Lecture 13  Differential Equations I

(Finite Difference Methods)

- mainly ODE's -

Introduction:

Types of Problems:
(easier)

(A) ODE
Ex. $M \frac{d^2 \phi}{dx^2} - k \phi(t)$
(simple harmonic oscillator)

(B) Linear
Ex. S110 wave equation

(C) Initial Value
$\phi(0) \phi'(0)$

more difficult

PDE
Ex. $\frac{\partial^2 \phi}{\partial x^2} - \frac{1}{c^2} \frac{\partial^2 \phi}{\partial t^2} = 0$
(wave equation)

Nonlinear
Ex. $\ddot{\phi} + \omega^2 \sin \phi = 0$ $\omega = \sqrt{\frac{g}{L}}$
(pendulum)

Boundary Value
$\phi(0) \phi(L)$

more difficult
This lecture emphasises:

- ODE (linear/non-linear)

  \[ \frac{\phi(t)}{x_k} = \frac{1}{h} (\phi_{k+1} - \phi_k) \]

  \[ \frac{\phi(t)}{x_k} = \frac{1}{2h} (\phi_{k+1} - 2\phi_k + \phi_{k-1}) \]

- Euler's (conceptual background)

- Runge-Kutta

- Midpoint & Richardson's Extrapolation

- Others: predictor-corrector leaping

- Stiffness problem: implicit methods

- Initial Value

- Boundary Value

  - Shooting Methods
  - Relaxation Methods

  PDE (next lecture)
All ODE in any order can be written into a set of 1st order ODE!

\[ \left\{ \frac{d^2y}{dt^2} + g(t) \frac{dy}{dt} = r(t) \right\} \quad (\text{2nd order}) \]

let \( z(t) = \frac{dy}{dt} \)
then \( \frac{dz}{dt} = \frac{d^2y}{dt^2} \)

\[ \Rightarrow \left\{ \begin{array}{l} \frac{dz}{dt} = r(t) - g(t) z(t) \\ \frac{dy}{dt} = z(t) \end{array} \right\} \quad (\text{1st-order}) \]

(\# trade off \( \Rightarrow \) increase of dim by 1)

So, in general, we can always write ODE as:

\[ y(t) = E(y, t) \]

\( E(x, t) \) could be linear or nonlinear!
Initial Value Problems

Basic Idea behind Numerical ODE Solver

→ Euler's Method:

\[ \dot{y}(t) = f(y, t) \]

Finite diff. \[ \Rightarrow \frac{y_{n+1} - y_n}{h} = f(y_n, t_n) \]

Euler Scheme \[ \Rightarrow y_{n+1} = y_n + hf(y_n, t_n) \]

\textbf{Note:} Only derivative at \( t_n \) is used to get to \( y_{n+1} \)!

(Asymmetric)

It is simple to see that:

By Taylor's expansion:

\[ y_{n+1} = y_n + hf(y_n) + h^2 \frac{f'(y_n)}{2!} + \ldots \]

\[ E \sim O(h^2) \]

\[ \therefore \text{So, Euler is a 1st order method!} \]
Applet is open in a separate window.

This java applet demonstrates properties of vector fields. You may select one of many vector fields from the Setup menu in the upper right.

The applet shows the potential surface of the vector field, with particles following the field vectors. You may click and drag with the mouse to rotate the view. Also some of the field selections have parameters which may be adjusted.

There is also a 3-D version of this applet (a version with 3-D fields, that is). This version only does 2-D fields, but unlike the 3-D version it can display the potential surface, curl, and divergence, and can also demonstrate Green's theorem and the divergence theorem.

Full Directions.

The source.

More applets.

Zip archive of this applet.

Version 1.3, posted 2/22/05

java@falstad.com
Modified & Improved Euler's Methods

→ Euler is asymmetric & first order

* Can try to make it more symmetric

and we will show that errors can be
cancelled to higher order ~ h^2!

---

**Graphical**

\[ y'(x_0) = f(x_0, y_0) \]

---

<table>
<thead>
<tr>
<th>Regular Euler</th>
</tr>
</thead>
<tbody>
<tr>
<td>move forward with derivative ( f(x_0, y_0) ) at initial pt.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modified Euler</th>
</tr>
</thead>
<tbody>
<tr>
<td>move forward with approx. derivative ( f(x_{\text{mid}}, y_{\text{mid}}) ) at ( x_{\text{mid}} = x_0 + \frac{h}{2} ).</td>
</tr>
</tbody>
</table>
improved Euler

- move forward with "average" derivative
  \[
  f_0 + \frac{f_n + f_{n+1}}{2}
  \]
  at \( x_0 \) & \( x_{0+h} \).

