In [4]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

1. Dynamical Systems

1.1. Phase diagram: Consider the dynamical system of a undamped pendulum with no driving force described by the differential equation $y'' + \sin(y) = 0$. We can rewrite it as

$$\begin{cases} y_1' = y_2 \\ y_2' = -\sin(y_1). \end{cases}$$

The phase portrait is a plot of a vector field which qualitatively shows how the solutions to these equations will go from a given starting point. To generate the phase portrait, we need to compute the derivatives y_1' and y_2' at $t = 0$ on a grid over the range of values for y_1 and y_2 we are interested in. Plot the derivatives as a vector at each $(y_1, y_2) \in [-2, 8] \times [-3, 3]$ which will show us the initial direction from each point (create a grid of 20×20 points).

In [2]:

```
# Use matplotlib.pyplot.quiver to plot a 2D field of arrows.
```

In [ ]:

```
def pendulum_ode(y, t):
    '''
    CODE HERE
    '''
    return []
```

1.2. Trajectories: The phase diagram shows the trajectories that a dynamical system can take through its phase space. Plot a few solutions on the vector field. Consider the solutions where $y_1(0) = 0$, and values of $y_2(0) = [0, 0.5, 1, 1.5, 2, 2.5]$, in other words let the pendulum start at an angle of zero, with various angular velocity.

In [ ]:

```
from scipy.integrate import odeint

angular_velocities = [0, 0.5, 1, 1.5, 2, 2.5]

for y20 in angular_velocities:
    tspan = np.linspace(0, 50, 200)

    '''
    CODE HERE
    '''
```

2. Brownian Motion

Consider a the stochastic process of Brownian motion, formally expressed as $dX(t) = b(t, X) \cdot dW(t)$.

It numeric simulation can be expressed as $X[n+1] = X[n] + dX = X[n] + b(t, X[n]) \cdot \xi\sqrt{dt}$, where ξ is normally distributed with mean 0 and variance 1.

2.1. Simulate and visualize a 2D random walk of your choice, meaning choose function $b(\cdot)$ as you like.

In [2]:

```
def generate_2D_brownian_walk(dt = 0.001, T = 1.):

    n = int(T / dt)  # Number of time steps.
    t = np.linspace(0., T, n)  # Vector of times.

    '''
    CODE HERE
    '''

    return
```

In [6]:

```
x,y = generate_2D_brownian_walk()
```

In [ ]:

```
fig = plt.figure()
plt.plot(x,y)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Simulation of Brownian motion")
plt.show()
```

In [ ]:

```
n_trajs = 5

fig = plt.figure(figsize=(12,12))
plt.xlabel("x")
plt.ylabel("y")
plt.title("Simulation of {} Brownian motions".format(n_trajs))

for j in range(n_trajs):
    x,y = generate_2D_brownian_walk(T=0.05)
    plt.plot(x,y)

plt.show()
```

In [ ]:

```

```

3. Stochastic Differential Equations

A stochastic differential equation (SDE) is a differential equation with at least one stochastic process term. The general form of a SDE is

$$dX(t) = a(t, X) \cdot dt + b(t, X) \cdot dW(t)$$

where $a(t, X)$ is a deterministic function known as **drift**, $b(t, X)$ is the **diffusion** coefficient and $dW(t)$ is the stochastic Wiener process (the Brownian motion of Exercise 2).

The solution can be expressed in the integral form

$$X(t) = X(0) + \int_0^t a(s, X(s))ds + \int_0^t b(s, X(s)dW(s),$$

where the second integral is an *Ito integral*.

Analytic solution

If we're lucky, a and b are such that we can solve for $X(t)$ analytically by computing the integrals. For example, in the special case of **Geometric Brownian Motion** where $a(t, x) = \mu x$ and $b(t, x) = \sigma x$, the SDE is

$$dX(t) = \mu X(t) \cdot dt + \sigma X(t) \cdot dW(t).$$

The corresponding solution can be expressed explicitly as follows:

$$X(t) = X(0) \exp\big((\mu - \sigma^2/2)t + \sigma W(t)\big).$$

A solution to an SDE is itself a stochastic function, meaning that, for any t, $X(t)$ is a random variable.

Define a function that simulates N different sample paths for $X(t)$, sample paths differ because of different realizations of the Brownian motion term.

