1 Ordinary Differential Equations

Physical systems evolves in space and time, and very often they are described by a ordinary differential equations (ODE) and/or partial differential equations (PDE). The difference between an ODE and a PDE is that an ODE only describes the changes in one spatial dimension *or* time, whereas a PDE describes a system that evolves in the $x-$, $y-$, $z-$ dimension and/or in time. In the following we will spend a significant amount of time to explore one of the simplest algorithm, Eulers method. Sometimes this is exactly the algorithm you would like to use, but with very little extra effort much more sophisticated algorithms can easily be implemented, such as the Runge-Kutta fourth order method. However, all these algorithms, will at some point run into the same kind of troubles if used reckless. Thus we will use the Eulers method as a playground, investigate when the algorithm run into trouble and suggests ways to fix it, these approaches can easily be extended to the higher order methods. Most of the other algorithms boils down to the same idea of extrapolating a function using derivatives multiplied with a small step size.

2 A Simple Model for Fluid Flow

Let us consider a simple example from chemical engineering, a continuous stirred tank reactor (CSTR), see figure **??**. The flow is incompressible ($q_{\rm out} = q_{\rm in}$), a fluid is entering on the top and exiting at the bottom, the tank has a fixed volume V. Assume that the tank is filled with saltwater, and that freshwater is pumped into it, how much time does it take before 90% of the saltwater is replaced with freshwater? The tank is *well mixed*, illustrated with the propeller, this means that at every time the concentration is uniform in the tank, i.e. that $C(t) = C_{\rm out}(t)$.

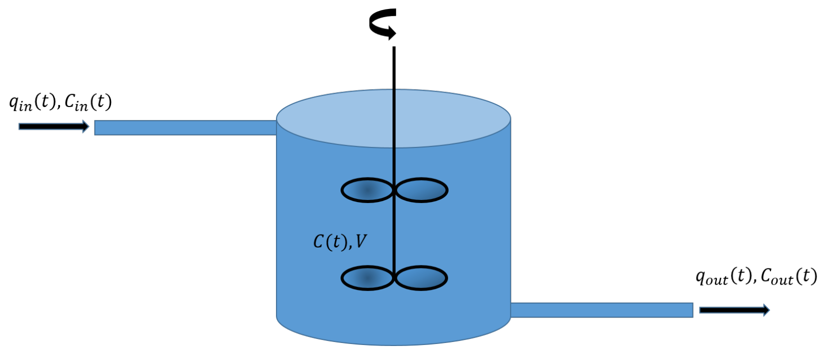

Figure 1: A continuous stirred tank model, $C(t) = C_{\rm out}(t)$, and $q_{\rm out} = q_{\rm in}$.

The concentration C is measured in gram of salt per liter water, and the flow rate q is liter of water per day. The model for the salt balance in this system can be described in words by:

$$\begin{aligned}[\text{accumulation of salt}] = [\text{salt into the system}] - [\text{salt out of the system}] \\ + [\text{generation of salt}]. \end{aligned} \tag{1}$$

In our case there are no generation of salt within the system so this term is zero. The flow of salt into the system during a time Δt is: $q_{\text{in}}(t) \cdot C_{\text{in}}(t) \cdot \Delta t = q(t) \cdot C_{\text{in}}(t) \cdot \Delta t$, the flow of salt out of the system is: $q_{\text{out}}(t) \cdot C_{\text{out}}(t) \cdot \Delta t = q(t) \cdot C(t) \cdot \Delta t$, and the accumulation during a time step is: $C(t + \Delta t) \cdot V - C(t) \cdot V$, hence:

$$C(t + \Delta t) \cdot V - C(t) \cdot V = q(t) \cdot C_{\text{in}}(t) \cdot \Delta t - q(t) \cdot C(t) \cdot \Delta t. \tag{2}$$

Note that it is not a priori apparent, which time the concentrations and flow rates on the right hand side should be evaluated at, we could have chosen to evaluate them at $t + \Delta t$, or at any time $t \in [t, t + \Delta t]$. We will return to this point later in this chapter. Dividing by Δt, and taking the limit $\Delta t \to 0$, we can write equation (**??**) as:

$$V \frac{dC(t)}{dt} = q(t) \left[C_{\text{in}}(t) - C(t) \right]. \tag{3}$$

Seawater contains about 35 gram salt/liter fluid, if we assume that the fresh water contains no salt, we have the boundary conditions $C_{\text{in}}(t) = 0$, $C(0) =$35gram/l. The equation (**??**) the reduces to:

$$V \frac{dC(t)}{dt} = -qC(t), \tag{4}$$

this equation can easily be solved, by dividing by C, multiplying by dt and integrating:

$$\begin{aligned} V \int_{C_0}^{C} \frac{dC}{C} &= -q \int_0^t dt, \\ C(t) &= C_0 e^{-t/\tau}, \text{ where } \tau \equiv \frac{V}{q}. \end{aligned} \tag{5}$$

This equation can be inverted to give $t = -\tau \ln[C(t)/C]$. If we assume that the volume of the tank is 1m^3=1000liters, and that the flow rate is 1 liter/min, we find that τ=1000min=0.69days and that it takes about $-0.69 \ln 0.9 \simeq 1.6$days to reduce the concentration by 90% to 3.5 gram/liter.

The CSTR

You might think that the CSTR is a very simple model, but this type of model is the basic building blocks in chemical engineering. By putting CSTR tanks in series and/or connecting them with pipes, the efficiency of manufacturing various type of chemicals can be investigated. Altough the CSTR is an idealized model for the part of a chemical factory, it is actually a *very good* model for fluid flow in a porous media. By connecting CSTR tanks in series, one can model how chemical tracers propagate in the subsurface. The physical reason for this is that dispersion in porous media will play the role of the propellers and mix the concentration uniformly.

3 Eulers Method

In a more complicated case, several tanks in series, with varying degree of flow rate or if salt was generated in the tank, it might be very hard to solve this equation analytically. Actually we already have developed a numerical algorithm to solve equation (**??**), before we arrived at equation (**??**) in equation (**??**). This is a special case of Eulers method, which is basically to replace the derivative in equation (**??**), with $(C(t+\Delta t)-C(t))/\Delta t$. By rewriting equation (**??**), so that we keep everything related to the new time step, $t+\Delta t$, on one side, we get:

$$VC(t+\Delta t) = VC(t) + qC_{\text{in}}(t) - qC(t) \tag{6}$$

$$,C(t+\Delta t) = C(t) + \frac{\Delta t}{\tau}\left[C_{\text{in}}(t) - C(t)\right], \tag{7}$$

we introduce the short hand notation: $C(t) = C_n$, and $C(t+\Delta t) = C_{n+1}$, hence the algorithm can be written more compact as:

$$C_{n+1} = \left(1 - \frac{\Delta t}{\tau}\right) C_n + \frac{\Delta t}{\tau} C_{\text{in},n}, \tag{8}$$

In the script below, we have implemented equation (**??**).

```python
def analytical(x):
    return np.exp(-x)

def euler_step(c_old, c_in, tau_inv,dt):
    fact=dt*tau_inv
    return (1-fact)*c_old+fact*c_in

def ode_solv(c_into,c_init,t_final,vol,q,dt):
    f=[];t=[]
    tau_inv = q/vol
    c_in    = c_into #freshwater into tank
    c_old   = c_init #seawater present
    ti=0.
    while(ti <= t_final):
        t.append(ti); f.append(c_old)
        c_new = euler_step(c_old,c_in,tau_inv,dt)
        c_old = c_new
        ti   += dt
    return t,f
```

