Computer methods in engineering
Ordinary differential equations (ODE)

Solutions to ordinary differential equations (ODE) of the form

$$y'(t) = f(t, y(t)) \tag{1}$$
$$y(t_0) = y_0 \quad , \tag{2}$$

where t is the variable and $y(t)$ is the function, can be estimated numerically using Euler methods. In this exercise we will solve an ODE problem using both forward and backward Euler methods, and using Heun's method. These numerical methods should be implemented in Python, and both the solution and the code should be part of what is handed in.

All three methods are iterative methods, where the value of y at a point t a distance away from the initial value t_0 is found by discretizing the distance between t_0 and t, and then iteratively moving from t_0 towards t. As with most numerical methods, also this method linearize the problem, where each iterative step is linear. A pseudo-code for a general algorithm is shown in Alg. 1.

Algorithm 1 General iterative numerical solver for ODE

Require: $f(t, y), y_0, t_0, h, N$

for $i = 1, \ldots, N$ **do**
  Set $t_i = t_{i-1} + h$.
  Calculate y_i using y_{i-1}, $f(t, y)$, t_{i-1} and t_i.
end for
return $\{t_i, y_i\}_{i=0}^{N}$.

Forward Euler method

The forward Euler method is one of the simplest methods to approximate an ODE numerically. In this method, we start with an expression for the derivative $y' = f(t, y(t))$ and a starting value $y(t_0) = y_0$. For a given step length h, we iterate this method by finding the next point $t_i = t_{i+1} + h$, and then find the next value y_{i+1} approximating y at t_{i+1} from the explicit expression

$$y_{i+1} = y_i + f(t_i, y_i)(t_{i+1} - t_i)$$

We then have the numerical approximation $y_{i+1} \simeq y(t_{i+1})$. The pseudo-code for this algorithm is shown in Alg. 2.

Algorithm 2 Forward Euler method

Require: $f(t,y), y_0, t_0, h, N$

for $i = 1, \ldots, N$ **do**
  Set $t_i = t_{i-1} + h$.
  Calculate $y_{i+1} = y_i + f(t_i, y_i)(t_i)(t_{i+1} - t_i)$
end for
return $\{t_i, y_i\}_{i=0}^{N}$.

Backward Euler method

The backward Euler method is very similar to the forward Euler method. It too is a method to approximate an ODE numerically. The main difference is that we use the derivative in the next point instead of the current point.

As with the forward method, we start with an expression for the derivative $y' = f(t, y(t))$ and a starting value $y(t_0) = y_0$. For a given step length h, we iterate this method by finding the next point $t_i = t_{i+1} + h$, and then find the next value y_{i+1} approximating y at t_{i+1} by solving the algebraic equation

$$y_{i+1} = y_i + f(t_{i+1}, y_{i+1})(t_{i+1} - t_i) \quad .$$

We then have the numerical approximation $y_{i+1} \simeq y(t_{i+1})$. The pseudo-code for this algorithm is shown in Alg. 3.

Algorithm 3 Backward Euler method

Require: $f(t,y), y_0, t_0, h, N$

for $i = 1, \ldots, N$ **do**
  Set $t_i = t_{i-1} + h$.
  Solve $y_{i+1} = y_i + f(t_{i+1}, y_{i+1})(t_{i+1} - t_i)$ for y_{i+1}.
end for
return $\{t_i, y_i\}_{i=0}^{N}$.

Heun's method

The Heun's method is a cross-breed between the forward and backward Euler methods. Instead of using the derivative in either the current or next step, it aims to obtain a derivative that is somewhat in between these to end-values.

As with the Euler methods, we start with an expression for the derivative $y' = f(t, y(t))$ and a starting value $y(t_0) = y_0$. For a given step length h, we iterate this method by first finding the next point $t_i = t_{i+1} + h$. Then we approximate the function value in the next step $\tilde{y}_{i+1}$ using Euler method:

$$\tilde{y}_{i+1} = y_i + f(t_i, y_i)(t_{i+1} - t_i) \quad .$$

With this approximation for the value in the next point, we can estimate the next point using a weighted average of the derivative in the current and next point:

$$y_{i+1} = y_i + \frac{f(t_i, y_i) + f(t_{i+1}, \tilde{y}_{i+1})}{2}(t_{i+1} - t_i) \quad .$$

We then have the numerical approximation $y_{i+1} \simeq y(t_{i+1})$. The pseudo-code for this algorithm is shown in Alg. 4.

