H. Ammari
K. Alexopoulos

Spring Term 2022

Numerical Analysis II

ETH Zürich
D-MATH

Problem Sheet 1

Problem 1.1 Simple Pendulum

(1.1a) A simple pendulum is one which can be considered to be a point mass suspended from a string or rod of negligible mass. Denote

- $l =$ length of the rod/string in meters
- $m =$ mass of the point mass in kilograms
- $g =$ accelaration due to gravity $= 9.81007 m/s^2$ according to Wikipedia.

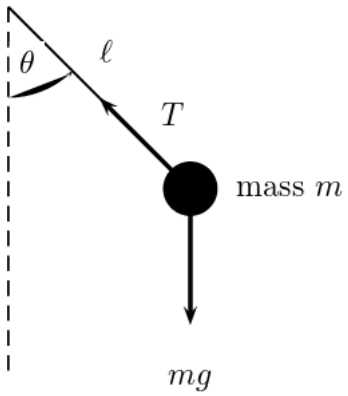

Figure 1.1: Simple pendulum

Using Newton's laws of motion, show that the angle $\theta(t)$, expressed as a function of time, satisfies the differential equation

$$\ddot{\theta} = -\frac{g \sin\theta}{l}. \tag{1.1.1}$$

Solution: Fix the coordinate system as shown in Figure 1.2. Let $(x(t), y(t))$ denote the position of the point mass at time t. By Newton's second law, we have that

$$\begin{cases} F_x = m\dfrac{\mathrm{d}^2 x}{\mathrm{d}t^2}, \\ F_y = m\dfrac{\mathrm{d}^2 y}{\mathrm{d}t^2}. \end{cases}$$

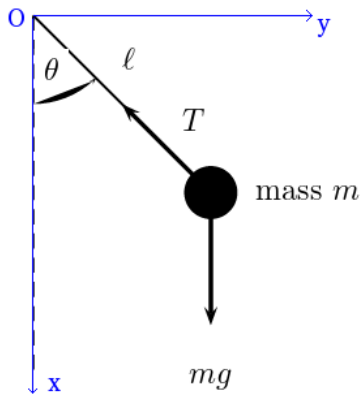

Figure 1.2: Coordinate system

Decomposing the forces acting on the mass gives the system

$$\begin{cases} F_x = mg - T\cos\theta, \\ F_y = -T\sin\theta. \end{cases}$$

Notice that (x, y) must obey the geometric constraints

$$\begin{cases} x = l\cos\theta, \\ y = l\sin\theta. \end{cases}$$

Taking the derivative of (x, y) with respect to t, we have

$$\begin{aligned} \dot{x} &= -l\sin\theta\dot{\theta}, \\ \ddot{x} &= -l\cos\theta\dot{\theta}^2 - l\sin\theta\ddot{\theta}, \\ \dot{y} &= l\cos\theta\dot{\theta}, \\ \ddot{y} &= -l\sin\theta\dot{\theta}^2 + l\cos\theta\ddot{\theta}. \end{aligned}$$

Combining everything together, we reach

$$\begin{aligned} -m(l\cos\theta\dot{\theta}^2 + l\sin\theta\ddot{\theta}) &= mg - T\cos\theta, \\ m(-l\sin\theta\dot{\theta}^2 + l\cos\theta\ddot{\theta}) &= -T\sin\theta. \end{aligned}$$

After eliminating T, the equation system becomes

$$ml(\sin^2\theta + \cos^2\theta)\ddot{\theta} = -mg\sin\theta,$$

that is

$$\ddot{\theta} = -\frac{g\sin\theta}{l}. \tag{1.1.2}$$

(1.1b) Write a Python script that solves your ODE for the initial conditions $\theta(0) = \pi/4$, $\dot{\theta}(0) = 0$. Plot the solution $\theta(t)$ for $t \in [0, 10]$.

HINT: You may need to convert the second-order ODE into first-order ODE system to use the ODE solver.

