Simulation and High-Performance Computing

Part 4: Higher-order Timestepping Methods

Steffen Börm

Christian-Albrechts-Universität zu Kiel

September 29th, 2020

Reminder: Explicit timestepping methods

Explicit Euler:

$$\tilde{y}(t+\delta) = y(t) + \delta\, f(t, y(t))$$

Reminder: Explicit timestepping methods

Explicit Euler:

$$\tilde{y}(t+\delta) = y(t) + \delta\, f(t, y(t))$$

$$\tilde{y}(t+\delta) = y(t) + \delta\, k_1,$$

$$k_1 := f(t, y(t)).$$

Reminder: Explicit timestepping methods

Explicit Euler:

$$\begin{aligned} \tilde{y}(t+\delta) &= y(t) + \delta\, f(t, y(t)) \\ \tilde{y}(t+\delta) &= y(t) + \delta\, k_1, \\ k_1 &:= f(t, y(t)). \end{aligned}$$

Runge's method:

$$\tilde{y}(t+\delta) = y(t) + \delta\, f(t + \tfrac{\delta}{2}, y(t) + \tfrac{\delta}{2} f(t, y(t)))$$

Reminder: Explicit timestepping methods

Explicit Euler:

$$\begin{aligned} \tilde{y}(t+\delta) &= y(t) + \delta\, f(t, y(t)) \\ \tilde{y}(t+\delta) &= y(t) + \delta\, k_1, \\ k_1 &:= f(t, y(t)). \end{aligned}$$

Runge's method:

$$\begin{aligned} \tilde{y}(t+\delta) &= y(t) + \delta\, f(t + \tfrac{\delta}{2}, y(t) + \tfrac{\delta}{2} f(t, y(t))) \\ \tilde{y}(t+\delta) &= y(t) + \delta\, k_2, \\ k_1 &:= f(t, y(t)), \\ k_2 &:= f(t + \tfrac{\delta}{2}, y(t) + \tfrac{\delta}{2} k_1). \end{aligned}$$

Reminder: Explicit timestepping methods

Explicit Euler:

$$\begin{aligned}
\tilde{y}(t+\delta) &= y(t) + \delta\, f(t, y(t))\\
\tilde{y}(t+\delta) &= y(t) + \delta\, k_1,\\
k_1 &:= f(t, y(t)).
\end{aligned}$$

Runge's method:

$$\begin{aligned}
\tilde{y}(t+\delta) &= y(t) + \delta\, f(t + \tfrac{\delta}{2}, y(t) + \tfrac{\delta}{2} f(t, y(t)))\\
\tilde{y}(t+\delta) &= y(t) + \delta\, k_2,\\
k_1 &:= f(t, y(t)),\\
k_2 &:= f(t + \tfrac{\delta}{2}, y(t) + \tfrac{\delta}{2} k_1).
\end{aligned}$$

Reminder: Implicit timestepping methods

Implicit Euler:

$$\tilde{y}(t+\delta) = y(t) + \delta\, f(t+\delta, \tilde{y}(t+\delta))$$

Reminder: Implicit timestepping methods

Implicit Euler:

$$\begin{aligned} \tilde{y}(t+\delta) &= y(t) + \delta\, f(t+\delta, \tilde{y}(t+\delta)) \\ \tilde{y}(t+\delta) &= y(t) + \delta\, k_1, \\ k_1 &:= f(t+\delta, \tilde{y}(t+\delta)), \end{aligned}$$

Reminder: Implicit timestepping methods

Implicit Euler:

$$
\begin{aligned}
\tilde{y}(t+\delta) &= y(t) + \delta\, f(t+\delta, \tilde{y}(t+\delta)) \\
\tilde{y}(t+\delta) &= y(t) + \delta\, k_1, \\
k_1 &:= f(t+\delta, \tilde{y}(t+\delta)), \\
k_1 &= f(t+\delta, y(t) + \delta\, k_1).
\end{aligned}
$$