Hint: Discretize a given time interval $[0, T]$ into n points t_i and compute the value of X at each t_i. To compute the value of W at those same points, remember that the increments $\Delta W(t_n) = W(t_n) - W(t_{n-1})$ are distributed according to $\sqrt{\Delta t} \cdot \mathcal{N}(0, 1)$. The sample path of Brownian motion W can be generated by summing the increments, and compute the exact solution X using the analytical solution above.

In [ ]:

- Compute the empirical approximation of the first two moments (mean and variance) over a large number of sample trajecories and **plot the confidence interval** over the trajectory space.

In [ ]:

3.1 Euler-Maruyana Method (E-M)

The integral form of Geometric Brownian Motion can be written as

$$X_{n+1} - X_n = \mu \int_{t_n}^{t_{n+1}} X(s)ds + \sigma \int_{t_n}^{t_{n+1}} X(s)dW(s).$$

The simplest approximation to this is the **Euler-Murayama method**, which is the stochastic generalization of the standard Euler method for ODEs:

$$X_{n+1} - X_n = \mu X_n \Delta t_n + \sigma X_n \Delta W_n.$$

- Compare the exact solution and the E-M approximation by simulating using two different values of Δt. Choose the "big Δt" values to be exact multiples of the smaller Δt values so that you can evaluate the processes at the same points.

In [ ]:

Comment: You should observe how the approximation gets "better" (ie. closer) as we reduce the size of our discrete time periods Δt.

3.2 Milstein method

The **Milstein method** increases the accuracy of the E-M approximation by adding a second-order "correction" term, which is derived from the stochastic Taylor series expansion of $X(t)$ by applying Ito's lemma to the $a(\cdot)$ and $b(\cdot)$ functions. he Milstein method yields the following differential form

$$X_{n+1} - X_n = a(X_n)\Delta t + b(X_n)\Delta W_n + \frac{1}{2} b'(X_n)b(X_n)\left((\Delta W_n)^2 - \Delta t\right),$$

which implies the following for our Geometric Brownian Motion example

$$X_{n+1} - X_n = \mu X_n \Delta t + \sigma X_n \Delta W_n + \frac{1}{2}\sigma^2 X_n \left((\Delta W_n)^2 - \Delta t\right).$$

- Simulate a single path using the same draw from the Brownian motion above, and plot it with the exact solution and the E-M approximation.

In [ ]:

Comment: Observe how the Milstein approximation looks to be closer than the corresponding E-M approximation. As mentioned above, there are many ways to formally define what we mean when we say that one stochastic process is close to another. We move on to this next.

3.3 Convergence

The concept of convergence formalizes what it means for one stochastic process to get closer to another as the discrete time steps Δt are reduced.

Weak convergence defines the following error term

$$e^w(\Delta t) = \sup_{t_n} \mid \mathbb{E}[X(t_n)] - \mathbb{E}[Y(t_n)] \mid.$$

The weak error term computes the error between the expected values of the two stochastic processes at a given point. So weak convergence captures the average behavior of the simulated approximations.

Strong convergence defines the following error term

$$e^s(\Delta t) = \sup_{t_n} \mathbb{E}[\mid X(t_n) - Y(t_n) \mid].$$

The strong error is the mean of errors, which captures the difference between the approximation and the exact solution for each individual sample path before the average is taken. Strong convergence is therefore more demanding than weak convergence.

We say that $X(t)$ exhibits strong or weak convergence to $Y(t)$ if the corresponding error tends to zero with Δt:

$$\lim_{\Delta t \to 0} e(\Delta t) = 0.$$

Choosing to require weak or strong convergence depends on the type of closeness we are interested in: “closeness” of the whole trajectory of the solution to the SDE (in which case we care about strong convergence) or in the expected value of some function of the process (which relates to weak convergence).

- Computes both the weak and the strong convergence errors for the E-M and for the Milstein approximations over a range of decreasing values of Δt. Simulate $10,000$ sample paths for each value of Δt. Plot the weak and strong error terms for each approximation against the Δt values on log-log axes.

In [13]:

```
dt_grid = [2 ** (R-10) for R in range(7)]
n_samples = 10000

# Loop over values of dt
for dt in dt_grid:


    # Generate sample paths
    for i in range(n_samples):

        # Create Brownian Motion

        # Exact solution

        # Euler-Maruyama

        # Milstein

        # Compute strong errors and add to those across from other sample pa

    # Compute mean of absolute errors and find maximum (strong error)

    # Compute error of means and find maximum (weak error)
```

In [8]:

```
# Plots
```

- Comment the obtained results.