TD4_Numerical_Solution_Of_ODEs

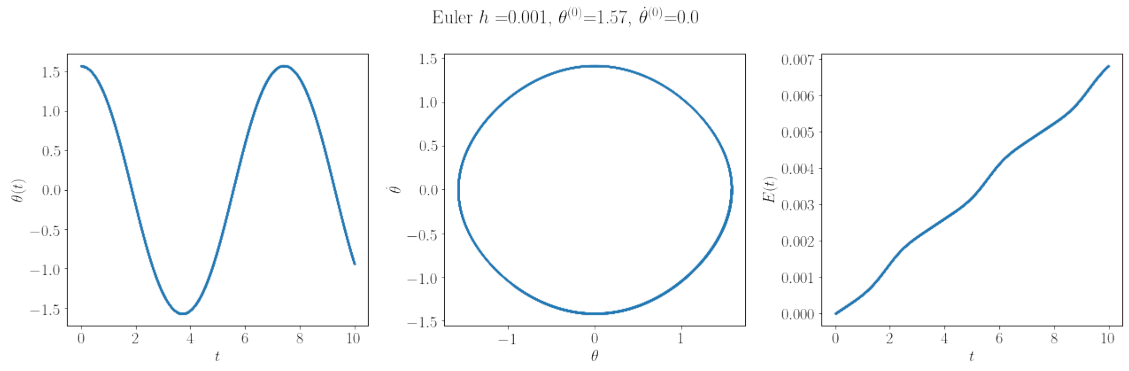
October 3, 2022

Question 1.1.1

Here the period of motion is $T = 2\pi$ in units of $\sqrt{l/g}$, so t_0 should be taken larger than T in order to observe the full physics of the pendulum.

[1]:
```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

#Direct input
plt.rc('text', usetex=True)
plt.rc('text.latex', preamble=r'\usepackage{amsmath} \usepackage{lmodern}␣
 ↪\usepackage{bm}')
#Options
params = {'text.usetex' : True,
          'font.size' : 18,
          'font.family' : 'lmodern',
          }
plt.rcParams.update(params)
mpl.rcParams['axes.linewidth'] = 1.

def syst_dyn(y,t):
    return np.array([y[1], - np.sin(y[0])])

def Euler(t0,h,yinit):
    t = [0.]
    y0 = [yinit[0]]
    y1 = [yinit[1]]
    tf = 0.
    y0f = yinit[0]
    y1f =yinit[1]
    while tf < t0:
        dy = h * syst_dyn(np.array([y0f, y1f]),tf)
        y0f += dy[0]
        y1f += dy[1]
        tf += h
        t += [tf]
        y0 += [y0f]
        y1 += [y1f]
    return np.array(t), np.vstack((np.array(y0), np.array(y1))).transpose()

def plot_sol_Euler(t0,h,yinit):
    t,y = Euler(t0,h,yinit)
    fig, ax = plt.subplots(1, 3, figsize = (18, 6), tight_layout = True)
    fig.suptitle(r'Euler $h=$' + str(h) + r', $\theta^{(0)}$=' + '{0:.2f}'.
 ↪format(yinit[0]
    ) + r', $\dot{\theta}^{(0)}$=' + str(yinit[1]))
    ax[0].plot(t, y[:,0], '-o', markersize = 0.5)
    ax[0].set_xlabel(r'$t$')
    ax[0].set_ylabel(r'$\theta(t)$')
    ax[1].plot(y[:,0], y[:,1], '-o', markersize = 0.5)
    ax[1].set_xlabel(r'$\theta$')
    ax[1].set_ylabel(r'$\dot{\theta}$')
    ax[2].plot(t, 0.5 * y[:,1] * y[:,1] - np.cos(y[:,0]), '-o', markersize = 0.
 ↪5)
    ax[2].set_xlabel(r'$t$')
    ax[2].set_ylabel(r'$E(t)$')
    fig.show()

plot_sol_Euler(10,0.001,np.array([np.pi / 2., 0.]))
```

```
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
```



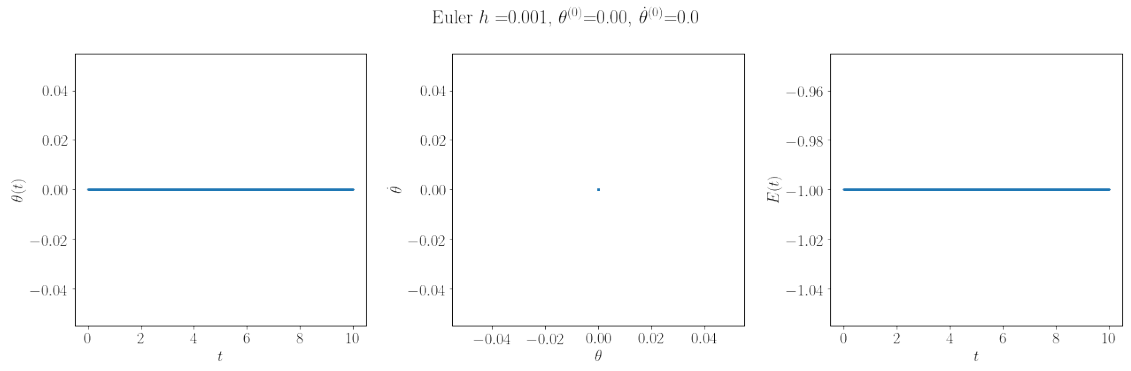
Question 1.1.2

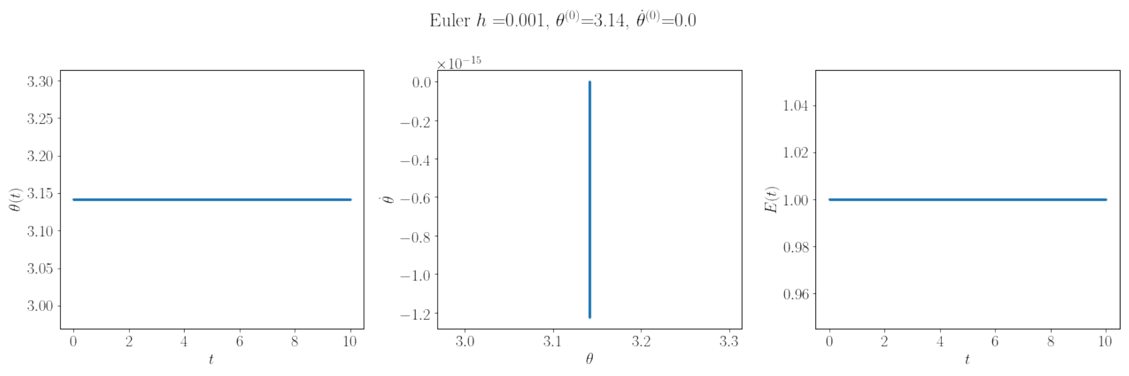
The equilibrium positions are found by setting $\mathbf{f}(\mathbf{y}, t) = \mathbf{0}$, resulting in $y_2 = 0$ and $\sin(y_1) = 0$ or $y_1 = 0, \pi$.

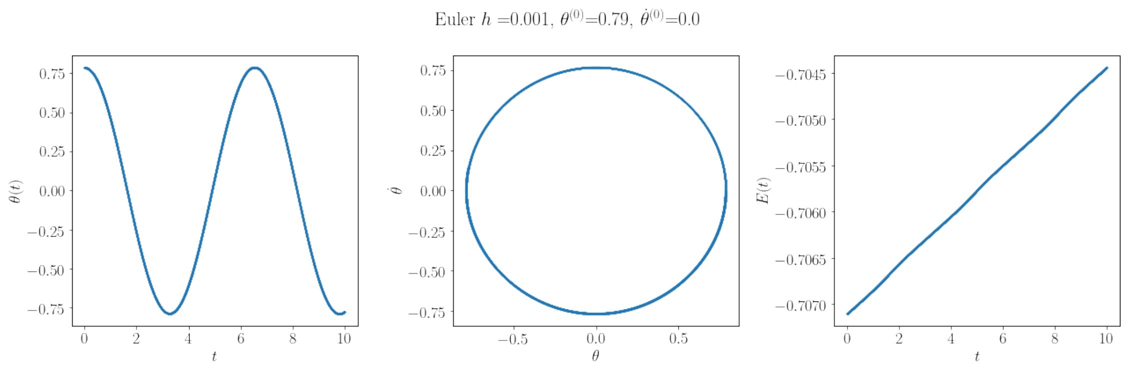
When $y_1^{(0)}$ increases, one observes that the trajectory in the phase portrait is no longer a circle, but is deformed. This is because we treat the pendulum beyond the small angle approximation.

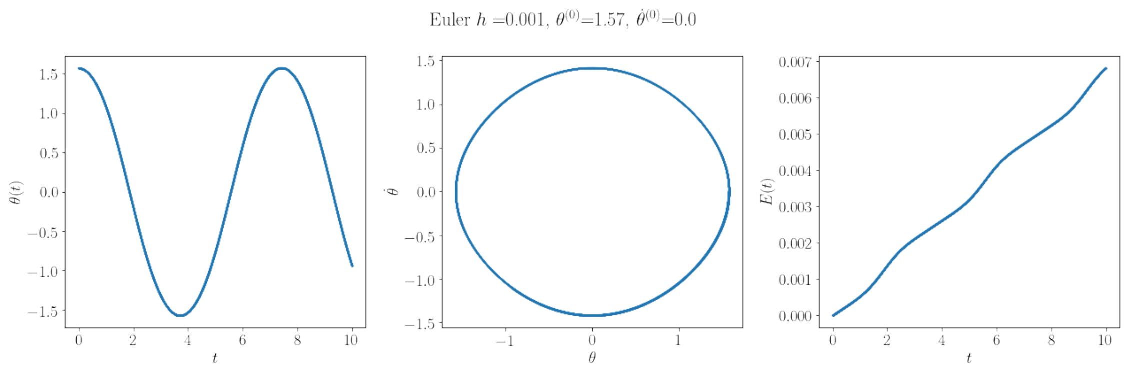
```
[2]: plot_sol_Euler(10, 0.001, np.array([0., 0.]))
     plot_sol_Euler(10, 0.001, np.array([np.pi, 0.]))
     plot_sol_Euler(10, 0.001, np.array([np.pi / 4., 0.]))
     plot_sol_Euler(10, 0.001, np.array([np.pi / 2., 0.]))
     plot_sol_Euler(10, 0.001, np.array([3. * np.pi / 4., 0.]))
```

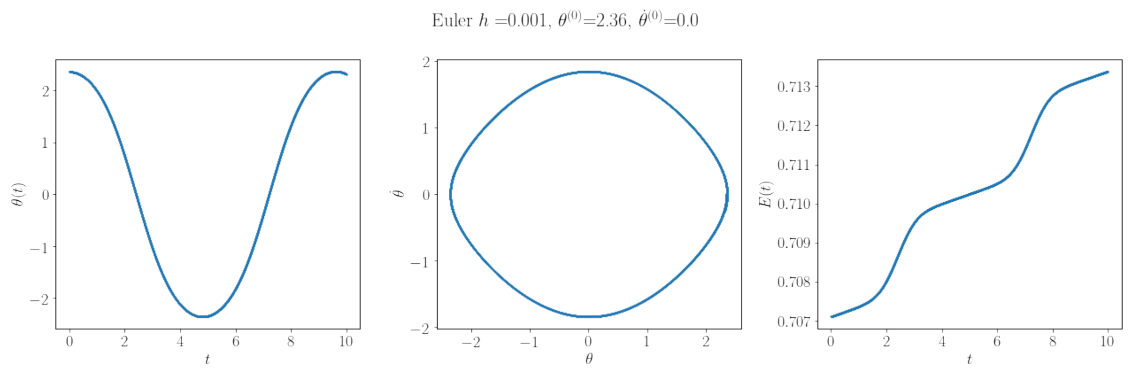
```
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
```




Euler h =0.001, $\theta^{(0)}$=3.14, $\dot{\theta}^{(0)}$=0.0
$\theta(t)$
t
$\times 10^{-15}$
$\dot{\theta}$
θ
$E(t)$
t

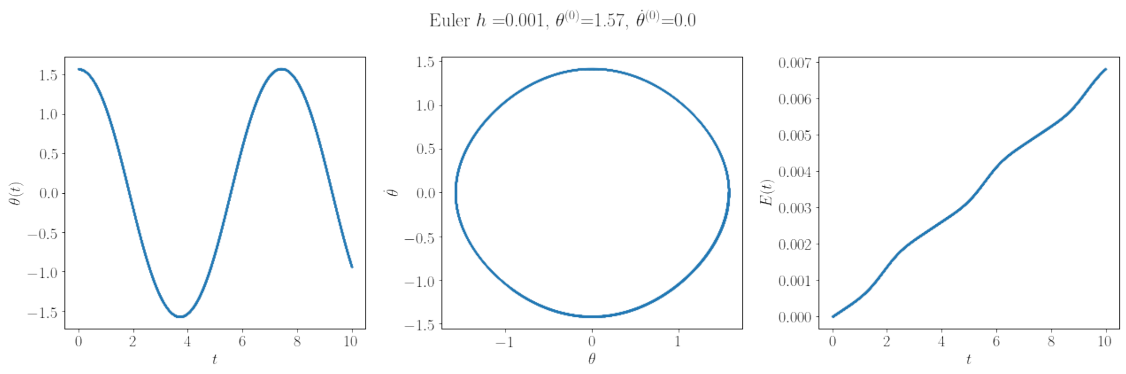

Euler h =0.001, $\theta^{(0)}$=0.79, $\dot{\theta}^{(0)}$=0.0
$\theta(t)$
t
$\dot{\theta}$
θ
$E(t)$
t

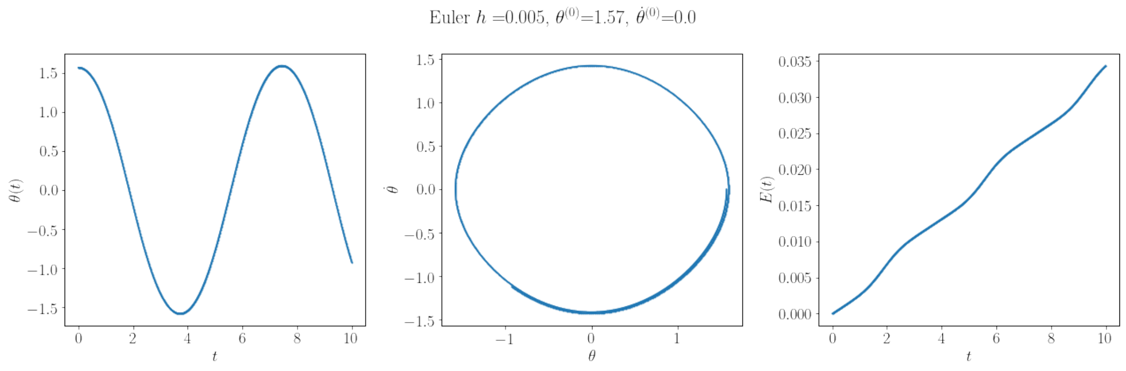

Euler h =0.001, $\theta^{(0)}$=1.57, $\dot{\theta}^{(0)}$=0.0
$\theta(t)$
t
$\dot{\theta}$
θ
$E(t)$
t