---

- let see how you do this?

**Modified**

- \( X_{\text{mid}} = x_0 + \frac{h}{2} \)
- \( Y_{\text{mid}} = y_0 + \frac{h}{2} f(x_0, y_0) \)
  (estimate midpt with regular Euler)
- then \( f(X_{\text{mid}}, Y_{\text{mid}}) \) (approx. derivative at midpt) can be evaluated.

**Lastly**

- \( y(x_{0+h}) = y_0 + hf(X_{\text{mid}}, Y_{\text{mid}}) \)
Improved:

- \( x_h = x_0 + h \)
- \( \tilde{y}_h = y_0 + hf(x_0, y_0) \) (approx. \( \tilde{y}_h \) by regular Euler)
- then take average of derivative
  - at \( x_0 \) (initial pt) & \( x_h \) (end pt).
  \[
  \bar{f} = \frac{f(x_0, y_0) + f(x_h, \tilde{y}_h)}{2}
  \]

Lastly:

- \( y(x_0+h) = y_0 + h \left( \frac{f(x_0+y_0) + f(x_0+h, y_0+hf_0)}{2} \right) \)

Modified & Improved Euler are 2nd order.

Write them in general form:

- \( y(x_0+h) = y_0 + h \left[ \alpha f(x_0, y_0) + \beta f(x_0+h, y_0+hf_0) \right] \)

Note that:

- Modified \( \alpha = 0 \), \( \beta = \frac{1}{2} \)
- Improved \( \alpha = \frac{1}{2} \), \( \beta = \frac{1}{2} \)

- Other choices are possible -
Now, let look at Taylor's expansion of 
\[ f(x_0 + \Delta x, y_0 + \Delta y) \]

\[ = f(x_0, y_0) \pm \frac{\partial f(x_0, y_0)}{\partial x} (\Delta x) \pm \frac{\partial f(x_0, y_0)}{\partial y} (\Delta y) \pm O(h^3) \]

Now, put this into our general order \( O \).

\[ y(x_0 + \Delta x) = y_0 + h(x + \Delta x) + h^2 \frac{\partial f}{\partial x} \left[ f_0 + \frac{\partial f}{\partial x} (x_0, y_0) \right] + O(h^3) \]

Compare with Taylor expansion of \( y(x_0 + \Delta x) \):

\[ y(x_0 + \Delta x) = y_0 + \frac{dy}{dx}(x_0, y_0) \Delta x + \frac{1}{2} \frac{d^2y}{dx^2}(x_0, y_0) (\Delta x)^2 + O(h^3) \]

Note: \( \frac{d}{dx} \frac{dy}{dx} = \frac{d}{dx} (f(x, y)) = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \frac{dy}{dx} \)

\[ = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} f \]
\[ \begin{align*}
\alpha + \beta &= 1 \\
\beta \gamma &= \frac{1}{2} \\
\gamma \delta &= \frac{1}{2}
\end{align*} \]

\[ \Rightarrow \text{Thus, Modified & Improved Euler methods are 2nd order in h!} \]
Check:

Taylor's expanding $Y_{n+1}$ around $Y_n$:

$$Y_{n+1} = Y_n + hf_n + \frac{h^2}{2} f'_n + O(h^3)$$

By finite difference,

$$\frac{(f_{n+\frac{1}{2}} - f_n)}{\frac{1}{2}h} \approx f'_n$$

Substitute this into $\circ$,

$$Y_{n+1} = Y_n + hf_n + \frac{h^2}{2} \left( \frac{h}{h} \left( f_{n+\frac{1}{2}} - f_n \right) \right) + O(h^3)$$

$$Y_{n+1} = Y_n + hf_{n+\frac{1}{2}} + O(h^3)$$

So, this "trick" step make method become 2nd order!

"RK - 2nd order" or "midpt" method will come back to this later!
Runge-Kutta Method
(higher order method)

Similar to Numerical Integrators (or Mod/Zip Gates),
one can try to cancel out higher order
terms in Taylor's expansion by
including other evaluation points in
the integration step. (Symmetrize)

2nd-order RK:

\[
\begin{cases}
  k_1 = hf(t_n, y_n) \\
  k_2 = hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\
  \Rightarrow y_{n+1} = y_n + k_2 + O(h^2)
\end{cases}
\]
4th order RK:

- Similarly, by evaluating \( f \) at different points and by combining them in specific ways, one can in principle cancel higher order terms order by order.