In figure **??** the result of the implementation is shown. Clearly we see that the results are dependent on the step size, as the step increases the numerical

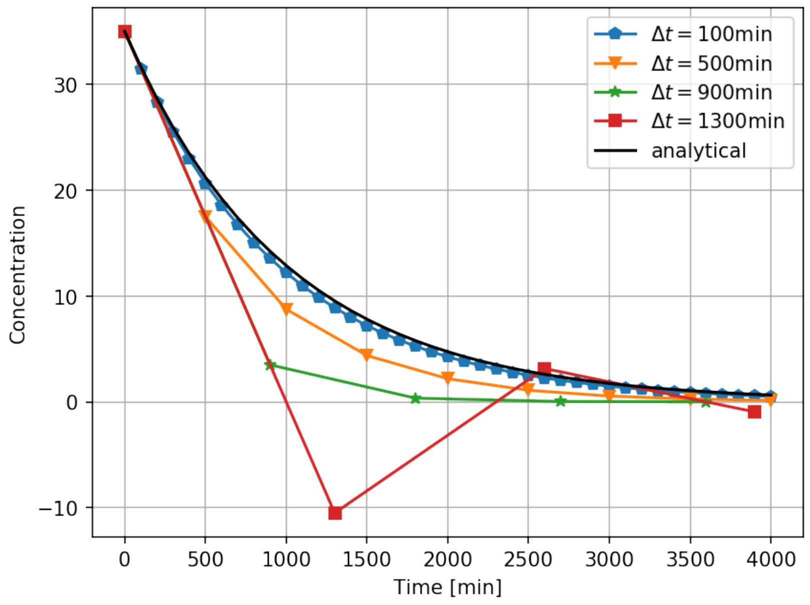

Figure 2: The concentration in the tank for different step size Δt.

solution deviates from the analytical solution. At some point the numerical algorithm fails completely, and produces results that have no meaning.

3.1 Error Analysis - Eulers Method

There are two obvious questions:

1. when does the algorithm produce unphysical results?
2. what is an appropriate step size?

Let us consider question **??** first, clearly when the concentrations gets negative the solution is unphysical. From equation (**??**), we see that when $\Delta t/\tau > 1$, there is a chance that concentrations become negative. For this specific case (the CSTR), there is a clear physical interpretation of this condition, inserting $\tau = V/q$, we can rewrite the condition $\Delta t/\tau < 1$ as $q\Delta t < V$. The volume into the tank during one time step is: $q\Delta t$, which means that whenever we *flush more than one tank volume through the tank during one time step, the algorithm fails.* When this happens the new concentration in the tank cannot be predicted from the old one, which makes sense because we could have flooded a completely different solution into the tank.

The second question, "what is an appropriate step size?", is a bit more difficult to answer. One strategy could be to simply use the results from chapter [Taylor], in that chapter we showed that by choosing a step size of 10^{-8} the truncation error has a minimum value (when using a first order Taylor approximation). How does the value 10^{-8} relate to a step size of 1min? In order to see the connection with equation (**??**), we need to the equation in a dimensionless form, by making the following substitution: $t \to t/\tau$:

$$\frac{dC(\tau)}{d\tau} = \left[C_{\mathrm{in}}(\tau) - C(\tau)\right]. \tag{9}$$

As we found earlier $\tau = 1000$min, thus a step size of 1min would correspond to a dimensionless time step of $\Delta t \to$1min/1000min$= 10^{-3}$. This number can be directly compared to the value 10^{-8}, which is the lowest value we can choose without getting into trouble with round off errors on the machine.

Dimensionless variables

It is a good idea (necessary) to formulate our equations in terms of dimensionless variables. The algorithms we develop can then be used in the same form if the system size changes, because the same step can be used regardless of the volume or the flow rate through the tank. Thus we do not need to rewrite the algorithm each time the physical system changes. This also means that if you use an algorithm developed by someone else (e.g. in Matlab or Python), you should always formulate the ODE system in dimensionless form before using the algorithm.

A second reason is that from a pure modeling point of view, dimensionless variables is a way of getting some understanding of what kind of combination of the physical parameters that describes the behavior of the system. For the case of the CSTR, there is a time scale $\tau = V/q$, which is an intrinsic measure of time in the system. No matter what the flow rate through the tank or the volume of the tank is, it will always take 0.1τ before the concentration in the tank is reduced by 90%.

As already mentioned a step size of 10^{-8}, is probably the smallest we can choose with respect to round off errors, but it is smaller than necessary and would lead to large simulation times. If it takes 1 second to run the simulation with a step size of 10^{-3}, it would take 10^5 seconds or 1 day with a step size of 10^{-8}. To continue the error analyses, we write our ODE for a general system as:

$$\frac{dy}{dt} = f(y,t), \tag{10}$$

or in discrete form:

$$\frac{y_{n+1} - y_n}{h} - \frac{h}{2}y''(\eta_n) = f(y,t).$$

$$y_{n+1} = y_n + hf(y,t) + \frac{h^2}{2}y''(\eta_n). \tag{11}$$

h is now the step size, equal to Δt if the derivative is with respect to t or Δx if the derivative is respect to x etc. Note that we have also included the error term related to the numerical derivative, $\eta_n \in [t_n, t_n + h]$. At each step we get an error term, and the distance between the true solution and our estimate, the *local error*, after N steps is:

$$\begin{aligned}\epsilon = \sum_{n=0}^{N-1} \frac{h^2}{2} y''(\eta_n) = \frac{h^2}{2} \sum_{n=0}^{N-1} f'(y_n, \eta_n) \simeq \frac{h}{2} \int_{t_0}^{t_f} f'(y, \eta) d\eta \\ = \frac{h}{2} \left[f(y(t_f), t_f) - f(y(t_0), t_0) \right]. \end{aligned} \tag{12}$$

Note that when we replace the sum with an integral in the equation above, this is only correct if the step size is not too large. From equation (**??**) we see that even if the error term on the numerical derivative is h^2, the error term on the solution is proportional to h (one order lower). This is because we accumulate errors for each step.

In the following we specialize to the CSTR, to see if we can gain some additional insight. First we change variables in equation (**??**): $y = C(t)/C_0$, and $x = t/\tau$, hence:

$$\frac{dy}{dx} = -y. \tag{13}$$

The solution to this equation is $y(x) = e^{-x}$, substituting back for the new variables y and x, we reproduce the result in equation (**??**). The total error, equation (**??**), for this case is

$$\epsilon = \frac{h}{2} \left[-y(x_f) + y(x_0) \right] = \frac{h}{2} \left[1 - e^{-x_f} \right], \tag{14}$$

we have assumed that $x_0 = t_0/\tau = 0$. This gives the estimated global error at time x_f. For $x_f = 0$, the numerical error is zero, this makes sense because at $x = 0$ we know the exact solution because of the initial conditions. When we move further away from the initial conditions, the numerical error increases, but equation (**??**) ensures us that as long as the step size is low enough we can get as close as possible to the true solution ,since the error scales as h (at some point we might run into trouble with round off error in the computer).