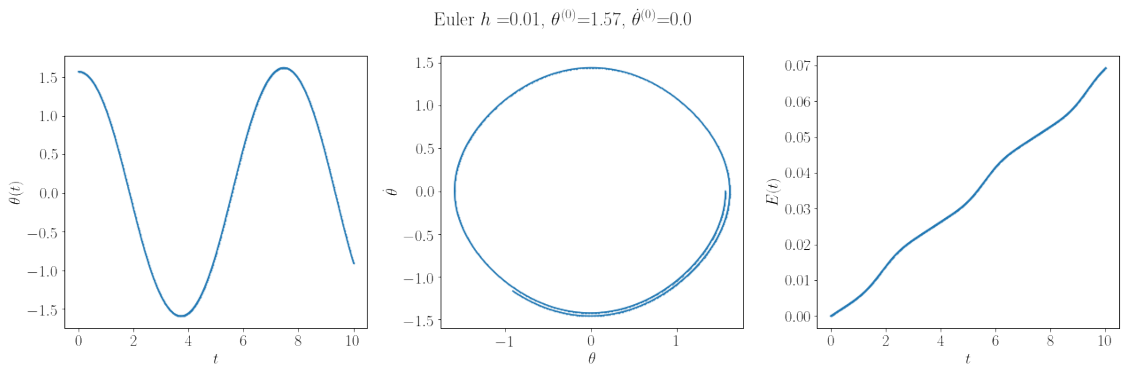


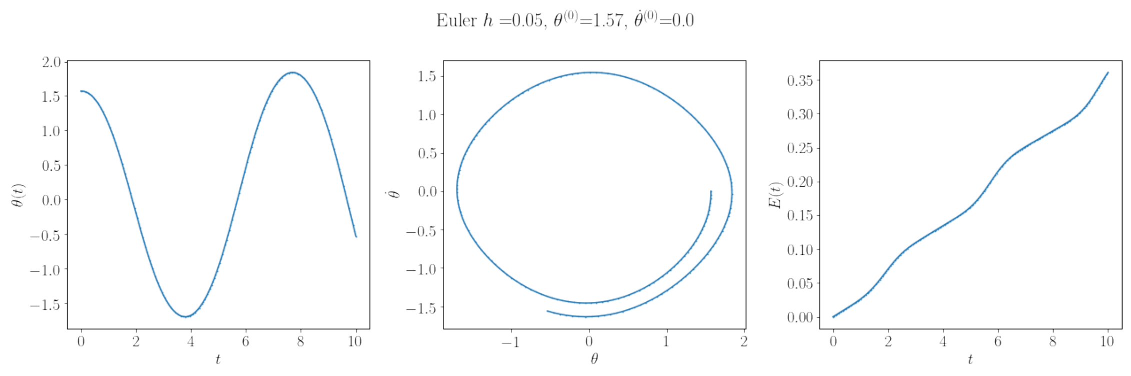
```
[3]: plot_sol_Euler(10, 0.001, np.array([np.pi / 2., 0.]))
     plot_sol_Euler(10, 0.005, np.array([np.pi / 2., 0.]))
     plot_sol_Euler(10, 0.01, np.array([np.pi / 2., 0.]))
     plot_sol_Euler(10, 0.05, np.array([np.pi / 2., 0.]))
     plot_sol_Euler(10, 0.1, np.array([np.pi / 2., 0.]))
```

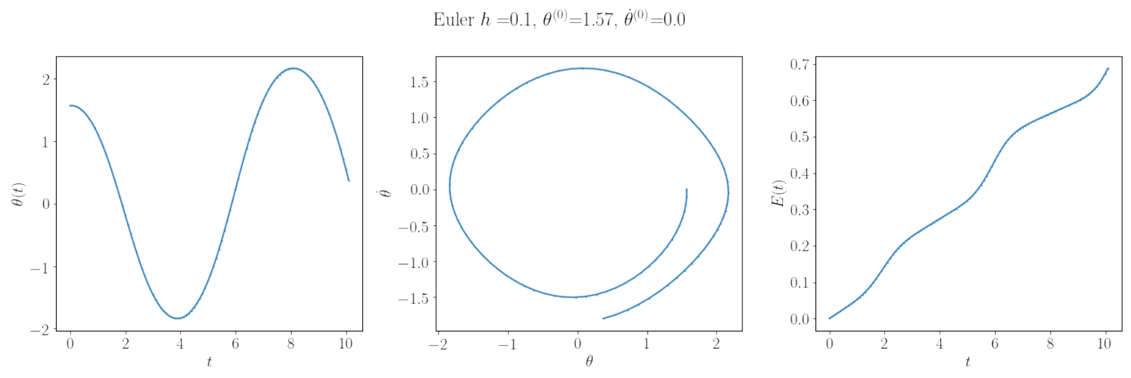
```
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
```


Euler h =0.001, $\theta^{(0)}$=1.57, $\dot{\theta}^{(0)}$=0.0
$\theta(t)$
t
$\dot{\theta}$
θ
$E(t)$
t


Euler h =0.005, $\theta^{(0)}$=1.57, $\dot{\theta}^{(0)}$=0.0
$\theta(t)$
t
$\dot{\theta}$
θ
$E(t)$
t


Euler h =0.01, $\theta^{(0)}$=1.57, $\dot{\theta}^{(0)}$=0.0
$\theta(t)$
t
$\dot{\theta}$
θ
$E(t)$
t





Question 1.1.3

There is a transition from a bounded state to an unbounded state when the initial velocity increases. The crossover happens when the system reaches $\theta = \pi$ with zero kinetic energy, namely, when

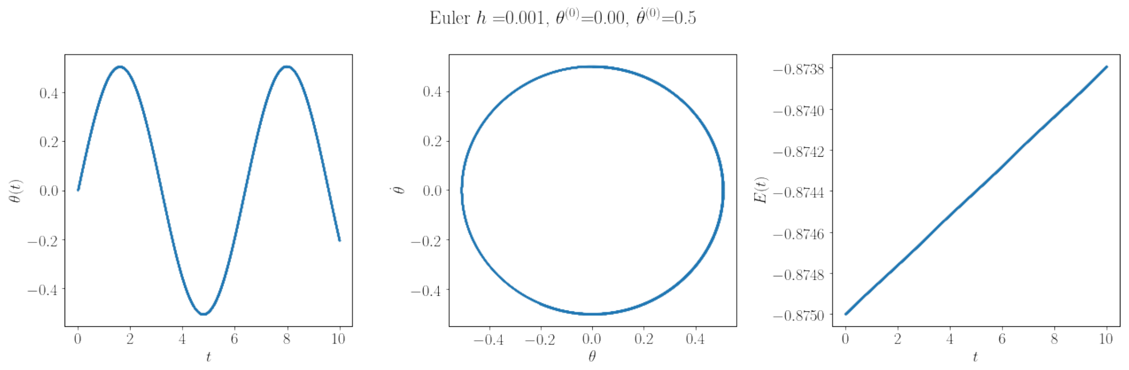
$$\frac{\left[\dot\theta^{(0)}\right]^2}{2} - 1 = 0 + 1,$$

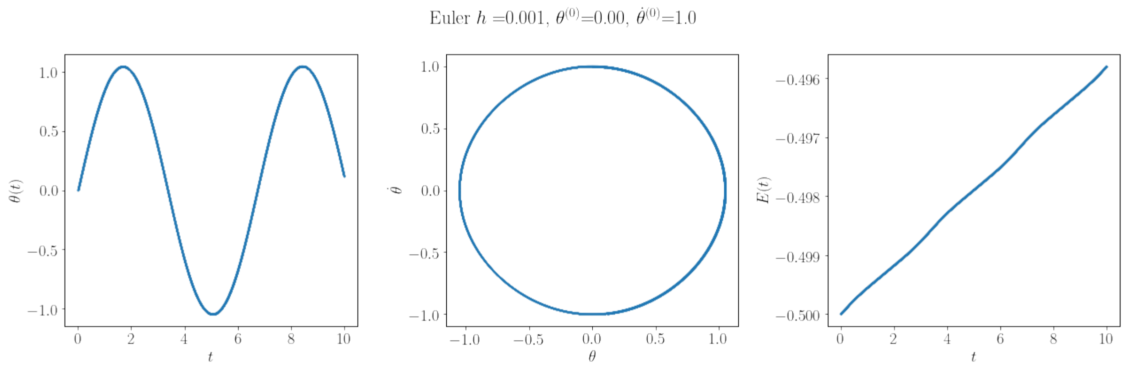
or $\dot\theta^{(0)} = \pm 2$.

```
[4]: plot_sol_Euler(10, 0.001, np.array([0., 0.5]))
plot_sol_Euler(10, 0.001, np.array([0., 1.]))
plot_sol_Euler(10, 0.001, np.array([0., 1.5]))
plot_sol_Euler(10, 0.001, np.array([0., 2.]))
plot_sol_Euler(10, 0.001, np.array([0., 2.5]))
plot_sol_Euler(10, 0.001, np.array([0., 3.]))
```

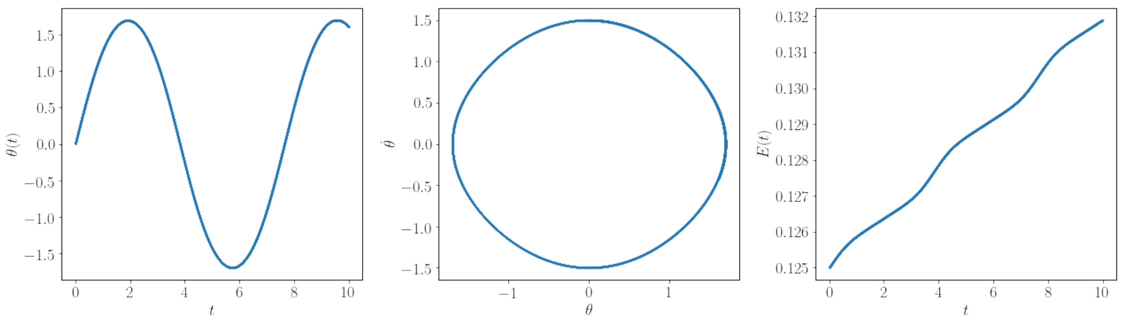
```
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
```

```
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
```

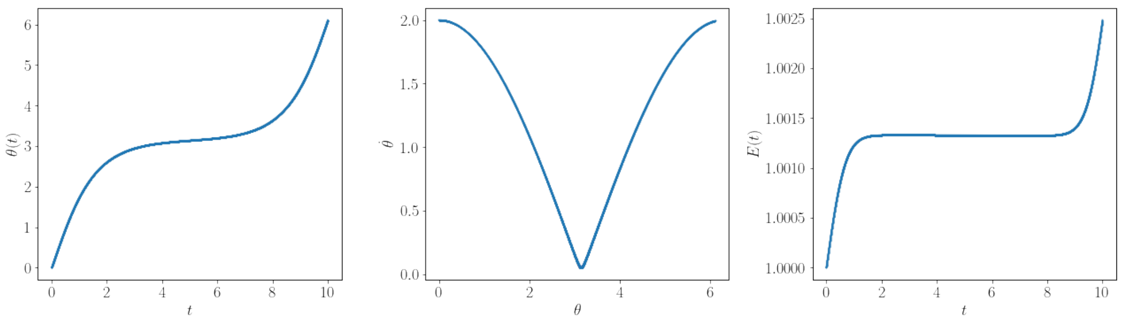




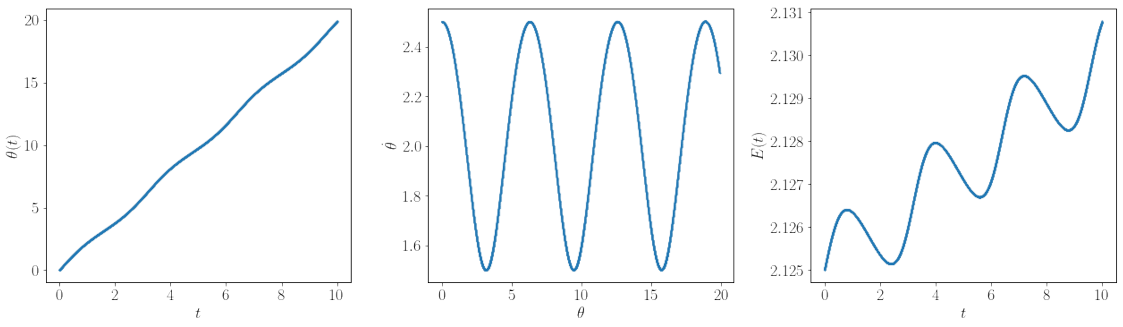
Euler h =0.001, $\theta^{(0)}$=0.00, $\dot{\theta}^{(0)}$=1.5

