Numerical methods for dynamical systems

Alexandre Chapoutot

ENSTA Paris
master CPS IP Paris

2020-2021
Differential equations

Many classes

- Ordinary Differential Equations (ODE)
  \[ \dot{y}(t) = f(t, y(t)) \]

- Differential-Algebraic equations (DAE)
  \[ \dot{y}(t) = f(t, y(t), x(t)) \]
  \[ 0 = g(t, y(t), x(t)) \]

- Delay Differential Equations (DDE)
  \[ \dot{y}(t) = f(t, y(t), y(t - \tau)) \]

- and others: partial differential equations, etc.

Remark

This talk focuses on ODE
High order vs first order and non-autonomous vs autonomous

- **High order vs first order**

\[ \ddot{y} = f(y, \dot{y}) \iff \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \begin{pmatrix} y_2 \\ f(y_1, y_2) \end{pmatrix} \text{ with } y_1 = y \text{ and } y_2 = \dot{y} . \]

- **Non-autonomous vs autonomous**

\[ \dot{y} = f(t, y) \iff \dot{z} = \begin{pmatrix} \dot{t} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} 1 \\ f(t, y) \end{pmatrix} = g(z) . \]
Consider an IVP for ODE, over the time interval \([0, t_{\text{end}}]\)

\[
\dot{y} = f(t, y) \quad \text{with} \quad y(0) = y_0
\]

IVP has a unique solution \(y(t; y_0)\) if \(f : \mathbb{R}^n \to \mathbb{R}^n\) is Lipschitz in \(y\)

\[
\forall t, \forall y_1, y_2 \in \mathbb{R}^n, \exists L > 0, \quad \| f(t, y_1) - f(t, y_2) \| \leq L \| y_1 - y_2 \|
\]

Goal of numerical integration

- Compute a sequence of time instants: \(t_0 = 0 < t_1 < \cdots < t_n = t_{\text{end}}\)
- Compute a sequence of values: \(y_0, y_1, \ldots, y_n\) such that

\[
\forall \ell \in [0, n], \quad y_\ell \approx y(t_\ell; y_0)
\]

- s.t. \(y_{\ell+1} \approx y(t_\ell + h; y_\ell)\) with an error \(O(h^{p+1})\) where
  - \(h\) is the integration step-size
  - \(p\) is the order of the method
### Simulation Algorithm

**Data:** $f$ the flow, $y_0$ initial condition, $t_0$ starting time, $t_{\text{end}}$ end time, $h$ integration step-size

$t \leftarrow t_0$;

$y \leftarrow y_0$;

**while** $t < t_{\text{end}}$ **do**

- Print($t$, $y$);
- $y \leftarrow \text{Euler}(f, t, y, h)$;
- $t \leftarrow t + h$;

**end**

with, the Euler's method defined by

$$y_{n+1} = y_n + hf(t_n, y_n) \quad \text{and} \quad t_{n+1} = t_n + h.$$
One-step methods: Runge-Kutta family

1. One-step methods: Runge-Kutta family
2. Building Runge-Kutta methods
3. Variable step-size methods
4. Solving algebraic equations in IRK
5. Implementation in Python
6. Special cases: symplectic integrator
Examples of Runge-Kutta methods

Single-step fixed step-size explicit Runge-Kutta method
e.g. explicit Trapezoidal method (or Heun’s method)\(^1\) is defined by:

\[
k_1 = f(t_\ell, y_\ell), \quad k_2 = f(t_\ell + \ell h, y_\ell + \ell h k_1)
\]

\[
y_{i+1} = y_\ell + h\left(\frac{1}{2} k_1 + \frac{1}{2} k_2\right)
\]

**Intuition**
- \(\dot{y} = t^2 + y^2\)
- \(y_0 = 0.46\)
- \(h = 1.0\)

dotted line is the exact solution.

\(^1\)example coming from “Geometric Numerical Integration”, Hairer, Lubich and Wanner, 2006.
Examples of Runge-Kutta methods

Single-step variable step-size explicit Runge-Kutta method

e.g. Bogacki-Shampine (ode23) is defined by:

\[ k_1 = f(t_n, y_n) \]
\[ k_2 = f(t_n + \frac{1}{2} h_n, y_n + \frac{1}{2} h k_1) \]
\[ k_3 = f(t_n + \frac{3}{4} h_n, y_n + \frac{3}{4} h k_2) \]
\[ y_{n+1} = y_n + h \left( \frac{2}{9} k_1 + \frac{1}{3} k_2 + \frac{4}{9} k_3 \right) \]
\[ k_4 = f(t_n + h_n, y_{n+1}) \]
\[ z_{n+1} = y_n + h \left( \frac{7}{24} k_1 + \frac{1}{4} k_2 + \frac{1}{3} k_3 + \frac{1}{8} k_4 \right) \]

