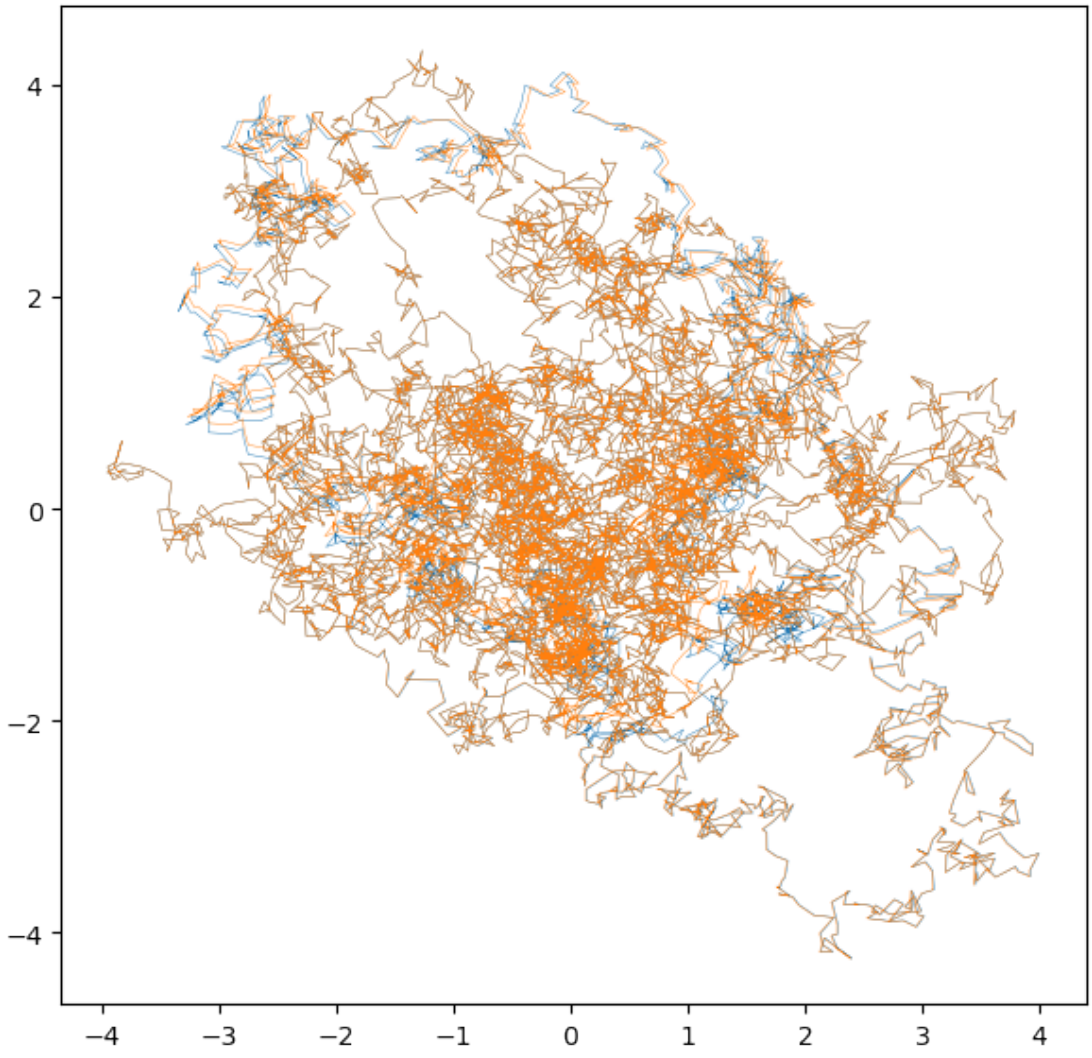

0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

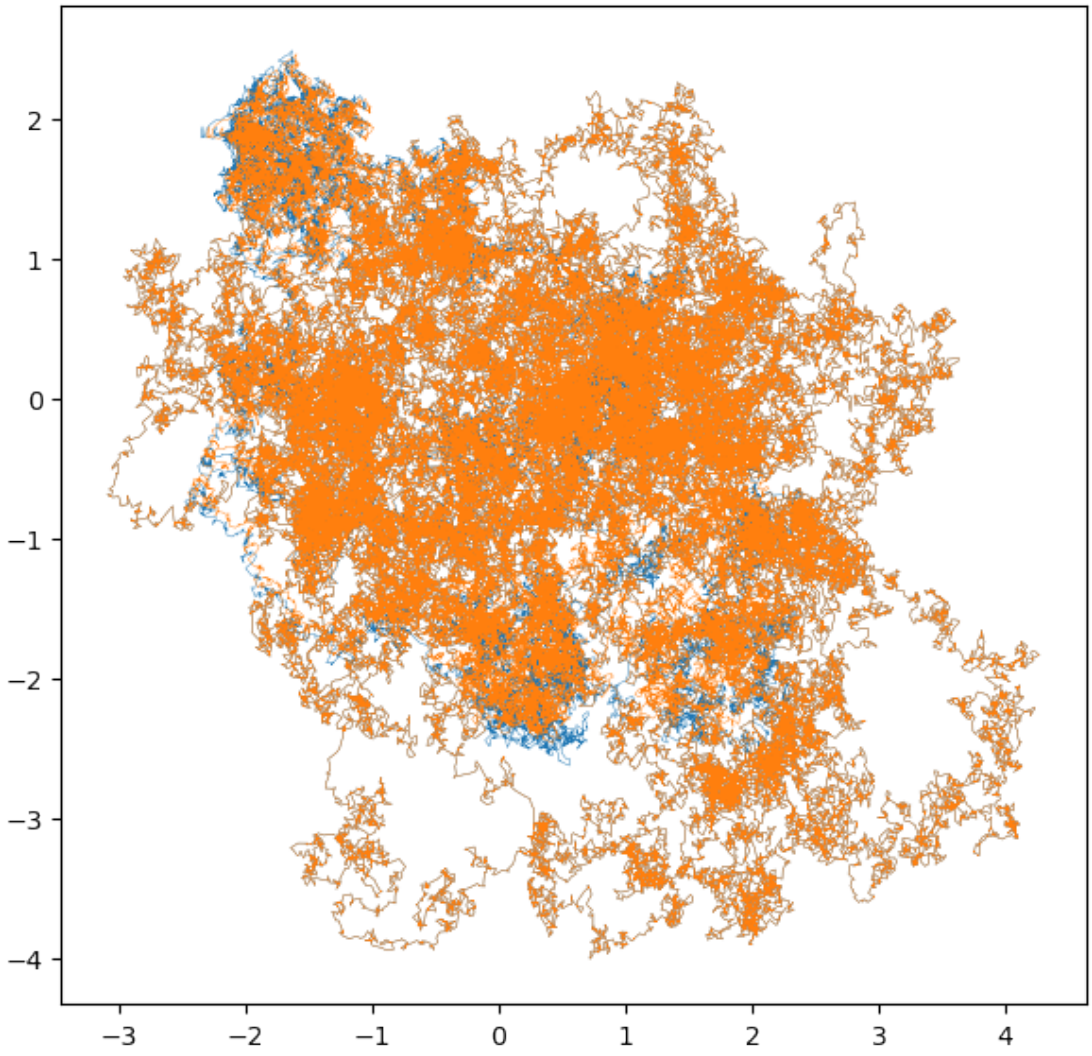


e) Physical Brownian motions in phase space (position+velocity)

```
In [16]: k = 2
         # number of copies
         x0 = np.array([[1.,.7],
                [0.,0.]])
         # initial value at time t=0
         gamma = 1.
         # damping parameter
         A = np.array([[0.,1.],[-1.,-gamma]])
         # drift matrix for physical Brownian motion in phase space (position,velocity)
         sigma = np.sqrt(2.*gamma)
         # volatility for physical Browmain motion

         tmax = 40.
```