- One particular choice is: 4th order RK

\[
\begin{align*}
    k_1 &= h f(x_n, y_n) \\
    k_2 &= h f(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \\
    k_3 &= h f(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \\
    k_4 &= h f(x_n + k_3, y_n + h) \\
    y_{n+1} &= y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5)
\end{align*}
\]

\( y(x) \)

\( x_n \)- \( x_n + \frac{h}{2} \)- \( x_{n+1} \)

\( y_n \)- \( y_{n+1} \)

\( * \) Check that 4th RK is \( O(h^5) \)!
RK4 → Scientific Workhorse
- Method of choice for 1st tries
- But not necessary the most accurate!

Adaptive Stepsize for RK4

(Improvement on Efficiency)

\[ y(t) \]

\[ \leftarrow A \rightarrow \leftarrow B \rightarrow t \]

\( y(t) \) varies quickly → need small steps
\( y(t) \) varies slowly → can use large steps.

* Optimize between accuracy & speed by vary step sizes!
Question: How to adaptively vary the step size?

⇒ Need to be able to estimate the truncation error with a given step size!

Solution:
- Start with a trial value of $h$
  - Calculate $y(t)$ in two different ways
    - 2 steps of $h$
    - 1 step of $2h$

Step doubling with correction

- big step
- 2 small steps

Total # of fn evaluations = 11 (1st pts are the same)
- just use 2 small steps

Overhead cost in add. computation = $\frac{11}{8} \approx 1.375$. 
But, it gives you a handle on the truncation error:

(1b)

one step: \( y(t + 2h) = y_1 + (2h)^5 \frac{y''(t)}{5!} + o(h^6) \)

two sm. steps: \( y(t + 2h) = y_2 + 2h^5 \frac{y''(t)}{5!} + o(h^6) \)

(Note: RK4 is a \( O(h^4) \) process)

(\( \Delta = y_2 - y_1 \) \( \sim O(h^5) \))

- \( \Delta = y_2 - y_1 \) \( \sim O(h^5) \)

- We want to adjust \( h \) so that this is not too small (efficiency) AND not too large (accuracy)

---

- Now, let say \( E_{\text{max}} \) - maximum tolerable truncation error

- Within the current interval \([t_k, t_{k+1}]\), we estimated \( \Delta(t_k, h_k) = y_2 - y_1 \) to be of \( O(h^5) \)

- Optimally, we want to set \( h_{k+1} \) for the next interval \([t_{k+1}, t_{k+2}]\) such that

\[ \Delta(t_{k+1}, h_{k+1}) = E_{\text{max}} \]
Since \( \Delta \alpha = h^5 \),

\[
\left| \frac{\Delta k_{h+1}}{\Delta k} \right| = \left| \frac{E_{\text{max}}}{\Delta (t_k, h_k)} \right| \propto \left( \frac{h_{k+1}}{h_k} \right)^5
\]

\[\Rightarrow h_{k+1} \leq h_k \left| \frac{E_{\text{max}}}{\Delta (t_k, h_k)} \right|^{\frac{1}{5}} \]

This means that:

\[\begin{cases} \text{If } \Delta k \geq E_{\text{max}}, & h_{k+1} \text{ will be scaled back smaller;} \\ \text{And, if } \Delta k < E_{\text{max}}, & h_{k+1} \text{ will be expanded bigger.} \end{cases}\]

Note: \( E_{\text{max}} \) should be stated in terms of fractional error

\[\text{Fractional error} \rightarrow 10^{-6}, 10^{-16}, \ldots\]

\[E_{\text{max}} \sim E \text{ single double}\]

\[\text{e.g. } E_{\text{max}} \text{ for oscillatory functions.}\]

A more strict criterion: (might be needed if \( E \) accumulated with steps)

\[\text{For small } h, \quad E_{\text{max}} \sim E \]

\[\text{fractional accuracy in increments of } y \quad \text{instead of } y \text{ itself!}\]
Kunge-Kutta-Fehlberg