Remark: the step-size \( h \) is adapted following \( \| y_{n+1} - z_{n+1} \| \)

\[ \begin{array}{cccc}
0 & 1/2 & 1/2 & 0 \\
1/2 & 3/4 & 0 & 3/4 \\
3/4 & 1 & 4/9 & 4/9 \\
1 & 2/9 & 1/3 & 4/9 \\
2/9 & 2/3 & 9 & 9 \\
7/24 & 1/4 & 1/3 & 1/8
\end{array} \]

\(^{1}\)example coming from “Geometric Numerical Integration”, Hairer, Lubich and Wanner, 2006.
Examples of Runge-Kutta methods

Single-step fixed step-size implicit Runge-Kutta method

e.g. Runge-Kutta Gauss method (order 4) is defined by:

\[
\begin{align*}
  k_1 &= f \left( t_n + \left( \frac{1}{2} - \frac{\sqrt{3}}{6} \right) h_n, \quad y_n + h \left( \frac{1}{4} k_1 + \left( \frac{1}{4} - \frac{\sqrt{3}}{6} \right) k_2 \right) \right) \quad (1a) \\
  k_2 &= f \left( t_n + \left( \frac{1}{2} + \frac{\sqrt{3}}{6} \right) h_n, \quad y_n + h \left( \left( \frac{1}{4} + \frac{\sqrt{3}}{6} \right) k_1 + \frac{1}{4} k_2 \right) \right) \quad (1b) \\
  y_{n+1} &= y_n + h \left( \frac{1}{2} k_1 + \frac{1}{2} k_2 \right) \quad (1c)
\end{align*}
\]

Remark: A non-linear system of equations must be solved at each step.

\[\text{\footnotesize \textsuperscript{1} example coming from “Geometric Numerical Integration”, Hairer, Lubich and Wanner, 2006.}\]
Runge-Kutta methods

$s$-stage Runge-Kutta methods are described by a Butcher tableau:

\[
\begin{array}{c|cccc}
  c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\
  \vdots & \vdots & \vdots & & \vdots \\
  c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\
  \hline
  b_1 & b_2 & \cdots & b_s \\
  b'_1 & b'_2 & \cdots & b'_s & (optional)
\end{array}
\]

Which induces the following recurrence formula:

\[
k_i = f \left( t_n + c_i h_n, y_n + h \sum_{j=1}^{s} a_{ij} k_j \right) \quad y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i \quad (2)
\]

- **Explicit** method (ERK) if \( a_{ij} = 0 \) is \( i \leq j \)
- **Diagonal Implicit** method (DIRK) if \( a_{ij} = 0 \) is \( i \leq j \) and at least one \( a_{ii} \neq 0 \)
- **Singly Diagonal implicit** method (SDIRK) if \( a_{ij} = 0 \) is \( i \leq j \) and all \( a_{ii} = \gamma \) are identical.
- **Implicit** method (IRK) otherwise
Building Runge-Kutta methods

1. One-step methods: Runge-Kutta family
2. Building Runge-Kutta methods
3. Variable step-size methods
4. Solving algebraic equations in IRK
5. Implementation in Python
6. Special cases: symplectic integrator
Every numerical method member of the Runge-Kutta family follows the condition order.

**Order condition**

This condition states that a method of this family is of order $p$ if and only if the $p + 1$ first coefficients of the Taylor expansion of the true solution and the Taylor expansion of the numerical methods are equal.

In other terms, a RK method has order $p$ if

$$y(t_n; y_{n-1}) - y_n = h^{p+1} \psi_f(y_n) + O(h^{p+2})$$
At a time instant $t_n$ the Taylor expansion of the true solution with the Lagrange remainder states that there exists $\xi \in ]t_n, t_{n+1}[$ such that:

$$y(t_{n+1}; y_0) = y(t_n; y_0) + \sum_{i=1}^{p} \frac{h_n^i}{i!} y^{(i)}(t_n; y_0) + O(h^{p+1})$$

$$= y(t_n; y_0) + \sum_{i=1}^{p} \frac{h_n^i}{i!} f(i-1)(t_n, y(t_n; y_0)) + O(h^{p+1})$$

The Taylor expansion (very complex expression) of the numerical solution is given by expanding, around $(t_n, y_n)$, the expression:

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i$$

Consequence of the condition order

The definition of RK methods (Butcher table coefficients) is based on the solution of under-determined system of algebraic equations.
Example: 3-stages explicit RK method (scalar IVP)

One considers a **scalar** ODE $\dot{y} = f(t, y)$ with $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$.