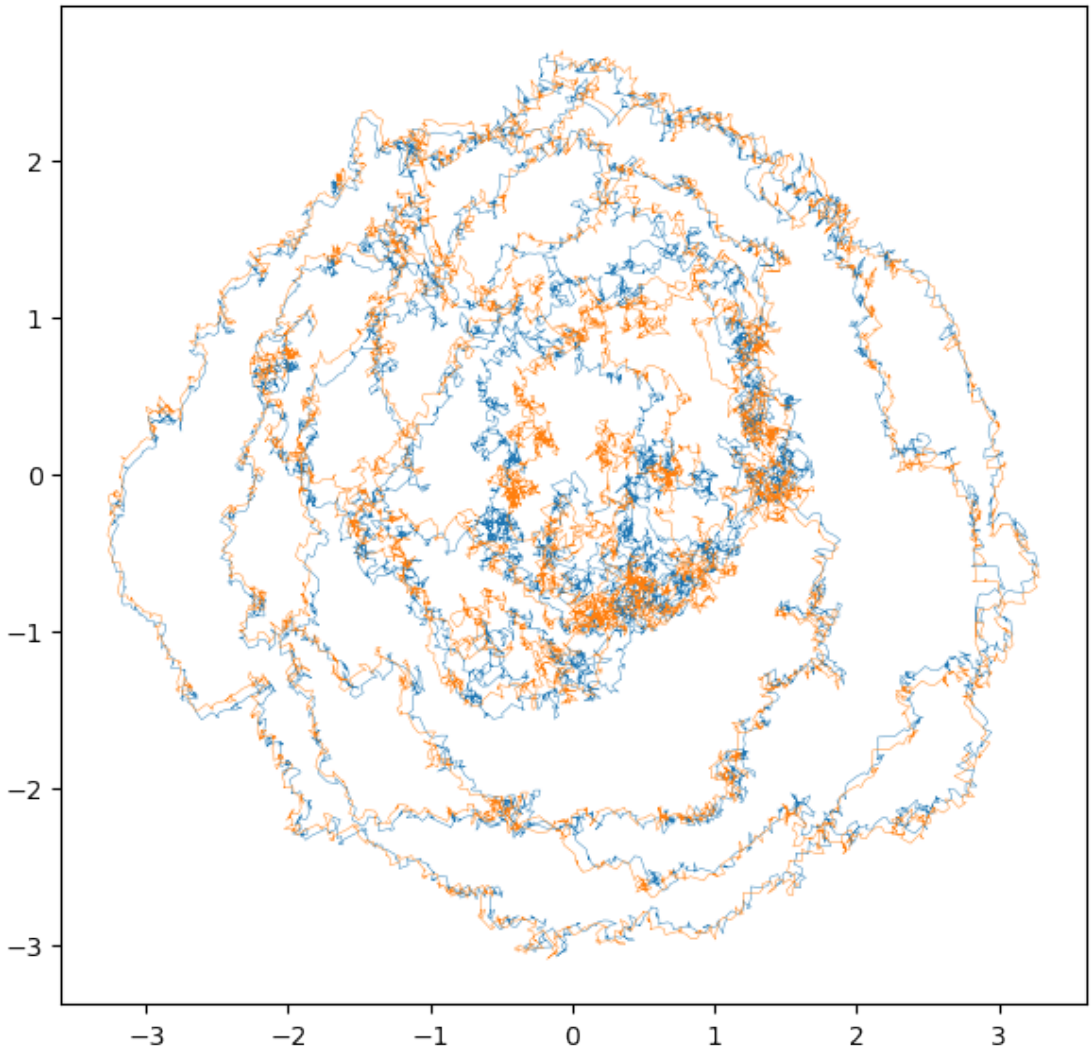
```
# simulation from time 0 to tmax
stepslist = [10000,100000]
# number of steps that will be simulated


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

    noise = np.random.randn(2,steps)*std
    # create a 2 times steps dimensional matrix of normal random numbers with varianc

    sde = np.ones((2,steps,k))
    sde[:,0,:] = x0


    for n in range(steps-1):
        for i in range(k):
            sde[:,n+1,i] = sde[:,n,i]+h*np.matmul(A,sde[:,n,i])+sigma*noise[:,n]


    plt.figure(figsize=(7,7), dpi=100)
    plt.plot(sde[0],sde[1],linewidth=.3)
    plt.show()
```