Reminder: Implicit timestepping methods

Implicit Euler:

$$\begin{aligned}
\tilde{y}(t+\delta) &= y(t) + \delta\, f(t+\delta, \tilde{y}(t+\delta))\\
\tilde{y}(t+\delta) &= y(t) + \delta\, k_1,\\
k_1 &:= f(t+\delta, \tilde{y}(t+\delta)),\\
k_1 &= f(t+\delta, y(t) + \delta\, k_1).
\end{aligned}$$

Crank-Nicolson method:

$$\tilde{y}(t+\delta) = y(t) + \tfrac{\delta}{2}\big(f(t, y(t)) + f(t+\delta, \tilde{y}(t+\delta))\big)$$

Reminder: Implicit timestepping methods

Implicit Euler:

$$\begin{aligned}
\tilde{y}(t+\delta) &= y(t) + \delta\, f(t+\delta, \tilde{y}(t+\delta)) \\
\tilde{y}(t+\delta) &= y(t) + \delta\, k_1, \\
k_1 &:= f(t+\delta, \tilde{y}(t+\delta)), \\
k_1 &= f(t+\delta, y(t) + \delta\, k_1).
\end{aligned}$$

Crank-Nicolson method:

$$\begin{aligned}
\tilde{y}(t+\delta) &= y(t) + \tfrac{\delta}{2}\big(f(t, y(t)) + f(t+\delta, \tilde{y}(t+\delta))\big) \\
\tilde{y}(t+\delta) &= y(t) + \tfrac{\delta}{2}(k_1 + k_2), \\
k_1 &= f(t, y(t)), \\
k_2 &= f(t+\delta, \tilde{y}(t+\delta)),
\end{aligned}$$

Reminder: Implicit timestepping methods

Implicit Euler:

$$\begin{aligned}
\tilde{y}(t+\delta) &= y(t) + \delta\, f(t+\delta, \tilde{y}(t+\delta)) \\
\tilde{y}(t+\delta) &= y(t) + \delta\, k_1, \\
k_1 &:= f(t+\delta, \tilde{y}(t+\delta)), \\
k_1 &= f(t+\delta, y(t) + \delta\, k_1).
\end{aligned}$$

Crank-Nicolson method:

$$\begin{aligned}
\tilde{y}(t+\delta) &= y(t) + \tfrac{\delta}{2}\big(f(t, y(t)) + f(t+\delta, \tilde{y}(t+\delta))\big) \\
\tilde{y}(t+\delta) &= y(t) + \tfrac{\delta}{2}(k_1 + k_2), \\
k_1 &= f(t, y(t)), \\
k_2 &= f(t+\delta, \tilde{y}(t+\delta)),
\end{aligned}$$

Reminder: Implicit timestepping methods

Implicit Euler:

$$\begin{aligned}
\tilde{y}(t+\delta) &= y(t) + \delta\, f(t+\delta, \tilde{y}(t+\delta)) \\
\tilde{y}(t+\delta) &= y(t) + \delta\, k_1, \\
k_1 &:= f(t+\delta, \tilde{y}(t+\delta)), \\
k_1 &= f(t+\delta, y(t) + \delta\, k_1).
\end{aligned}$$

Crank-Nicolson method:

$$\begin{aligned}
\tilde{y}(t+\delta) &= y(t) + \tfrac{\delta}{2}\big(f(t, y(t)) + f(t+\delta, \tilde{y}(t+\delta))\big) \\
\tilde{y}(t+\delta) &= y(t) + \tfrac{\delta}{2}(k_1 + k_2), \\
k_1 &= f(t, y(t)), \\
k_2 &= f(t+\delta, \tilde{y}(t+\delta)), \\
k_2 &= f(t+\delta, y(t) + \tfrac{\delta}{2}(k_1 + k_2)).
\end{aligned}$$

General Runge-Kutta method

Observation: All single-step methods introduced in this lecture so far share a common form:

$$\tilde{y}(t+\delta) = y(t) + \delta \sum_{i=1}^{n} b_i\, k_i,$$

$$k_i = f\left(t + \delta\, c_i, y(t) + \delta \sum_{j=1}^{n} a_{ij} k_j\right) \qquad \text{for all } i \in [1:n].$$