One tries to determine the coefficients $b_i \ (i = 1, 2, 3)$, $c_2$, $c_3$, $a_{32}$ such that

$$y_{n+1} = y_n + h(b_1 k_1 + b_2 k_2 + b_3 k_3)$$
$$k_1 = f(t_n, y_n)$$
$$k_2 = f(t_n + c_2 h, y_n + hc_2 k_1)$$
$$k_3 = f(t_n + c_3 h, y_n + h(c_3 - a_{32})k_1 + ha_{32} k_2)$$

Some notations (evaluation at point $(t_n, y(t_n))$):

$$f = f(t, y) \quad f_t = \frac{\partial f(t, y)}{\partial t} \quad f_{tt} = \frac{\partial^2 f(t, y)}{\partial t^2} \quad f_{ty} = \frac{\partial f(t, y)}{\partial t \partial y} \quad \ldots$$

**Note:** in Butcher tableau we always have the **row-sum condition**

$$c_i = \sum_{j=1}^{s} a_{ij}, \quad i = 1, 2, \ldots, s.$$
Taylor expansion of $y(t_{n+1})$, the exact solution, around $t_n$:

$$y(t_{n+1}) = y(t_n) + hy^{(1)}(t_n) + \frac{h^2}{2} y^{(2)}(t_n) + \frac{h^3}{6} y^{(3)}(t_n) + \mathcal{O}(h^4)$$

Moreover,

$$y^{(1)}(t_n) = f$$
$$y^{(2)}(t_n) = f_t + f_y \dot{y} = f_t + ff_y$$
$$y^{(3)}(t_n) = f_{tt} + f_{ty} f + f (f_{ty} + f_{yy} f) + f_y (f_y + ff_y)$$
$$= f_{tt} + 2ff_{ty} + f^2 f_{yy} + f_y (f_t + ff_y)$$

With $F = f_t + ff_y$ and $G = f_{tt} + 2ff_{ty} + f^2 f_{ty}$, one has:

$$y(t_{n+1}) = y(t_n) + hf + \frac{h^2}{2} F + \frac{h^3}{6} (Ff_y + G) + \mathcal{O}(h^4)$$
Example: 3-stages explicit RK method (scalar IVP)

Taylor expansion $k_i$ around $t_n$

$$k_2 = f + h c_2 \left( f_t + k_1 f_y \right) + \frac{h^2}{2} c_2^2 \left( f_{tt} + 2 k_1 f_{ty} + k_1^2 f_{yy} \right) + O(h^3)$$

$$= f + h c_2 F + \frac{h^2}{2} c_2^2 G + O(h^3)$$

$$k_3 = f + h \left\{ c_3 f_t + [(c_3 - a_{32}) k_1 + a_{32} k_2] f_y \right\}$$

$$+ \frac{h^2}{2} \left\{ c_3^2 f_{tt} + 2 c_3 [(c_3 - a_{32}) k_1 + a_{32} k_2] f_{ty}$$

$$+ [(c_3 - a_{32}) k_1 + a_{32} k_2]^2 f_{yy} \right\} + O(h^3)$$

$$= f + h c_3 F + h^2 (c_2 a_{32} F f_y + \frac{1}{2} c_3^2 G + O(h^3) \ (\text{substituting } k_1 = f \text{ and } k_2)$$

Taylor expansion of $y_{n+1}$ (localizing assumption $y_n = y(t_n)$)

$$y_{n+1} = y(t_n) + h (b_1 + b_2 + b_3) f + h^2 (b_2 c_2 + b_3 c_3) F$$

$$+ \frac{h^3}{2} \left[ 2 b_3 c_2 a_{32} F f_y + (b_2 c_2^2 + b_3 c_3^2) G \right] + O(h^4)$$
Building one stage method

We fix $b_2 = b_3 = 0$, so one gets

$$y_{n+1} = y(t_n) + hb_1 f + O(h^2)$$

In consequence $b_1 = 1$ (by identification) so one gets Euler’s method (order 1)
Building two stages method

We fix $b_3 = 0$, so one gets

$$y_{n+1} = y(t_n) + h(b_1 + b_2)f + h^2 b_2 c_2 F + \frac{1}{2} h^3 b_2 c_2^2 G + O(h^3)$$

In consequence to get order 2 methods, we need to solve

$$b_1 + b_2 = 1 \quad b_2 c_2 = \frac{1}{2}$$

Remark: there is a (singly) infinite number of solutions.