- Alternative to doubling method for
  adaptive step size control.
  (Advantage: 6 fn calls instead of 11)
  With six fn calls, Fehlberg found a way
  to construct
  - a 5th order \( \frac{\partial y}{\partial t} \)
  AND - a 4th order \( \frac{\partial y}{\partial t} \)
  - SO, just like doubling method, we can
    get a handle on truncation error
    with these two estimates.

\[ \Delta_k = y_k - \hat{y}_k \sim \Delta(\Delta_k) \]

\[ \Delta_k \text{ determined by the larger 4th order} \]

Again, \[ h_{K+1} < h_K \left( \frac{\epsilon_{\text{max}}}{\Delta_k} \right)^{1/5} \]
In the case when the total error $E$ accumulated with the # of steps

$\Rightarrow$ reducing step size $h$ (more steps),

we need to require

$E_{max}$ to be smaller accordingly!

$\Rightarrow E_{max} \rightarrow E_{max} h$

With this scaling, we need to adjust

$$\frac{E_{max}}{\Delta k} \Rightarrow \frac{E_{max} h_{k+1}}{\Delta k} \sim \left(\frac{h_{k+1}}{h_k}\right)^5 \Rightarrow \frac{h_k E_{max}}{\Delta k} \sim \left(\frac{h_{k+1}}{h_k}\right)^4$$

$\Rightarrow h_{k+1} \sim h_k \left(\frac{h_k E_{max}}{\Delta k}\right)^{1/4}$

Press suggestion:

Combination of the two exponents

$$h_{k+1} = \left\{\begin{array}{ll}
S h_k \left(\frac{E_{max}}{\Delta k}\right)^{1/5} & (E_{max} \geq \Delta k) \\
S h_k \left(\frac{E_{hic}}{\Delta k}\right)^{1/4} & (E_{hic} < \Delta k)
\end{array}\right.$$
\[ \Rightarrow \text{the scaling should be } 1/4 \text{ instead of } 1/5 \]

when \( h \to 0 \).

A workable balance:

\[
\begin{align*}
S h k \left| \frac{E_{\text{err}}}{\Delta k} \right|^{1/4} & \quad \text{for } E_{\text{err}} \geq \Delta k \text{ expand} \\
S h k \left| \frac{E_{\text{err}}}{\Delta k} \right|^{1/4} & \quad \text{for } E_{\text{err}} < \Delta k \text{ reduce} \\
\end{align*}
\]

where \( S \) is a safety factor \( \leq 1 \).

Richardson's Extrapolation to \( h \to 0 \)

(Bulirsch-Stoer Method)

Recall \( \text{Romberg integration} \): we use information of truncation error due to step size \( h \) to extrapolate result to \( h \to 0 \).

We can also use this idea for ODE solver:

\[
Y(t) \xrightarrow{\text{result with } h} \xrightarrow{\text{result with } h/2} \ldots \xrightarrow{\text{extrapolated result for } h = 0} \]

\[ \text{ knowing } \Delta \sim O(h^4) \]

results from \( h + h/2 \)

\[ \Rightarrow \text{better than } h/2 \text{ alone.} \]
In principle, we can use any methods described before as the kernel for this scheme.

Recall **Mid-pt Method**:

\[ Y_{n+1} = Y_n + h f(t_n + \frac{1}{2}h, Y_n + \frac{1}{2}k_i) \]

\[ k_i = hf(t_n + \frac{1}{2}h, Y_n) \]

For simplicity, assume we used fixed step size \( h \).

Given \( Y_n(T, h) \) \( \rightarrow \) result for step size \( h \).

**Modified Mid-pt**

Recall **extrapolation** (Neville's algorithm)

\[ \begin{align*}
  h & \quad Y_1 \\
  \frac{h}{2} & \quad Y_2 > Y_{12} \\
  \frac{h}{4} & \quad Y_3 > Y_{23} > Y_{123} \\
  \frac{h}{8} & \quad Y_4 > Y_{34} > Y_{234} \\
\end{align*} \]

Starting values

\[ y(t) \]

\[ y(t) \]
Modified Midpt

\[
\begin{align*}
\text{steps} & : 0 \rightarrow n \rightarrow T \\
\text{intermediate approx.} & \begin{cases}
Z_0 = y(0) \\
Z_i = Z_0 + hf(0, Z_0) \\
Z_{m+1} = Z_m + h f(x + mh, Z_m) & m = 1, 2, \ldots, n-1
\end{cases} \\
\text{Final approx.} & : Y_h(T, h) = \frac{1}{2} \left( Z_n + Z_{n-1} + h f(H, Z_n) \right)
\end{align*}
\]

\[\Rightarrow O(h^2) \quad \text{in particular,} \]
\[y_n - y(T) = \epsilon_n h^2 \quad \text{even powers only!} \]