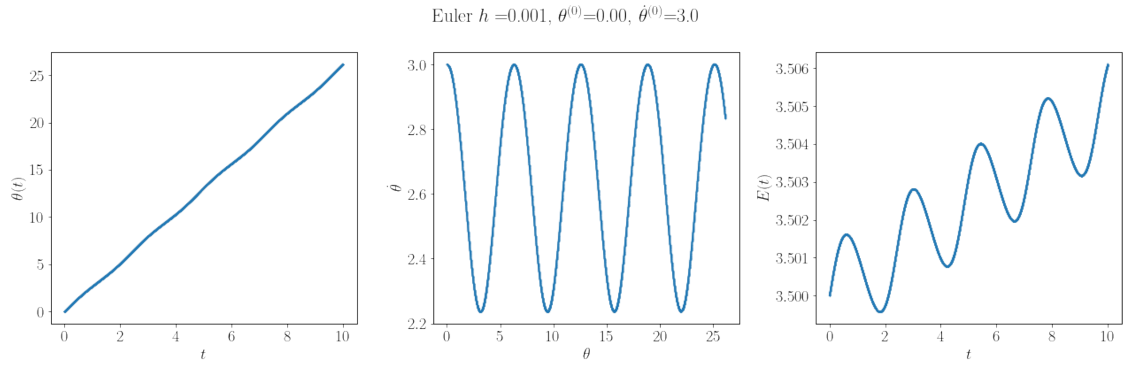


Euler h =0.001, $\theta^{(0)}$=0.00, $\dot{\theta}^{(0)}$=2.0



Euler h =0.001, $\theta^{(0)}$=0.00, $\dot{\theta}^{(0)}$=2.5





Question 1.1.4

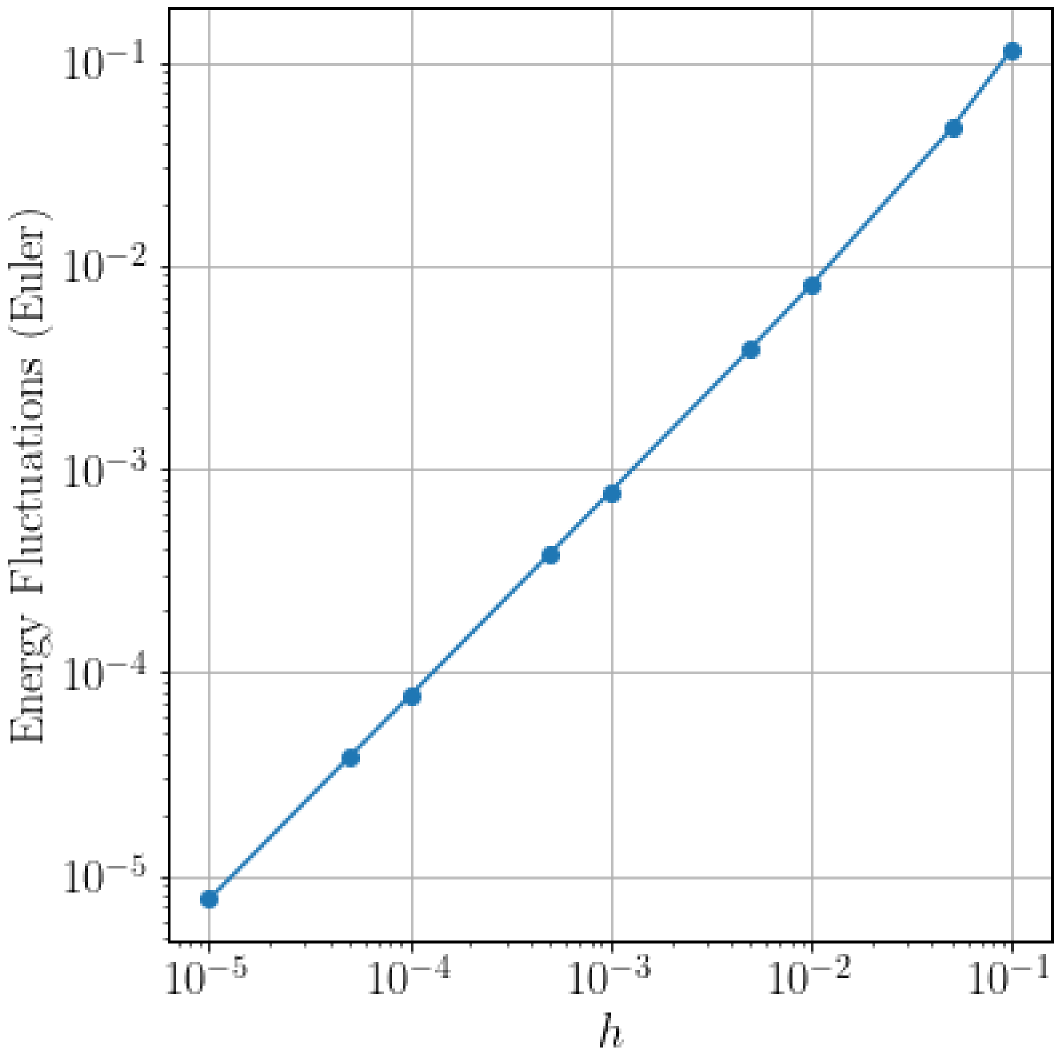
When h is too large (typically $h \gtrsim 10^{-3}$), the trajectory is not closed in the phase portrait, because of too large errors in the integration of the equations of motion. In particular, one can check that the energy is never conserved but grows more and more as h increases. This is a typical feature in the integration of equations of motion. Usually, changes in the total energy are tolerated when they are smaller than 10^{-4} in relative value. Their growth with h depends on the algorithm to integrate the equations of motion.

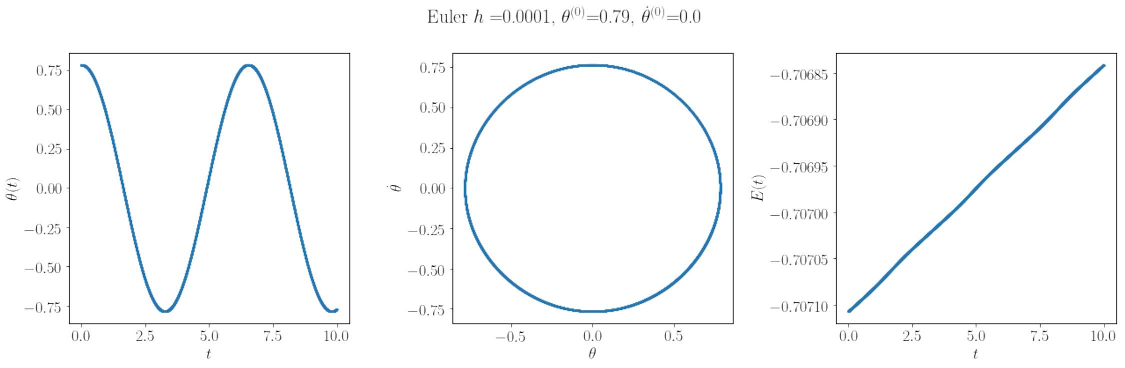
For the Euler method, energy fluctuations increase linearly with h, and h should be taken of the order of 10^{-4} to meet the above criterion (although it may vary from one initial condition to another).

```
[5]: H_arr = np.array([0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.
      ↪1])
     En_exc = []
     for h in H_arr:
         t, y = Euler(10, h, np.array([np.pi / 4., 0.]))
         En_exc += [np.std(0.5 * y[:,1] * y[:,1] - np.cos(y[:,0]))]
     En_exc = np.array(En_exc)
     fig, ax = plt.subplots(figsize = (6, 6), tight_layout = True)
     ax.loglog(H_arr, En_exc, '-o')
     ax.set_xlabel(r'$h$')
     ax.set_ylabel(r'Energy Fluctuations (Euler)')
     ax.grid()
     fig.show()

     plot_sol_Euler(10, H_arr[np.argmin(np.absolute(En_exc - 1e-4))], np.array([np.
      ↪pi / 4., 0.]))
```

```
/tmp/ipykernel_3144492/148167345.py:12: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/991533669.py:50: UserWarning: Matplotlib is currently
```

```
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
```





Question 1.2

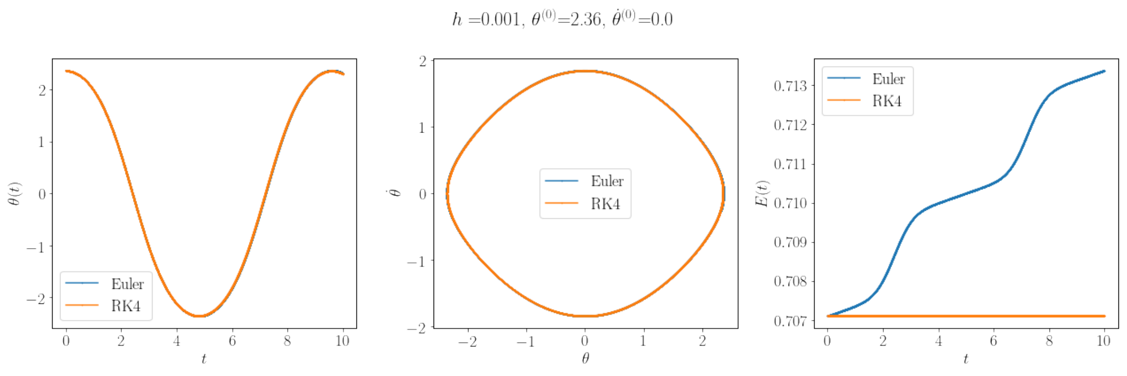
Energy conservation is far improved thanks to the RK4 method, and the trajectory is always closed after one period, contrary to the Euler method when h is increased too significantly.

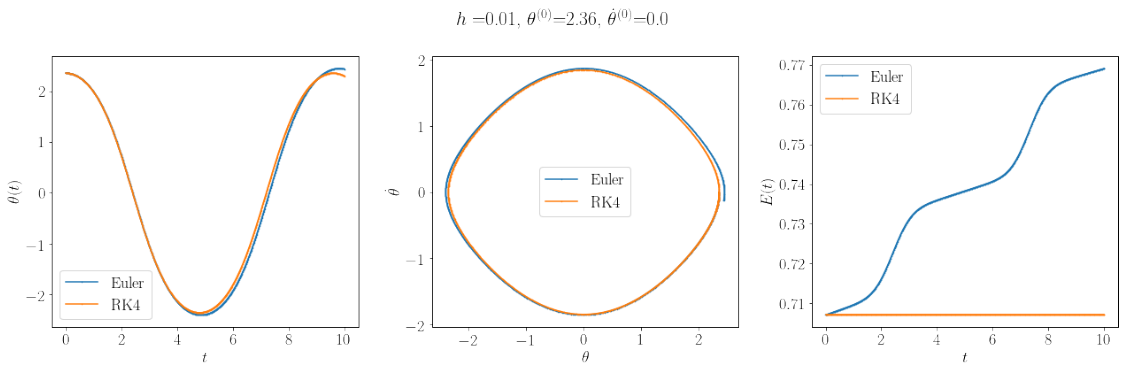
The RK4 method is thus more stable, but contains more operations at each timestep: there is a compromise between the rapidity and the accuracy of calculation, which is a generic feature in Computational Physics.

[6]:
```
def RK4(t0,h,yinit):
    t = [0.]
    y0 = [yinit[0]]
    y1 = [yinit[1]]
    tf = 0.
    y0f = yinit[0]
    y1f = yinit[1]
    while tf < t0:
        yf = np.array([y0f, y1f])
        k1 = h * syst_dyn(yf,tf)
        k2 = h * syst_dyn(yf + 0.5 * k1,tf + 0.5 * h)
        k3 = h * syst_dyn(yf + 0.5 * k2,tf + 0.5 * h)
        k4 = h * syst_dyn(yf + k3, tf + h)
        y0f += ( k1[0] + k2[0] + k2[0] + k3[0] + k3[0] + k4[0] ) / 6.
        y1f += ( k1[1] + k2[1] + k2[1] + k3[1] + k3[1] + k4[1] ) / 6.
        tf += h
        t += [tf]
        y0 += [y0f]
        y1 += [y1f]
    return np.array(t), np.vstack((np.array(y0), np.array(y1))).transpose()

def plot_sol_RK4(t0,h,yinit):
    t, y = RK4(t0,h,yinit)
    fig, ax = plt.subplots(1, 3, figsize = (18, 6), tight_layout = True)
    fig.suptitle(r'RK4 $h=$' + str(h) + r', $\theta^{(0)}$=' + '{0:.2f}'.
 ↪format(yinit[0]
    ) + r', $\dot{\theta}^{(0)}$=' + str(yinit[1]))
    ax[0].plot(t, y[:,0], '-o', markersize = 0.5)
    ax[0].set_xlabel(r'$t$')
    ax[0].set_ylabel(r'$\theta(t)$')
    ax[1].plot(y[:,0], y[:,1], '-o', markersize = 0.5)
    ax[1].set_xlabel(r'$\theta$')
    ax[1].set_ylabel(r'$\dot{\theta}$')
    ax[2].plot(t, 0.5 * y[:,1] * y[:,1] - np.cos(y[:,0]), '-o', markersize = 0.
 ↪5)
    ax[2].set_xlabel(r'$t$')
    ax[2].set_ylabel(r'$E(t)$')
    fig.show()

def plot_sol_RK4_Euler(t0,h,yinit):
    t, y = RK4(t0,h,yinit)
    tE, yE = Euler(t0,h,yinit)
    fig, ax = plt.subplots(1, 3, figsize = (18, 6), tight_layout = True)
    fig.suptitle(r'$h=$' + str(h) + r', $\theta^{(0)}$=' + '{0:.2f}'.
 ↪format(yinit[0]
    ) + r', $\dot{\theta}^{(0)}$=' + str(yinit[1]))
    ax[0].plot(tE, yE[:,0], '-s', markersize = 0.5, label = 'Euler')
    ax[0].plot(t, y[:,0], '-o', markersize = 0.5, label = 'RK4')
    ax[0].set_xlabel(r'$t$')
    ax[0].set_ylabel(r'$\theta(t)$')
    ax[0].legend()
    ax[1].plot(yE[:,0], yE[:,1], '-s', markersize = 0.5, label = 'Euler')
    ax[1].plot(y[:,0], y[:,1], '-o', markersize = 0.5, label = 'RK4')
    ax[1].set_xlabel(r'$\theta$')
    ax[1].set_ylabel(r'$\dot{\theta}$')
    ax[1].legend()
    ax[2].plot(tE, 0.5 * yE[:,1] * yE[:,1] - np.cos(yE[:,0]), '-s', markersize␣
 ↪= 0.5, label = 'Euler')
    ax[2].plot(t, 0.5 * y[:,1] * y[:,1] - np.cos(y[:,0]), '-o', markersize = 0.
 ↪5, label = 'RK4')
    ax[2].set_xlabel(r'$t$')
    ax[2].set_ylabel(r'$E(t)$')
    ax[2].legend()
    fig.show()

plot_sol_RK4_Euler(10, 0.001, np.array([3. * np.pi / 4., 0.]))
plot_sol_RK4_Euler(10, 0.01, np.array([3. * np.pi / 4., 0.]))
plot_sol_RK4_Euler(10, 0.1, np.array([3. * np.pi / 4., 0.]))

plot_sol_RK4_Euler(10, 0.001, np.array([0., 2.5]))
plot_sol_RK4_Euler(10, 0.01, np.array([0., 2.5]))
plot_sol_RK4_Euler(10, 0.1, np.array([0., 2.5]))
```

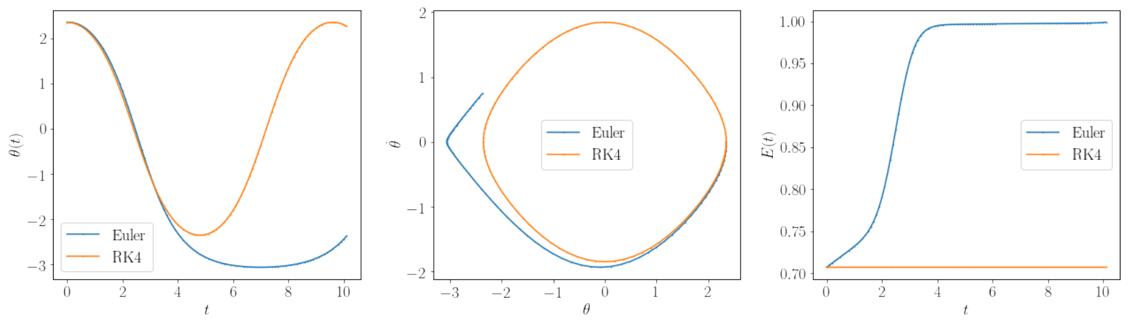
```
/tmp/ipykernel_3144492/1828600938.py:59: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/1828600938.py:59: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/1828600938.py:59: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
```

```
  fig.show()
/tmp/ipykernel_3144492/1828600938.py:59: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/1828600938.py:59: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/1828600938.py:59: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
```