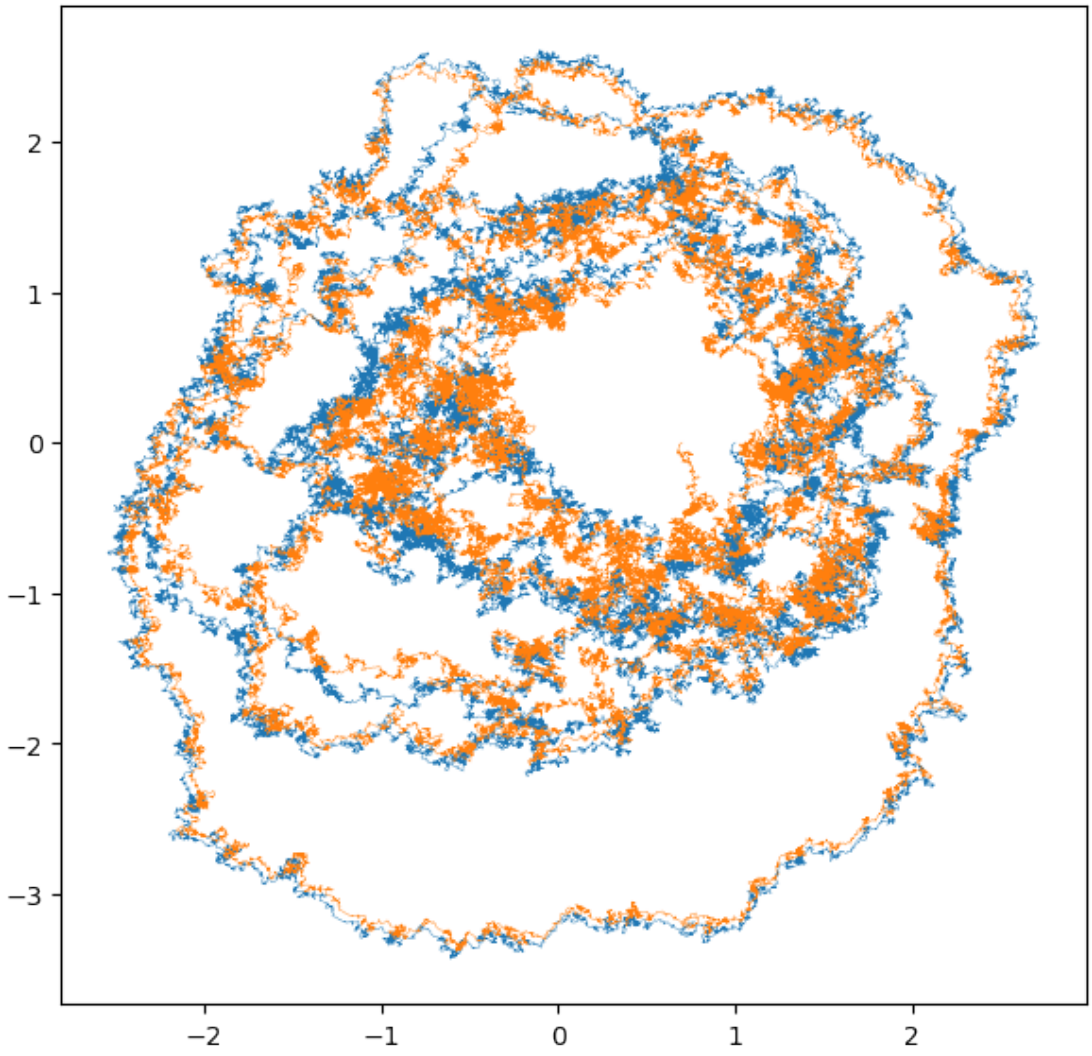

4
2
0
−2
−4
−4
−3
−2
−1
0
1
2
3
4



In [17]:
```
k = 2
# number of copies
x0 = np.array([[1.,.7],
       [0.,0.]])
# initial value at time t=0
gamma = .1
# little damping and noise
A = np.array([[0.,1.],[-1.,-gamma]])
# drift matrix for physical Brownian motion in phase space
# (first coordinate position, second coordinate velocity)
# (=harmonic oscillator with friction and noise)
sigma = np.sqrt(2.*gamma)
# volatility for physical Browmain motion
```