\text{Simplest extrapolation:}

\[\text{assume we have } y_n \text{ and } y_{n/2} \text{ (both } O(h^2)\text{)} \]

\[\Rightarrow y(T) = \frac{4y_n - y_{n/2}}{3} \quad \text{this estimate is} \]
\[4\text{th order accurate!} \]
Rational Functions:

\[ Y_n = \frac{P_m}{Q_n} = \frac{a_0 + \ldots + a_m x^m}{b_0 + \ldots + b_n x^n} \]

Using the trick with small differences:

\( \Delta_{m,k} = Y_{k \ldots (k+m)} - Y_{k \ldots (k+m-1)} \)

\( \Theta_{m,k} = Y_{k \ldots (k+m)} - Y_{(k+1) \ldots (k+m)} \)

\( \rightarrow \) recurrence relation:

\[ \Theta_{m+1,k} = \frac{\left( \Delta_{m(k-1)} - \Theta_{m,k} \right) \Delta_{m(k+1)}}{h_k \Theta_{m,k} - \Delta_{m(k+1)}} \]

\[ \Delta_{m+1,k} = \frac{h_k}{h_{m+k+1}} \Theta_{m,k} \left( \Delta_{m(k+1)} - \Theta_{m,k} \right) \]

I. C. \( \Theta_{0,k} = \Delta_{0,k} = Y(h_k) \)

\( h_k = \frac{T}{n_k} \)

\( n_k = 2, 4, 6, 8, 10, 12, 14, \ldots \)
At each level $m$, $\Delta$, $\Theta$ are the corrections that give extrapolation one order higher.

The final answer $Y_{\text{final}}$ is equal to the sum of any element $Y_i$ + a set of $\Delta$ and/or $\Theta$ to get you thru the tree!

**Stiffness**

- Problem with a different time scale in ODE
- Occurs whenever there are more than one 1st-order differential eq.

Example:

- $u' = 998u + 1998v$
- $v' = -998u - 1999v$

with I.C.: $u(0) = 1$; $v(0) = 0$
By means of the transformation:

\[ u = 2y - z \quad u = -y + z \]

we find:

\[ m = 2e^{-x} - e^{-1000x} \]
\[ v = -e^{-x} + e^{-1000x} \]

\[ \uparrow \quad \uparrow \]

two very different time scales!

In real soln: the \( e^{-1000x} \) term will die away quickly! \( \Rightarrow \) Not important in soln for \( x \) large.

\[ \checkmark \text{ However, numerically it is unstable unless } h < \frac{1}{1000}! \]

\[ \Rightarrow \text{ To see this, let consider a simpler case} \]
\[ y_0 = 1 \]

\[ y' = -cy \quad c > 0 \quad \Rightarrow y(t) = e^{-ct} \]

Tuler scheme (other in principle similar)

\[ y_{n+1} = y_n + h y' = (1 - ch) y_n \]
This is explicit \( \Rightarrow \) \( Y_{nt1} \) is a fn of \( Y_n \).

\[ \text{**Note:**} \quad \text{\( Y_{nt1} \) will only be stable (not blow up)} \]

If \( |1-ch| < 1 \)

\[ \left\{ \begin{array}{l}
1-ch < 1 \\
\frac{h}{20} \\
1-ch > -1 \\
-ch > -2 
\end{array} \right. \Rightarrow 0 < h < \frac{2}{c} \]

(\( h < \frac{2}{c} \) to be stable) (2/100)

A simple cure is to use implicit differencing:

backward Euler scheme:

\[
Y_{nt1} = Y_n + h Y_{nt1}' \\
Y_{nt1} = \frac{Y_n}{1+ch}
\]

In this case, \( \frac{1}{1+ch} < 1 \) for all \( h > 0 \)

And, \( Y_{nt1} \) is stable, i.e. \( Y_{nt1} \to 0 \) as \( n \to \infty \)

as required by the actual soln:

\( Y = e^{-ct} \to 0 \) as \( t \to \infty \).