Two particular solutions of order 2:

\[
\begin{array}{cc|cc}
0 & 1 & \frac{1}{2} & 0 \\
1 & \frac{1}{2} & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{cc|cc}
0 & 1 & \frac{1}{2} & \frac{1}{2} \\
1 & \frac{1}{2} & \frac{1}{2} & 0 \\
\end{array}
\]
Building three stages method

In consequence to get order 3 methods, we need to solve

\[ b_1 + b_2 + b_3 = 1 \quad b_2 c_2 + b_3 c_3 = \frac{1}{2} \]
\[ b_2 c_2^2 + b_3 c_3^2 = \frac{1}{3} \quad b_3 c_2 a_{32} = \frac{1}{6} \]

**Remark:** there is a (doubly) infinite number of solutions.

Two particular solutions of order 3:

\[
\begin{array}{ccc|ccc}
0 & 1/3 & 1/3 & 0 & 2/3 & 0 \\
1/3 & 1/3 & 2/3 & \frac{1}{4} & 0 & \frac{3}{4} \\
\hline
\end{array}
\]

\[
\begin{array}{ccc|ccc}
0 & 1/2 & 1/2 & 1 & -1 & 2 \\
\frac{1}{2} & \frac{1}{2} & \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \\
\hline
\end{array}
\]
Relation between order and stage

**Limitation of ERK**

$s$-stage ERK method cannot have order greater than $s$

Moreover, this upper bound is reached for order less or equal to 4. For now, we only know:

<table>
<thead>
<tr>
<th>order</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>[12, 17]</td>
<td>[13, 17]</td>
</tr>
<tr>
<td>cond</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>17</td>
<td>37</td>
<td>85</td>
<td>200</td>
<td>486</td>
<td>1205</td>
</tr>
</tbody>
</table>

**Remark:** order 10 is highest order known for an ERK (with rational coefficients).

**Optimal order for IRK methods**

We know $s$-stage method having order $2s$ (Gauss’ methods).
Note on building IRK Gauss’ method

\[ \dot{y} = f(y) \quad \text{with} \quad y(0) = y_0 \iff y(t) = y_0 + \int_{t_n}^{t_{n+1}} f(y(s)) ds \]

We solve this equation using quadrature formula.

IRK Gauss method is associated to a collocation method (polynomial approximation of the integral) such that for \( i, j = 1, \ldots, s \):

\[ a_{ij} = \int_0^{c_i} \ell_j(t) dt \quad \text{and} \quad b_j = \int_0^1 \ell_j(t) dt \]

with \( \ell_j(t) = \prod_{k \neq j} \frac{t-c_k}{c_j-c_k} \) the Lagrange polynomial.

And the \( c_i \) are chosen as the solution of the Shifted Legendre polynomial of degree \( s \):

\[ P_s(x) = (-1)^s \sum_{k=0}^{s} \binom{s}{k} \binom{s+k}{s} (-x)^k \]

1, \( x \), \( 0.5(3x^2 - 1) \), \( 0.5(5x^3 - 3x) \), etc.
Example (order 3): Radau family ($2s - 1$)

Based on different polynomial, one can build different IRK methods with a particular structure. For examples, Radau family consider as endpoints of time interval either 0 or 1.

**Radau IA** (0 endpoint)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\frac{1}{4}$</td>
<td>$-\frac{1}{4}$</td>
</tr>
<tr>
<td>$\frac{2}{3}$</td>
<td>$\frac{1}{4}$</td>
<td>$\frac{5}{12}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\frac{1}{4}$</td>
<td>$\frac{3}{4}$</td>
</tr>
</tbody>
</table>

**Radau IIA** (1 endpoint)

<table>
<thead>
<tr>
<th></th>
<th>$\frac{5}{12}$</th>
<th>$-\frac{1}{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\frac{1}{3}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$\frac{3}{4}$</td>
<td>$\frac{1}{4}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$\frac{3}{4}$</td>
<td>$\frac{1}{4}$</td>
</tr>
</tbody>
</table>
Example (order 4): Lobatto family \((2s − 2)\)

Based on different polynomial, one can build different IRK methods with a particular structure. For examples, Lobatto family always consider 0 and 1 as endpoints of time interval.