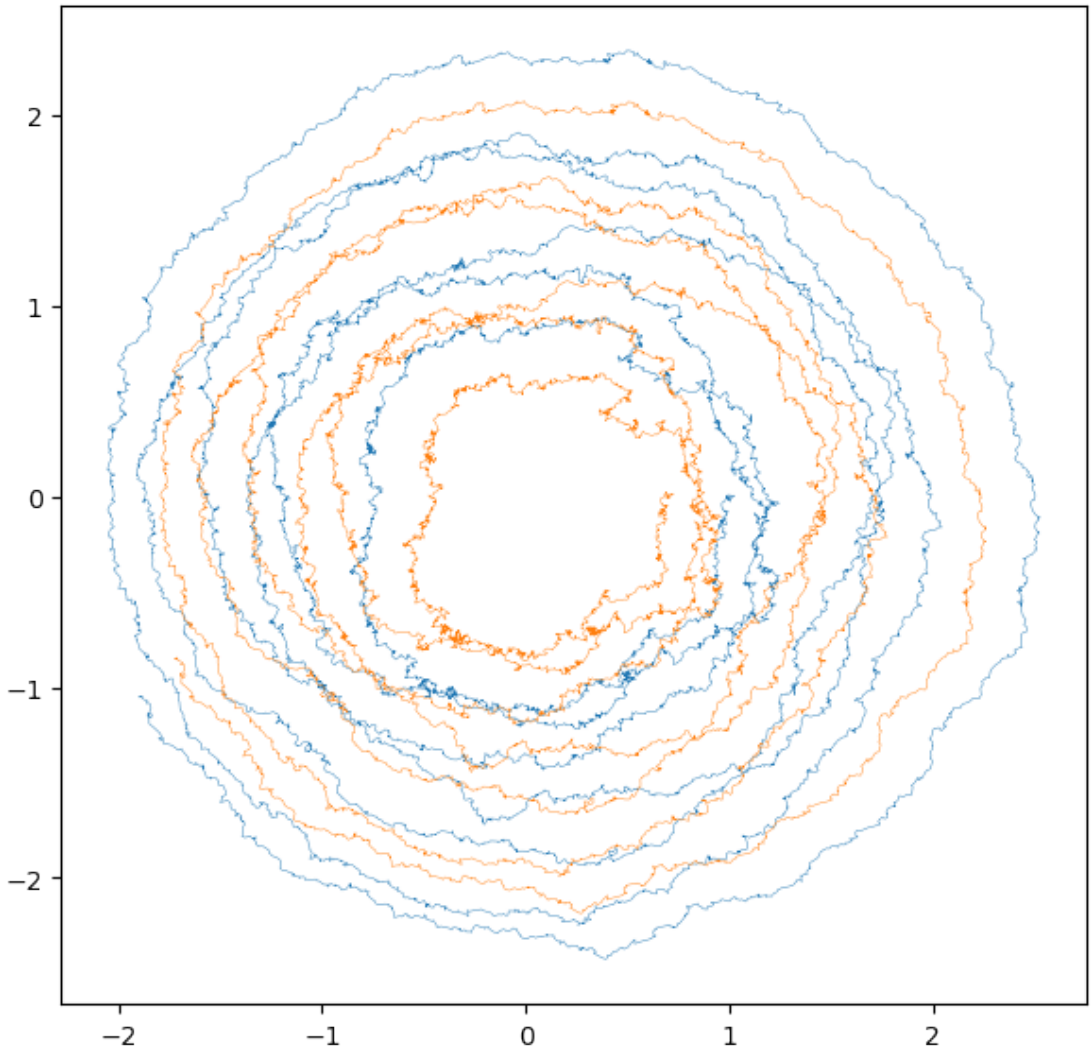
```
tmax = 40.
# simulation from time 0 to tmax
stepslist = [10000,100000]
# number of steps that will be simulated


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

    noise = np.random.randn(2,steps)*std
    # create a 2 times steps dimensional matrix of normal random numbers with varianc

    sde = np.ones((2,steps,k))
    sde[:,0,:] = x0


    for n in range(steps-1):
        for i in range(k):
            sde[:,n+1,i] = sde[:,n,i]+h*np.matmul(A,sde[:,n,i])+sigma*noise[:,n]


    plt.figure(figsize=(7,7), dpi=100)
    plt.plot(sde[0],sde[1],linewidth=.3)
    plt.show()
```