Can we prove directly that we get the analytical result? In this case it is fairly simple, if we use Eulers method on equation (**??**), we get:

$$\begin{aligned}\frac{y_{n+1} - y_n}{h} = -y_n f. \\ y_{n+1} = (1 - h) y_n, \end{aligned} \tag{15}$$

or alternatively:

$$\begin{aligned} y_1 &= (1-h)y_0, \\ y_2 &= (1-h)y_1 = (1-h)^2 y_0, \\ &\vdots \\ y_{N+1} &= (1-h)^N y_0 = (1-h)^{x_f/h} y_0. \end{aligned} \tag{16}$$

In the last equation, we have used the the fact the number of steps, N, is equal to the simulation time divided by the step size, hence: $N = x_f/h$. From calculus, the equation above is one of the well known limits for the exponential function: $\lim_{x\to\infty}(1+k/x)^{mx} = e^{mk}$, hence:

$$y_n = (1-h)^{x_f/h} y_0 \to e^{-x_f}, \tag{17}$$

when $h \to 0$. Below is an implementation of the Euler algorithm in this simple case, we also estimate the local error, and global error after N steps.

```python
import matplotlib.pyplot as plt
import numpy as np
def euler(tf,h):
    t=[];f=[]
    ti=0.;fi=1.
    t.append(ti);f.append(fi)
    global_err=0.
    while(ti<= tf):
        ti+=h
        fi=fi*(1-h)
        global_err += abs(np.exp(-ti)-fi)
        t.append(ti);f.append(fi)
    print("error= ", np.exp(-ti)-fi," est.err=", .5*h*(1-np.exp(-ti)))
    print("global error=",global_err)
    return t,f

t,f=euler(1.,1e-5)
```

By changing the step size h, you can easily verify that the local error systematically increases or decreases proportional to h. Something curious happens with the global error when the step size is changed, it does not change very much. The global error involves a second sum over all the local error, which can be approximated as a second integration in equation (**??**):

$$\epsilon_{\text{global}} = \frac{1}{2}\int_0^{x_f} \left[-y(x) + y(0)\right] dx = \frac{1}{2}\left[x_f + e^{-x_f} - 1\right]. \tag{18}$$

Note that the global error does not go to zero when the step size decreases, which can easily be verified by changing the step size. This is strange, but can be understood by the following argument: when the step size decreases the local error scales as $\sim h$, but the number of steps scales as $1/h$, so the global error must scale as $h \times 1/h$ or some constant value. Usually it is much easier to control the local error than the global error, this should be kept in mind if you ever encounter a problem where it is important control the global error. For the higher order methods that we will discuss later in this chapter, the global error will go to zero when h decreases.

The answer to our original question, what is an appropriate step size?, will depend on what you want to achieve, but in most practical situations you would specify a local error that is acceptable for the problem under investigation and then choose a step size where the local error always is lower than this value. In the next subsection we will investigate how to achieve this in practice.

3.2 Adaptive step size - Eulers Method

We want to be sure that we use a step size that achieves a certain accuracy in our numerical solution, but at the same time do not waste simulation time using a too low step size. The following approach is similar to the one we derived for the Romberg integration, and a special case of what is known as Richardson Extrapolation. The method is easily extended to higher order methods.

We start by using Eulers method to get from y_0 to y_1 by taking one (large) step, h:

$$y_1^* = y_0 + hf(y_0, t_0), \tag{19}$$

where we have used a * to indicate that we have used a step size of size h. We know that Eulers algorithm is accurate to second order. Our estimate of the new value, y_1^* should (if the step size is not too large) then be related to the true solution $y(t_1)$ in the following way:

$$y(t_1) = y_1^* + ch^2. \tag{20}$$

The constant c is unknown, but it can be found by taking two smaller steps of size $h/2$. Again, if the steps are not too large, our new estimate of the value y_1 will be related to the true solution as:

$$y(t_1) = y_1 + 2c\left(\frac{h}{2}\right)^2. \tag{21}$$

The factor 2 in front of c is because we now need to take two steps, and we accumulate a total error of $2c(h/2)^2 = ch^2/2$. It might not be completely obvious that the constant c should be the same in equation (**??**) and (**??**). If you are not convinced, there is an exercise at the end of the chapter. The term ch^2 is used to represent the truncation error, the constant c is in principle unknown, but can be estimated by taking two steps of size $h/2$:

$$\Delta \equiv y_1^* - y_1 = c\frac{h^2}{2}. \tag{22}$$

Now we have everything we need: We want the local error to be smaller than some tolerance, ε, or equivalently that $\Delta \leq \varepsilon$. To achieve this we need an optimal step size, h', that gives us exactly the desired error:

$$\varepsilon = c\frac{h'^2}{2}. \tag{23}$$

$$h' = h\sqrt{\left|\frac{\varepsilon}{\Delta}\right|}. \tag{24}$$

Equation (**??**) serves two purposes, if the estimated error Δ is higher than the tolerance, ε, we have specified it will give us an estimate for the step size we should choose in order to achieve a higher accuracy, if on the other hand $\Delta > \varepsilon$, then we get an estimate for the next, larger step. Before the implementation we note, as we did for the Romberg integration, that equation (**??**) also gives us an estimate for the error term that we can add to equation (**??**) as an improved estimate of y_1. This we get for free and will make our Euler algorithm accurate to h^3, hence the improved Euler step, $\hat{y_1}$, will now be improved if we *subtract* the error term from our previous estimate:

$$\hat{y_1} = y_1 - \Delta = 2y_1 - y_1^*. \tag{25}$$

Below is an implementation of the adaptive Euler algorithm:

```python
def one_step(c_old, c_in,h):
    return (1-h)*c_old+h*c_in

def adaptive_euler(c_into,c_init,t_final,tol=1e-4):
    f=[];t=[]
    c_in    = c_into #freshwater into tank
    c_old   = c_init #seawater present
    ti=0.; h_new=1e-3;
    toli=1.; # a high init tolerance to enter while loop
    no_steps=0
    global_err=0.
    while(ti <= t_final):
        t.append(ti); f.append(c_old)
        while(toli>tol):# first two small steps
            hi=h_new
            k1 = one_step(c_old,c_in,hi*.5)
            k2 = one_step(k1,c_in,hi*.5)
            # ... and one large step
            k3 = one_step(c_old,c_in,hi)
            toli = abs(k3-k2)
            h_new=hi*np.sqrt(tol/toli)
            no_steps+=3
        toli=1.
        c_old=2*k2-k3 # higher order correction
 # normal Euler, uncomment and inspect the global error
#        c_old = k2
        ti   += hi
        global_err += abs(np.exp(-ti)-c_old)
    print("No steps=", no_steps, "Global Error=", global_err)
    return t,f
```

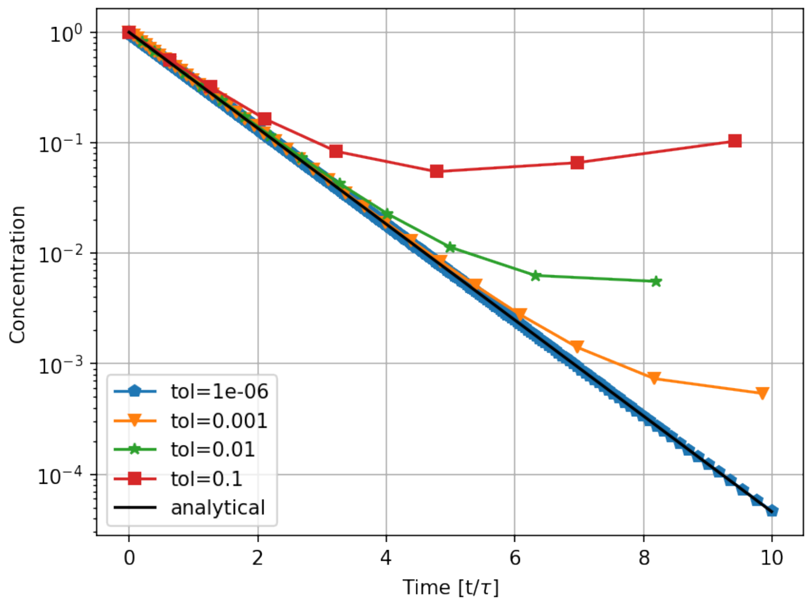

Figure 3: The concentration in the tank using adaptive Euler. Number of Euler steps are: 3006, 117, 48 and 36 for the different step sizes.