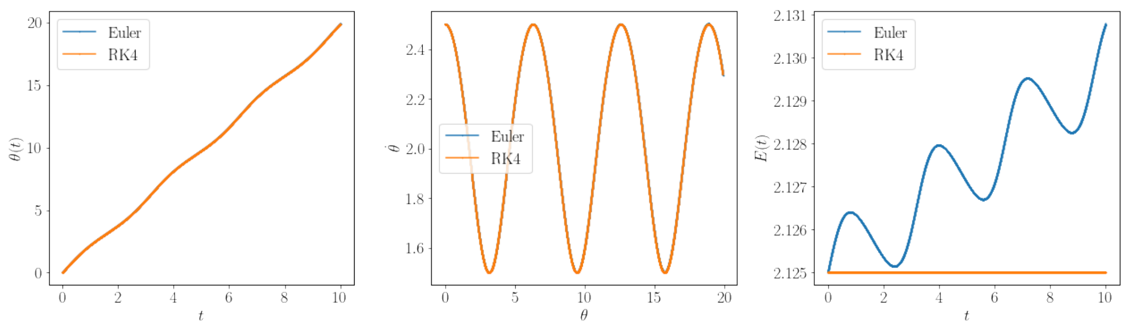




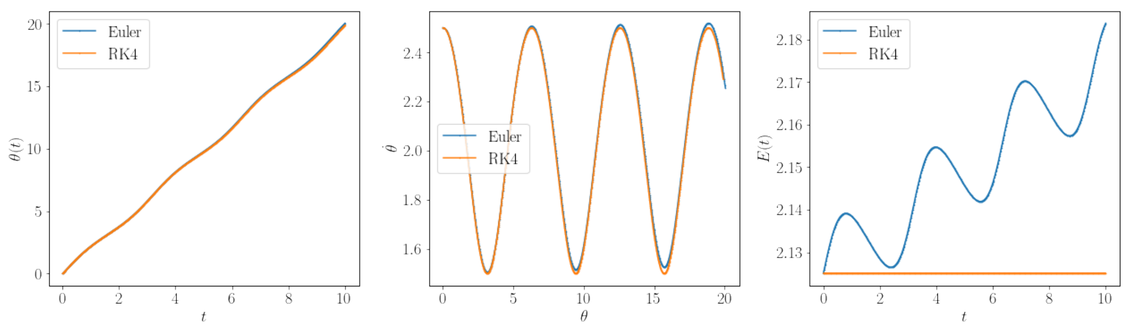
$h = 0.1$, $\theta^{(0)} = 2.36$, $\dot{\theta}^{(0)} = 0.0$

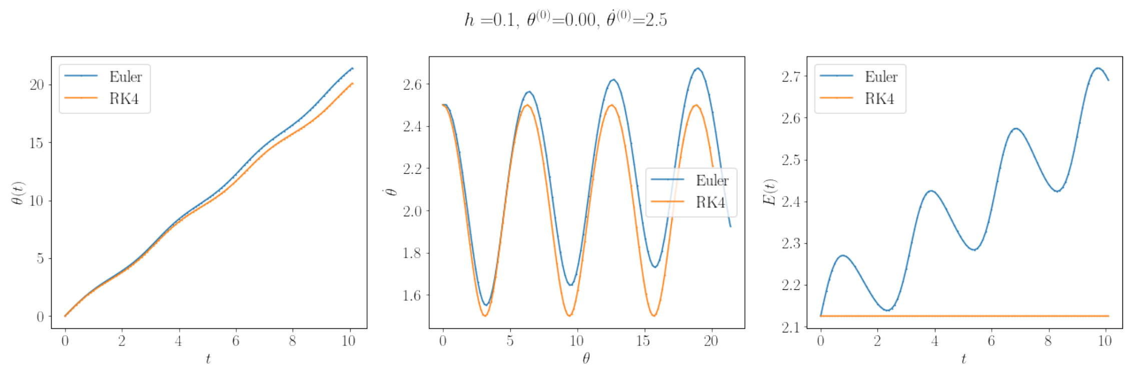


$h = 0.001$, $\theta^{(0)} = 0.00$, $\dot{\theta}^{(0)} = 2.5$



$h = 0.01$, $\theta^{(0)} = 0.00$, $\dot{\theta}^{(0)} = 2.5$



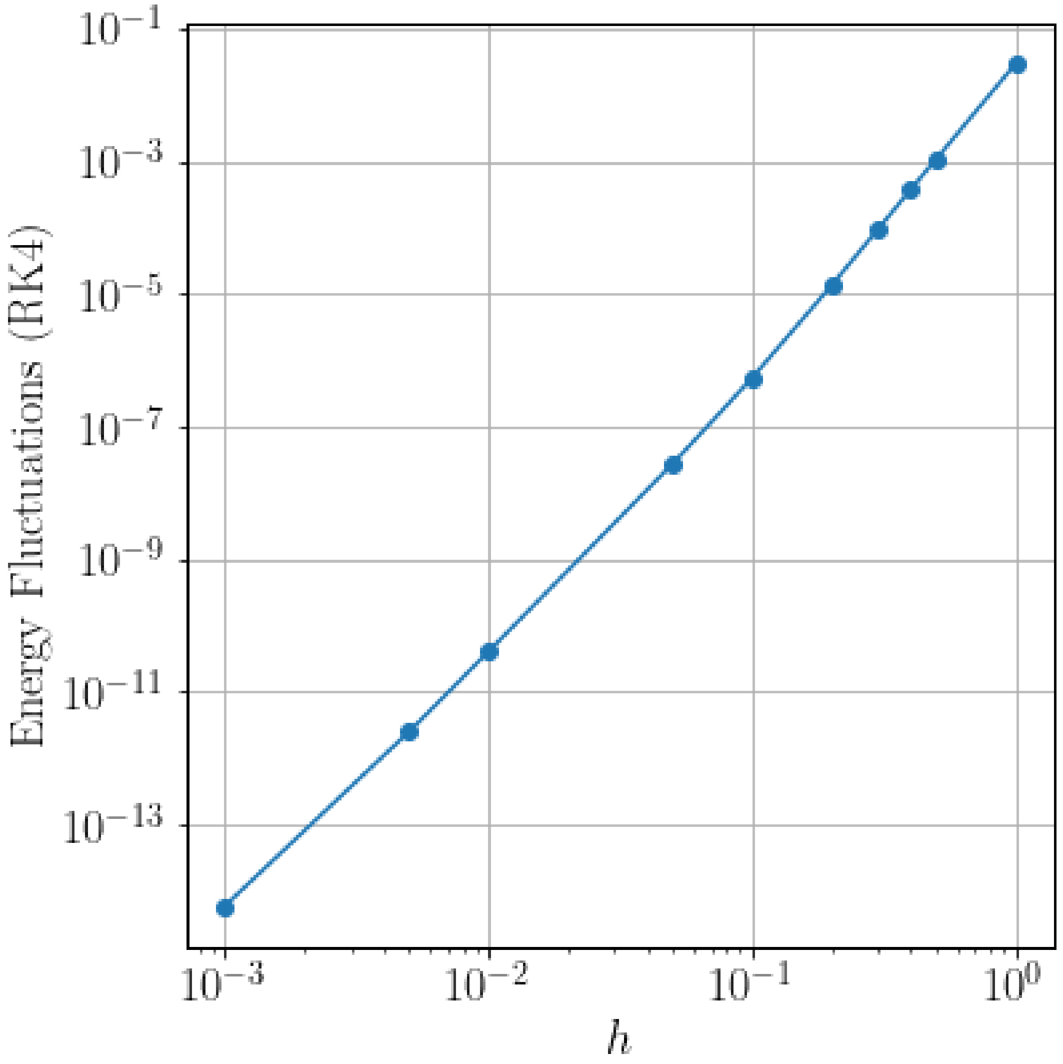


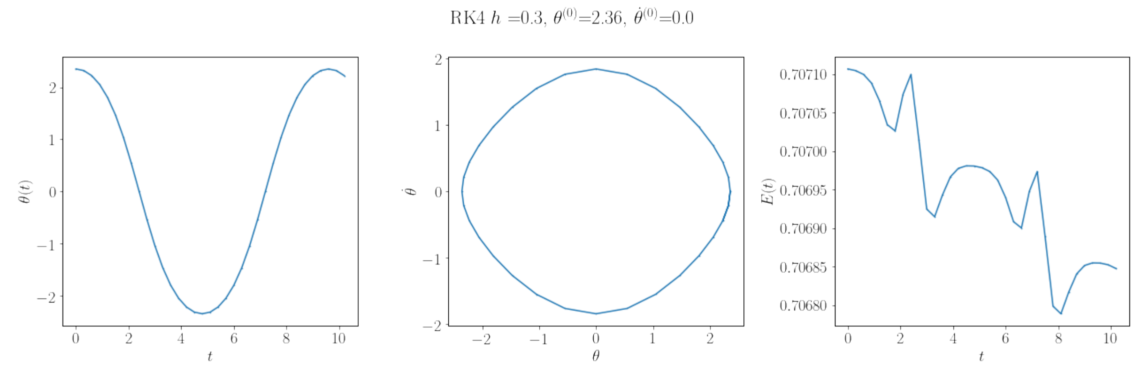
We can also study the energy fluctuations in the case of the RK4 method, to determine the maximum allowed value of h. In this case, we find that $h \simeq 0.3$ is reasonable (although still depending on the initial condition), because energy fluctuations now grow as h^4 (instead of h for the Euler method). However, for such a large value of h, the number of points on the trajectory is low, resulting in rough curves. To obtain smoother trajectories, one can take $h = 0.01$ for which energy fluctuations are of the order of $10^{-10} - 10^{-12}$ (depending again on the initial condition).

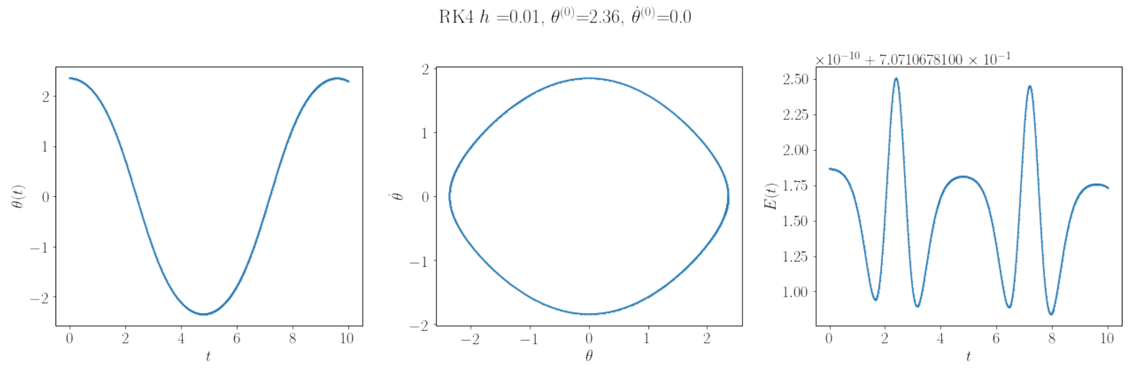
[7]:
```
H_arr = np.array([0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 1.])
En_exc = []
for h in H_arr:
    t, y = RK4(10, h, np.array([3. * np.pi / 4., 0.]))
    En_exc += [np.std(0.5 * y[:,1] * y[:,1] - np.cos(y[:,0]))]
En_exc = np.array(En_exc)
fig, ax = plt.subplots(figsize = (6, 6), tight_layout = True)
ax.loglog(H_arr, En_exc, '-o')
ax.set_xlabel(r'$h$')
ax.set_ylabel(r'Energy Fluctuations (RK4)')
ax.grid()
fig.show()

plot_sol_RK4(10, H_arr[np.argmin(np.absolute(En_exc - 1e-4))], np.array([3. *␣
 ↪np.pi / 4., 0.]))
plot_sol_RK4(10, 0.01, np.array([3. * np.pi / 4., 0.]))
```

```
/tmp/ipykernel_3144492/1213378129.py:12: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/1828600938.py:36: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/1828600938.py:36: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
```

```
cannot show the figure.
  fig.show()
```







Question 1.3

A little manipulation of the two proposed equations shows that the system can be written as

$$\begin{cases} y_{1,n+1} = y_{1,n} + hy_{2,n} - \dfrac{h^2}{2}\sin\left(\dfrac{y_{1,n}+y_{1,n+1}}{2}\right), \\ y_{2,n+1} = y_{2,n} - h\sin\left(\dfrac{y_{1,n}+y_{1,n+1}}{2}\right). \end{cases}$$

The first equation is a closed equation for $y_{1,n+1}$ which amounts to find the root of the function

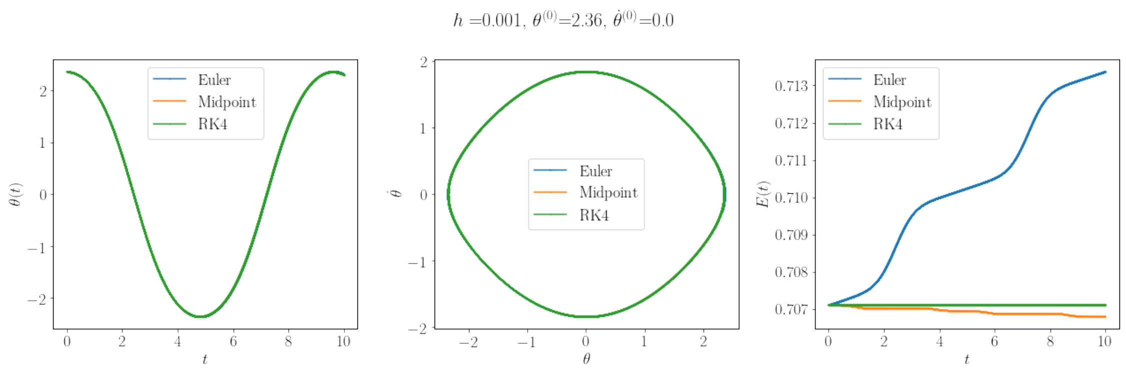
$$g(y) = y - y_{1,n} - hy_{2,n} + \frac{h^2}{2}\sin\left(\frac{y_{1,n}+y}{2}\right).$$

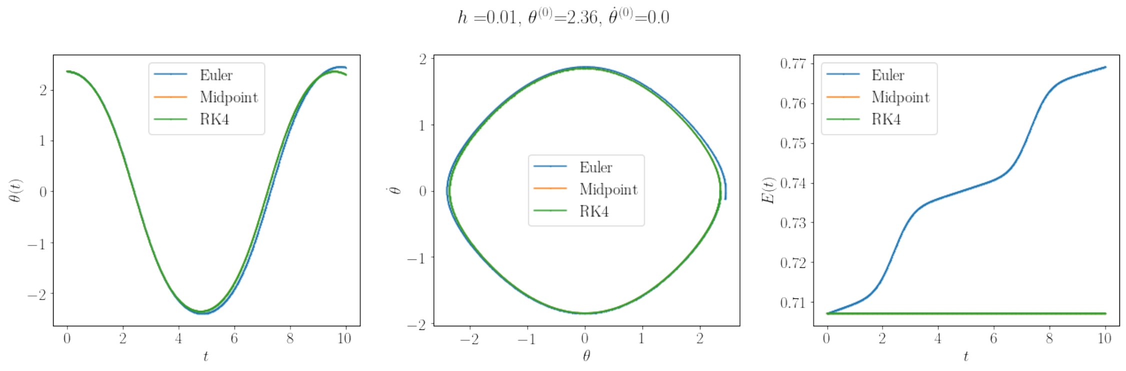
The midpoint method is accurate, like the RK4 method.