\( \text{Stability for explicit method is true for linear system} \)

and it gives better performance in general.
For general nonlinear systems:

\[ \chi' = f(\chi) \]

\[ \Rightarrow \quad \chi_{n+1} = \chi_n + h f(\chi_n) \]

\[ \Rightarrow \quad \text{One can solve the implicit equation directly} \]

or \underline{Linearize}:

\[ \chi_{n+1} = \chi_n + h \left[ f(\chi_n) + Df|_{\chi_n} \cdot (\chi_{n+1} - \chi_n) \right] \]

\[ [1 - h Df](\chi_{n+1} - \chi_n) = h f(\chi_n) \]

\[ \Rightarrow \quad \chi_{n+1} = \chi_n + h \left[ 1 - h Df \right]^{-1} f(\chi_n) \]

\[ \uparrow \]

Cost of implicit \( \rightarrow \) need to evaluate inverse!

higher order implicit methods:

Generalization of RK4 \( \rightarrow \) Rosenbrock

of BS \( \rightarrow \) Bader and Empfindhard

[Press]
The Leapfrog Integrator

Let consider a 2nd order ODE (e.g. Newton's Equation) where the acceleration per unit mass of a particle is given by \( f(x) \), i.e.

\[
\frac{d^2 x}{dt^2} = f(x) \quad \Rightarrow \quad \left\{ \begin{array}{l}
\frac{dx}{dt} = v \\
\frac{dv}{dt} = f(x)
\end{array} \right.
\]

The Leapfrog integrator for this system can be defined as:

\[
\begin{align*}
X_{i+1} &= X_i + V_{i+\frac{1}{2}} \Delta t \\
V_{i+\frac{1}{2}} &= V_i + \frac{1}{2} \Delta t \left( f(X_{i+1}) \right) \\
V_{i+1} &= V_i + \frac{1}{2} \Delta t \left( f(X_{i+1}) \right)
\end{align*}
\]

where \( V(t) = \frac{dx}{dt}(t) \)

\( V_{i+1/2} = V(t + \frac{1}{2} \Delta t) \)
Note:

- symmetric with respect to the ways that $x$ and $v$ are advanced in time.

\[ x_i \rightarrow x_{i+1} \text{ with } v_{i+1/2} \text{ (at midpoint)} \]

\[ v_{i+1/2} \rightarrow v_{i+3/2} \text{ with } x_{i+1} \text{ (at midpoint)} \]

- $x$ and $v$ advance in a staggered leapfrog manner.

- need "self-starting":

- to get $x_i$, one needs $v_{1/2}$

from Euler, RK4, etc.

- $x(t)$ is 2nd order in accuracy:

\[ x(t+dt) = x(t) + \left[ v(t) + \frac{1}{2} dt f(t) + O(dt^2) \right] dt \]

\[ x(t+dt) = x(t) + v(t) dt + \frac{1}{2} f(t) dt^2 + O(dt^3) \]
Leapfrog is time-reversible!

One can invert the iteration \( i \) explicitly,

\[
\begin{align*}
\frac{v_{i+1/2}}{v_i} + f(x_{i+1})(-dt) \\
\frac{x_{i+1}}{x_i} + v_{i+1/2}(-dt)
\end{align*}
\]

These are precisely the steps (the same functional evaluations) that we took to advance the system in the first place.

\( +dt \rightarrow -dt \), we get back to exactly the same starting point!

Note: Euler or RK4 - the derivatives are evaluated not symmetrically at different times.

E.g. Euler:

forward:

\[
\begin{align*}
\frac{x_{i+1}}{x_i} = & \quad x_i + v_i dt \\
\frac{v_{i+1}}{v_i} = & \quad v_i + f(x_i) dt
\end{align*}
\]

backward:

\[
\begin{align*}
\frac{v_i}{v_{i+1}} = & \quad v_{i+1} + f(x_{i+1}) dt \\
\frac{x_i}{x_{i+1}} = & \quad x_{i+1} + v_{i+1}(-dt)
\end{align*}
\]
Time reversibility in \textit{Leapfrog} is explicit!

In Euler RK4, time reversibility is only approximated (up to prescribed precision).

- Recall that in a physical system, the total energy of the system is conserved \textit{iff} its Hamiltonian is time invariant.

- Thus, for problems where the explicit conservation of energy is required, explicit time reversibility in the ODE solver is important. E.g., long-time solutions of particles in a conservative force field.

- Another way to say this is that:

  \textit{Euler/RK4} has intrinsic numerical dissipation, while the \textit{Leapfrog} method is "amplitude" conserving.