### Lobatto IIIA

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/2</td>
<td>5/24</td>
<td>1/3</td>
<td>1/24</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1/6</td>
<td>2/3</td>
<td>1/6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1/6</td>
<td>2/3</td>
<td>1/6</td>
<td></td>
</tr>
</tbody>
</table>

### Lobatto IIIB

<table>
<thead>
<tr>
<th></th>
<th>1/6</th>
<th>−1/6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/2</td>
<td>1/6</td>
<td>1/3</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1/6</td>
<td>5/6</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1/6</td>
<td>2/3</td>
<td>1/6</td>
</tr>
</tbody>
</table>

### Lobatto IIIC

<table>
<thead>
<tr>
<th></th>
<th>1/6</th>
<th>−1/3</th>
<th>1/6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/2</td>
<td>1/6</td>
<td>5/12</td>
<td>−1/12</td>
</tr>
<tr>
<td>1</td>
<td>1/6</td>
<td>2/3</td>
<td>1/6</td>
</tr>
<tr>
<td></td>
<td>1/6</td>
<td>2/3</td>
<td>1/6</td>
</tr>
</tbody>
</table>
Variable step-size methods

1. One-step methods: Runge-Kutta family

2. Building Runge-Kutta methods

3. Variable step-size methods

4. Solving algebraic equations in IRK

5. Implementation in Python

6. Special cases: symplectic integrator
Local error estimation in ERK

**Goal:** monitoring the step length to
- increase it if the norm of the error is below a given tolerance (with a factor)
- decrease it if the norm of the error is above a given tolerance

The trade-off between Accuracy versus Performance

**An old solution:** *Richardson extrapolation*, from a method of order $p$:
- solve the IVP for one step $h$ to get $\tilde{y}_n$
- solve the IVP for two steps $h/2$ to get $\tilde{z}_n$
- error estimation if given by: $(\tilde{z}_n - \tilde{y}_n)/(2^{p+1} - 1)$

Drawback: time consuming

**Other approach**

embedding two ERK (or suitable IRK) methods of order $p$ and $p + 1$ and compute the difference, as

$$y_{n+1} - y_{n+1} = [y(t_{n+1}) - y_{n+1}] - [y(t_{n+1}) - y^*_{n+1}] = h^{p+1}\psi_f(y_n) + O(h^{p+2})$$

with $y_{n+1}$ solution of order $p$ and $y^*_{n+1}$ solution of order $p^* > p$
Example: explicit Runge-Kutta-Fehlberg (RKF45)

Fehlberg’s method of order 4 and 5

<table>
<thead>
<tr>
<th></th>
<th>(1/4)</th>
<th>(1/4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3/8)</td>
<td>(3/32)</td>
<td>(9/32)</td>
</tr>
<tr>
<td>(12/13)</td>
<td>(1932/2197)</td>
<td>(-7200/2197)</td>
</tr>
<tr>
<td>(1)</td>
<td>(439/216)</td>
<td>(-8)</td>
</tr>
<tr>
<td>(1/2)</td>
<td>(-8/27)</td>
<td>(2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>(25/216)</th>
<th>(0)</th>
<th>(1408/2565)</th>
<th>(2197/4104)</th>
<th>(-1/5)</th>
<th>(0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(16/135)</td>
<td>(0)</td>
<td>(-128/4275)</td>
<td>(-2197/75240)</td>
<td>(1/50)</td>
<td>(2/55)</td>
<td></td>
</tr>
</tbody>
</table>

Remark

- coefficient chosen to minimize the coefficient of the Taylor expansion remainder
Example: explicit DOPRI54

Dormand-Prince’s method of order 5 and 4

<table>
<thead>
<tr>
<th>0</th>
<th>1/5</th>
<th>1/5</th>
<th>3/40</th>
<th>9/40</th>
<th>44/55</th>
<th>56/15</th>
<th>32/9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5/6</td>
<td>5/6</td>
<td>3/40</td>
<td>9/40</td>
<td>44/55</td>
<td>56/15</td>
<td>32/9</td>
</tr>
<tr>
<td>8</td>
<td>19372/6561</td>
<td>25360/6561</td>
<td>64448/6561</td>
<td>212/729</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9017/3168</td>
<td>355/33</td>
<td>46732/5247</td>
<td>49/176</td>
<td>5103/18656</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>35/384</td>
<td>0</td>
<td>500/1113</td>
<td>125/192</td>
<td>2187/6784</td>
<td>11/84</td>
<td></td>
</tr>
</tbody>
</table>

| 5179/57600 | 0 | 7571/16695 | 393/640 | 92097/339200 | 187/2100 | 1/40 |
| 35/384 | 0 | 500/1113 | 125/192 | 2187/6784 | 11/84 | 0 |

Remarks

- 7 stage for an order 5 method but **FSAL (First Same As Last)**
- **Local extrapolation** order 5 approximation is used to solve the next step
### Example (order 4): SDIRK Family

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>(\frac{1}{4})</th>
<th>(\frac{1}{4})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\frac{1}{4})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{2})</td>
<td>(\frac{1}{4})</td>
</tr>
<tr>
<td>(\frac{1}{20})</td>
<td>(\frac{17}{50})</td>
<td>(-\frac{1}{25})</td>
<td>(\frac{1}{4})</td>
</tr>
<tr>
<td>(\frac{1}{2})</td>
<td>(\frac{371}{1360})</td>
<td>(-\frac{137}{2720})</td>
<td>(\frac{15}{544})</td>
</tr>
<tr>
<td>(1)</td>
<td>(\frac{25}{24})</td>
<td>(-\frac{49}{48})</td>
<td>(\frac{125}{16})</td>
</tr>
</tbody>
</table>

\[
\begin{array}{cccc}
\frac{25}{24} & -\frac{49}{48} & \frac{125}{16} & -\frac{85}{12} & 1 \\
\frac{59}{48} & -\frac{17}{96} & \frac{225}{32} & -\frac{85}{12} & 0 \\
\end{array}
\]

**Remarks:**
- This an embedded implicit RK method (difficult to find one for IRK)
- Simplification of the iteration to solve the fixpoint equations
Step size control - simple case

Simple strategy:

\[
\text{err} = \| \mathbf{y}_{n+1} - \mathbf{z}_{n+1} \|
\]

with \( \mathbf{y}_{n+1} \) the approximation of order \( p \) and \( \mathbf{z}_{n+1} \) the approximation of order \( p + 1 \).

Simple step-size update strategy

From an user defined tolerance TOL:

- if \( \text{err} > \text{TOL} \) then solve IVP with \( h/2 \)
- if \( \text{err} \leq \frac{\text{TOL}}{2^{p+1}} \) then solve IVP with \( 2h \)
Validation of the integration step

For adaptive step-size method: for all continuous state variables

\[ \text{err} = \| y_{n+1} - z_{n+1} \| \leq \max (\text{atol}, \text{rtol} \times \max (\| y_{n+1} \|, \| y_n \|)) \cdot \]

**Estimated error** \hspace{1cm} **Maximal acceptable error**

**Note:** different norms can be considered.

Strategy:

- **Success:** may increase the step-size: \( h_{n+1} = h_n \times \sqrt[\frac{p+1}{\text{err}}} \) (\( p \) is the minimal order of the embedded methods).

- **Failure:** reduce the step-size \( h_n \) in general only a division by 2, and restart the integration step with the new step-size.

**Remark**

The reduction of the step-size is done until the \( h_{\min} \) is reached. In that case a simulation error may happen.
Some care is necessary to reduce probability the next step is rejected:

\[ h_{n+1} = h_n \min \left( \text{facmax}, \max \left( \text{facmin}, \text{fac}^{p+1} \sqrt{1/\text{err}} \right) \right) \]

and to prevent that \( h \) is increased or decreased too quickly.

Usually:

- \( \text{fac} = 0.8, 0.9, (0.25)^{1/(p+1)}, (0.38)^{1/(p+1)} \)
- \( \text{facmax} \) is between 1.5 and 5
- \( \text{facmin} \) is equal to 0.5

**Note**

after a rejection (i.e., a valid step coming from a reject step) it is advisable to let \( h \) unchanged.
Solving algebraic equations in IRK

1. One-step methods: Runge-Kutta family

2. Building Runge-Kutta methods

3. Variable step-size methods

4. Solving algebraic equations in IRK

5. Implementation in Python

6. Special cases: symplectic integrator
The $k_i, i = 1, \ldots, s$, form a nonlinear system of equations in,

$$k_i = f \left( t_n + c_i h_n, y_n + h \sum_{j=1}^{s} a_{ij} k_j \right) \quad y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i$$

### Theorem: existence and uniqueness of the solution

Let $f : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ be continuous and satisfy a Lipschitz conditions with constant $L$ (w.r.t. $y$). If

$$h < \frac{1}{L \max_i \sum_j |a_{ij}|}$$

there exists a unique solution which can be obtained by iteration.