General Runge-Kutta method

Observation: All single-step methods introduced in this lecture so far share a common form:

$$\tilde{y}(t+\delta) = y(t) + \delta \sum_{i=1}^{n} b_i\, k_i,$$

$$k_i = f\left(t + \delta\, c_i, y(t) + \delta \sum_{j=1}^{n} a_{ij} k_j\right) \qquad \text{for all } i \in [1:n].$$

- Explicit method if $a_{ij} = 0$ for $j \geq i$,
- Semi-implicit method if $a_{ij} = 0$ for $j > i$,
- Implicit method otherwise.

Butcher tableaus

Idea: Collect all coefficients in a simple representation.

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ c_n & a_{n1} & \cdots & a_{nn} \\ \hline & b_1 & \cdots & b_n \end{array}$$

Butcher tableaus

Idea: Collect all coefficients in a simple representation.

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ c_n & a_{n1} & \cdots & a_{nn} \\ \hline & b_1 & \cdots & b_n \end{array}$$

Explicit and implicit Euler:

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array} \qquad\qquad \begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

Butcher tableaus

Idea: Collect all coefficients in a simple representation.

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ c_n & a_{n1} & \cdots & a_{nn} \\ \hline & b_1 & \cdots & b_n \end{array}$$

Explicit and implicit Euler:

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array} \qquad \begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

Runge and Crank-Nicolson:

$$\begin{array}{c|cc} 0 & 0 & \\ 1/2 & 1/2 & 0 \\ \hline & 0 & 1 \end{array} \qquad \begin{array}{c|cc} 0 & 0 & \\ 1 & 1/2 & 1/2 \\ \hline & 1/2 & 1/2 \end{array}$$

Classic fourth-order Runge-Kutta

Idea: Based on Simpson's quadrature.

$$
\begin{array}{r|rrrr}
0 & 0 & & & \\
1/2 & 1/2 & 0 & & \\
1/2 & 0 & 1/2 & 0 & \\
1 & 0 & 0 & 1 & 0 \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}
$$

Classic fourth-order Runge-Kutta

Idea: Based on Simpson's quadrature.

$$\begin{array}{c|cccc} 0 & 0 & & & \\ 1/2 & 1/2 & 0 & & \\ 1/2 & 0 & 1/2 & 0 & \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Algorithm:

$$\begin{aligned} k_1 &:= f(t, y(t)), \\ k_2 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_1), \\ k_3 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_2), \\ k_4 &:= f(t + \delta, y(t) + \delta\, k_3), \\ \tilde{y}(t+\delta) &:= y(t) + \delta \left(\tfrac{1}{6} k_1 + \tfrac{1}{3} k_2 + \tfrac{1}{3} k_3 + \tfrac{1}{6} k_4 \right). \end{aligned}$$

Classic fourth-order Runge-Kutta

Idea: Based on Simpson's quadrature.

$$\begin{array}{c|cccc} 0 & 0 & & & \\ 1/2 & 1/2 & 0 & & \\ 1/2 & 0 & 1/2 & 0 & \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Algorithm:

$$\begin{aligned} k_1 &:= f(t, y(t)), \\ k_2 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_1), \\ k_3 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_2), \\ k_4 &:= f(t + \delta, y(t) + \delta\, k_3), \\ \tilde{y}(t+\delta) &:= y(t) + \delta\left(\tfrac{1}{6}k_1 + \tfrac{1}{3}k_2 + \tfrac{1}{3}k_3 + \tfrac{1}{6}k_4\right). \end{aligned}$$

Classic fourth-order Runge-Kutta

Idea: Based on Simpson's quadrature.

$$\begin{array}{c|cccc}
0 & 0 & & & \\
1/2 & 1/2 & 0 & & \\
1/2 & 0 & 1/2 & 0 & \\
1 & 0 & 0 & 1 & 0 \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}$$