Algorithm 4 Heun's method

Require: $f(t,y), y_0, t_0, h, N$

for $i = 1, \ldots, N$ **do**
  Set $t_i = t_{i-1} + h$.
  Calculate $\tilde{y}_{i+1} = y_i + f(t_i, y_i)(t_{i+1} - t_i)$
  Calculate $y_{i+1} = y_i + \frac{f(t_i,y_i)+f(t_{i+1},\tilde{y}_{i+1})}{2}(t_{i+1} - t_i)$
end for
return $\{t_i, y_i\}_{i=0}^{N}$.

Problem 1

Assume a reservoir filled with oil, with one water injector and one producer. For this exercise we assume the reservoir is so small that we can assume that it is always fully mixed, in the sense that we have the same saturation of oil and water everywhere. Let s_o be the oil saturation of the reservoir. Further, assume that we have the same constant rate of water injected into the reservoir as the produced rate, Q. In other words, both the injector and producer have the same rate Q. Finally, assume that he producer produce with a phase fraction s_o (a consequence of the phases in the reservoir being instantly mixed when water is injected. This also assumes straight relative permeability curves).

Let V_p be the total pore volume of the reservoir. Initially, we thus have a volume V_p of oil in the reservoir at time $t_0 = 0$ (which gives $s_o = 1$ at $t = 0$). However, the oil leaves the reservoir at a rate $Q_o = Qs_o$. We thus have the differential equation

$$V_p \frac{\partial s_o}{\partial t} = -Qs_o \quad . \tag{3}$$

For the tasks below, assume that the reservoir has a pore volume $V_p = 1 \times 10^3\,\mathrm{m}^3$, and that the wells inject and produce with a rate $Q = 1 \times 10^{-3}\,\mathrm{m}^3/\mathrm{s}$.

a) Reformulate Eq. (3) as a ordinary differential equation with initial condition of the form

$$y'(t) = f(t, y(t)) \tag{4}$$
$$y(t_0) = y_0 \quad . \tag{5}$$

Hint: Let $y = s_o$.

Solution:

If $y = s_o$, then we have the equation

$$y'(t) = -\frac{Q}{V_p} y \tag{6}$$
$$y(0) = 1 \quad . \tag{7}$$

Alternatively, this can be written as

$$\frac{ds_o}{dt} = -\frac{Q}{V_p} s_o(t) \tag{8}$$
$$s_o(0) = 1 \quad . \tag{9}$$

b) Create a Python code to solve the ODE equation using a forward Euler method.

Solution:

The explicit expression in the forward Euler method is

$$y_{i+1} = y_i + f(t_i, y_i)(t_{i+1} - t_i) \quad .$$

In our case, this gives

$$y_{i+1} = y_i - \frac{Q}{V_p} y_i \Delta t \tag{10}$$

$$= y_i \left(1 - \frac{Q}{V_p} \Delta t\right) \tag{11}$$

Using the variable values listed above, we can then create the following code to solve this ODE:

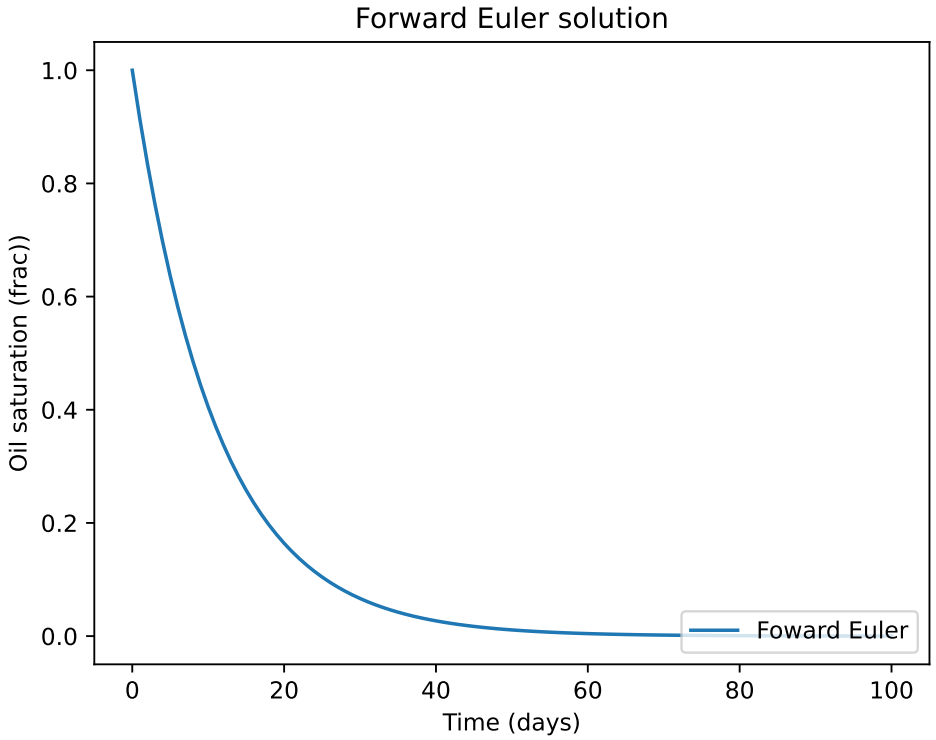
```
import numpy as np
import matplotlib.pyplot as plt

class EulerMethods:
  def __init__(self, fTt, iN, fQ, fVp):
    # Total time
    self.fTt = fTt
    #Number of time intervals
    self.iN = iN
    #Flow rate
    self.fQ = fQ
    #Pore volume
    self.fVp = fVp
    #Time interval
    self.fh = fTt / (iN - 1)
    #Time array
    self.aft = np.linspace(0.0, fTt, iN)

  def calc_forward_euler(self):
    self.afyf = np.zeros(self.iN)
    self.afyf[0] = 1.0
    for ii in range(1, self.iN):
      self.afyf[ii] = self.afyf[ii-1] * (1 - self.fQ * self.fh / self.fVp)

  def plot_forward_euler(self):
    plt.plot(self.aft/(60*60*24), self.afyf, label="Foward Euler")
    plt.xlabel("Time (days)")
    plt.ylabel("Oil saturation (frac))")
    plt.legend(loc='lower right')
    plt.title("Forward Euler solution")
    plt.savefig("forwardEuler.pdf")
    plt.clf()
```

This code can be run as follows:

```
fTt = 60.0 * 60 * 24 * 100
iN = 101
fQ = 1E-3
fVp = 1E3
fOilSat = 0.1
```

```
hEuler = EulerMethods(fTt, iN, fQ, fVp)
hEuler.calc_forward_euler()
hEuler.plot_forward_euler()
```

Running this code give the output shown in the Forward Euler solution figure below.



c) Set up the equation to be solved for the backward Euler method, and solve this equation. Use this solution to create a Python code to solve the ODE equation using a backward Euler method.