**Remark:** in case of stiff problems (see lecture on DAE), larger value of $L$ is bad as it may cause numerical instability in iterations.
Quick remainder on Newton-Raphson methods

**Goal of the method**
finding zeroes of non-linear continuously differentiable functions $G : \mathbb{R}^n \rightarrow \mathbb{R}^n$

Based on the idea (in 1D) to approximate non-linear function by its tangent equation and starting from a sufficiently close solution $x_0$ we can produce an approximation $x_1$ closer to the solution, such that

$$x_1 = x_0 - J_G^{-1}(x_0)G(x_0)$$

where $J_G$ is the Jacobian matrix of $G$. This process is repeated until we are close enough

Usually instead of computing inverse of matrices, it is more efficient to solve the linear system (e.g., with LU decomposition)

$$J_G(x_0)\delta x = -G(x_0) \quad \text{with unknown} \quad \delta x = x_1 - x_0$$

and so $x_1 = x_0 + \delta x$
Reformulating non-linear system of $k_i$'s

Solution of the nonlinear system of equations using Newton’s method: first we can rewrite the system:

$$k_i = f \left( t_n + c_i h_n, y_n + h \sum_{j=1}^{s} a_{ij} k_j \right)$$

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i$$

with $k_i' = y_n + h \sum_{j=1}^{s} a_{ij} k_j$ into

$$k_i' = y_n + h \sum_{j=1}^{s} a_{ij} f(t_n + c_i h_n, k_j')$$

$$y_{n+1} = y_n + h \sum_{j=1}^{s} b_i f(t_n + c_i h_n, k_j')$$
Reformulating non-linear system of $k_i$'s

Next, let $z_i = k'_i - y_n$ we have:

$$
\begin{pmatrix}
    z_1 \\
    \vdots \\
    z_s
\end{pmatrix} =
\begin{pmatrix}
    a_{11} & \cdots & a_{1s} \\
    \vdots & \ddots & \vdots \\
    a_{s1} & \cdots & a_{ss}
\end{pmatrix}
\begin{pmatrix}
    \text{hf}(t_n + c_1 h_n, y_n + z_1) \\
    \vdots \\
    \text{hf}(t_n + c_s h_n, y_n + z_s)
\end{pmatrix}
$$

$$
\begin{pmatrix}
    z_1 \\
    \vdots \\
    z_s
\end{pmatrix} = hAF(z)
$$

hence, with $z^k$ the solution of Equation (3):

$$
y_{n+1} = y_n + \sum_{i=1}^{s} d_i z_i^k \quad \text{with} \quad (d_1, \cdots, d_s) = (b_1, \cdots, b_s)A^{-1}
$$

with $A = \{a_{ij}\}$ if $A$ is invertible (it is the case for Gauss' method).
Reformulating non-linear system of $k_i$’s

Now we have to solve:

$$g(z) = 0 \quad \text{with} \quad g(z) = z - hAF(z)$$

with Newton’s method where Jacobian matrix $\nabla g(z)$ of $g$ is:

$$\nabla g(z) = \begin{pmatrix} I - ha_{11}J(z_1) & -ha_{12}J(z_2) & \cdots & -ha_{1s}J(z_s) \\ -ha_{21}J(z_1) & I - ha_{22}J(z_2) & \cdots & -ha_{2s}J(z_s) \\ \vdots & \vdots & \ddots & \vdots \\ -ha_{1s}J(z_1) & -ha_{2s}J(z_2) & \cdots & I - ha_{ss}J(z_s) \end{pmatrix}$$

with $J(z_i) = \frac{\partial f}{\partial y}(t_n + c_i h_n, y_n + z_i)$. And the Newton iteration is defined by:

$$z^{k+1} = z^k + p_k \quad \text{with} \quad p_k \text{ solution of } \nabla g(z^k)p = -g(z^k)$$

Remarks: Usually we use $\frac{\partial f}{\partial y}(t_n, y_n) \approx \frac{\partial f}{\partial y}(t_n + c_i h_n, y_n + z_i)$ and we have strategy to update the computation of $\nabla g(z)$
Implementation in Python