Algorithm:

$$\begin{aligned}
k_1 &:= f(t, y(t)), \\
k_2 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_1), \\
k_3 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_2), \\
k_4 &:= f(t + \delta, y(t) + \delta\, k_3), \\
\tilde{y}(t + \delta) &:= y(t) + \delta \left(\tfrac{1}{6} k_1 + \tfrac{1}{3} k_2 + \tfrac{1}{3} k_3 + \tfrac{1}{6} k_4 \right).
\end{aligned}$$

Classic fourth-order Runge-Kutta

Idea: Based on Simpson's quadrature.

$$\begin{array}{c|cccc} 0 & 0 & & & \\ 1/2 & 1/2 & 0 & & \\ 1/2 & 0 & 1/2 & 0 & \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Algorithm:

$$\begin{aligned} k_1 &:= f(t, y(t)), \\ k_2 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_1), \\ k_3 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_2), \\ k_4 &:= f(t + \delta, y(t) + \delta\, k_3), \\ \tilde{y}(t+\delta) &:= y(t) + \delta\left(\tfrac{1}{6}k_1 + \tfrac{1}{3}k_2 + \tfrac{1}{3}k_3 + \tfrac{1}{6}k_4\right). \end{aligned}$$

Classic fourth-order Runge-Kutta

Idea: Based on Simpson's quadrature.

$$
\begin{array}{c|cccc}
0 & 0 & & & \\
1/2 & 1/2 & 0 & & \\
1/2 & 0 & 1/2 & 0 & \\
1 & 0 & 0 & 1 & 0 \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}
$$

Algorithm:

$$
\begin{aligned}
k_1 &:= f(t, y(t)), \\
k_2 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_1), \\
k_3 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_2), \\
k_4 &:= f(t + \delta, y(t) + \delta\, k_3), \\
\tilde{y}(t+\delta) &:= y(t) + \delta\left(\tfrac{1}{6}k_1 + \tfrac{1}{3}k_2 + \tfrac{1}{3}k_3 + \tfrac{1}{6}k_4\right).
\end{aligned}
$$

Classic fourth-order Runge-Kutta

Idea: Based on Simpson's quadrature.

$$
\begin{array}{c|cccc}
0 & 0 & & & \\
1/2 & 1/2 & 0 & & \\
1/2 & 0 & 1/2 & 0 & \\
1 & 0 & 0 & 1 & 0 \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}
$$

Algorithm:

$$
\begin{aligned}
k_1 &:= f(t, y(t)), \\
k_2 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_1), \\
k_3 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_2), \\
k_4 &:= f(t + \delta, y(t) + \delta\, k_3), \\
\tilde{y}(t+\delta) &:= y(t) + \delta \left(\tfrac{1}{6} k_1 + \tfrac{1}{3} k_2 + \tfrac{1}{3} k_3 + \tfrac{1}{6} k_4\right).
\end{aligned}
$$

Classic fourth-order Runge-Kutta

Idea: Based on Simpson's quadrature.

$$\begin{array}{c|cccc}
0 & 0 & & & \\
1/2 & 1/2 & 0 & & \\
1/2 & 0 & 1/2 & 0 & \\
1 & 0 & 0 & 1 & 0 \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}$$

Algorithm:

$$\begin{aligned}
k_1 &:= f(t, y(t)), \\
k_2 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_1), \\
k_3 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_2), \\
k_4 &:= f(t + \delta, y(t) + \delta\, k_3), \\
\tilde{y}(t+\delta) &:= y(t) + \delta \left(\tfrac{1}{6} k_1 + \tfrac{1}{3} k_2 + \tfrac{1}{3} k_3 + \tfrac{1}{6} k_4 \right).
\end{aligned}$$

Classic fourth-order Runge-Kutta

Idea: Based on Simpson's quadrature.