Solution:
The implicit expression in the forward Euler method is

$$y_{i+1} = y_i + f(t_{i+1}, y_{i+1})(t_{i+1} - t_i) \quad .$$

In our case, this gives

$$y_{i+1} = y_i - \frac{Q}{V_p} y_{i+1} \Delta t \tag{12}$$

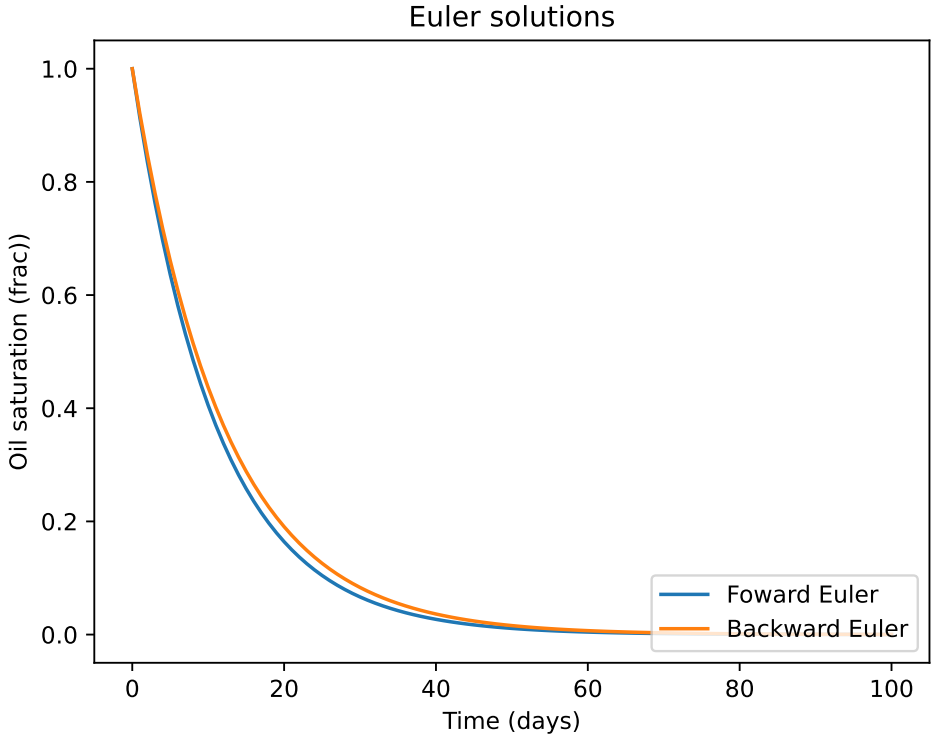
$$\left(1 + \frac{Q\Delta t}{V_p}\right) y_{i+1} = y_i \tag{13}$$

$$y_{i+1} = \frac{y_i}{1 + \frac{Q\Delta t}{V_p}} \tag{14}$$

We can then create the following code to solve this ODE:

```
def calc_backward_euler(self):
  self.afyb = np.zeros(iN)
  self.afyb[0] = 1.0
  for ii in range(1, self.iN):
    self.afyb[ii] = self.afyb[ii-1] / (1 + self.fQ * self.fh / self.fVp)

def plot_backward_euler(self):
  plt.plot(self.aft/(60*60*24), self.afyf, label="Foward Euler")
  plt.plot(self.aft/(60*60*24), self.afyb, label="Backward Euler")
  plt.xlabel("Time (days)")
  plt.ylabel("Oil saturation (frac))")
  plt.legend(loc='lower right')
  plt.title("Euler solutions")
  plt.savefig("backwardEuler.pdf")
```

Running this code give the output shown in the Forward Euler solution figure below.



d) Create a Python code to solve the ODE equation using Heun's method.

Solution:

The explicit expression in the forward Euler method gives us an approximation for the next point as

$$\tilde{y}_{i+1} = y_i + f(t_i, y_i)(t_{i+1} - t_i) \quad .$$

In our case, this is given as

$$\tilde{y}_{i+1} = y_i - \frac{Q}{V_p} y_i \Delta t \tag{15}$$

$$= y_i \left(1 - \frac{Q}{V_p} \Delta t\right) \tag{16}$$

We can estimate the next point from an average derivative as

$$y_{i+1} = y_i + \frac{f(y_i, t_i) + f(\tilde{y}_{i+1}, t_{i+1})}{2} (t_{i+1} - t_i) \tag{17}$$

$$= y_i - \frac{\frac{Q}{V_p} y_i + \frac{Q}{V_p} y_i \left(1 - \frac{Q}{V_p} \Delta t\right)}{2} \Delta t \tag{18}$$

$$= y_i - \frac{Q}{V_p} y_i \frac{2 - \frac{Q}{V_p} \Delta t}{2} \Delta t \tag{19}$$

$$= y_i \left(1 - \frac{Q \Delta t}{V_p} \frac{2 - \frac{Q \Delta t}{V_p}}{2}\right) \tag{20}$$