Solution:

Listing 1.1: simplependulum.py

```
import numpy as np
from scipy.integrate import ode
import matplotlib.pyplot as plt

def g():
    return 9.81007
def l():
    return 1.0
def fun(t, y):
    return np.array([y[1], - g() * np.sin(y[0])/l()])
t0, t1 = 0, 10                          # start and end
t = np.linspace(t0, t1, 100)            # the points of
   evaluation of solution
y0 = [np.pi/4, 0]                       # initial value
y = np.zeros((len(t), len(y0)))         # array for solution
y[0, :] = y0
r = ode(fun)
r.set_initial_value(y0, t0)             # set initial values
for i in range(1, t.size):
   y[i, :] = r.integrate(t[i])          # compute each time step
plt.plot(t, y[:,0])
plt.show()
```

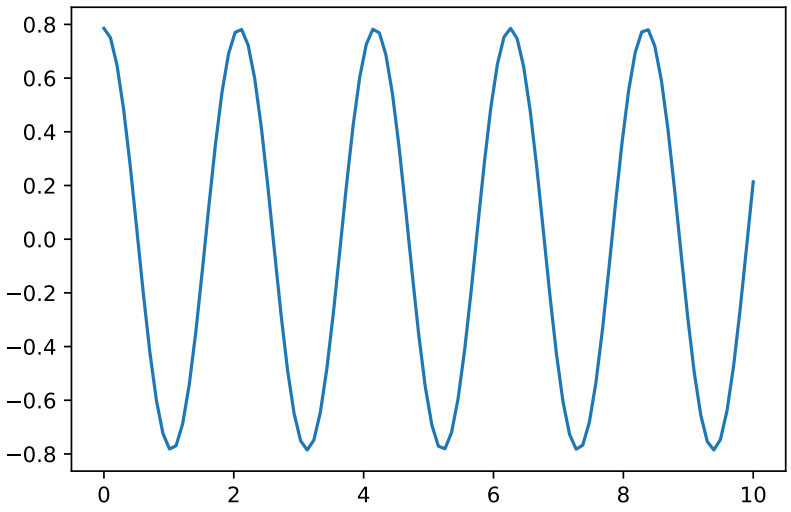

Figure 1.3: Plot of $\theta(t)$, from `simplependulum.py`

(1.1c) Recall the Taylor expansion of $\sin\theta = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} + \ldots$. Thus, if θ is small we can substitute $\sin\theta \approx \theta$ in our equation (1.1.1). Repeat the analysis of (1.1b) for the approximated

equation:

$$\ddot{\theta} = -\frac{g\theta}{l}. \tag{1.1.3}$$

Now, consider the case where $\theta(0) \approx \pi$, i.e. pendulum is almost inverted. If we let $\phi = \theta - \pi$ then we have that $\sin\theta = -\sin\phi = -\phi + \frac{\phi^3}{3!} - \frac{\phi^5}{5!} + \dots$ whenever $\phi \approx 0$.

Using a small angle approximation around π, solve the ODE and plot the solution in the case that $\theta(0) = 3\pi/4, \dot{\theta}(0) = 0$. What's wrong?

Solution: (1.1.3) is solved using `simplependulumapprox.py`, the solution is shown in Figure 1.4.

Listing 1.2: simplependulumapprox.py

```
import numpy as np
from scipy.integrate import ode
import matplotlib.pyplot as plt

def g():
    return 9.81007
def l():
    return 1.0
def fun(t, y):
    return np.array([y[1], - g() * y[0]/l()])
t0, t1 = 0, 10                          # start and end
t = np.linspace(t0, t1, 100)            # the points of
   evaluation of solution
y0 = [np.pi/4, 0]                       # initial value
y = np.zeros((len(t), len(y0)))         # array for solution
y[0, :] = y0
r = ode(fun)
r.set_initial_value(y0, t0)             # set initial values
for i in range(1, t.size):
   y[i, :] = r.integrate(t[i])          # compute each time step
plt.plot(t, y[:,0])
plt.show()
```

When the pendulum is almost inverted, its small angle approximation equation is:

$$\ddot{\phi} = \frac{g\phi}{l},$$

where $\phi = \theta - \pi$. This is solved using `simplependulumapproxinverted.py`, the solution is shown in Figure 1.5. The reason that the outcome is wrong is that the direction of the force T (as shown in Figure 1.2) is different in the cases $\theta > \pi/2$ and $\theta < \pi/2$.