$$\begin{array}{c|cccc} 0 & 0 & & & \\ 1/2 & 1/2 & 0 & & \\ 1/2 & 0 & 1/2 & 0 & \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Algorithm:

$$\begin{aligned} k_1 &:= f(t, y(t)), \\ k_2 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_1), \\ k_3 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_2), \\ k_4 &:= f(t + \delta, y(t) + \delta\, k_3), \\ \tilde{y}(t+\delta) &:= y(t) + \delta \left(\tfrac{1}{6} k_1 + \tfrac{1}{3} k_2 + \tfrac{1}{3} k_3 + \tfrac{1}{6} k_4 \right). \end{aligned}$$

Classic fourth-order Runge-Kutta

Idea: Based on Simpson's quadrature.

$$\begin{array}{c|cccc} 0 & 0 & & & \\ 1/2 & 1/2 & 0 & & \\ 1/2 & 0 & 1/2 & 0 & \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Algorithm:

$$\begin{aligned} k_1 &:= f(t, y(t)), \\ k_2 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_1), \\ k_3 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_2), \\ k_4 &:= f(t + \delta, y(t) + \delta\, k_3), \\ \tilde{y}(t+\delta) &:= y(t) + \delta \left(\tfrac{1}{6} k_1 + \tfrac{1}{3} k_2 + \tfrac{1}{3} k_3 + \tfrac{1}{6} k_4 \right). \end{aligned}$$

Classic fourth-order Runge-Kutta

Idea: Based on Simpson's quadrature.

$$
\begin{array}{c|cccc}
0 & 0 & & & \\
1/2 & 1/2 & 0 & & \\
1/2 & 0 & 1/2 & 0 & \\
1 & 0 & 0 & 1 & 0 \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}
$$

Algorithm:

$$
\begin{aligned}
k_1 &:= f(t, y(t)), \\
k_2 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_1), \\
k_3 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_2), \\
k_4 &:= f(t + \delta, y(t) + \delta\, k_3), \\
\tilde{y}(t+\delta) &:= y(t) + \delta \left(\tfrac{1}{6} k_1 + \tfrac{1}{3} k_2 + \tfrac{1}{3} k_3 + \tfrac{1}{6} k_4 \right).
\end{aligned}
$$

Classic fourth-order Runge-Kutta

Idea: Based on Simpson's quadrature.

$$
\begin{array}{c|cccc}
0 & 0 & & & \\
1/2 & 1/2 & 0 & & \\
1/2 & 0 & 1/2 & 0 & \\
1 & 0 & 0 & 1 & 0 \\
\hline
 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}
$$

Algorithm:

$$
\begin{aligned}
k_1 &:= f(t, y(t)), \\
k_2 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_1), \\
k_3 &:= f(t + \tfrac{1}{2}\delta, y(t) + \tfrac{1}{2}\delta\, k_2), \\
k_4 &:= f(t + \delta, y(t) + \delta\, k_3), \\
\tilde{y}(t+\delta) &:= y(t) + \delta \left(\tfrac{1}{6} k_1 + \tfrac{1}{3} k_2 + \tfrac{1}{3} k_3 + \tfrac{1}{6} k_4 \right).
\end{aligned}
$$

Classic Runge-Kutta: Implementation

Approach: Store approximted derivatives $k_1, \ldots, k_4$ in auxiliary variables.

```
dx1 = v;
dv1 = -c/m * x;

dx2 = v + 0.5 * delta * dv1;
dv2 = -c/m * (x + 0.5 * delta * dx1);

dx3 = v + 0.5 * delta * dv2;
dv3 = -c/m * (x + 0.5 * delta * dx2);

dx4 = v + delta * dv3;
dv4 = -c/m * (x + delta * dx3);

x += delta * (dx1 + 2.0 * dx2 + 2.0 * dx3 + dx4) / 6.0;
v += delta * (dv1 + 2.0 * dv2 + 2.0 * dv3 + dv4) / 6.0;
```

Experiment: Crank-Nicolson vs Runge-Kutta

Approach: Start at $t = 0$, perform successive timesteps to reach $t = 10$.