In figure **??** the result of the implementation is shown. Note that the number of steps for an accuracy of 10^{-6} is only about 3000. Without knowing anything about the accuracy, we would have to assume that we needed a step size of the order of h in order to reach a local accuracy of h because of equation (**??**). In the current case, we would have needed 10^7 steps, which would lead to unnecessary long simulation times.

4 Runge-Kutta Methods

The Euler method only have an accuracy of order h, and a global that do not go to zero as the step size decrease. As indicated the The Runge-Kutta methods may be motivated by inspecting the Euler method in figure **??**. The Euler method uses information from the prevoous time step to estimate the value at the new time step. The Runge Kutta methods uses the information about the slope between the points t_n and $t_n + h$. By inspecting figure **??**, we clearly see that by using the slope at $t_n + h/2$ would give us a significant improvement. The

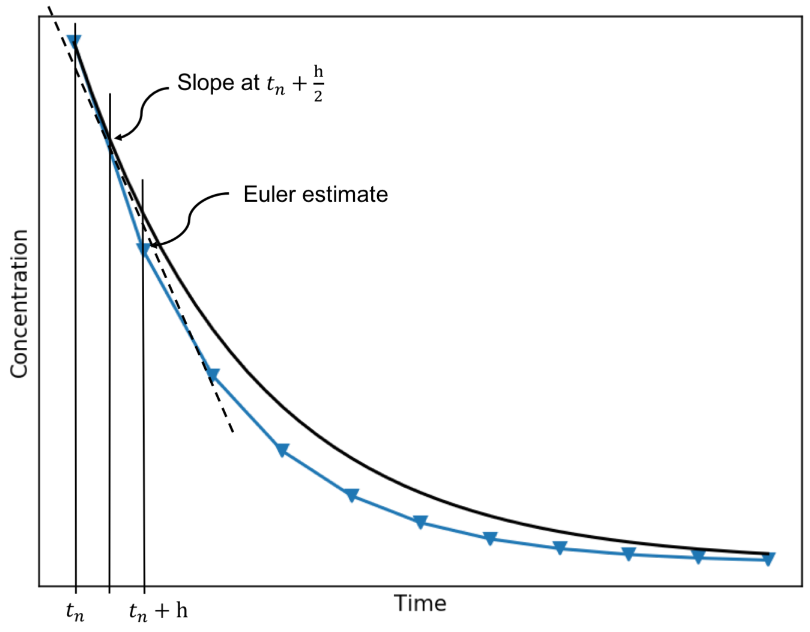

Figure 4: Illustration of the Euler algorithm, and a motivation for using the slope a distance from the t_n.

2. order Runge-Kutta method can be derived by Taylor expanding the solution around $t_n + h/2$, we dot this by setting $t_n + h = t_n + h/2 + h/2$:

$$y(t_n + h) = y(t_n + \frac{h}{2}) + \frac{h}{2} \left.\frac{dy}{dt}\right|_{t=t_n+h/2} + \frac{h^2}{4} \left.\frac{d^2y}{dt^2}\right|_{t=t_n+h/2} + \mathcal{O}(h^3). \tag{26}$$

Similarly we can expand the solution in $y(t_n)$ about $t_n + h/2$, by setting $t_n = t_n + h/2 - h/2$:

$$y(t_n) = y(t_n + \frac{h}{2}) - \frac{h}{2} \left.\frac{dy}{dt}\right|_{t=t_n+h/2} + \frac{h^2}{4} \left.\frac{d^2y}{dt^2}\right|_{t=t_n+h/2} - \mathcal{O}(h^3). \tag{27}$$

Substracting these two equations the term $y(t_n + \frac{h}{2})$, and all even powers in the derivative cancels out:

$$\begin{aligned} y(t_n + h) &= y(t_n) + h \left.\frac{dy}{dt}\right|_{t=t_n+h/2} + \mathcal{O}(h^3), \\ y(t_n + h) &= y(t_n) + hf(y_{n+h/2}, t_n + h/2) + \mathcal{O}(h^3). \end{aligned} \tag{28}$$

In the last equation, we have used equation (**??**). Note that we now have an expression that is very similar to Eulers algorithm, but it is accurate to order h^3. There is one problem, and that is that the function f is to be evaluated at the point $y_{n+1/2} = y(t_n + h/2)$ which we do not know. This can be fixed by using Eulers algorithm: $y_{n+1/2} = y_n + h/2f(y_n, t_n)$. We can do this even if Eulers algorithm is only accurate to order h^2, because the f in equation (**??**) is multiplied by h, and thus our algorithm is still accurate up to order h^3.

The 2. order Runge-Kutta:

$$\begin{aligned} k_1 &= hf(y_n, t_n) \\ k_2 &= hf(y_n + \frac{1}{2}k_1, t_n + h/2) \\ y_{n+1} &= y_n + k_2 \end{aligned} \tag{29}$$

```
def fm(c_old,c_in):
    return c_in-c_old

def rk2_step(c_old, c_in, h):
    k1=h*fm(c_old,c_in)
    k2=h*fm(c_old+0.5*k1,c_in)
    return c_old+k2

def ode_solv(c_into,c_init,t_final,h):
    f=[];t=[]
    c_in  = c_into #freshwater into tank
    c_old = c_init #seawater present
    ti=0.
    while(ti <= t_final):
        t.append(ti); f.append(c_old)
        c_new = rk2_step(c_old,c_in,h)
        c_old = c_new
        ti   += h
    return t,f
```

In figure **??** the result of the implementation is shown. Note that when comparing Runge-Kutta 2. order with Eulers method, see figure **??** and **??**, we of course have the obvious result that a larger step size can be taken, without loosing numerical accuracy. It is also worth noting that we can take steps that is larger than the tank volume. Eulers method failed whenever the time step was larger than one tank volume ($h = t/\tau > 1$), whereas the Runge-Kutta method finds a physical solution for step sizes lower than twice the tank volume. If the step size is larger, we see that the concentration in the tank increases, which is clearly unphysical.

