Lecture 2-2: Differential Equations

(Adapted from slides by Gerald Fux)

Zejian Li
(li.zejian@ictp.it)

23. Oct. 2024

Introduction

- **Ordinary Differential Equations (ODE)**: Differential equations for functions depending on only one variable.
  - Order of ODE: the highest appearing order of derivative of the function
  - System of ODE: coupled differential equation for multiple functions (each depending on only one variable).

$$\text{Bacteria growth} \quad \frac{\mathrm{d}w}{\mathrm{d}t} = \eta w \quad \text{is a 1st order ODE}$$

$$\text{(Damped) harmonic oscillator} \quad \frac{\mathrm{d}^2 x}{\mathrm{d}t^2} = -\gamma \frac{\mathrm{d}x}{\mathrm{d}t} - \frac{k}{m} x \quad \text{is a 2nd order ODE}$$

- **Partial Differential Equations (PDE)**: Differential equations for functions depending on multiple variables. For example: Maxwell differential equations are a system of first order PDE.

We want to find the unknown function(s) [e.g. $w(t)$, $x(t)$, or $\vec{E}(\vec{r}, t)$ & $\vec{B}(\vec{r}, t)$] for specific initial conditions.

Numerical Methods for Differential Equation

- Often analytical solutions are complicated, hard to find, or unknown.
- Even more often there exist no analytical solutions, and a numerical solution is necessary.
- Numerical method idea:
  - Start with the initial conditions.
  - Take a small step: Calculate an approximate value of the function for a small increment of the independent variable.
  - Take another small step: Calculate the next approximate value of the function for another small increment of the independent variable.
  - ... and so on ...

Euler Method - Idea

For 1st order ODE, which have the form:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(t, x) \quad \text{with} \quad x(t_0) = x_0$$

For example:

$$\text{radioactive decay:} \quad \frac{\mathrm{d}x}{\mathrm{d}t} = f(\cancel{t}, x) = -\gamma x$$

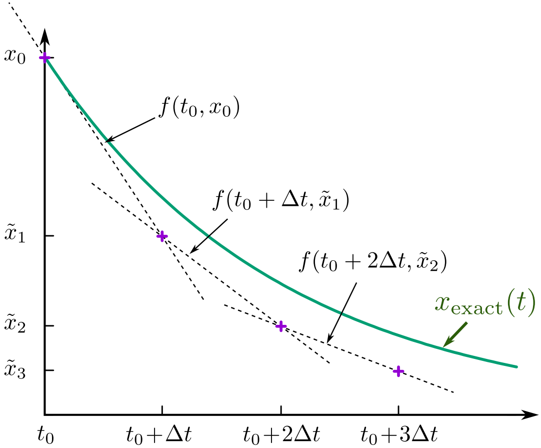
Consider the Taylor expansion at $t = t_0$ of the solution $x(t)$:

$$x(t_0 + \Delta t) = x(t_0) + \Delta t \cdot \left.\frac{\mathrm{d}x}{\mathrm{d}t}\right|_{t=t_0} + \mathcal{O}(\Delta t^2)$$

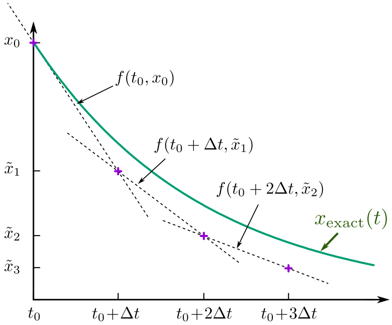
$$x(t_0 + \Delta t) \quad \approx \quad x(t_0) + f(t_0, x_0) \cdot \Delta t$$

Euler Method - Sketch

$$x(t_0 + \Delta t) \quad \approx \quad x(t_0) + f(t_0, x_0) \cdot \Delta t$$



Euler Method - Algorithm



Algorithm: Euler Method

Input: function $f(t, x)$, initial values t_0, x_0, step size Δt, and number of steps N

1. set $x := x_0$ and $t := t_0$
2. repeat N times:
   - set $fx := f(t, x)$
   - set $x := x + fx \cdot \Delta t$
   - set $t := t + \Delta t$

Output: the sequence of t and x approximating the solution of $\frac{\mathrm{d}x}{\mathrm{d}t} = f(t, x)$.

