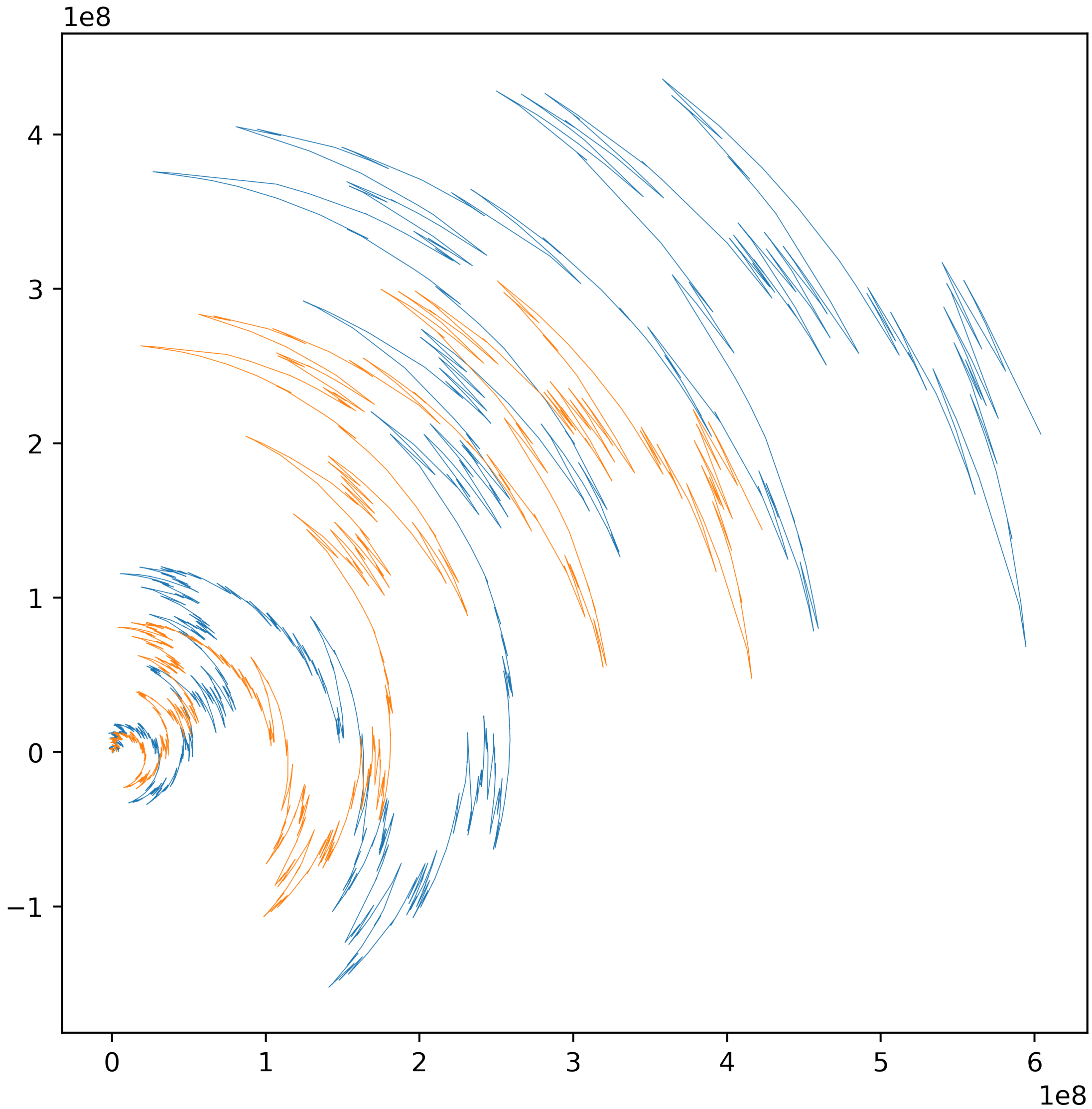
Random rotations Ito vs Stratonovich

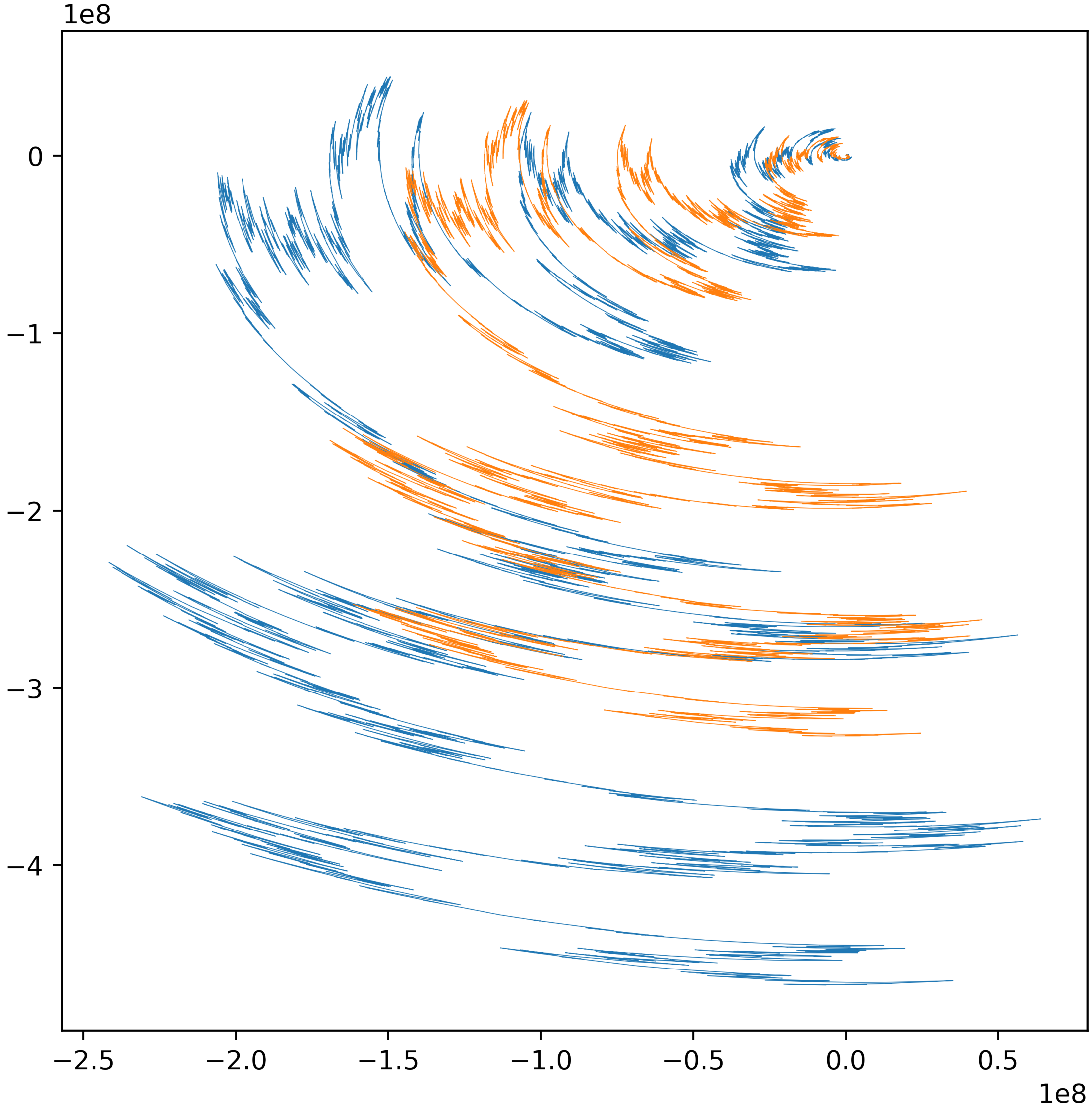
December 13, 2018

a) Euler discretization of Ito SDE without correction term

In [2]:
```
# Euler-Maryuama Approximation for solution of SDE
# dZ_t = A Z_t dB_t
# in R^2 where B_t is a one-dimensional BM and A is an antisymmetric matrix
# 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.,-1.],[1.,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(steps)*std
    # create a steps dimensional vector of normal random numbers with variance h
```

```
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]+np.matmul(A,sde[:,n,i])*noise[n]


plt.figure(figsize=(7,7), dpi=500)
plt.plot(sde[0],sde[1],linewidth=.3)
plt.show()
```