The Runge-Kutta fourth order method is one of he most used methods, it is accurate to order h^4, and has an error of order h^5. The development of the

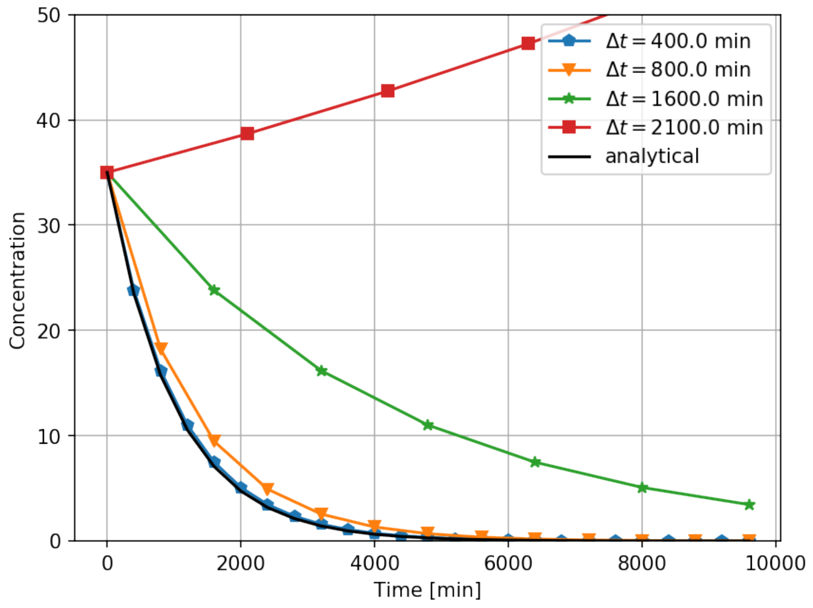

Figure 5: The concentration in the tank for different step size Δt.

algorithm itself is similar to the 2. order method, but of course more involved. We just quote the result:

The 4. order Runge-Kutta:

$$
\begin{aligned}
k_1 &= hf(y_n, t_n)\\
k_2 &= hf(y_n + \frac{1}{2}k_1, t_n + h/2)\\
k_3 &= hf(y_n + \frac{1}{2}k_2, t_n + h/2)\\
k_4 &= hf(y_n + k_3, t_n + h)\\
y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad (30)
\end{aligned}
$$

```
def fm(c_old,c_in):
    return c_in-c_old

def rk4_step(c_old, c_in, h):
    k1=h*fm(c_old,c_in)
    k2=h*fm(c_old+0.5*k1,c_in)
    k3=h*fm(c_old+0.5*k2,c_in)
    k4=h*fm(c_old+    k3,c_in)
    return c_old+(k1+2*k2+2*k3+k4)/6

def ode_solv(c_into,c_init,t_final,h):
    f=[];t=[]
    c_in  = c_into #freshwater into tank
    c_old = c_init #seawater present
    ti=0.
    while(ti <= t_final):
        t.append(ti); f.append(c_old)
        c_new = rk4_step(c_old,c_in,h)
        c_old = c_new
        ti   += h
    return t,f
```

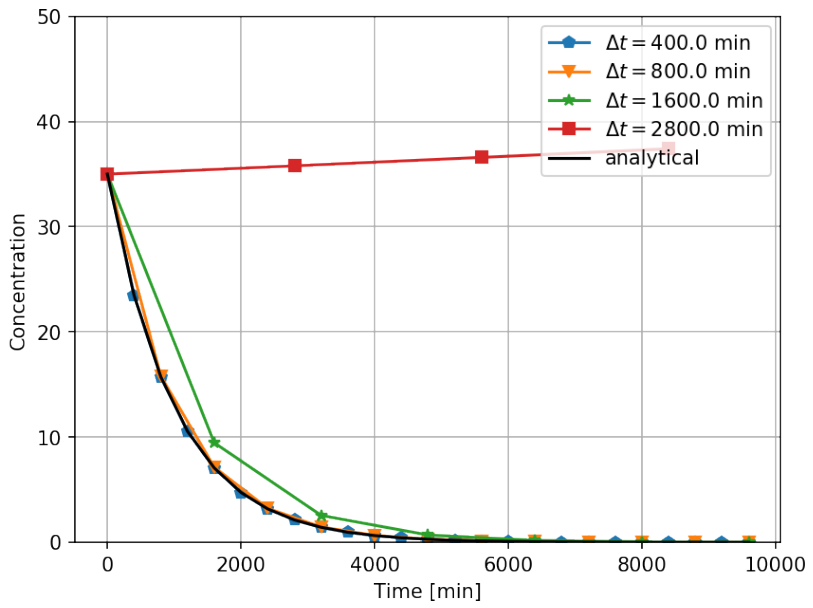

Figure 6: The concentration in the tank for different step size Δt.

In figure **??** the result of the implementation is shown.

4.1 Adaptive step size - Runge-Kutta Method

Just as we did with Eulers method, we can implement an adaptive method. The derivation is exactly the same, but this time our method is accurate to fourth order, hence the error term is of order h^5. We start by taking one large step of size h:

$$y_1^* = y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + ch^5, \tag{31}$$

where $k_1, \ldots k_4$ is given in equation (**??**). Next, we take two small steps. As with Eulers method, we then have to insert our estimate for $y_{1/2}$ into the expression for f. Then the f after two half steps will contain terms proportional to f inside the argument, but we can safely remove those terms because they have a factor of h in front and the net effect is that they are of the order h^6. Thus we are sure that the constant c in front of the error term is the same whether we take a step of size h or two steps of size $h/2$, hence:

$$y_1 = y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + 2c\left(\frac{h}{2}\right)^5. \tag{32}$$

Subtracting equation (**??**) and (**??**), we find an expression similar to equation (**??**):

$$\Delta \equiv y_1^* - y_1 = c\frac{15}{16}h^5, \tag{33}$$

or $c = 16\Delta/(15h^5)$. For the Euler scheme, Δ also happened to be equal to the truncation error, but in this case it is:

$$\varepsilon = 2c\left(\frac{h}{2}\right)^5 = \frac{\Delta}{15} \tag{34}$$

we want the local error, ε, to be smaller than some tolerance, ε_0. The optimal step size, h', that gives us exactly the desired error is then:

$$\varepsilon_0 = 2c\left(\frac{h'}{2}\right)^5. \tag{35}$$

Dividing equation (**??**) by equation (**??**), we can estimate the optimal step size:

$$h' = h\left|\frac{\varepsilon_0}{\varepsilon}\right|^{1/5}, \tag{36}$$

where the expression for ε can be found in equation (**??**). Below is an implementation

```
def fm(c_old,c_in):
    return c_in-c_old

def rk4_step(c_old, c_in, h):
    k1=h*fm(c_old,c_in)
    k2=h*fm(c_old+0.5*k1,c_in)
    k3=h*fm(c_old+0.5*k2,c_in)
    k4=h*fm(c_old+    k3,c_in)
    return c_old+(k1+2*k2+2*k3+k4)/6

def adaptive_ode_solv(c_into,c_init,t_final,tol=1e-4):
    f=[];t=[]
    tau_inv = q/vol
    c_in    = c_into #freshwater into tank
    c_old   = c_init #seawater present
    ti=0.; h_new=1;
    toli=1.; # a high init tolerance to enter while loop
    no_steps=0
    global_err=0.
    while(ti <= t_final):
        t.append(ti); f.append(c_old)
        while(toli>tol):# first two small steps
            hi=h_new
            k1 = rk4_step(c_old,c_in,hi*.5)
            k2 = rk4_step(k1,c_in,hi*.5)
            # ... and one large step
            k3 = rk4_step(c_old,c_in,hi)
            toli = abs(k3-k2)/15
            h_new=min(hi*(tol/toli)**(0.2),1)
            no_steps+=3
        toli=1.
        c_old=k2-(k3-k2)/30
        ti   += hi
        global_err += abs(np.exp(-ti)-c_old)
    print("No steps=", no_steps, "Global Error=", global_err)
    return t,f
```

In figure **??** the result of the implementation is shown. Note that we put a safety limit on the step size 'min(hi*(tol/toli)**(0.2),1)'. In general we can use the same procedure any method of order h^p, and you can easily verify that:

Error term and step size for h^p:

$$\epsilon = \frac{|\Delta|}{2^p - 1} = \frac{|y_1^* - y_1|}{2^p - 1}, \tag{37}$$

$$h' = \beta h \left|\frac{\varepsilon}{\varepsilon_0}\right|^{\frac{1}{p+1}}, \tag{38}$$

where β is a safety factor $\beta \simeq 0.8, 0.9$, and you should always be careful that the step size do not become too large so that the method breaks down. This can happens when ε is very low, which may happen if $y_1^* \simeq y_1$ and if $y_1^* \simeq y_1 \simeq 0$.