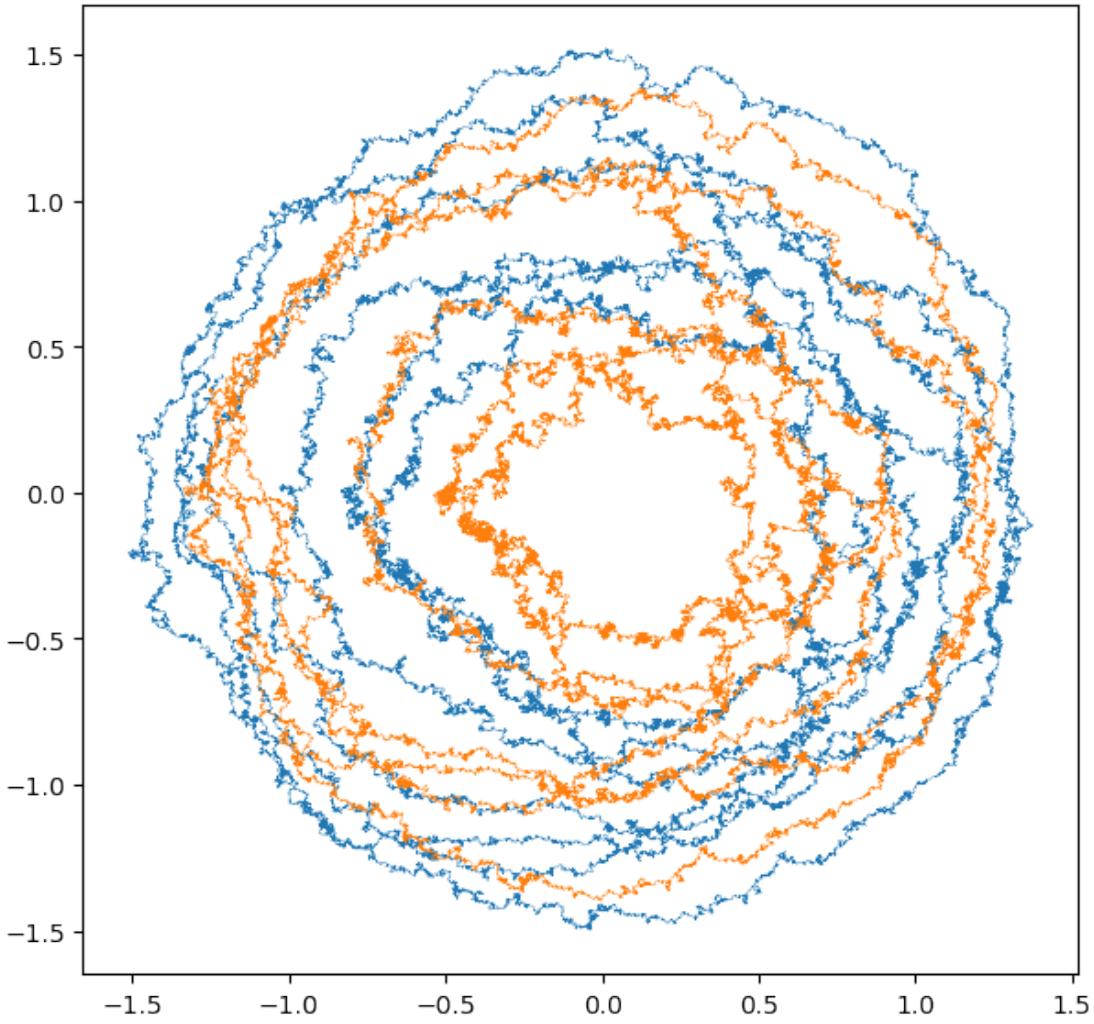

2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3




```
In [18]: k = 2
         # number of copies
         x0 = np.array([[1.,.7],
                [0.,0.]])
         # initial value at time t=0
         gamma = .01
         # very little damping and noise
         A = np.array([[0.,1.],[-1.,-gamma]])
         # drift matrix for physical Brownian motion in phase space (position,velocity)
         sigma = np.sqrt(2.*gamma)
         # volatility for physical Brownain motion

         tmax = 40.
         # simulation from time 0 to tmax
```

```
stepslist = [10000,100000]
# number of steps that will be simulated


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

    noise = np.random.randn(2,steps)*std
    # create a 2 times steps dimensional matrix of normal random numbers with varianc

    sde = np.ones((2,steps,k))
    sde[:,0,:] = x0


    for n in range(steps-1):
        for i in range(k):
            sde[:,n+1,i] = sde[:,n,i]+h*np.matmul(A,sde[:,n,i])+sigma*noise[:,n]


    plt.figure(figsize=(7,7), dpi=100)
    plt.plot(sde[0],sde[1],linewidth=.3)
    plt.show()
```


2
1
0
−1
−2
−2
−1
0
1
2



In [ ]: