Solving Ordinary Differential Equations (ODE)
Part I - ODE of order 1

Dr. Sudipa Upadhaya, Dept. of Physics, Ramsaday College

We shall try to develop our skills in numerically solving Ordinary Differential equations.

Introduction

Differential equation is considered to be one of the most powerful mathematical tools to understand and predict the behaviour of a dynamical system. A dynamical system is some system with some state, usually expressed by a set of variables, that evolves in time. An oscillating pendulum or the weather of a region are examples of dynamical systems. (In the present circumstances) The pertinent problem of spreading of a disease also can be attributed to the same. We can use basic laws of physics, or plain intuition, to express mathematical rules that govern the evolution of such a system in time. These rules take the form of *differential equations*.

We are all familiar with algebraic equations like $ax^2 + bx + c = 0$, where the unknowns a, b and c are constants. In a differential equation, the unknown is a function, and a differential equation will usually involve this function and one or more of its derivatives. When this function depends on a single independent variable, the equation is called an *Ordinary Differential Equation (ODE)*. We have seen examples in classes. We also know that, the highest order derivative appearing in such equations determines the order of the ODE (*Caution!!* This is not the order of the numerical methods we implement to find solutions).

First I shall start out preparing for ODEs and solving them using Forward Euler method or simply the Euler Method, details of which have already been discussed in class. Then I shall explain numerically how to solve ODEs with the Euler method, both single first-order ODE and systems of first-order ODEs, which in particular will be important to solve 2^{nd} order ODE. After the "warm-up" application - filling of a water tank, I shall demonstrate the mathematical and programming details through one specific application: spreading of diseases.

Filling a Water Tank: Two Cases

Problem : A 25 L tank is to be filled with water in two different ways. In the first case, the water volume that enters the tank per unit time (rate of volume increase) is piecewise constant, while in the second case, it is continuously increasing. For each of these two cases, develop a code that can predict how the total water volume V in the tank will develop with time t over a period of 3 s. Calculations must be based on the information given: the initial volume of water (1 L in both cases), and the volume of water entering the tank per unit time.

Case 1: Piecewise Constant Rate

In this simpler case, there is initially 1 L of water in the tank, i.e., V (0) = 1 L,
while the rates of volume increase are given as:

$$r = 1\ Ls^{-1}\ ,\ 0s < t < 1s\ ,$$
$$r = 3\ Ls^{-1}\ ,\ 1s \leq t < 2s\ ,$$
$$r = 7\ Ls^{-1}\ ,\ 2s \leq t \leq 3s\ .$$

Before turning to the programming, we should work out the exact solution by hand for this problem, since that is rather straight forward. Such a solution will ofcourse be useful for verifying our implementation. In fact, comparing program output to these hand calculations should suffice for this particular problem.

Exact Solution by Hand: For each of the given sub- intervals (on the time axis), the total volume V of water in the tank will increase linearly.

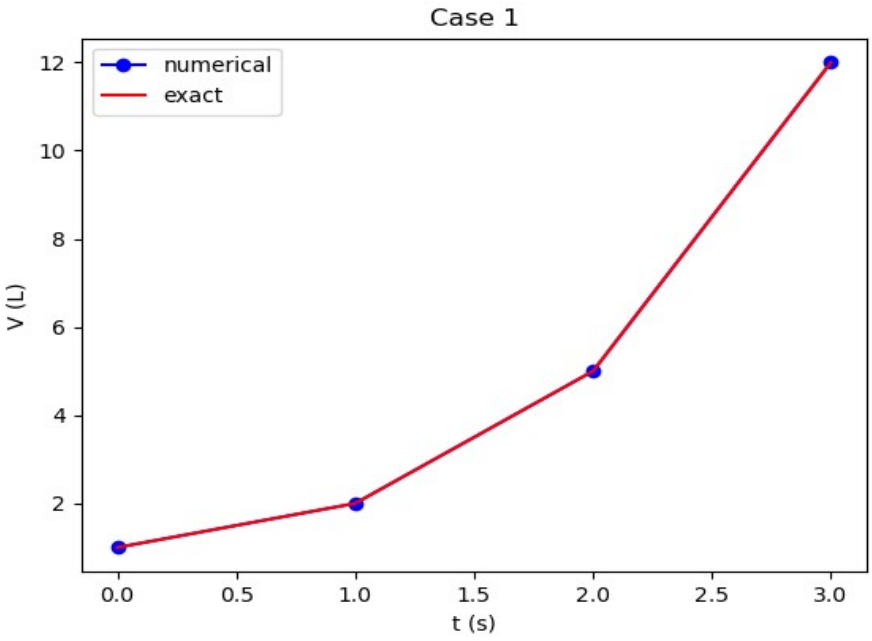
$$V\ (0) = 1\ L\ ,$$
$$V\ (1) = 1\ L + (1\ s)(1\ Ls^{-1}) = 2\ L\ ,$$
$$V\ (2) = 2\ L + (1\ s)(3\ Ls^{-1}) = 5\ L\ ,$$
$$V\ (3) = 5\ L + (1\ s)(7\ Ls^{-1}) = 12\ L\ .$$

Numerical solution using Euler Method :

```
import numpy as np
import matplotlib.pyplot as plt
a = 0.0; b = 3.0                    #time interval
N = 3                                    #number of time steps
dt = (b - a)/N                      #time step (s)
V_exact = [1.0, 2.0, 5.0, 12.0]          #exact volumes (L)
```

```
V = np.zeros(4)                          #numerically computed volume
(L)
V[0] = 1                                 #initial volume
r = np.zeros(3)                          #rates of volume increase
(L/s)
r[0] = 1; r[1] = 3; r[2] = 7

for i in [0, 1, 2]:
     V[i+1] = V[i] + dt*r[i]
time = [0, 1, 2, 3]
plt.plot(time, V, 'bo-', time, V_exact, 'r')
plt.title('Case 1')
plt.legend(['numerical','exact'], loc='upper left')
plt.xlabel('t (s)')
plt.ylabel('V (L)')
plt.show()
```

The resulting plot looks like :



Let us now evaluate the other case of continuously increasing rate.

Case 2: Continuously Increasing Rate

This case is a bit more tricky. As in the previous case, there is 1 L of water in the tank from the start. When the tank fills up,

however, the rate of change in volume of water increases, i.e. r can be approximated to be equal to the current volume of water, i.e., r = V (in units of Ls^{-1}). So, for example, at the time when there is 2 L in the tank, water enters the tank at 2 Ls^{-1}, when there is 2.1 L in the tank, water comes in at 2.1 Ls^{-1}, and so on. Thus, r does not remain constant now for any period of time within the 3 s interval, it increases continuously and produces a gradually steeper curve for V(t). The information we have is , V(0) = 1 L,
and, for the rate (in Ls^{-1}),

$$r(t) = V(t), \quad 0s < t \leq 3s \ .$$

We can easily check the exact solution in this case, which is $V(t) = e^t$. This allows us to easily check out the performance of any computational idea that we might try.

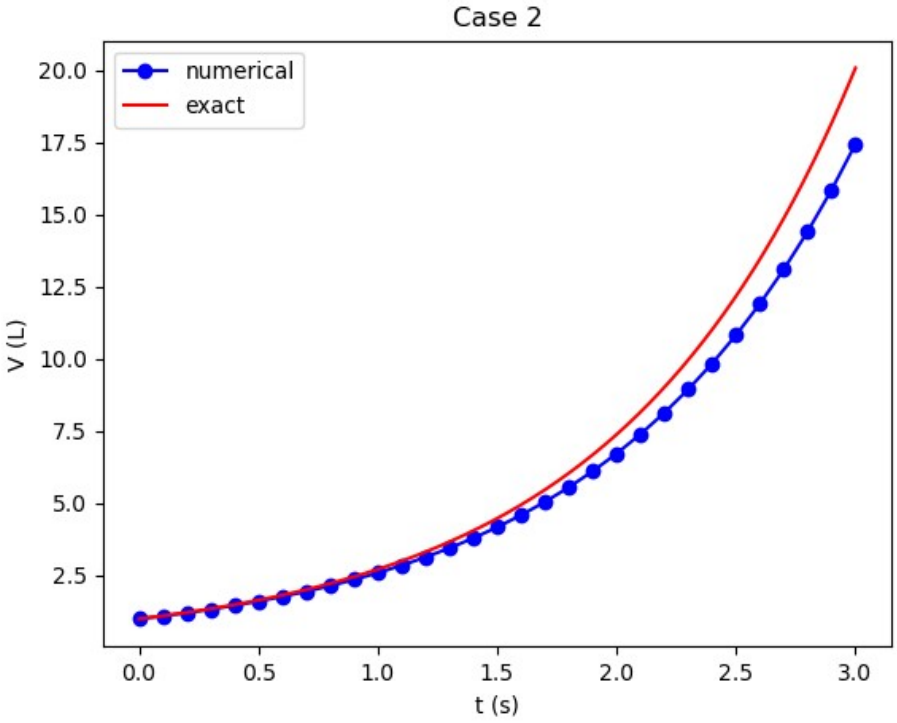
The First Time Step - In the very first time step, since we are given the initial volume V (0) = 1 L, we do know the correct volume and correct rate at t = 0, since r(t) = V(t). Thus, using this information and pretending that r stays constant as time increases, we will be able to compute a straight line segment for the very first time step (some Δt must be chosen). This straight line segment will then become tangent to the true solution curve when t = 0 (*Try to remember the discussion we had in class, regarding Euler Method*). The computed volume at the end of the first time step will have an error, but if our time step is not too large, the straight line segment will stay close to the true solution curve and the error in V should be "small".

The Second Time Step - What about the second time step? Well, the volume we computed (with an error) at the end of the first time step, must now serve as the starting volume and ("constant") rate for the second time step. This allows us to compute an approximation to the volume also at the end of the second time step. If errors are "small", also the second straight line segment should be close to the true solution curve.

We realize that, in this way, we can work our way all along the total time interval. Immediately, we suspect that the error may grow with the number of time steps, but since the total time interval is not too large, and since we may choose a very small time step, this could still work!

```
import numpy as np
import matplotlib.pyplot as plt
a = 0.0; b = 3.0                #time interval
N = 30                          #number of time steps
dt = (b - a)/N                  #time step(s)
V = np.zeros(N+1)               #numerically computed volume (L)
V[0] = 1                        #initial volume

for i in range(0, N, 1):
     V[i+1] = V[i] + dt*V[i]    # ...r is V now

time_exact = np.linspace(a, b, 1000)
V_exact = np.exp(time_exact)    # make exact solution (for plotting)

time = np.linspace(0, 3, N+1)
plt.plot(time, V, 'bo-', time_exact, V_exact, 'r')
plt.title('Case 2')
plt.legend(['numerical','exact'], loc='upper left')
plt.xlabel('t (s)')
plt.ylabel('V (L)')
plt.show()
```

I considered a time step $\Delta t = 0.1s$ for a first try. To plot the exact solution, I just picked 1000 points in time, which can be considered "large enough" to get a representative curve. The resulting plots look like :



This looks promising! Not surprisingly, the error grows with time. However, the time step is not particularly small, so we should expect much more accurate computations if Δt is reduced. I skip showing the corresponding results here. You can try on your own for fun.

Exercise 1. Write the differential equations for the problem we have just solved numerically, for both the cases.

Exercise 2. Write similar code for case 2 using Modified Euler Algorithm.

P.T.O.

Spreading of Disease: A System of First Order ODEs

This section will explain how one can apply mathematics and programming to solve a system of first-order ODEs. This will be required to solve higher order differential equations. We will do this by investigating the spreading of disease, say Flu in a local community (!! A pertinent topic, as you may generalize to the present circumstances with additional conditions as required). The mathematical model is now a system of three differential equations with three unknown functions. To derive such a model, we can use mainly intuition, so no specific background knowledge of diseases is required. I am skipping here the model formulation. If you find any difficulty to understand, I can further upload notes on that or can discuss in class.

Let us imagine a local housing complex as a small and closed society. Suddenly, one or more of the people get the flu. We expect that the flu may spread quite effectively or die out. The question is how many of the persons will be infected. Some quite simple mathematics can help us to achieve insight into the dynamics of how the disease spreads.

Let the function S(t) stand for how many individuals (called *Susceptibles*), at time t, have the chance to get infected. 't' may count hours or days, for instance. Another category, I, comprises of the individuals that are infected. Let I(t) count how many there are in category I at time t. Finally, an individual can have finite probability to recover from the disease or to die. In either case, the individual is moved from the I category to a category we call the removed category, labelled with R. We let R(t) count the number of individuals in the R category at time t. We assume for this initial study that, those who enter the R category, cannot leave this category. To summarize, the spreading of this disease is essentially the dynamics of moving individuals from S to I and then to the R category:

The time interval Δt can be chosen and varied to see the effects

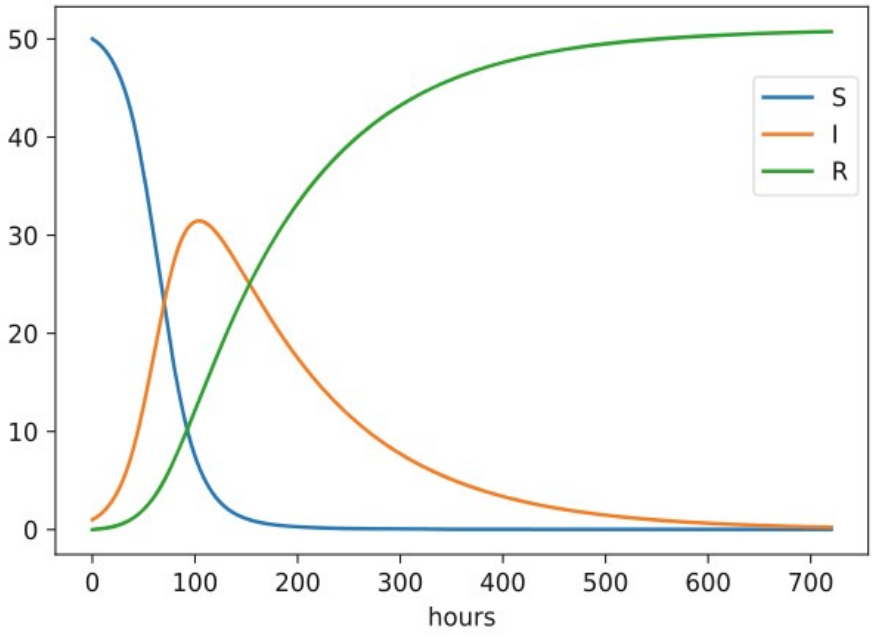
accordingly. The three differential equations are:,