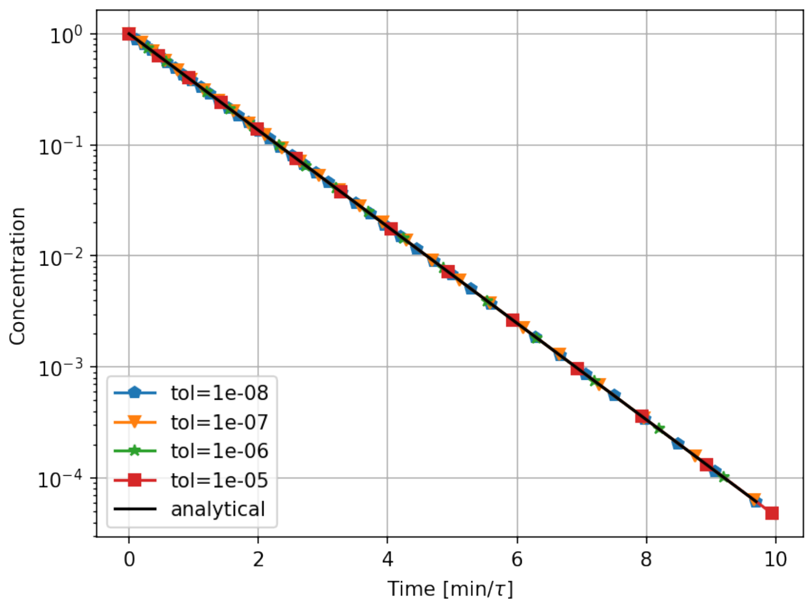

Figure 7: The concentration in the tank for different step size Δt. Number of rk4 steps are: 138, 99, 72 and 66 for the different step sizes.

4.2 Conservation of Mass

A mathematical model of a physical system should always be formulated in such a way that it is consistent with the laws of nature. In practical situations this statement is usually equivalent to state that the mathematical model should respect conservation laws. The conservation laws can be conservation of mass, energy, momentum, electrical charge, etc. In our example with the mixing tank, we were able to derive an expression for the concentration of salt out of the tank, equation (**??**), by *demanding* conservation of mass (see equation (**??**)).

A natural question to ask is then: If our mathematical model respect conservation of mass, are we sure that our solution method respect conservation of mass? We of course expect that when the grid spacing approaches zero our numerical solution will get closer and closer to the analytical solution. Clearly when $\Delta x \to 0$, the mass is conserved. So what is the problem? The problem is that in many practical problems we cannot always have a mesh that is small enough to ensure that our solution always is close enough to the analytical solution. The physical system we consider might be very complicated (e.g. a model for the earth climate), and our ODE system could be a very small part of a very big system. A very good test of any code is to investigate if the code respect the conservation laws. If we know that our implementation respect e.g. mass conservation at the discrete level, we can easily test mass conservation by summing up all the mass entering, and subtracting the mass out of and present in our system. If the mass is not conserved exactly, there is a good chance that there is a bug in our implementation.

If we now turn to our system, we know that the total amount of salt in the system when we start is $C(0)V$. The amount entering is zero, and the amount leaving each time step is $q(t)C(t)\Delta t$. Thus we should expect that if we add the amount of salt in the tank to the amount that has left the system we should always get an amount that is equal to the original amount. Alternatively, we expect $\int_{t_0}^{t} qC(t)dt + C(t)V - C(0)V = 0$. Adding the following code in the `while(ti <= t_final):` loop:

```
mout += 0.5*(c_old+c_new)*q*dt
mbal  = (c_new*vol+mout-vol*c_init)/(vol*c_init)
```

it is possible to calculate the amount of mass lost (note that we have used the trapezoidal formula to calculate the integral). In the table below the fraction of mass lost relative to the original amount is shown for the various numerical methods.

Δt	h	Euler	RK 2. order	RK 4. order
900	0.9	-0.4500	0.3682	0.0776
500	0.5	-0.2500	0.0833	0.0215
100	0.1	-0.0500	0.0026	0.0008
10	0.01	-0.0050	2.5E-05	8.3E-06

We clearly see from the table that the Runge-Kutta methods performs better than Eulers method, but *all of the methods violates mass balance.*

This might not be a surprise as we know that our numerical solution is always an approximation to the analytical solution. How can we then formulate an algorithm that will respect conservation laws at the discrete level? It turns out that for Eulers method it is not so difficult. Eulers algorithm at the discrete level (see equation (**??**)) is actually a two-step process: first we inject the fresh water while we remove the "old" fluid *and then we mix.* By thinking about the problem this way, it makes more sense to calculate the mass out of the tank as $\sum_k q_k C_k \Delta t_k$. If we in our implementation calculates the mass out of the tank as:

```
mout += c_old*q*dt
mbal  = (c_new*vol+mout-vol*c_init)/(vol*c_init)
```

We easily find that the mass is exactly conserved at every time for Eulers method. The concentration in the tank will of course not be any closer to the analytical solution, but if our mixing tank was part of a much bigger system we could make sure that the mass would always be conserved if we make sure that the mass out of the tank and into the next part of the system was equal to $qC(t)\Delta t$.

5 Solving a set of ODE equations

What happens if we have more than one equation that needs to be solved? If we continue with our current example, we might be interested in what would happen if we had multiple tanks in series. This could be a very simple model to describe the cleaning of a salty lake by injecting fresh water into it, but at the same time this lake was connected to two nearby fresh water lakes, as illustrated in figure **??**. The weakest part of the model is the assumption about complete mixing, in a practical situation we could enforce complete mixing with the salty water in the first tank by injecting fresh water at multiple point in the lake. For the two next lakes, the degree of mixing is not obvious, but salt water is heavier than fresh water and therefore it would sink and mix with the fresh water. Thus if the flow rate was slow, one might imaging that a more or less complete mixing could occur. Our model then could answer questions like, how long time would it take before most of the salt water is removed from the first lake, and how much time would it take before most of the salt water was cleared from the whole system? The answer to these questions would give practical input on how much and how fast one should inject the fresh water to clean up the system. If we had data from an actual system, we could compare our model predictions with data from the physical system, and investigate if our model description was correct.

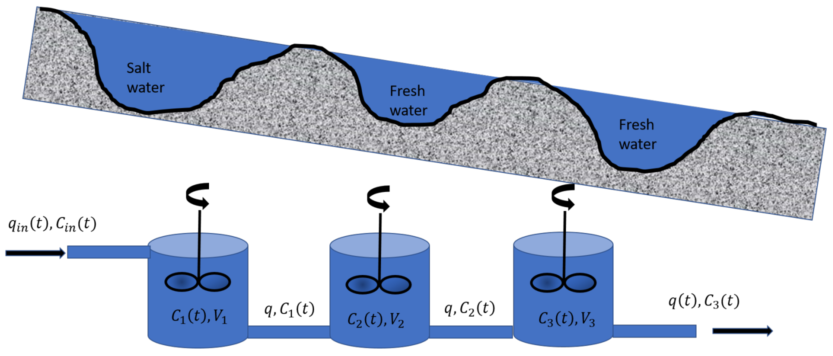

Figure 8: A simple model for cleaning a salty lake that is connected to two lakes down stream.