Listing 1.3: simplependuluminverted.py

```
import numpy as np
from scipy.integrate import ode
import matplotlib.pyplot as plt
```

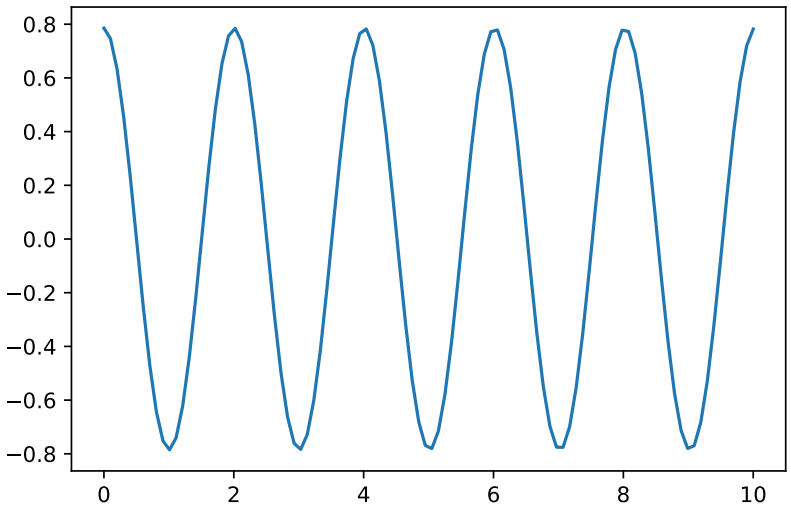

Figure 1.4: Plot of approximated $\theta(t)$, from `simplependulumapprox.py`

```
4
5  def g():
6      return 9.81007
7  def l():
8      return 1.0
9  def fun(t, y):
10     return np.array([y[1], g() * y[0]/l()])
11 t0, t1 = 0, 10                          # start and end
12 t = np.linspace(t0, t1, 100)           # the points of
       evaluation of solution
13 y0 = [-np.pi/4, 0]                      # initial value
14 y = np.zeros((len(t), len(y0)))         # array for solution
15 y[0, :] = y0
16 r = ode(fun)
17 r.set_initial_value(y0, t0)            # set initial values
18 for i in range(1, t.size):
19     y[i, :] = r.integrate(t[i])         # compute each time step
20 plt.plot(t, y[:,0]+np.pi)
21 plt.show()
```

Problem 1.2 Population dynamics

In biological applications, the population P of certain organisms at time t is sometimes assumed to obey the equation

$$\frac{\mathrm{d}P}{\mathrm{d}t} = aP(1 - \frac{P}{E}), \tag{1.2.1}$$

where a and E are positive constants. This model is sometimes called the logistic growth model. P needs to be non-negative.

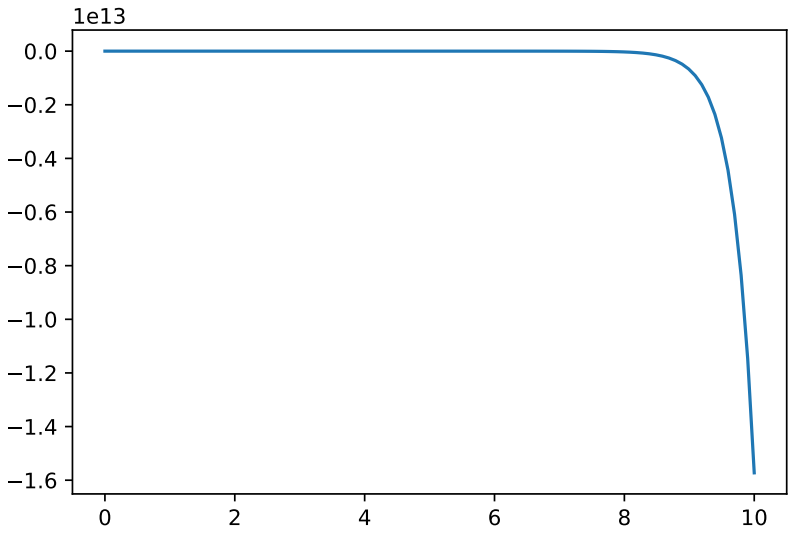

Figure 1.5: Plot of $\theta(t)$ for the inverted pendulum, from `simplependuluminverted.py`

(1.2a) Find the equilibrium solutions, i.e. solutions that do not vary in time.

Solution: An equilibrium solution is such that $\frac{\mathrm{d}P}{\mathrm{d}t} = 0$, thus $P = 0$ or $P = E$.

(1.2b) For which values of P is P increasing or decreasing as a function of t? Using (1.2.1), find an expression for $\frac{\mathrm{d}^2P}{\mathrm{d}t^2}$ in terms of P and the constants a and E. For which values of P is P convex $\left(\frac{\mathrm{d}^2P}{\mathrm{d}t^2} > 0\right)$ or concave $\left(\frac{\mathrm{d}^2P}{\mathrm{d}t^2} < 0\right)$ as a function of t?

Solution: P is monotone increasing in the region region $0 < P < E$ and monotone decreasing if $P > E$.

The second derivative of P is

$$\begin{aligned}\frac{\mathrm{d}^2P}{\mathrm{d}t^2} &= \frac{\mathrm{d}}{\mathrm{d}t}\frac{\mathrm{d}P}{\mathrm{d}t} \\ &= (a - \frac{2aP}{E})\frac{\mathrm{d}P}{\mathrm{d}t} \\ &= a^2P(1 - \frac{2P}{E})(1 - \frac{P}{E}).\end{aligned}$$

So, P is convex when $P > E$ or $0 < P < E/2$ and P is concave when $E/2 < P < E$.

(1.2c) Let $E = 100$ and $a = 0.1$. Write a Python script to solve (1.2.1) and plot the graph of P as a function of t over the interval $[0, 100]$. Use $P_0 = 10$ and $P_1 = 150$ as initial values, plotting the two curves on the same axes. What's the behaviour of P as t becomes large?

Solution:

See `population.py` and Figure 1.6.

Listing 1.4: population.py

```
1 import numpy as np
```

```
from scipy.integrate import ode
import matplotlib.pyplot as plt

def E():
    return 100
def a():
    return 0.1
def fun(t, y):
    return a()*y*(1-y/E())
t0, t1 = 0, 100                    # start and end
t = np.linspace(t0, t1, 100)       # the points of
   evaluation of solution
init = [10, 150]                   # initial values
y = np.zeros((len(t),len(init)))   # array for solution
for i in range(len(init)):
    y[0, i] = init[i]
    r = ode(fun)
    r.set_initial_value(init[i], t0)          # set initial
       values
    for j in range(1, t.size):
        y[j,i] = r.integrate(t[j])      # compute each time
           step
plt.plot(t, y)
plt.show()
```

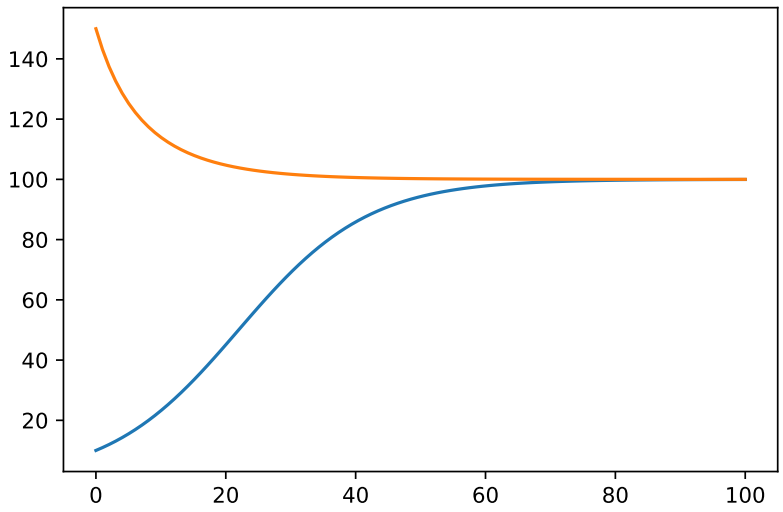

Figure 1.6: Population $P(t)$, for initial values $P_0 = 10$ and $P_1 = 150$.

For large t, both solutions converge to 100, which is the positive equilibrium solution (since $E = 100$).

Published on 2 March 2022.
To be submitted by 10 March 2022.