We can then create the following code to solve this ODE:

```
def calc_heun(self):
  self.afyh = np.zeros(iN)
  self.afyh[0] = 1.0
  for ii in range(1, self.iN):
    self.afyh[ii] = self.afyh[ii-1] * (1 - self.fQ * self.fh / self.fVp * ...
        (2 - self.fQ * self.fh / self.fVp) / 2)

def plot_heun(self):
  plt.plot(self.aft/(60*60*24), self.afyf, label="Foward Euler")
  plt.plot(self.aft/(60*60*24), self.afyb, label="Backward Euler")
  plt.plot(self.aft/(60*60*24), self.afyh, label="Heun's method")
  plt.xlabel("Time (days)")
  plt.ylabel("Oil saturation (frac))")
  plt.legend(loc='lower right')
  plt.title("Euler and Heun's solutions")
  plt.savefig("heun.pdf")
```

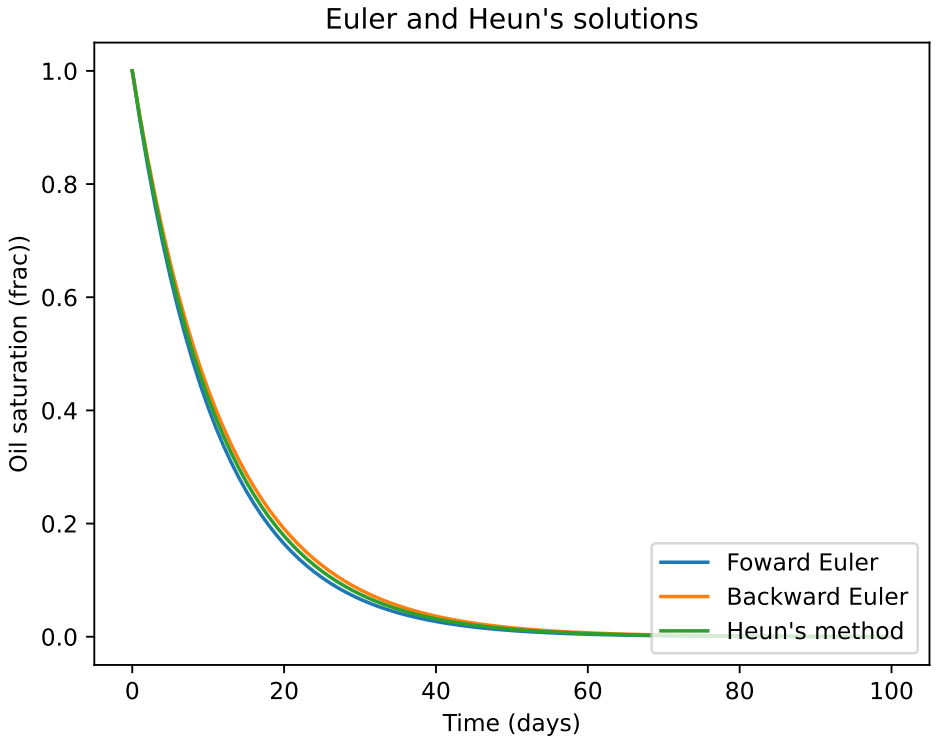
Running this code give the output shown below.

e) Estimate the time when the oil saturation in the reservoir has been reduced to 10% with the different methods, all using a step length of $h = 1\,\mathrm{d}$.

Solution:

We interpolate our arrays to find when the oil saturation becomes 10%. This is done by the following code:

```
def find_oil_saturation(self, fOilSat):
  # #np.interp expects a monotonically increasing
  # #curve, so we need to use the negative values.
  fWcf = np.interp(-fOilSat, -self.afyf, self.aft) / (60*60*24)
  fWcb = np.interp(-fOilSat, -self.afyb, self.aft) / (60*60*24)
  fWch = np.interp(-fOilSat, -self.afyh, self.aft) / (60*60*24)
  print(str(fOilSat)+' oil saturation after these days:', fWcf, fWcb, fWch)
```



The code produce the following result:

```
0.1 oil saturation after these days:  25.49294573059946 27.792604887132764
26.694964050718784
```

f) Estimate the time when the production reach a 90% water cut with the different methods, all using a step length of $h = 1$ d.

Solution:

This is a trick question. The same solutions as above.

g) Find the analytical solution to the ODE. From the analytical solution, find the exact oil saturation (according to the equations) after 100 days. Calculate the error for the three different methods when using three different step lengths of one second, one minute and one day.