For simplicity we will assume that all the lakes have the same volume, V. The governing equations follows as before, by assuming mass balance (equation (**??**)):

$$\begin{aligned}
C_1(t+\Delta t)\cdot V - C_1(t)\cdot V &= q(t)\cdot C_{\rm in}(t)\cdot\Delta t - q(t)\cdot C_1(t)\cdot\Delta t.\\
C_2(t+\Delta t)\cdot V - C_2(t)\cdot V &= q(t)\cdot C_1(t)\cdot\Delta t - q(t)\cdot C_2(t)\cdot\Delta t.\\
C_3(t+\Delta t)\cdot V - C_3(t)\cdot V &= q(t)\cdot C_2(t)\cdot\Delta t - q(t)\cdot C_3(t)\cdot\Delta t. \qquad (39)
\end{aligned}$$

Taking the limit $\Delta t \to 0$, we can write equation (**??**) as:

$$V\frac{dC_1(t)}{dt} = q(t)\left[C_{\rm in}(t) - C_1(t)\right]. \tag{40}$$

$$V\frac{dC_2(t)}{dt} = q(t)\left[C_1(t) - C_2(t)\right]. \tag{41}$$

$$V\frac{dC_3(t)}{dt} = q(t)\left[C_2(t) - C_3(t)\right]. \tag{42}$$

Let us first derive the analytical solution, only the first tank is filled with salt water $C_1(0) = C_0$, $C_2(0) = C_3(0) = 0$, and $C_{\rm in} = 0$. The solution to equation (**??**) is, as before $C_1(t) = C_0 e^{-t/\tau}$, inserting this equation into equation (**??**) we find:

$$V\frac{dC_2(t)}{dt} = q(t)\left[C_0 e^{-t/\tau} - C_2(t)\right], \tag{43}$$

$$\frac{d}{dt}\left[e^{t/\tau}C_2\right] = \frac{C_0}{\tau}, \tag{44}$$

$$C_2(t) = \frac{C_0 t}{\tau}e^{-t/\tau}. \tag{45}$$

where we have use the technique of integrating factors when going from equation (**??**) to (**??**). Inserting equation (**??**) into equation (**??**), solving the equation in a similar way as for C_2 we find:

$$V\frac{dC_3(t)}{dt} = q(t)\left[\frac{C_0 t}{\tau}e^{-t/\tau} - C_3(t)\right], \tag{46}$$

$$\frac{d}{dt}\left[e^{t/\tau}C_3\right] = \frac{C_0 t}{\tau}, \tag{47}$$

$$C_3(t) = \frac{C_0 t^2}{2\tau^2}e^{-t/\tau}. \tag{48}$$

The numerical solution follows the exact same pattern as before if we introduce a vector notation. Before doing that, we rescale the time $t \to t/\tau$ and the concentrations, $\hat{C_i} = C_i/C_0$ for $i = 1, 2, 3$, hence:

$$\frac{d}{dt}\begin{bmatrix}\hat{C_1(t)}\\ \hat{C_1}\\ \hat{C_1}\end{bmatrix} = \begin{bmatrix}C_{\rm in}(t) - C_1(t)\\ C_1(t) - C_2(t)\\ C_2(t) - C_3(t)\end{bmatrix} \tag{49}$$

first we rescale the and that there are three tanks in seriesizeBefore we proceed, let us take a look at a situation where there

6 Stiff sets of ODE

As already mentioned a couple of times, our system could be part of a much larger system. To illustrate this, let us now assume that we have two tanks in

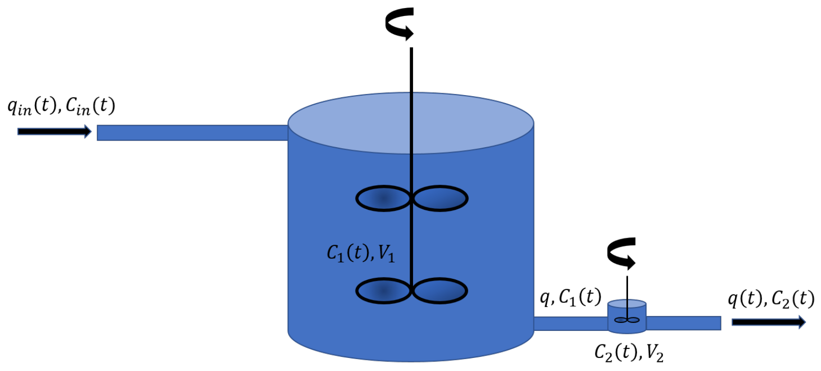

Figure 9: A continuous stirred tank model with a sampling vessel.

series. The first tank is similar to our original tank, but the second tank is a sampling tank, 1000 times smaller.

The governing equations can be found by requiring mass balance for each of the tanks (see equation (**??**):

$$C_1(t+\Delta t)\cdot V_1 - C_1(t)\cdot V_1 = q(t)\cdot C_{\text{in}}(t)\cdot\Delta t - q(t)\cdot C_1(t)\cdot\Delta t.$$
$$C_2(t+\Delta t)\cdot V_2 - C_2(t)\cdot V_2 = q(t)\cdot C_1(t)\cdot\Delta t - q(t)\cdot C_2(t)\cdot\Delta t. \qquad (50)$$

Taking the limit $\Delta t \to 0$, we can write equation (**??**) as:

$$V_1\frac{dC_1(t)}{dt} = q(t)\left[C_{\text{in}}(t) - C_1(t)\right]. \qquad (51)$$
$$V_2\frac{dC_2(t)}{dt} = q(t)\left[C_1(t) - C_2(t)\right]. \qquad (52)$$

Assume, as before that both tanks are filled with seawater, and fresh water is flooded into the tank, i.e. $C_{\text{in}} = 0$. Before we start on the numerical solution, let us first find the analytical solution: As before the solution for the first tank (equation (**??**)) is:

$$C_1(t) = C_0 e^{-t/\tau_1}, \qquad (53)$$

where $\tau_1 \equiv V_1/q$. Inserting this equation into equation (**??**), we get:

$$\frac{dC_2(t)}{dt} = \frac{1}{\tau_2}\left[C_0 e^{-t/\tau_1} - C_2(t)\right],$$
$$\frac{d}{dt}\left[e^{t/\tau_2}C_2\right] = \frac{C_0}{\tau_2}e^{-t(1/\tau_1+1/\tau_2)}, \qquad (54)$$
$$C_2(t) = \frac{C_0}{1-\frac{\tau_2}{\tau_1}}\left[e^{-t/\tau_1} - \frac{\tau_2}{\tau_1}e^{-t/\tau_2}\right], \qquad (55)$$

where $\tau_2 \equiv V_2/q$. To arrive at equation (**??**) we have used the technique of integrating factors, and in equation (**??**) we have used the initial condition $C(0) = C_0$.