1. One-step methods: Runge-Kutta family
2. Building Runge-Kutta methods
3. Variable step-size methods
4. Solving algebraic equations in IRK
5. Implementation in Python
6. Special cases: symplectic integrator
def euler_one_step (f, t, y, h):
    return y + h * f(t, y)

def heun_one_step (f, t, y, h):
y1 = euler_one_step (f, t, y, h)
    return y + h * 0.5 * (f(t, y) + f(t+h, y1))

def solve (f, t0, y0, tend, nsteps):
h = (tend - t0) / nsteps
    time = np.linspace(t0, tend, nsteps)
yn = y0
    y = []
    print(h)
    for t in time:
        y.append(yn)
        # change the method here
        yn = heun_one_step (f, t, yn, h)
    return [time, y]

def dynamic (t, y):
    return np.array([-y[1], y[0]])
Implementation of fixed step size IRK

```python
def backward_euler_one_step(f, t, y, h):
    yn = y; err = 10000000
    while (err > 1e-14):
        yn1 = y + h * f(t + h, yn)
        err = LA.norm(yn1 - yn, 2)
        yn = yn1
    return yn1

def implicit_trapezoidal_one_step(f, t, y, h):
    aux = lambda yn: y + h * 0.5 * (f(t, y) + f(t+h, yn)) - yn
    res = fsolve(aux, y)
    return res

def solve(f, t0, y0, tend, nsteps):
    h = (tend - t0) / nsteps
    time = np.linspace(t0, tend, nsteps)
    yn = y0; y = []
    for t in time:
        y.append(yn)
        yn = implicit_trapezoidal_one_step(f, t, yn, h)
    return [time, y]

def dynamic(t, y):
    return np.array([-y[1], y[0]])

[t, y] = solve(dynamic, 0.0, np.array([1., 0.]), 2*np.pi*10, 100)
```
Implementation of variable step size ERK

```python
def heun_euler_one_step(f, t, y, h):
    k1 = f(t, y); k2 = f(t + h, y + h * k1); ynp1 = y + h * 0.5 * (k1 + k2)
    znp1 = y + h * k1; err = ynp1 - znp1
    return (ynp1, err)

def compute_h(err, hn, order, tolerance):
    if LA.norm(err, 2) <= (tolerance / pow(2.0, order + 1)):
        return 2 * hn
    else:
        return hn

def solve(f, t0, y0, tend, tolerance):
    t = t0; yn = y0; hn = 0.5; y = [y0]; time = [t0]; h = [hn]
    while t <= tend:
        (yn, err) = heun_euler_one_step(f, t, yn, hn)
        if LA.norm(err, 2) <= tolerance:
            y.append(yn); t = t + hn; time.append(t)
            hn = compute_h(err, hn, 1, tolerance); h.append(hn)
        else:
            hn = hn / 2.0
    return [time, y, h]

def dynamic(t, y):
    return np.array([-y[1], y[0]])

[t, y, h] = solve(dynamic, 0.0, np.array([1.0, 0.0]), 2*np.pi*10, 1e-2)
```
Special cases: symplectic integrator

1. One-step methods: Runge-Kutta family
2. Building Runge-Kutta methods
3. Variable step-size methods
4. Solving algebraic equations in IRK
5. Implementation in Python
6. Special cases: symplectic integrator
We consider **conservative** (i.e., energy is preserved) Hamiltonian dynamical systems of the form

\[ H(q, p) = K(p) + V(q) \]

where \( H \) the Hamiltonian, \( K \) is the kinetic energy and \( V \) is the potential energy.

And so can be write as

\[
\begin{align*}
\frac{dq}{dt} &= \frac{\partial H}{\partial p} \\
\frac{dp}{dt} &= -\frac{\partial H}{\partial q}
\end{align*}
\]

**Classical example: harmonic oscillator**

We have

\[ H = \frac{1}{2} p^2 + \frac{1}{2} q^2 \]

so

\[
\begin{align*}
\frac{dq}{dt} &= p \\
\frac{dq}{dt} &= -q
\end{align*}
\]
Symplectic Euler’s method

- Applying directly explicit Euler’s method on conservative Hamiltonian system cannot guaranteed the preservation of energy along the simulation.
- But we can do a small change to make the Euler’s method symplectic i.e., energy preserving as

**Solution 1**

\[ q_{n+1} = q_n + \frac{\partial K}{\partial p}(p_n) \]
\[ p_{n+1} = p_n + \frac{\partial K}{\partial p}(q_{n+1}) \]

Note: \( q \) has to be solved first

**Solution 2**

\[ q_{n+1} = q_n + \frac{\partial K}{\partial p}(p_{n+1}) \]
\[ p_{n+1} = p_n + \frac{\partial K}{\partial p}(q_n) \]

Note: \( p \) has to be solved first

**Note:** In that case, it is a fixed step-size explicit order 1 method