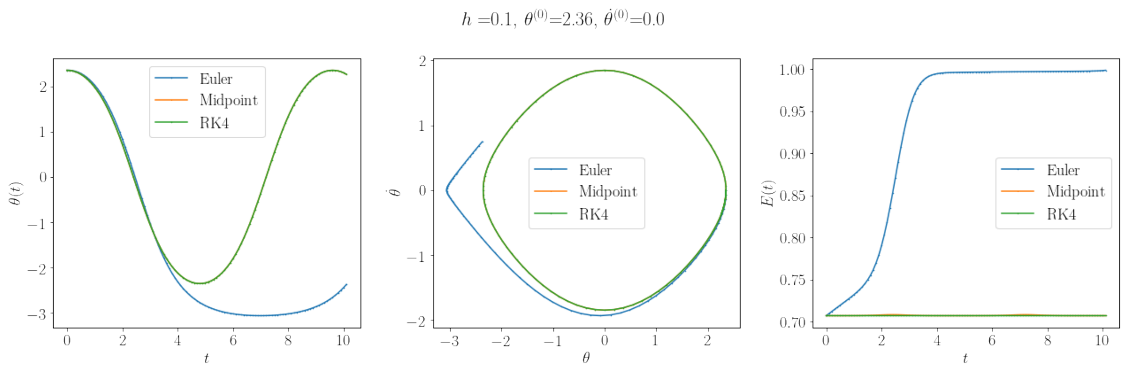
[8]:
```
def midpoint(t0,h,eta,yinit):
    t = [0.]
    y0 = [yinit[0]]
    y1 = [yinit[1]]
    tf = 0.
    y0f = yinit[0]
    y1f = yinit[1]
    while tf < t0:
        y0_new_before = y0f
        y1_new_before = y1f
        y0_new = y0f + h * y1f - 0.5 * h * h * np.sin(0.5 * ( y0f +␣
↪y0_new_before ))
        y1_new = y1f - h * np.sin(0.5 * ( y0f + y0_new_before ))
        while max(np.absolute(y1_new - y1_new_before), np.absolute(y0_new -␣
↪y0_new_before)) >= eta:
            y0_new_before = y0_new
            y1_new_before = y1_new
            y0_new = y0f + h * y1f - 0.5 * h * h * np.sin(0.5 * ( y0f +␣
↪y0_new_before ))
```

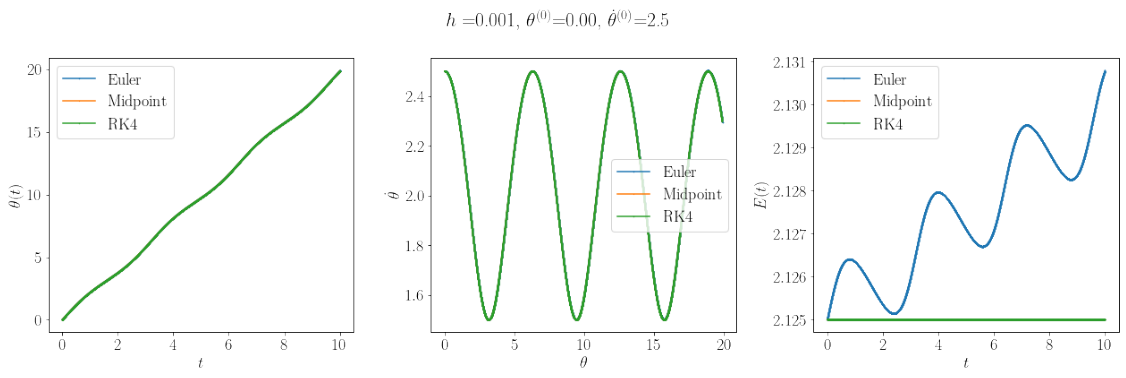
```
            y1_new = y1f - h * np.sin(0.5 * ( y0f + y0_new_before ))
        y0f = y0_new
        y1f = y1_new
        tf += h
        t += [tf]
        y0 += [y0f]
        y1 += [y1f]
    return np.array(t), np.vstack((np.array(y0), np.array(y1))).transpose()

def plot_sol_midpoint(t0,h,eta,yinit):
    t, y = midpoint(t0,h,eta,yinit)
    fig, ax = plt.subplots(1, 3, figsize = (18, 6), tight_layout = True)
    fig.suptitle(r'Midpoint $h=$' + str(h) + r', $\theta^{(0)}$=' + '{0:.2f}'.
 ↪format(yinit[0]
    ) + r', $\dot{\theta}^{(0)}$=' + str(yinit[1]))
    ax[0].plot(t, y[:,0], '-o', markersize = 0.5)
    ax[0].set_xlabel(r'$t$')
    ax[0].set_ylabel(r'$\theta(t)$')
    ax[1].plot(y[:,0], y[:,1], '-o', markersize = 0.5)
    ax[1].set_xlabel(r'$\theta$')
    ax[1].set_ylabel(r'$\dot{\theta}$')
    ax[2].plot(t, 0.5 * y[:,1] * y[:,1] - np.cos(y[:,0]), '-o', markersize = 0.
 ↪5)
    ax[2].set_xlabel(r'$t$')
    ax[2].set_ylabel(r'$E(t)$')
    fig.show()

def plot_sol_RK4_midpoint_Euler(t0,h,eta,yinit):
    t, y = RK4(t0,h,yinit)
    tE, yE = Euler(t0,h,yinit)
    tM, yM = midpoint(t0,h,eta,yinit)
    fig, ax = plt.subplots(1, 3, figsize = (18, 6), tight_layout = True)
    fig.suptitle(r'$h=$' + str(h) + r', $\theta^{(0)}$=' + '{0:.2f}'.
 ↪format(yinit[0]
    ) + r', $\dot{\theta}^{(0)}$=' + str(yinit[1]))
    ax[0].plot(tE, yE[:,0], '-s', markersize = 0.5, label = 'Euler')
    ax[0].plot(tM, yM[:,0], '-*', markersize = 0.5, label = 'Midpoint')
    ax[0].plot(t, y[:,0], '-o', markersize = 0.5, label = 'RK4')
    ax[0].set_xlabel(r'$t$')
    ax[0].set_ylabel(r'$\theta(t)$')
    ax[0].legend()
    ax[1].plot(yE[:,0], yE[:,1], '-s', markersize = 0.5, label = 'Euler')
    ax[1].plot(yM[:,0], yM[:,1], '-*', markersize = 0.5, label = 'Midpoint')
    ax[1].plot(y[:,0], y[:,1], '-o', markersize = 0.5, label = 'RK4')
    ax[1].set_xlabel(r'$\theta$')
    ax[1].set_ylabel(r'$\dot{\theta}$')
    ax[1].legend()
    ax[2].plot(tE, 0.5 * yE[:,1] * yE[:,1] - np.cos(yE[:,0]), '-s', markersize␣
↪= 0.5, label = 'Euler')
    ax[2].plot(tM, 0.5 * yM[:,1] * yM[:,1] - np.cos(yM[:,0]), '-*', markersize␣
↪= 0.5, label = 'Midpoint')
    ax[2].plot(t, 0.5 * y[:,1] * y[:,1] - np.cos(y[:,0]), '-o', markersize = 0.
↪5, label = 'RK4')
    ax[2].set_xlabel(r'$t$')
    ax[2].set_ylabel(r'$E(t)$')
    ax[2].legend()
    fig.show()

plot_sol_RK4_midpoint_Euler(10, 0.001, 0.001, np.array([3. * np.pi / 4., 0.]))
plot_sol_RK4_midpoint_Euler(10, 0.01, 0.001, np.array([3. * np.pi / 4., 0.]))
plot_sol_RK4_midpoint_Euler(10, 0.1, 0.001, np.array([3. * np.pi / 4., 0.]))

plot_sol_RK4_midpoint_Euler(10, 0.001, 0.001, np.array([0., 2.5]))
plot_sol_RK4_midpoint_Euler(10, 0.01, 0.001, np.array([0., 2.5]))
plot_sol_RK4_midpoint_Euler(10, 0.1, 0.001, np.array([0., 2.5]))
```

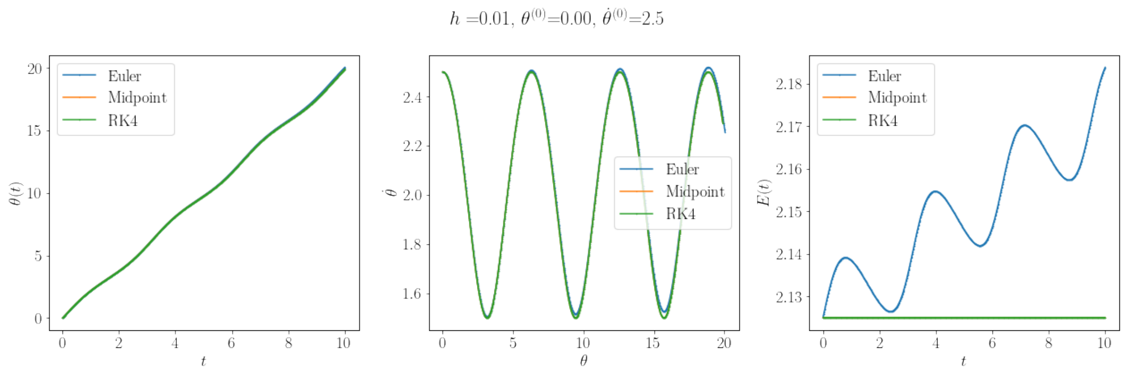
```
/tmp/ipykernel_3144492/2128641198.py:67: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/2128641198.py:67: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/2128641198.py:67: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/2128641198.py:67: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/2128641198.py:67: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/2128641198.py:67: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
```

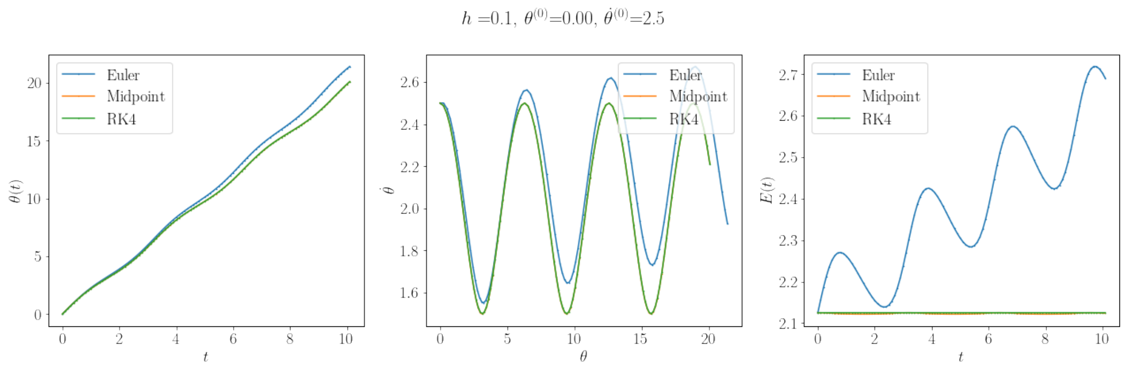

h =0.001, $\theta^{(0)}$=2.36, $\dot{\theta}^{(0)}$=0.0
Euler
Midpoint
RK4
$\theta(t)$
t
$\dot{\theta}$
θ
$E(t)$


h =0.01, $\theta^{(0)}$=2.36, $\dot{\theta}^{(0)}$=0.0
Euler
Midpoint
RK4
$\theta(t)$
t
$\dot{\theta}$
θ
$E(t)$


h =0.1, $\theta^{(0)}$=2.36, $\dot{\theta}^{(0)}$=0.0
Euler
Midpoint
RK4
$\theta(t)$
t
$\dot{\theta}$
θ
$E(t)$