$$S' = -\beta SI,$$
$$I' = \beta SI - \gamma I,$$
$$R' = \gamma I$$

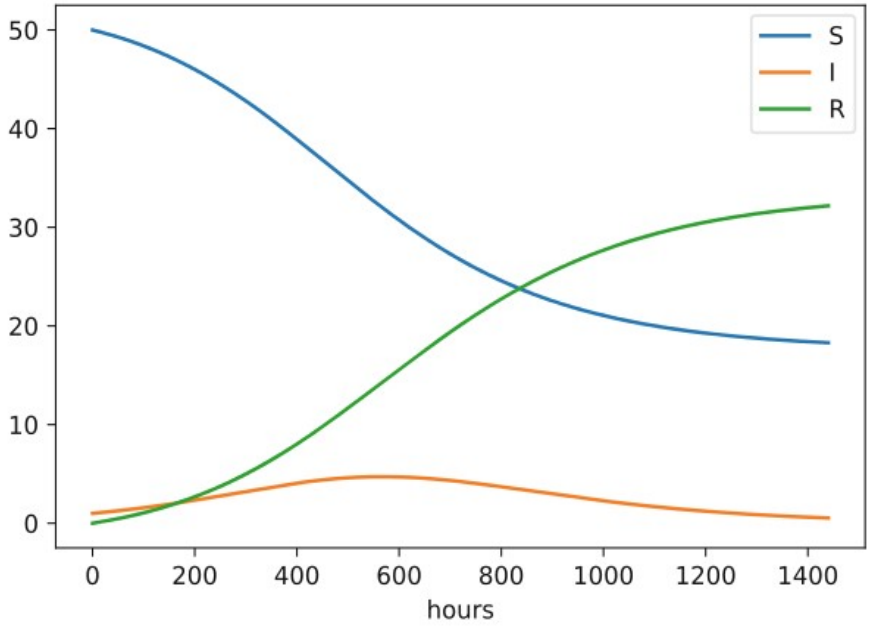
where, β is a parameter reflecting the probability of people getting infected during a time interval of unit length. γ is the probability that one individual recovers or dies in a unit time interval.

```
import numpy as np
import matplotlib.pyplot as plt
# Time unit: 1 h
beta = 10./(40*8*24)
gamma = 3./(15*24)
dt = 0.1                    # 6 min
D = 30                      # Simulate for D days
N_t = int(D*24/dt)          # Corresponding no of time steps

t = np.linspace(0, N_t*dt, N_t+1)
S = np.zeros(N_t+1)
I = np.zeros(N_t+1)
R = np.zeros(N_t+1)

# Initial condition
S[0] = 50
I[0] = 1
R[0] = 0

# Step equations forward in time
for n in range(N_t):
     S[n+1] = S[n] - dt*beta*S[n]*I[n]
     I[n+1] = I[n] + dt*beta*S[n]*I[n] - dt*gamma*I[n]
     R[n+1] = R[n] + dt*gamma*I[n]
fig = plt.figure()
l1, l2, l3 = plt.plot(t, S, t, I, t, R)
fig.legend((l1, l2, l3), ('S', 'I', 'R'), 'center right')
plt.xlabel('hours')
plt.savefig('tmp.pdf'); plt.savefig('tmp.png')
plt.show()
```

The methods to find β and γ are driven by the initial conditions in the model formalism, that I shall discuss in class, if needed/if anyone feels interested and if time permits. With these choices of parameters, the resulting plots look like,



Note : We can experiment with β and γ to see whether we get an outbreak of the disease or not. Imagine that a "wash your hands" campaign became successful and that some other housing complex in this case, experienced a reduction of β by a factor of 5. With this lower β the disease spreads very slowly so I simulated for 60 days. The curves appear as,



Further conditions like, some other people from state R become susceptible or use of vaccination etc. may be incorporated as per the situations demand. We shall skip them for the time-being and can resume if needed or time permits.

Exercise 3 : Redo this problem of spreading of disease using RK2 method. Explain all the plots.

Exercise 4 : Write down the differential equation for an LR circuit and solve numerically using RK4 mechanism. Choose the parameters conveniently and vary them to see the effects. Also plot the deviation of the numerical results from the exact ones and see the effects of reduced step sizes.

Exercise 5 : Solve Radioactive decay equation using Euler and RK4 methods. Choose parameters conveniently. Save the plots in png and pdf formats. Also save the results in data files for both the methods to plot the difference between them.