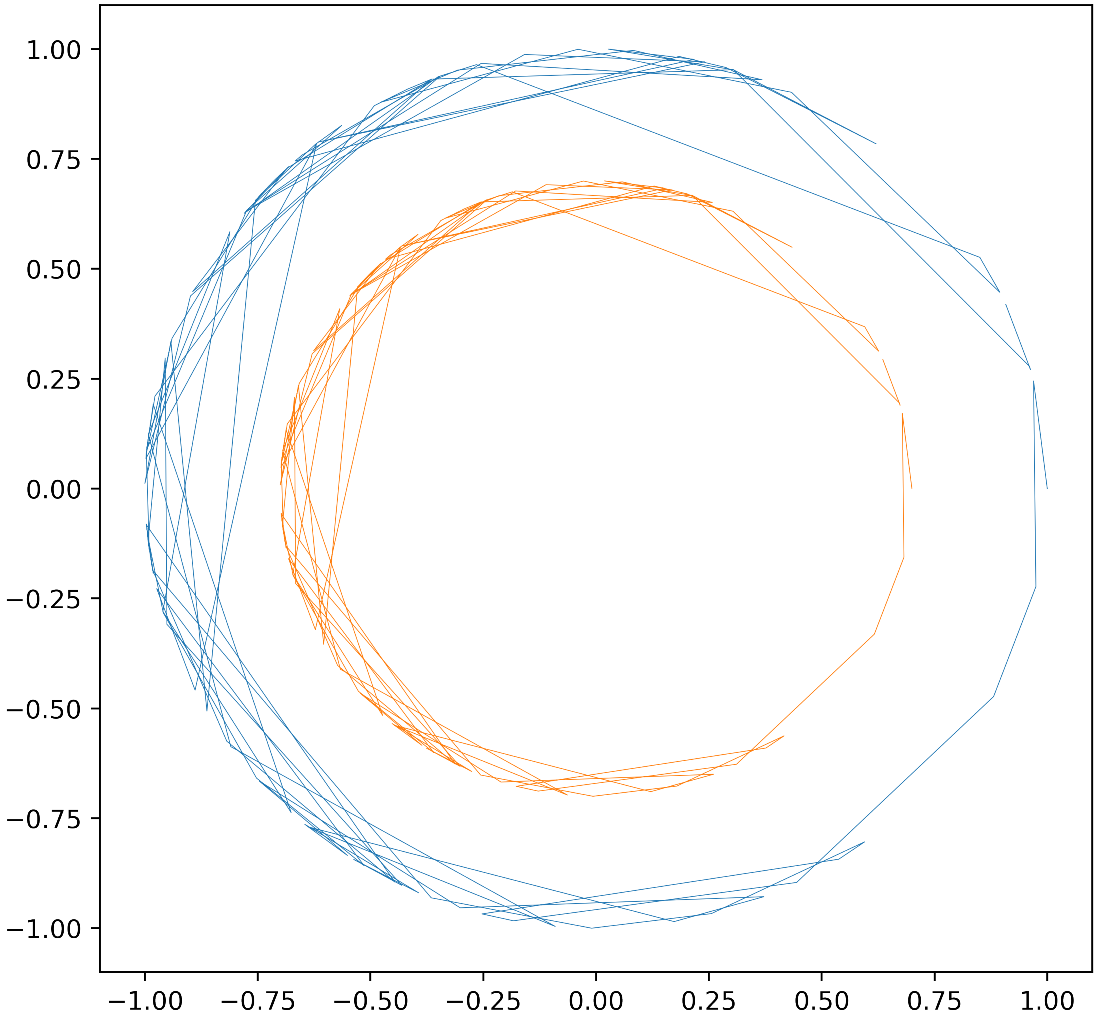


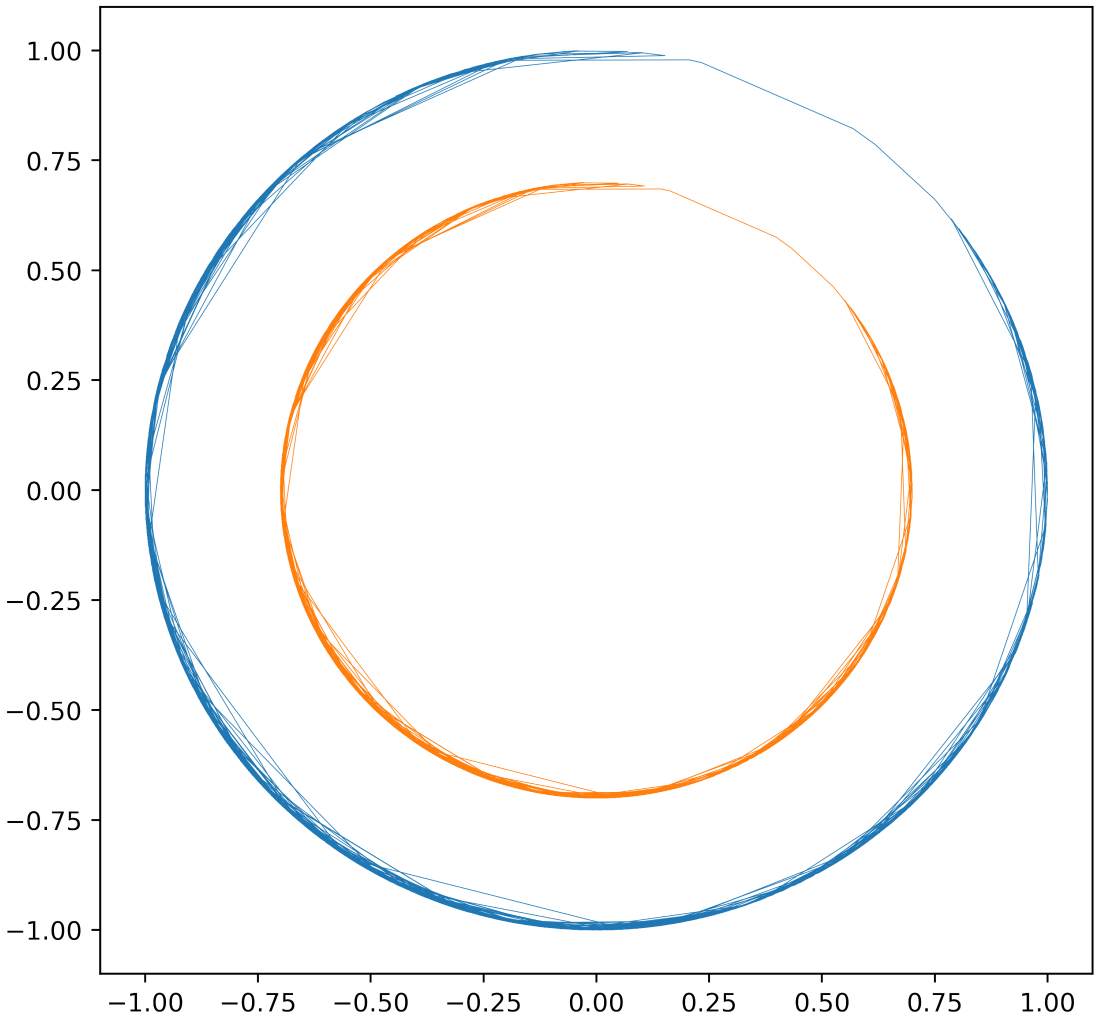


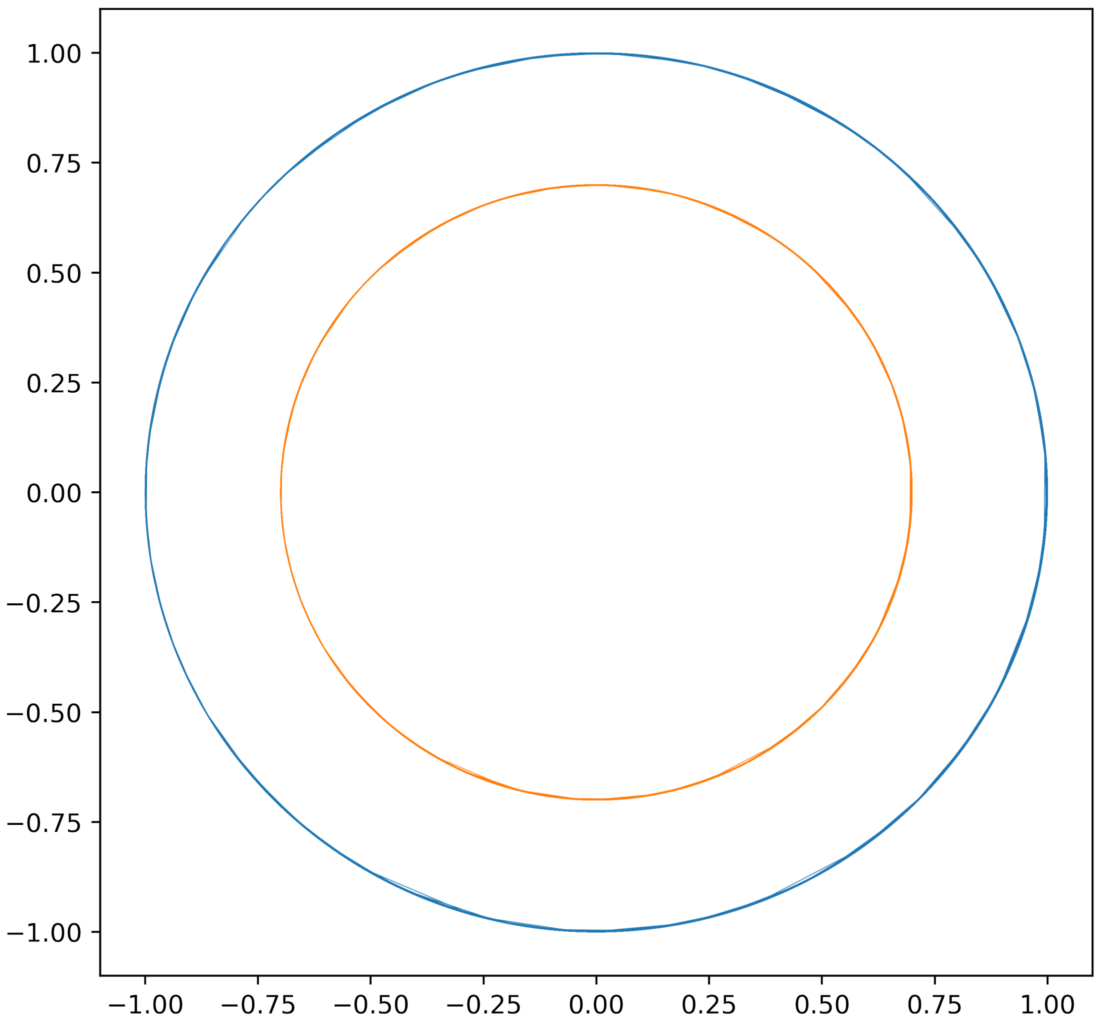
b) Discretization of Stratonovich SDE

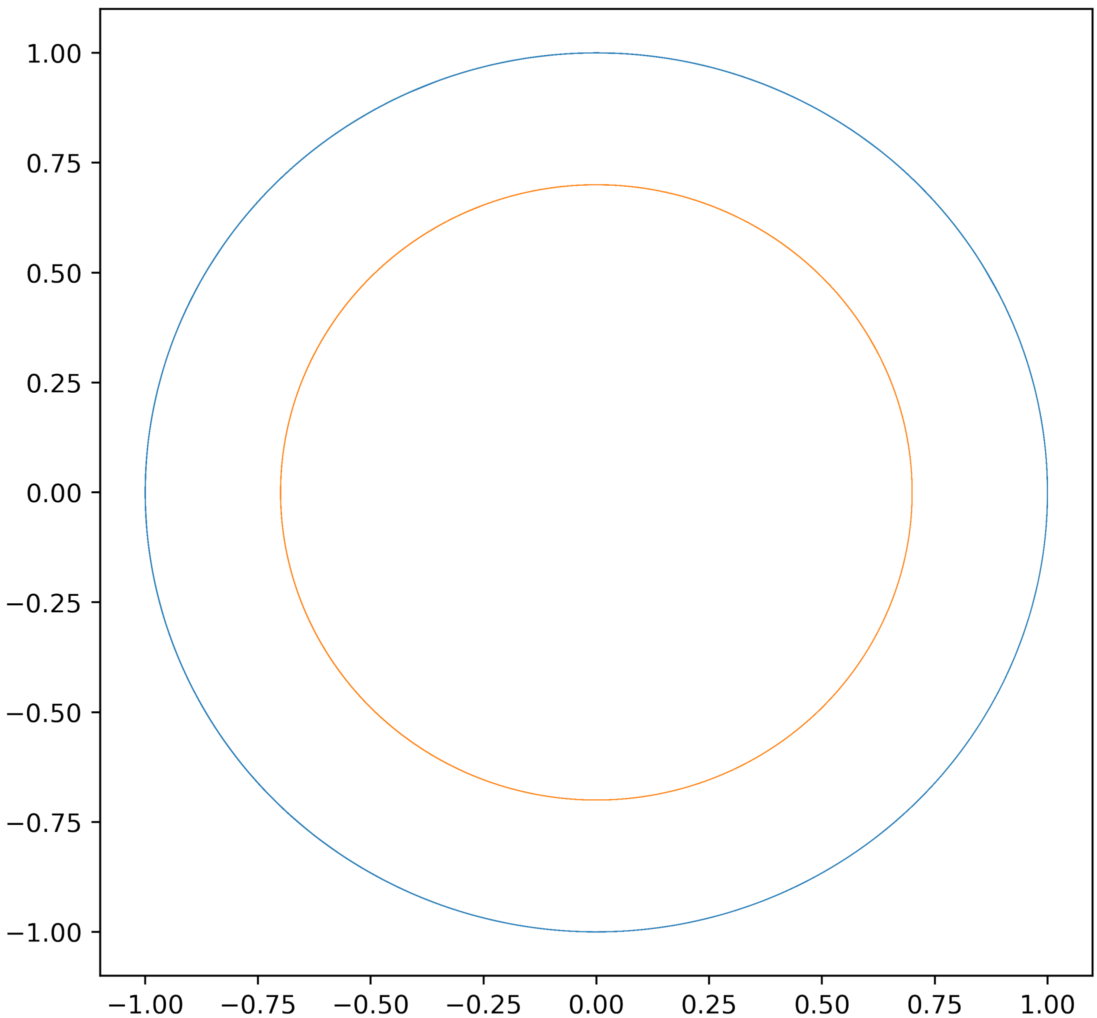
```
In [4]: # Discretization for Stratonovich equation
        # dZ_t = A Z_t dB_t
        # in R^2 where B_t is a one-dimensional BM and A is an antisymmetric matrix
        # Simulation of flow starting from several initial conditions
        #
        # The Euler type discretization of the Stratonovich SDE can be written as
        # Z' = (I-(A/2)dB)^{-1}Z + (I-(A/2)dB)^{-1}(A/2)Z dB

        import numpy as np
```

```python
# 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]

from scipy import linalg

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 Brownian motion
I = np.array([[1.,0.],[0.,1.]])
# identity matrix
sigma = 1.
# volatility

tmax = 40.
# simulation from time 0 to tmax
stepslist = [100,1000,10000,100000]
# 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(steps)*std
    # create a steps dimensional vector of normal random numbers with variance h

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


    for n in range(steps-1):
        for i in range(k):
            B = linalg.inv(I-A*noise[n]/2.)
            C = np.matmul(B,A)/2.
            sde[:,n+1,i] = np.matmul(B,sde[:,n,i])+np.matmul(C,sde[:,n,i])*noise[n]


    plt.figure(figsize=(7,7), dpi=500)
    plt.plot(sde[0],sde[1],linewidth=.3)
    plt.show()
```


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


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


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


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