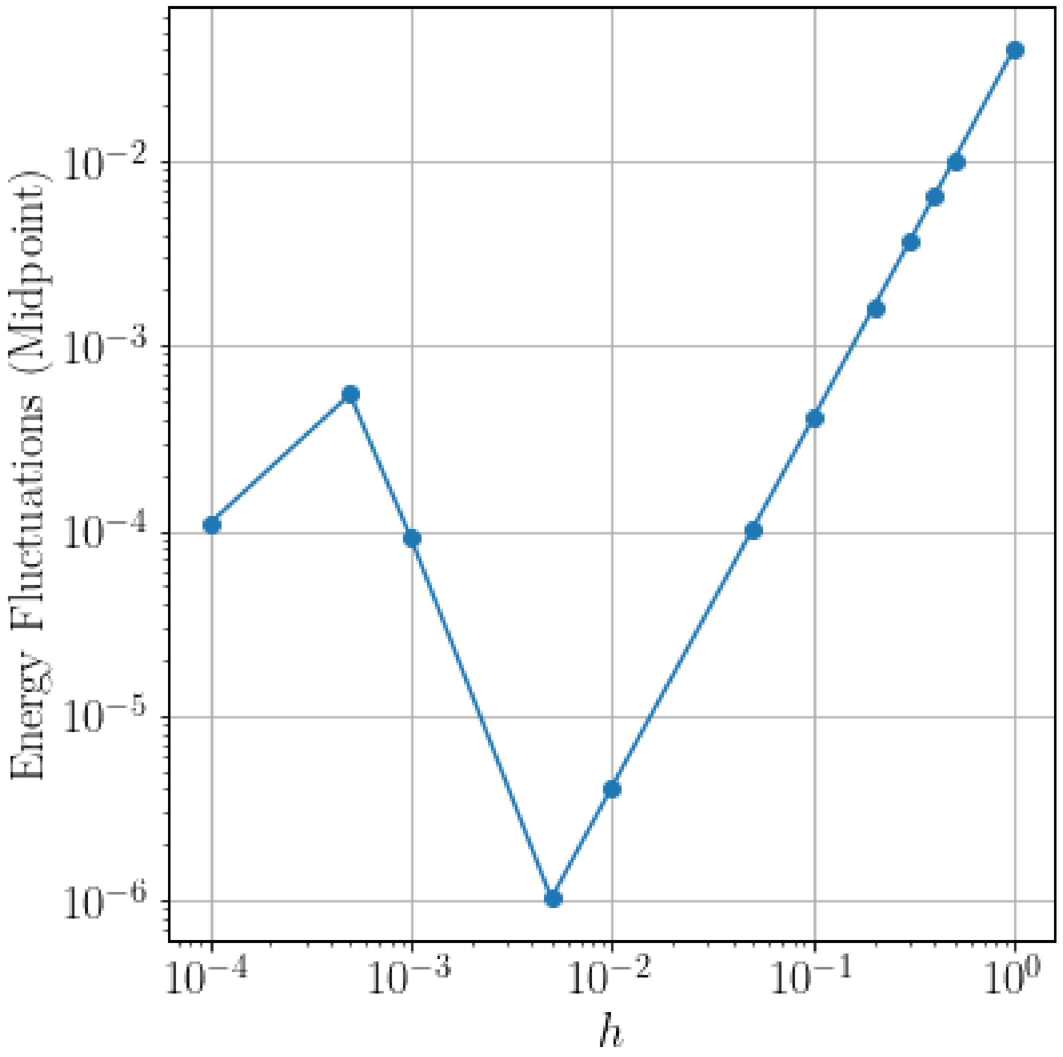


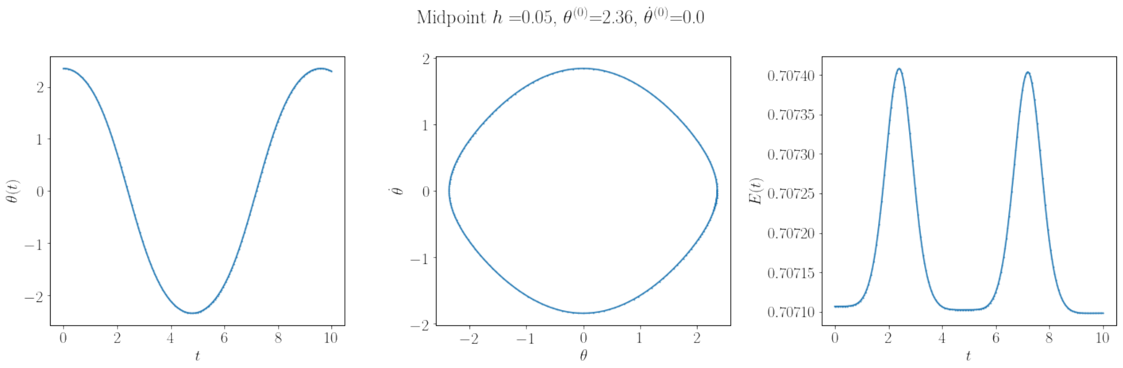
To quantify the performance of the different algorithms, we measure the energy fluctuations for the midpoint method. We find that they grow as h^2, and as such this method is better than the Euler method but less efficient than the RK4 method.

```
[9]: H_arr = np.array([0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4,␣
      ↪0.5, 1.])
     En_exc = []
```

```python
for h in H_arr:
    t, y = midpoint(10, h, 0.001, np.array([3. * np.pi / 4., 0.]))
    En_exc += [np.std(0.5 * y[:,1] * y[:,1] - np.cos(y[:,0]))]
En_exc = np.array(En_exc)
fig, ax = plt.subplots(figsize = (6, 6), tight_layout = True)
ax.loglog(H_arr, En_exc, '-o')
ax.set_xlabel(r'$h$')
ax.set_ylabel(r'Energy Fluctuations (Midpoint)')
ax.grid()
fig.show()

plot_sol_midpoint(10, H_arr[np.argmin(np.absolute(En_exc - 1e-4))], 0.001, np.
 ↪array([3. * np.pi / 4., 0.]))
```

```
/tmp/ipykernel_3144492/3132936213.py:12: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/2128641198.py:40: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
```

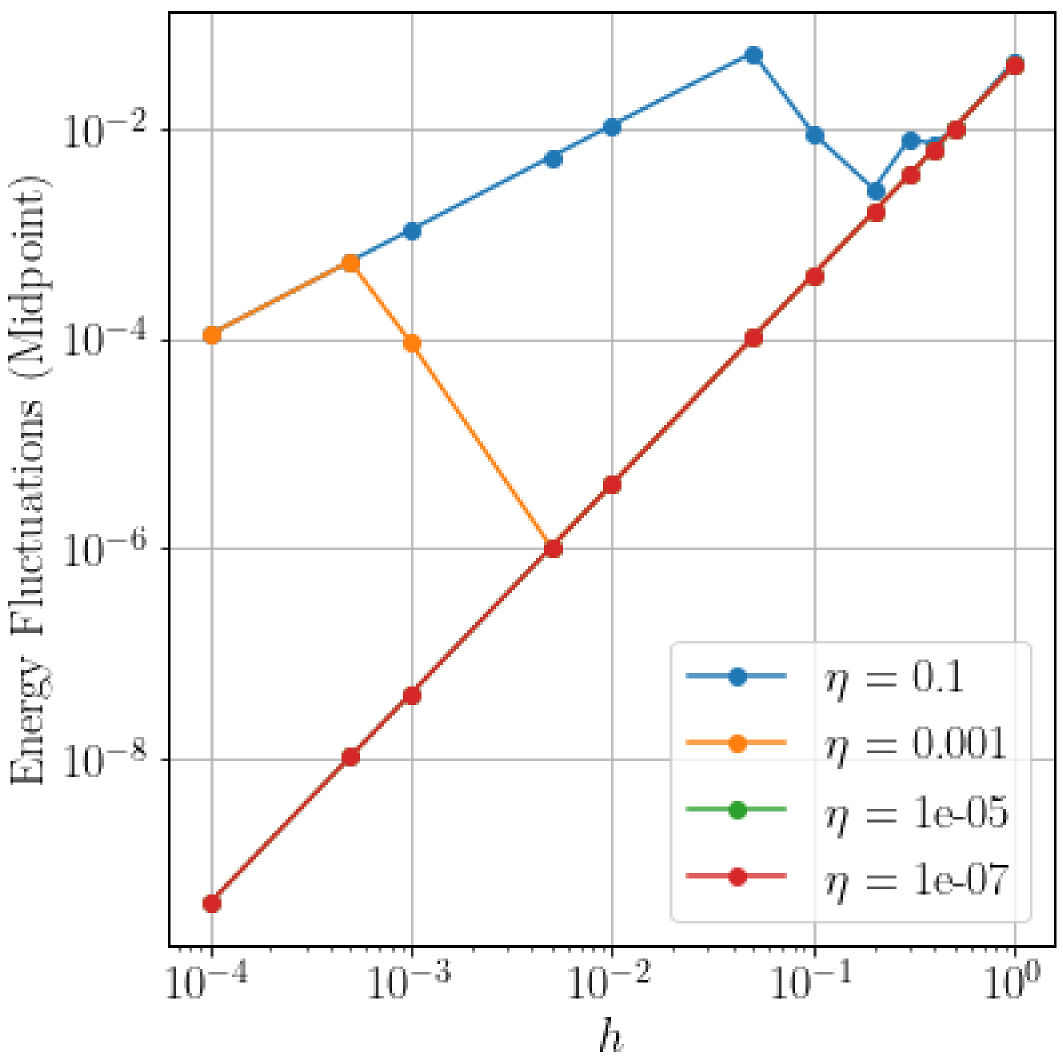


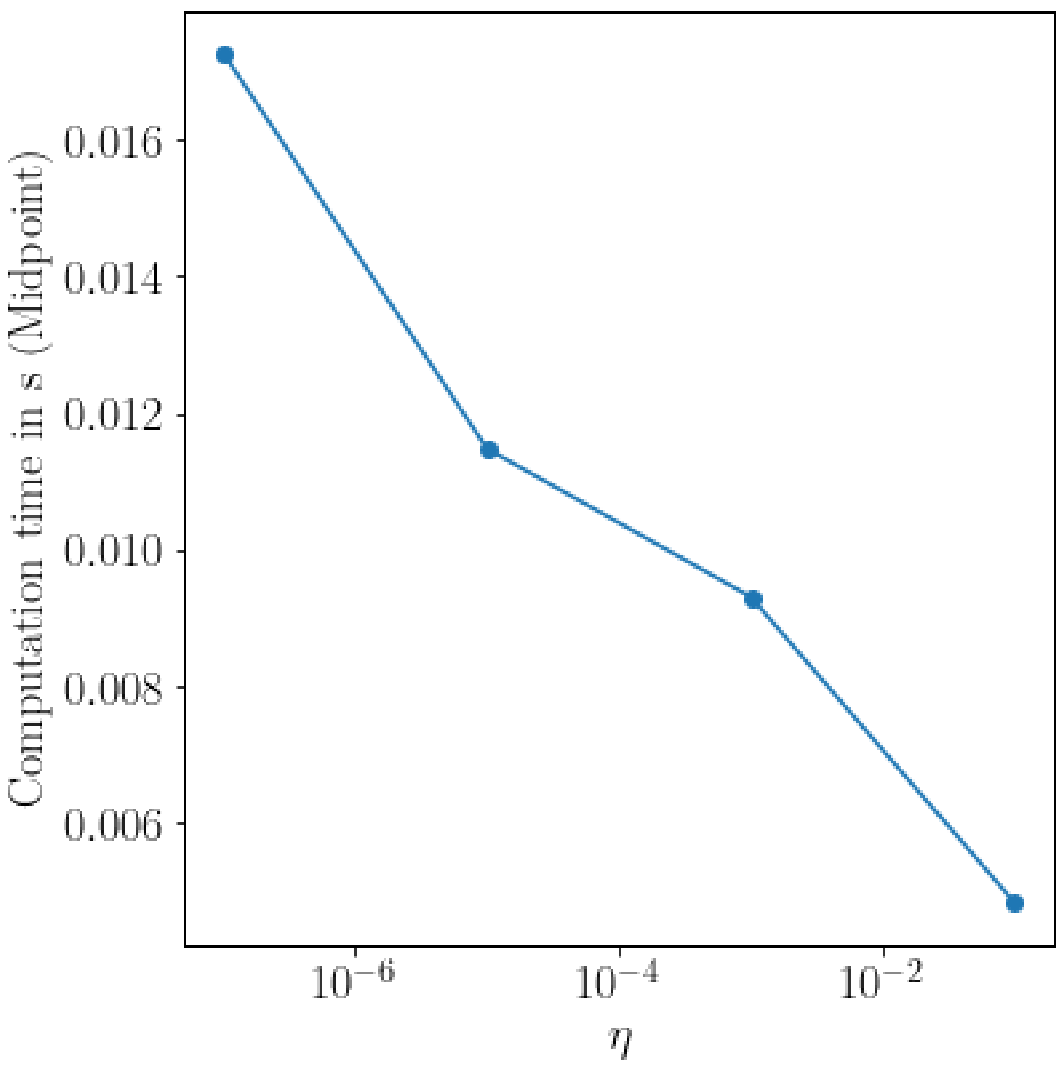


We finally need to analyse the role of η. We thus monitor the energy fluctuations and the total time to integrate the equations of motion. We find that energy fluctuations are almost insensitive to the value of η, as soon as $\eta \lesssim 10^{-3}$. Instead, the computation time midly grows with decreasing η (suggesting that the convergence between two iterations is quite fast and a quite small value of $\eta \simeq 10^{-5}$ can be taken without divergence of the calculation time).

[10]:
```python
import timeit

H_arr = np.array([0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4,␣
 ↪0.5, 1.])
eta_arr = np.array([1e-1, 1e-3, 1e-5, 1e-7])
En_exc = np.zeros((len(eta_arr), len(H_arr)))
for i in range(len(H_arr)):
    for j in range(len(eta_arr)):
        t, y = midpoint(10, H_arr[i], eta_arr[j], np.array([3. * np.pi / 4., 0.
 ↪]))
        En_exc[j,i] += [np.std(0.5 * y[:,1] * y[:,1] - np.cos(y[:,0]))]
fig, ax = plt.subplots(figsize = (6, 6), tight_layout = True)
for j in range(len(eta_arr)):
   ax.loglog(H_arr, En_exc[j,:], '-o', label = r'$\eta$ = ' + '{0}'.
 ↪format(eta_arr[j]))
ax.set_xlabel(r'$h$')
ax.set_ylabel(r'Energy Fluctuations (Midpoint)')
ax.legend()
ax.grid()
fig.show()

time_comp = []
for eta in eta_arr:
    start = timeit.default_timer()
    midpoint(10, 0.01, eta, np.array([3. * np.pi / 4., 0.]))
    stop = timeit.default_timer()
    time_comp += [ stop - start ] # time in seconds
time_comp = np.array(time_comp)
fig, ax = plt.subplots(figsize = (6, 6), tight_layout = True)
ax.semilogx(eta_arr, time_comp, '-o')
ax.set_xlabel(r'$\eta$')
ax.set_ylabel(r'Computation time in s (Midpoint)')
fig.show()
```

```
/tmp/ipykernel_3144492/2853120865.py:17: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/2853120865.py:30: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
```


Energy Fluctuations (Midpoint)
h
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
η = 0.1
η = 0.001
η = 1e-05
η = 1e-07



Question 2.1

In the case of large-enough μ, the Van der Pol oscillator behaves as a relaxation oscillator (this is what is meant when one says that the Van der Pol oscillator is "stiff"). To see that, we set $y = \dot{x}/\mu + F(x)$ with $F(x) = x^3/3 - x$. We then obtain that

$$\begin{cases} \dot{y} = -x/\mu, \\ \dot{x} = \mu\left[y - F(x)\right]. \end{cases}$$

If $\mu \gg 1$, then starting from a point (x, y) with $x > 0$ and $y > 0$, $y - F(x)$ is of order 1 and the velocity in the x-direction is very large. The system thus converges horizontally in the plane (x, y) to the curve of $F(x)$ and follows it up to its minimum at $x = 1$. At this point, $|\dot{x}|$ starts to be very large again, until the system jumps to the curve $F(x)$ in the region $x < 0$. It then follows the curve $F(x)$ up to its maximum at $x = -1$ and jumps again to the curve $F(x)$ in the region $x > 0$. Overall, the system behaves as a relaxation oscillator with a period given by twice the time to follow the curve $F(x)$ to its minimum:

$$T = 2\int_{x_0}^{1} \frac{\mathrm{d}y}{\dot{y}} = 2\mu\int_{1}^{x_0} \frac{\mathrm{d}y}{x} = 2\mu\int_{1}^{x_0} \frac{\mathrm{d}y}{x} = 2\mu\int_{1}^{x_0} \frac{\mathrm{d}x}{x}F'(x),$$

where $F(x_0) = F(-1) = 2/3$. It is easy to show that $x_0 = 2$, so that

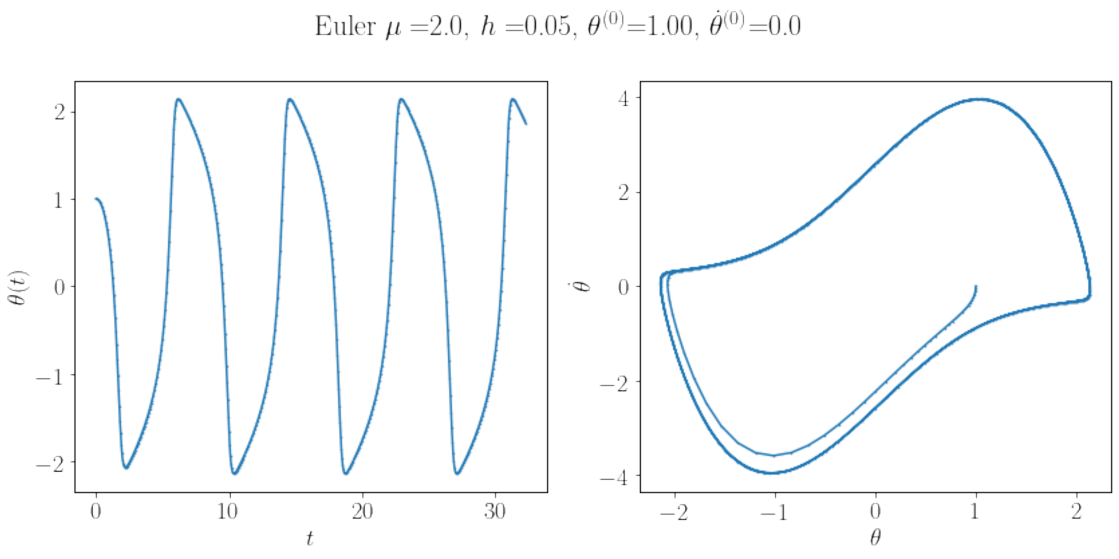
$$T = 2\mu\int_{1}^{2} \mathrm{d}x \left(x - \frac{1}{x}\right) = 2\mu\left[\frac{x^2}{2} - \ln x\right]_1^2 = \mu\,(3 - 2\ln 2)\,.$$

As a result, in order to observe the physics of the Van der Pol oscillator, one should perform a simulation of duration larger than T. However, this calculation just gives a crude estimate of the period as it is obtained in the limit of very large μ.