Next, we will consider the numerical solution. You might think that two equations is much harder to solve than one, but all the algorithms we have developed are easily extended two more than one variable First of all we have two equations instead of one, this has very little practical impact because we just need to In our example with the CSTR the conservation of mass is violated because the mass out of the tank is always $qC(t)$, but the reason for the violation of the conservation of mass is that if a too large volume of fluid is injected into the tank, the injected is quite easy to understand why: A volume of fresh water (zero concentration of salt) is injected into the tank, and an equally amount of fluid is removed from the tank. If the time step is large, there will be a significant volume of fluid injected into the tank, and because we assume that all concentrations are mixed instantaneously the concentration will drop in the tank. Thus the concentration out of the tank should be the new concentration, not the old one. Thus instead of using equation (**??**), we should use

$$C(t+\Delta t)\cdot V - C(t)\cdot V = q(t)\cdot C_{\text{in}}(t)\cdot\Delta t - q(t+\Delta t)\cdot C(t+\Delta t)\cdot\Delta t. \quad (56)$$

Note that the only difference is that $C(t+\Delta t)$ now enters on the right hand side. In this simple case, we can easily invert the equation to find an expression for $C(t+\Delta t)$:

$$C(t+\Delta t) = \frac{C(t)+\frac{\Delta t}{\tau}C_{\text{in}}(t)}{1+\frac{\Delta t}{\tau}}. \quad (57)$$

Note that we have put $\tau = q/V$, and we also assume that the flow rate is constant (at least turing the time step Δt). Below is an implementation of equation (**??**):

```python
def one_step_imp(c_old, c_in, tau_inv,dt):
    fact=dt*tau_inv
    return (c_old+fact*c_in)/(1+fact)

def euler(c_into,c_init,t_final,vol,q,dt):
    f=[];t=[]
    tau_inv = q/vol
    c_in    = c_into #freshwater into tank
    c_old   = c_init #seawater present
    ti=0.
    while(ti <= t_final):
        t.append(ti); f.append(c_old)
        c_new = one_step_imp(c_old,c_in,tau_inv,dt)
        c_old = c_new
        ti   += dt
    return t,f
```

In figure **??** the result of the implementation is shown.

At the There are a few more important points that we would should mention, and that is the stability of the Eulers method and the

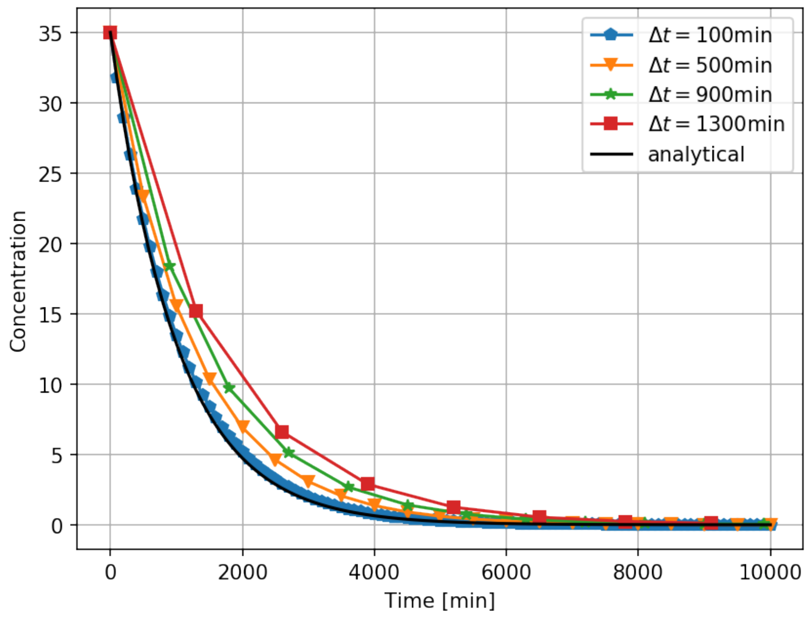

Figure 10: The concentration in the tank for different step size Δt.

Exercise 1: Truncation Error

What is the local error after one step? Remember that the true solution $y(t)$, obeys the following equation:

$$\frac{dy}{dt} = f(y,t). \tag{58}$$

We will also assume (for simplicity) that in our starting point $t = t_0$, the numerical solution, y_0, is equal to the true solution, $y(t_0)$, hence $y(t_0) = y_0$. Show that the truncation error follows Richardson Extrapolation Make an adaptive algorithm for second order equation

Hint. Wolframalpha can perhaps compute the integral.

a) Subexercises are numbered a), b), etc.

Answer. Short answer to subexercise a).

Solution. The local error, is the difference between the numerical solution and the true solution:

$$\begin{aligned}\epsilon^* &= y(t_0+h) - y_1^* = y(t_0) + y'(t_0)h + \frac{1}{2}y''(t_0)h^2 + \mathcal{O}(h^3)\\ &\quad - \left[y_0 + hf(y_0, t_0+h)\right],\end{aligned} \tag{59}$$

where we have used Taylor expansion to expand the true solution around t_0, and equation (**??**). Using equation (**??**) to replace $y'(t_0)$ with $f(y_0, t_0)$, we find:

$$\epsilon^* = y(t_0 + h) - y_1^* = \frac{1}{2}y''(t_0)h^2 \equiv ch^2, \tag{60}$$

where we have ignored terms of higher order than h^2, and defined c as $c = y''(t_0)/2$. Next we take two steps of size $h/2$ to reach y_1:

$$y_{1/2} = y_0 + \frac{h}{2}f(y_0, t_0), \tag{61}$$

$$y_1 = y_{1/2} + \frac{h}{2}f(y_{1/2}, t_0 + h/2), \tag{62}$$

$$y_1 = y_0 + \frac{h}{2}f(y_0, t_0) + \frac{h}{2}f(y_0 + \frac{h}{2}f(y_0, t_0), t_0 + h/2). \tag{63}$$

Note that we have inserted equation (**??**) into equation (**??**) to arrive at equation (**??**). The truncation error in this case is, as before:

$$\begin{aligned}\epsilon = y(t_0 + h) - y_1 &= y(t_0) + y'(t_0)h + \frac{1}{2}y''(t_0)h^2 + \mathcal{O}(h^3)\\ &- \left[y_0 + \frac{h}{2}f(y_0, t_0) + \frac{h}{2}f(y_0 + \frac{h}{2}f(y_0, t_0), t_0 + h/2)\right].\end{aligned} \tag{64}$$

This equation is slightly more complicated, due to the term involving f inside the last parenthesis, we can use Taylor expansion to exand it about (y_0, t_0):

$$\begin{aligned}f(y_0 + \frac{h}{2}f(y_0, t_0), t_0 + h/2) &= f(y_0, t_0)\\ &+ \frac{h}{2}f(y_0, t_0)\left.\frac{\partial f}{\partial y}\right|_{y=y_0, t=t_0} + \frac{h}{2}\left.\frac{\partial f}{\partial t}\right|_{y=y_0, t=t_0} + \mathcal{O}(h^2).\end{aligned} \tag{65}$$

It turns out that this equation is related to $y''(t_0, y_0)$, which can be seen by differentiating equation (**??**):

$$\frac{d^2y}{dt^2} = \frac{df(y,t)}{dt} = \frac{\partial f(y,t)}{\partial y}\frac{dy}{dt} + \frac{\partial f(y,t)}{\partial t} = \frac{\partial f(y,t)}{\partial y}f(y,t) + \frac{\partial f(y,t)}{\partial t}. \tag{66}$$

Hence, equation (**??**) can be written:

$$f(y_0 + \frac{h}{2}f(y_0, t_0), t_0 + h/2) = f(y_0, t_0) + \frac{h}{2}y''(t_0, y_0), \tag{67}$$

hence the truncation error in equation (**??**) can finally be written:

$$\epsilon = y(t_1) - y_1 = \frac{h^2}{4}y''(y_0, t_0) = \frac{1}{2}ch^2, \tag{68}$$

Here goes a full solution of the whole exercise.

Remarks. At the very end of the exercise it may be appropriate to summarize and give some perspectives. The text inside the `!bremarks` and `!eremarks` directives is always typeset at the end of the exercise.