Euler Method - Fortran Implementation Sketch

```fortran
subroutine euler_method(t0, x0, dt, N, tlist, xlist)
  ! ... variable declarations and allocations ...
  x = x0
  t = t0
  tlist(1) = t
  xlist(1) = x
  do i = 1, N
    fx = f(t,x)
    x = x + fx * dt
    t = t + dt
    tlist(i+1) = t
    xlist(i+1) = x
  end do
end subroutine euler_method
```

Improved Euler Method: Midpoint Method

In a similar spirit to the improvement from the left Riemann sum to the midpoint sum for numerical integration, one can also improve the Euler method for differential equations:

$$x(t_0 + \Delta t) \approx x(t_0) + \Delta t \cdot \left.\frac{\mathrm{d}x}{\mathrm{d}t}\right|_{t=t_M} + \mathcal{O}(\Delta t^3) \quad \text{with} \quad t_M = t_0 + \Delta t/2$$

$$\begin{aligned} x(t_0 + \Delta t) &\approx x(t_0) + f(t_M, x_M) \cdot \Delta t \\ &\text{with} \\ t_M &:= t_0 + \Delta t/2 \\ x_M &:= x(t_0) + f(t_0, x_0) \cdot \Delta t/2 \end{aligned}$$

Higher Order ODE $\to$ System of 1st Order ODE

Euler method and midpoint method only work for 1st order ODEs. However, ...

It is always possible to rewrite a higher order ODE as a system of 1st order ODEs!

For example, the equation of motion for the angle x of a friction-less (stiff) pendulum

$$\frac{\mathrm{d}^2 x}{\mathrm{d}t^2} = -\sin(x)$$

can be rewritten as the system of two first order ODEs (for the angle x and the angular velocity v):

$$\frac{\mathrm{d}x}{\mathrm{d}t} = v$$
$$\frac{\mathrm{d}v}{\mathrm{d}t} = -sin(x).$$

Euler Method - Algorithm for a System

Algorithm: Euler Method for a System x and y

Input: functions $f_x(t,x,y)$, $f_y(t,x,y)$, initial values t_0, x_0, y_0, step size Δt, and number of steps N

1. set $x := x_0$, $y := y_0$ and $t := t_0$
2. repeat N times:
    - set $fx := f_x(t,x,y)$
    - set $fy := f_y(t,x,y)$
    - set $x := x + fx \cdot \Delta t$
    - set $y := y + fy \cdot \Delta t$
    - set $t := t + \Delta t$

Output: the sequence of t, x, and y which approximates the solution of

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f_x(t,x,y) \qquad \frac{\mathrm{d}y}{\mathrm{d}t} = f_y(t,x,y).$$

Assignment 12

Write a program that solves the differential equation $\frac{\mathrm{d}x}{\mathrm{d}t} = -x$ using 1) Euler and 2) midpoint methods.

- Create separate subroutines for the two methods (see slide 7).
- Solve with initial conditions `x(0.0) = 1.0` and `dt=0.1, N=100` and store the results in arrays `tlist` and `xlist`.
- The program should create a file for each method (`euler.txt` and `midpoint.txt`) with the result in two columns: `t` and `x(t)`.
- Plot the result x versus t with your favorite plotting program.

Bonus question:

- Expand your subroutines and solve the equation of motion for the angle x of a friction-less physical pendulum: $\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -\sin(x)$ using the two methods, with initial conditions `t0=0, x0=1.0, v0=0.0` and `dt = 0.1, N=300`.

Submit the graphs and your code as `Ass12.YourLastName.f90` to `li.zejian@ictp.it` before the next lesson.

Reminder for plotting:

- You can use whatever program you like to plot the dynamics. A very simple way is to use `gnuplot`.
  - If the file 'data.txt' has two columns t and x, then you can plot $x(t)$ with:
    `$ gnuplot -p -e "plot 'data.txt' using 1:2 with lines"`
  - If the file 'data.txt' has three columns t, x, and v, then you can plot $x(t)$ and $v(t)$ with:
    `$ gnuplot -p -e "plot for [col=2:3] 'data.txt' using 1:col with lines"`

  Side note:
  - Usually we need a much smaller `dt` to obtain reliable results. Here we use the relatively large `dt=0.1` to better compare the performance of the difference methods.