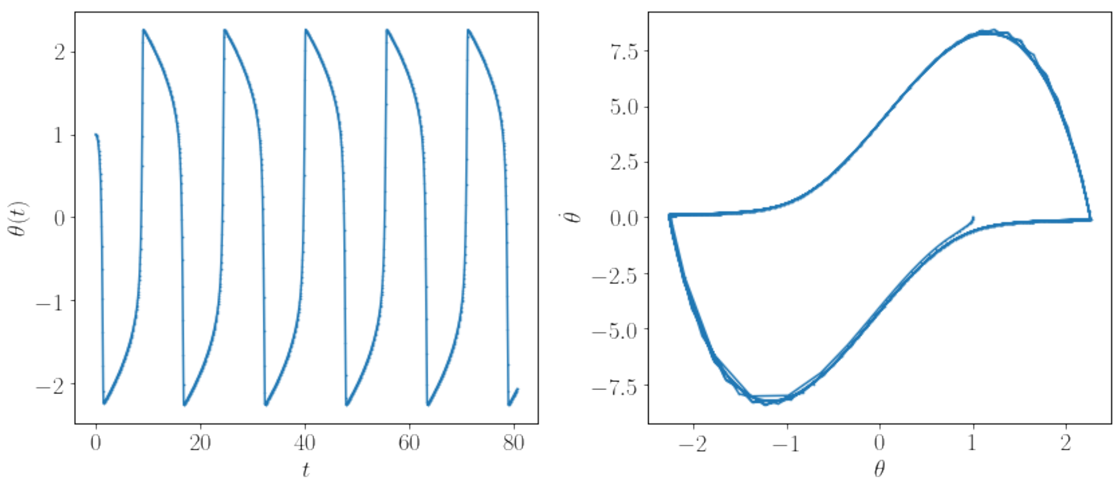
Instead, if μ is small but non zero, the physics should be closer to the one of the harmonic oscillator, with quasi-sinusoidal oscillations.

[11]:
```python
def syst_dyn_vdP(y,t,mu):
    return np.array([y[1], - y[0] + mu * y[1] * ( 1. - y[0] * y[0] )])

def Euler_vdP(t0,h,mu,yinit):
    t = [0.]
    y0 = [yinit[0]]
    y1 = [yinit[1]]
    tf = 0.
    y0f = yinit[0]
    y1f = yinit[1]
    while tf < t0:
        dy = h * syst_dyn_vdP(np.array([y0f,y1f]), tf, mu)
        y0f += dy[0]
        y1f += dy[1]
        tf += h
        t += [tf]
        y0 += [y0f]
        y1 += [y1f]
    return np.array(t), np.vstack((np.array(y0), np.array(y1))).transpose()

def plot_sol_Euler_vdP(t0,h,mu,yinit):
    t, y = Euler_vdP(t0,h,mu,yinit)
    fig, ax = plt.subplots(1, 2, figsize = (12, 6), tight_layout = True)
    fig.suptitle(r'Euler $\mu=$' + str(mu) + r', $h=$' + str(h) + r',␣
 ↪$\theta^{(0)}$=' + '{0:.2f}'.format(yinit[0]
    ) + r', $\dot{\theta}^{(0)}$=' + str(yinit[1]))
    ax[0].plot(t, y[:,0], '-o', markersize = 0.5)
    ax[0].set_xlabel(r'$t$')
    ax[0].set_ylabel(r'$\theta(t)$')
    ax[1].plot(y[:,0], y[:,1], '-o', markersize = 0.5)
    ax[1].set_xlabel(r'$\theta$')
    ax[1].set_ylabel(r'$\dot{\theta}$')
    fig.show()

def T(mu):
    return max(mu * ( 3. - 2. * np.log(2.) ), 1.)

plot_sol_Euler_vdP(10 * T(2.), 0.05, 2., np.array([1., 0.]))
plot_sol_Euler_vdP(10 * T(5.), 0.05, 5., np.array([1., 0.]))
plot_sol_Euler_vdP(10 * T(10.), 0.05, 10., np.array([1., 0.]))
```

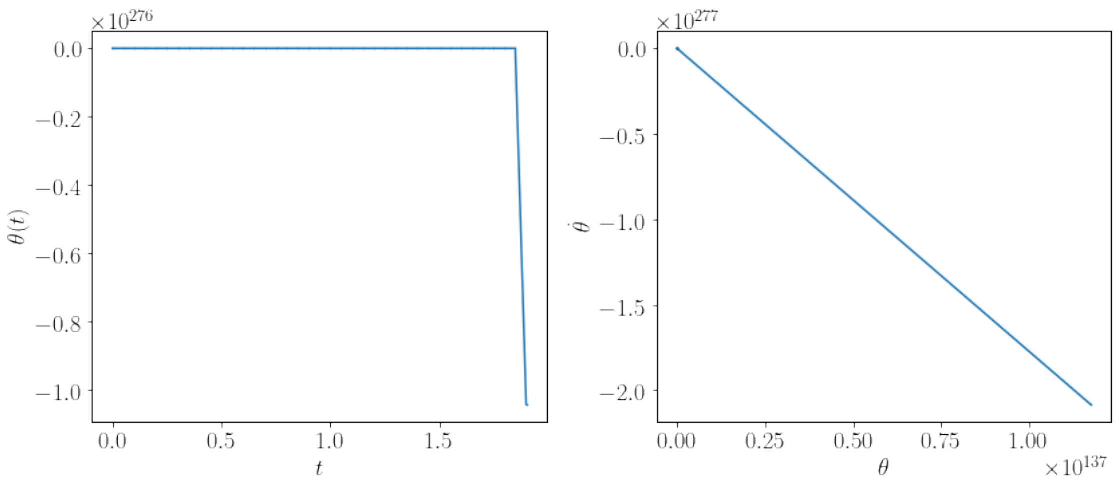
```
/tmp/ipykernel_3144492/721846563.py:32: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/721846563.py:32: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/721846563.py:2: RuntimeWarning: overflow encountered in
double_scalars
  return np.array([y[1], - y[0] + mu * y[1] * ( 1. - y[0] * y[0] )])
/tmp/ipykernel_3144492/721846563.py:14: RuntimeWarning: invalid value
encountered in double_scalars
  y1f += dy[1]
/tmp/ipykernel_3144492/721846563.py:32: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
```



Euler μ =5.0, h =0.05, $\theta^{(0)}$=1.00, $\dot{\theta}^{(0)}$=0.0



Euler μ =10.0, h =0.05, $\theta^{(0)}$=1.00, $\dot{\theta}^{(0)}$=0.0

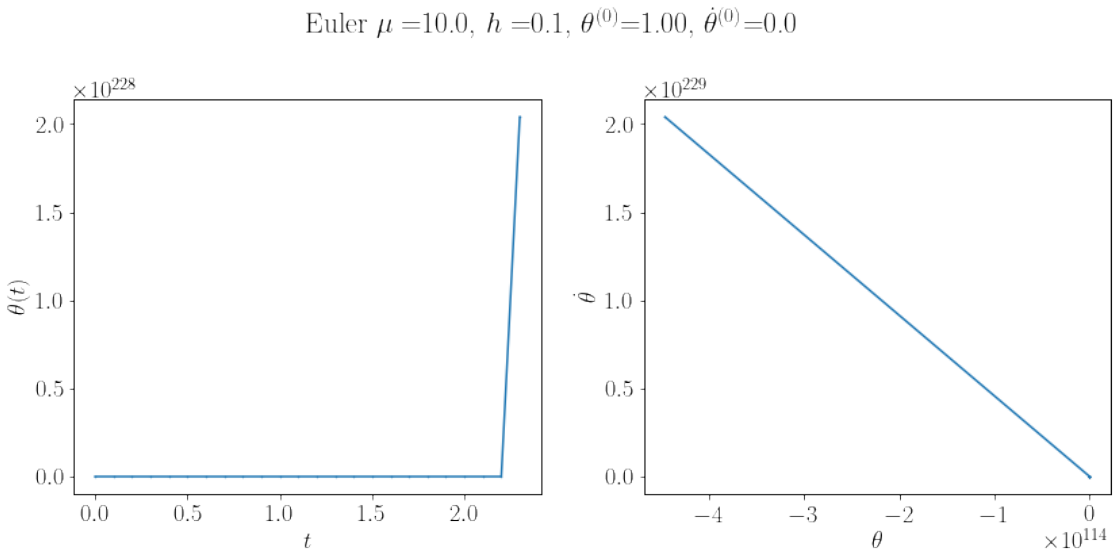


As μ gets larger, the integration timestep h has to be decreased in order to correctly integrate the equations of motion when there is a switch event.

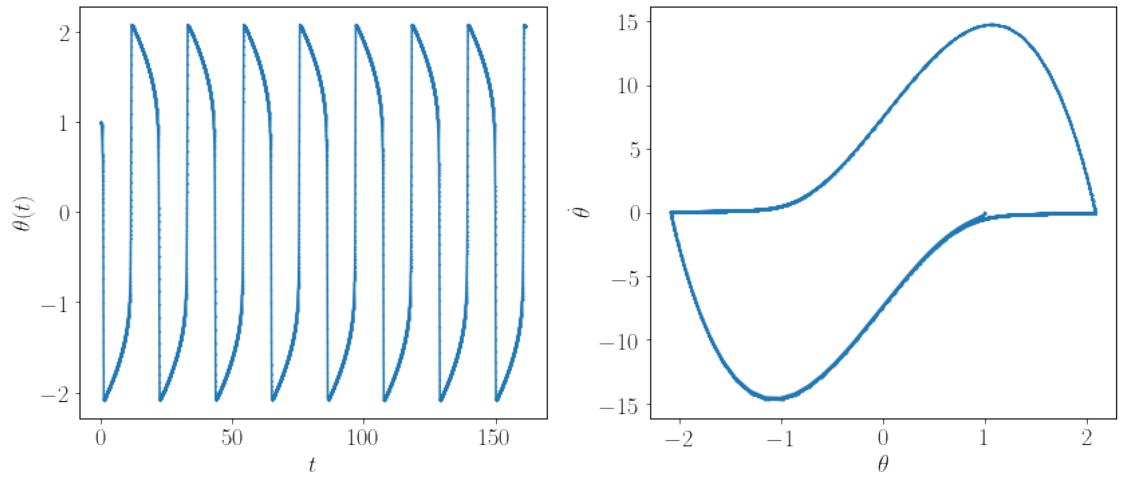
[12]:
```
plot_sol_Euler_vdP(10 * T(10.), 0.1, 10., np.array([1., 0.]))
plot_sol_Euler_vdP(10 * T(10.), 0.01, 10., np.array([1., 0.]))
plot_sol_Euler_vdP(10 * T(10.), 0.001, 10., np.array([1., 0.]))
```

```
/tmp/ipykernel_3144492/721846563.py:2: RuntimeWarning: overflow encountered in
double_scalars
  return np.array([y[1], - y[0] + mu * y[1] * ( 1. - y[0] * y[0] )])
/tmp/ipykernel_3144492/721846563.py:14: RuntimeWarning: invalid value
```

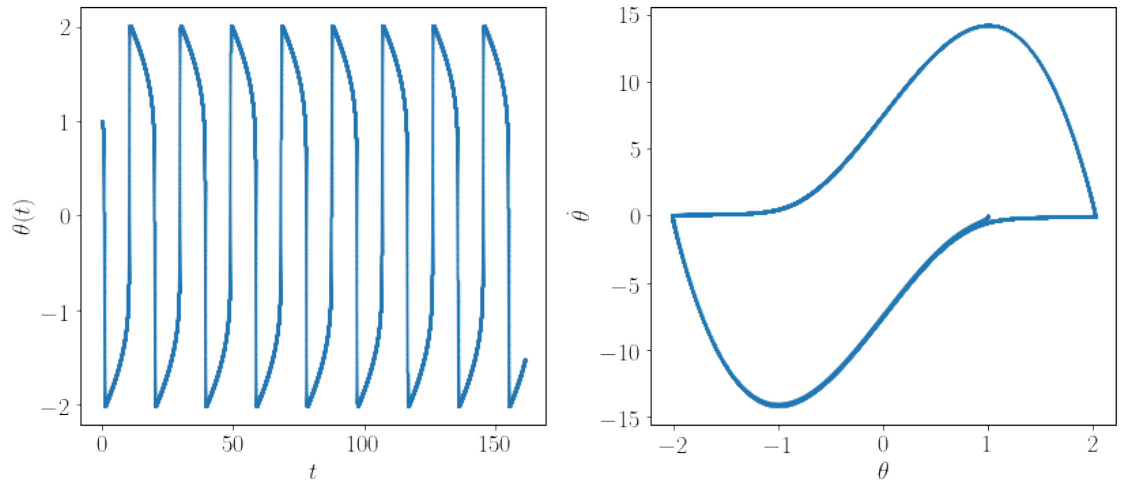
```
encountered in double_scalars
  y1f += dy[1]
/tmp/ipykernel_3144492/721846563.py:32: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/721846563.py:32: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/721846563.py:32: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
```