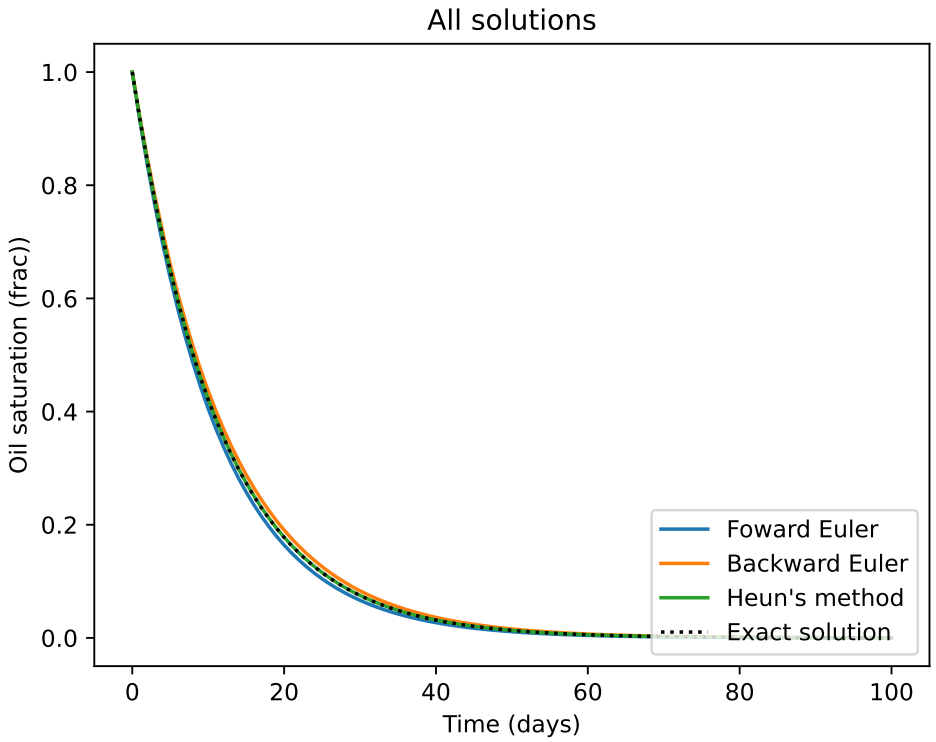
Solution:

We observe that the exact solution is $s_o = \exp -\frac{Qt}{V_p}$. *This can be calculated and plotted with the following code:*

```
def plot_all(self):
  plt.plot(self.aft/(60*60*24), self.afyf, label="Foward Euler")
  plt.plot(self.aft/(60*60*24), self.afyb, label="Backward Euler")
  plt.plot(self.aft/(60*60*24), self.afyh, label="Heun's method")
  plt.plot(self.aft/(60*60*24), self.exact_saturation(self.aft), ...
      label="Exact solution", color='k', linestyle='dotted')
  plt.xlabel("Time (days)")
  plt.ylabel("Oil saturation (frac))")
  plt.legend(loc='lower right')
```

```
    plt.title("All solutions")
    plt.savefig("all.pdf")
```

If we plot this against the other solutions we get the plot below.



By rearranging the equations we have

$$0.1 = \exp -\frac{Qt}{V_p} \tag{21}$$

$$\ln 0.1 = -\frac{Qt}{V_p} \tag{22}$$

$$t = -\frac{V_p \ln 0.1}{Q} \tag{23}$$

This can be calculated with the following code:

```
def exact_solution(self, fOilSat):
  return -self.fVp * np.log(fOilSat) / (self.fQ * 60 * 60 * 24)

def error_calculation(self, fOilSat):
  print(self.exact_solution(fOilSat))
  print('Errors:', np.abs(fWcf - self.exact_solution(fOilSat)),
      np.abs(fWcb - self.exact_solution(fOilSat)),
      np.abs(fWch - self.exact_solution(fOilSat)))
```

And it gives the following output:

```
Errors:  1.1573446975723698 1.1423144589609358 0.04467362254695573
```

Thus, we see that for this example, the forward method is the worst, the backward is better, and the Heun's method is the best.