	Crank-Nic		Runge-Kutta	
δ	error	ratio	error	ratio
$1/2$	9.2_{-2}		8.1_{-4}	
$1/4$	2.7_{-2}	3.4	1.2_{-4}	6.9
$1/8$	7.0_{-3}	3.9	9.2_{-6}	12.6
$1/16$	1.8_{-3}	4.0	6.4_{-7}	14.5
$1/32$	4.4_{-4}	4.0	4.1_{-8}	15.3
$1/64$	1.1_{-4}	4.0	2.6_{-9}	15.7
$1/128$	2.8_{-5}	4.0	1.7_{-10}	15.8
$1/256$	6.9_{-6}	4.0	1.1_{-11}	15.9
$1/512$	1.7_{-6}	4.0	6.6_{-13}	15.9

Observation: Classic Runge-Kutta is indeed of fourth order, since the error behaves like δ^4.

Multistep methods

Problem: Higher-order Runge-Kutta methods are computationally expensive.

Idea: Re-use results computed in previous steps in order to save time.

Approach: We let $t_i := t_0 + \delta\, i$.
An m-step method computes $y(t_{i+1})$ based on $y(t_i), \ldots, y(t_{i-m+1})$ (and maybe additional data corresponding to these previous states).

Multistep methods

Problem: Higher-order Runge-Kutta methods are computationally expensive.

Idea: Re-use results computed in previous steps in order to save time.

Approach: We let $t_i := t_0 + \delta\, i$.
An m-step method computes $y(t_{i+1})$ based on $y(t_i), \ldots, y(t_{i-m+1})$ (and maybe additional data corresponding to these previous states).

Example: Leapfrog method, $y(t+\delta)$ depends on $y(t)$ and $y(t+\frac{\delta}{2})$.

Adams-Bashforth method

Idea: Fundamental theorem of calculus states

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} y'(s)\,ds.$$

Adams-Bashforth method

Idea: Fundamental theorem of calculus states

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} y'(s)\,ds.$$

Approximate the interval by replacing y' with the interpolating polynomial

$$p(s) = \sum_{j=0}^{m} y'(t_{i-j})\,\ell_{i,j}(s)$$

in the points $t_i, t_{i-1}, \ldots, t_{i-m}$ with Lagrange polynomials $\ell_{i,0}, \ldots, \ell_{i,m}$.

Adams-Bashforth method

Idea: Fundamental theorem of calculus states

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} y'(s)\,ds.$$

Approximate the interval by replacing y' with the interpolating polynomial

$$p(s) = \sum_{j=0}^{m} y'(t_{i-j})\,\ell_{i,j}(s)$$

in the points $t_i, t_{i-1}, \ldots, t_{i-m}$ with Lagrange polynomials $\ell_{i,0}, \ldots, \ell_{i,m}$.

$$y(t_{i+1}) \approx y(t_i) + \int_{t_i}^{t_{i+1}} p(s)\,ds = y(t_i) + \sum_{j=0}^{m} y'(t_{i-j}) \underbrace{\int_{t_i}^{t_{i+1}} \ell_{i,j}(s)\,ds}_{=:a_{ij}}$$

$$= y(t_i) + \sum_{j=0}^{m} a_{ij}\, f(t_{i-j}, y(t_{i-j})).$$

Adams-Bashforth coefficients

Equidistant points $t_i = t_0 + \delta\, i$ imply

$$\ell_{i,j}(s) = \prod_{\substack{k=0\\k\neq j}}^{m} \frac{s - t_{i-k}}{t_{i-j} - t_{i-k}} = \prod_{\substack{k=0\\k\neq j}}^{m} \frac{s - t_i + \delta k}{\delta(k-j)}$$

Adams-Bashforth coefficients

Equidistant points $t_i = t_0 + \delta\, i$ imply

$$\ell_{i,j}(s) = \prod_{\substack{k=0\\k\neq j}}^{m} \frac{s - t_{i-k}}{t_{i-j} - t_{i-k}} = \prod_{\substack{k=0\\k\neq j}}^{m} \frac{s - t_i + \delta k}{\delta(k-j)} = \prod_{\substack{k=0\\k\neq j}}^{m} \frac{\hat{s} + k}{k - j}$$

with $s = \delta\hat{s} + t_i$.