Euler μ =10.0, h =0.01, $\theta^{(0)}$=1.00, $\dot{\theta}^{(0)}$=0.0



Euler μ =10.0, h =0.001, $\theta^{(0)}$=1.00, $\dot{\theta}^{(0)}$=0.0



Question 2.2

To determine $(y_{1,n+1},\ y_{2,n+1})$, we have to solve the following set of equations:

$$\begin{cases} y_{1,n+1} = y_{1,n} + \dfrac{h}{2}\left(y_{2,n} + y_{2,n+1}\right) \\ y_{2,n+1} = y_{2,n} + \dfrac{\mu h}{2}\left[\left(1 - y_{1,n}^2\right) y_{2,n} + \left(1 - y_{1,n+1}^2\right) y_{2,n+1}\right] - \dfrac{h}{2}\left(y_{1,n} + y_{1,n+1}\right) \end{cases}.$$

The latter can be transformed into a third-order polynomial equation for $y_{1,n+1}$ (or $y_{2,n+1}$):

$$\begin{cases} y_{2,n+1} = \dfrac{2}{h}\left(y_{1,n+1} - y_{1,n}\right) - y_{2,n} \\ y_{1,n+1}^3 - \left(y_{1,n} + \dfrac{h}{2}y_{2,n}\right) y_{1,n+1}^2 + \left(\dfrac{h}{2\mu} + \dfrac{2}{\mu h} - 1\right) y_{1,n+1} + y_{1,n}\left(1 + \dfrac{h}{2\mu} - \dfrac{2}{\mu h}\right) + \left(\dfrac{h}{2}y_{1,n}^2 - \dfrac{2}{\mu}\right) y_{2,n} = 0 \end{cases}.$$

To solve the cubic equation, we introduce the auxiliary variable $u = y_{1,n+1} - [y_{1,n} - (h/2)y_{2,n}]/3$. The equation verified by u is then

$$u^3 + Px + Q = 0,$$

with

$$\begin{cases} P = \dfrac{h}{2\mu} + \dfrac{2}{\mu h} - 1 - \dfrac{1}{3}\left(y_{1,n} + \dfrac{h}{2}y_{2,n}\right)^2 \\ Q = -\dfrac{2}{27}\left(y_{1,n} + \dfrac{h}{2}y_{2,n}\right)^3 + \dfrac{1}{3}\left(y_{1,n} + \dfrac{h}{2}y_{2,n}\right)\left(\dfrac{h}{2\mu} + \dfrac{2}{\mu h} - 1\right) + y_{1,n}\left(1 + \dfrac{h}{2\mu} - \dfrac{2}{\mu h}\right) + \left(\dfrac{h}{2}y_{1,n}^2 - \dfrac{2}{\mu}\right) y_{2,n} \end{cases}.$$

By studying the function $f : u \mapsto u^3 + Pu + Q$, one can show that it has only one real root when $27Q^2 + 4P^3 > 0$ (and two complex conjugate roots), and three real roots otherwise. In the limit $h \to 0$, $P \to +\infty$ and $Q \to -\infty$, so that there is a single real root, which corresponds to $y_{1,n+1} - [y_{1,n} - (h/2)y_{2,n}]/3$.

To find the real root in this situation, we write $u = u_1 + u_2$, and we get $u_1^3 + u_2^3 + (3u_1u_2 + P)(u_1 + u_2) + Q = 0$ and we impose that $3u_1u_2 + P = 0$, leading to the set of two equations:

$$\begin{cases} u_1 u_2 = -\dfrac{P}{3} \\ u_1^3 + u_2^3 = -Q \end{cases} \implies \begin{cases} u_1^3 u_2^3 = -\left(\dfrac{P}{3}\right)^3 \\ u_1^3 + u_2^3 = -Q \end{cases}.$$

As a consequence u_1^3 and u_2^3 are the two roots of the quadratic equation $X^2 + QX - (P/3)^3 = 0$, whose discriminant is $\Delta = Q^2 + 4(P/3)^3 > 0$, leading to

$$u_1^3 = -\frac{Q}{2} + \sqrt{\left(\frac{Q}{2}\right)^2 + \left(\frac{P}{3}\right)^3}, \quad u_2^3 = -\frac{Q}{2} - \sqrt{\left(\frac{Q}{2}\right)^2 + \left(\frac{P}{3}\right)^3},$$

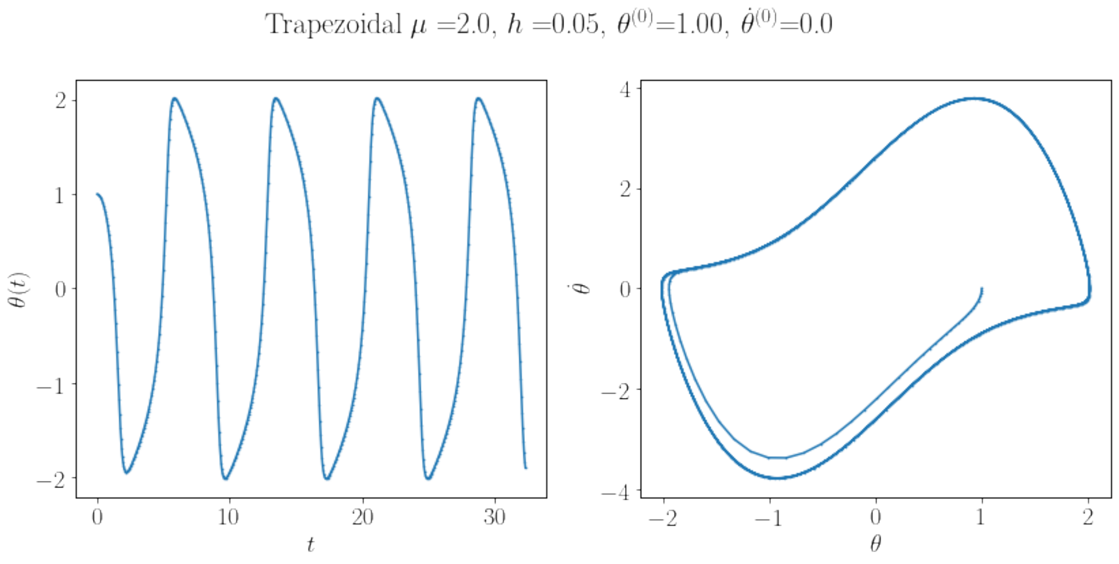
and eventually to

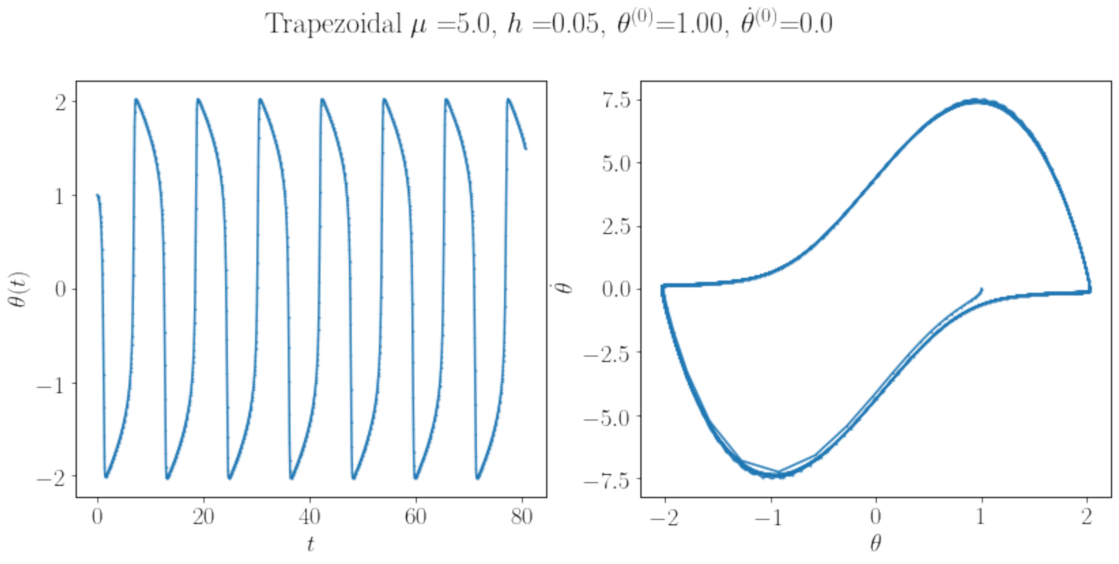
$$\begin{cases} y_{1,n+1} = \dfrac{1}{3}\left(y_{1,n} + \dfrac{h}{2}y_{2,n}\right) + u_1 + u_2 = \dfrac{1}{3}\left(y_{1,n} + \dfrac{h}{2}y_{2,n}\right) + \left[\dfrac{Q}{2} + \sqrt{\left(\dfrac{Q}{2}\right)^2 + \left(\dfrac{P}{3}\right)^3}\right]^{1/3} + \left[\dfrac{Q}{2} - \sqrt{\left(\dfrac{Q}{2}\right)^2 + \cdots}\right. \\ y_{2,n+1} = -\dfrac{2}{3}\left(y_{2,n} + \dfrac{2}{h}y_{1,n}\right) + \dfrac{2}{h}\left[\dfrac{Q}{2} + \sqrt{\left(\dfrac{Q}{2}\right)^2 + \left(\dfrac{P}{3}\right)^3}\right]^{1/3} + \dfrac{2}{h}\left[\dfrac{Q}{2} - \sqrt{\left(\dfrac{Q}{2}\right)^2 + \left(\dfrac{P}{3}\right)^3}\right]^{1/3} \end{cases}$$

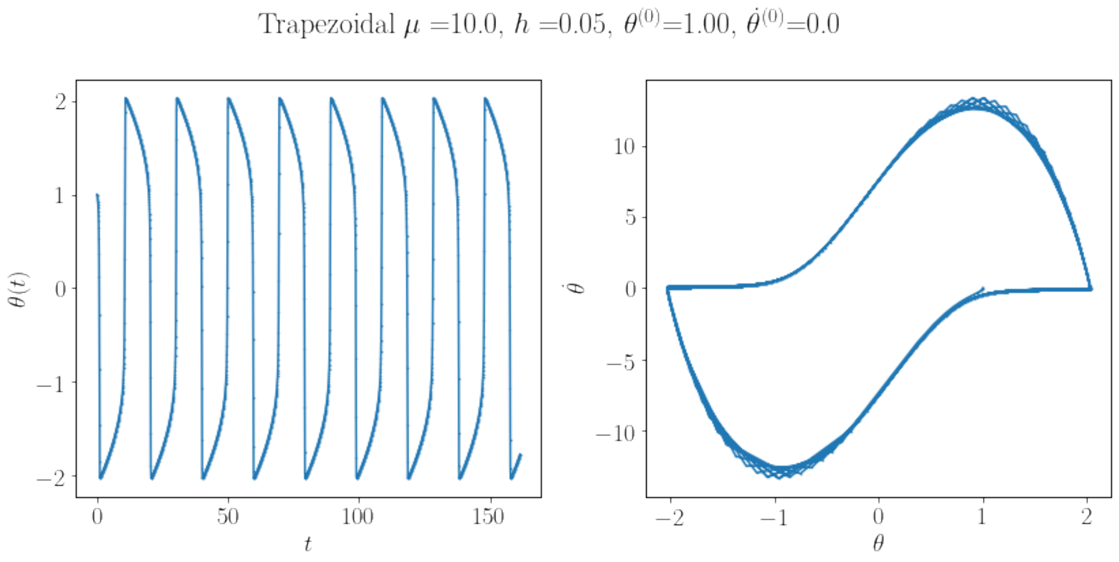
The integration scheme works better than the Euler procedure at equal timestep h.

[13]:
```python
def trapez_vdP(t0,h,mu,yinit):
    t = [0.]
    y0 = [yinit[0]]
    y1 = [yinit[1]]
    tf = 0.
    y0f = yinit[0]
    y1f = yinit[1]
    while tf < t0:
        tmp = y0f + 0.5 * h * y1f
        PP = 0.5 * h / mu + 2. / mu / h - 1. - tmp * tmp / 3.
        QQ = - 2. * ( tmp / 3. ) ** 3 + tmp / 3. * ( 0.5 * h / mu + 2. / mu / h␣
↪- 1. )
        QQ += y0f * (1. + 0.5 * h / mu - 2. / mu / h ) + ( 0.5 * h * y0f * y0f␣
↪- 2. / mu ) * y1f
        delta = np.sqrt(( 0.5 * QQ ) ** 2 + ( PP / 3. ) ** 3)
        u1 = - 0.5 * QQ - delta
        if u1 > 0.:
            u1 = u1 ** ( 1. / 3. )
        else:
            u1 = - ( - u1 ) ** ( 1. / 3. )
        u2 = - 0.5 * QQ + delta
        if u2 > 0.:
            u2 = u2 ** ( 1. / 3. )
        else:
            u2 = - ( - u2 ) ** ( 1. / 3. )
        y0f = tmp / 3. + u1 + u2
        y1f = - 4. / h * tmp / 3. + 2. / h * ( u1 + u2 )
        tf += h
        t += [tf]
        y0 += [y0f]
        y1 += [y1f]
    return np.array(t), np.vstack((np.array(y0), np.array(y1))).transpose()

def plot_sol_trapez_vdP(t0,h,mu,yinit):
    t, y = trapez_vdP(t0,h,mu,yinit)
    fig, ax = plt.subplots(1, 2, figsize = (12, 6), tight_layout = True)
    fig.suptitle(r'Trapezoidal $\mu=$' + str(mu) + r', $h=$' + str(h) + r',␣
↪$\theta^{(0)}$=' + '{0:.2f}'.format(yinit[0]
    ) + r', $\dot{\theta}^{(0)}$=' + str(yinit[1]))
    ax[0].plot(t, y[:,0], '-o', markersize = 0.5)
    ax[0].set_xlabel(r'$t$')
    ax[0].set_ylabel(r'$\theta(t)$')
    ax[1].plot(y[:,0], y[:,1], '-o', markersize = 0.5)
    ax[1].set_xlabel(r'$\theta$')
    ax[1].set_ylabel(r'$\dot{\theta}$')
    fig.show()

plot_sol_trapez_vdP(10 * T(2.), 0.05, 2., np.array([1., 0.]))
plot_sol_trapez_vdP(10 * T(5.), 0.05, 5., np.array([1., 0.]))
plot_sol_trapez_vdP(10 * T(10.), 0.05, 10., np.array([1., 0.]))
```

```
/tmp/ipykernel_3144492/437391730.py:43: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/437391730.py:43: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
/tmp/ipykernel_3144492/437391730.py:43: UserWarning: Matplotlib is currently
using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so
cannot show the figure.
  fig.show()
```






Trapezoidal μ =10.0, h =0.05, $\theta^{(0)}$=1.00, $\dot{\theta}^{(0)}$=0.0
$\theta(t)$
t
0
50
100
150
2
1
0
−1
−2
$\dot{\theta}$
θ
10
5
0
−5
−10
−2
−1
0
1
2