Adams-Bashforth coefficients

Equidistant points $t_i = t_0 + \delta\, i$ imply

$$\ell_{i,j}(s) = \prod_{\substack{k=0 \\ k \neq j}}^{m} \frac{s - t_{i-k}}{t_{i-j} - t_{i-k}} = \prod_{\substack{k=0 \\ k \neq j}}^{m} \frac{s - t_i + \delta k}{\delta(k-j)} = \prod_{\substack{k=0 \\ k \neq j}}^{m} \frac{\hat{s} + k}{k - j}$$

with $s = \delta \hat{s} + t_i$.

Coefficients given by

$$a_{ij} = \int_{t_i}^{t_{i+1}} \ell_{i,j}(s)\, ds$$

Adams-Bashforth coefficients

Equidistant points $t_i = t_0 + \delta\, i$ imply

$$\ell_{i,j}(s) = \prod_{\substack{k=0\\k\neq j}}^{m} \frac{s - t_{i-k}}{t_{i-j} - t_{i-k}} = \prod_{\substack{k=0\\k\neq j}}^{m} \frac{s - t_i + \delta k}{\delta(k-j)} = \prod_{\substack{k=0\\k\neq j}}^{m} \frac{\hat{s} + k}{k - j}$$

with $s = \delta\hat{s} + t_i$.

Coefficients given by

$$a_{ij} = \int_{t_i}^{t_{i+1}} \ell_{i,j}(s)\, ds = \delta \int_0^1 \ell_{i,j}(\delta\hat{s} + t_i)\, d\hat{s}$$

Adams-Bashforth coefficients

Equidistant points $t_i = t_0 + \delta\, i$ imply

$$\ell_{i,j}(s) = \prod_{\substack{k=0\\k\neq j}}^{m} \frac{s - t_{i-k}}{t_{i-j} - t_{i-k}} = \prod_{\substack{k=0\\k\neq j}}^{m} \frac{s - t_i + \delta k}{\delta(k-j)} = \prod_{\substack{k=0\\k\neq j}}^{m} \frac{\hat{s} + k}{k - j}$$

with $s = \delta\hat{s} + t_i$.

Coefficients given by

$$a_{ij} = \int_{t_i}^{t_{i+1}} \ell_{i,j}(s)\, ds = \delta \int_0^1 \ell_{i,j}(\delta\hat{s} + t_i)\, d\hat{s} = \delta \underbrace{\int_0^1 \prod_{\substack{k=0\\k\neq j}}^{m} \frac{\hat{s} + k}{k - j}\, d\hat{s}}_{=: w_j} = \delta\, w_j.$$

Adams-Bashforth coefficients

Equidistant points $t_i = t_0 + \delta\, i$ imply

$$\ell_{i,j}(s) = \prod_{\substack{k=0\\k\neq j}}^{m} \frac{s - t_{i-k}}{t_{i-j} - t_{i-k}} = \prod_{\substack{k=0\\k\neq j}}^{m} \frac{s - t_i + \delta k}{\delta(k-j)} = \prod_{\substack{k=0\\k\neq j}}^{m} \frac{\hat{s} + k}{k - j}$$

with $s = \delta\hat{s} + t_i$.

Coefficients given by

$$a_{ij} = \int_{t_i}^{t_{i+1}} \ell_{i,j}(s)\, ds = \delta \int_0^1 \ell_{i,j}(\delta\hat{s} + t_i)\, d\hat{s} = \delta \underbrace{\int_0^1 \prod_{\substack{k=0\\k\neq j}}^{m} \frac{\hat{s} + k}{k - j}\, d\hat{s}}_{=:w_j} = \delta\, w_j.$$

The same coefficients are used in all timesteps.

Computing the weights

Idea: Monomials $p_i(t) = t^i$ have to be integrated exactly,

$$\sum_{j=0}^{m} w_j \, p_i(-j) = \int_0^1 p_i(s) \, ds.$$

Example: Interpolation points $0, -1, -2, -3$, weights have to solve

$$\begin{aligned}
w_0 + w_1 + w_2 + w_3 &= 1, & (p_0(t) = 1)\\
-w_1 - 2\,w_2 - 3\,w_3 &= 1/2, & (p_1(t) = t)\\
w_1 + 4\,w_2 + 9\,w_3 &= 1/3, & (p_2(t) = t^2)\\
-w_1 - 8\,w_2 - 27\,w3 &= 1/4. & (p_3(t) = t^3)
\end{aligned}$$

The solution is $w_0 = \frac{55}{24}$, $w_1 = -\frac{59}{24}$, $w_2 = \frac{37}{24}$, $w_3 = -\frac{9}{24}$.

Adams-Bashforth algorithm

Idea: Store $y_i := \tilde{y}(t_i)$ and $f_i := f(t_i, \tilde{y}(t_i))$.

$$y_{i+1} := y_i + \delta \sum_{j=0}^{m} w_j \, f_{i-j},$$
$$f_{i+1} := f(t_{i+1}, y_{i+1}).$$

Observation: We require only one evaluation of f per step.

Storage: We have to store the derivatives $f_i, \dots, f_{i-m}$.
Older derivatives can be cyclically overwritten.

Problem: We need approximations for the first $m+1$ states before we can start the Adams-Bashforth algorithm.

Adams-Bashforth: Implementation

Approach: Store previous states in arrays `x`, `v` and previous derivatives in arrays `dx` and `dv`.

```
for(i=3; i<n; i++) {
  x[(i+1)%4] = x[i%4] + delta * (w0 * dx[i%4]
                                 + w1 * dx[(i-1)%4]
                                 + w2 * dx[(i-2)%4]
                                 + w3 * dx[(i-3)%4]);
  v[(i+1)%4] = v[i%4] + delta * (w0 * dv[i%4]
                                 + w1 * dv[(i-1)%4]
                                 + w2 * dv[(i-2)%4]
                                 + w3 * dv[(i-3)%4]);

  dx[(i+1)%4] = v[(i+1)%4];
  dv[(i+1)%4] = -c/m * x[(i+1)%4];
}
```

Experiment: Runge-Kutta vs Adams-Bashforth

Approach: Start at $t = 0$, perform successive timesteps to reach $t = 10$.

	Runge-Kutta		Adams-Bash	
δ	error	ratio	error	ratio
$1/2$	8.1_{-4}		2.0_{-2}	
$1/4$	1.2_{-4}	6.9	2.3_{-3}	8.7
$1/8$	9.2_{-6}	12.6	3.0_{-4}	7.5
$1/16$	6.4_{-7}	14.5	2.4_{-5}	12.7
$1/32$	4.1_{-8}	15.3	1.7_{-6}	14.5
$1/64$	2.6_{-9}	15.7	1.1_{-7}	15.3
$1/128$	1.7_{-10}	15.8	6.9_{-9}	15.7
$1/256$	1.1_{-11}	15.9	4.4_{-10}	15.8
$1/512$	6.6_{-13}	15.9	2.7_{-11}	15.9

Observation: Both methods of fourth order.

Summary

Runge-Kutta methods use multiple intermediate derivatives.

$$k_i := f\left(t + \delta\, c_i, y(t) + \delta \sum_{j=1}^{s} a_{ij}\, k_j\right) \qquad \text{for all } i \in [1:s],$$

$$\tilde{y}(t+\delta) := y(t) + \delta \sum_{i=1}^{s} b_i\, k_i$$

Example: Classic Runge-Kutta method reaches fourth-order accuracy.

Multistep methods re-use previous states and derivatives.

$$\tilde{y}(t+\delta) = y(t) + \delta \sum_{j=0}^{m} w_j\, f(t - \delta\, j, \tilde{y}(t - \delta\, j)).$$

Example: Adams-Bashforth methods of any order can be constructed by solving a system of linear equations.