FH Bielefeld
University of
Applied Sciences

GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

Bernhard Bachmann

Faculty of Engineering and Mathematics
Bielefeld University of Applied Sciences, Germany

6th February 2023

Version 1.0

GBODE

The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

GBODE stands for Generic Bi-rate Ordinary Differential Equation solver and is highly configurable supporting the following features:

Single-rate Mode of GBODE:

- Generic implementation for any (implicit, explicit) Runge-Kutta scheme
- Different methods for step size control
- Support of different extrapolation schemes
- Reliable event handling
- Efficient solving of non-linear equations (incl. sparse matrix handling)

GBODE stands for Generic Bi-rate Ordinary Differential Equation solver and is highly configurable supporting the following features:

Single-rate Mode of GBODE:

- Generic implementation for any (implicit, explicit) Runge-Kutta scheme
- Different methods for step size control
- Support of different extrapolation schemes
- Reliable event handling
- Efficient solving of non-linear equations (incl. sparse matrix handling)

GBODE

The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

GBODE stands for Generic Bi-rate Ordinary Differential Equation solver and is highly configurable supporting the following features:

Single-rate Mode of GBODE:

- Generic implementation for any (implicit, explicit) Runge-Kutta scheme
- Different methods for step size control
- Support of different extrapolation schemes
- Reliable event handling
- Efficient solving of non-linear equations (incl. sparse matrix handling)

GBODE

The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

GBODE stands for Generic Bi-rate Ordinary Differential Equation solver and is highly configurable supporting the following features:

Single-rate Mode of GBODE:

- Generic implementation for any (implicit, explicit) Runge-Kutta scheme
- Different methods for step size control
- Support of different extrapolation schemes
- Reliable event handling
- Efficient solving of non-linear equations (incl. sparse matrix handling)

GBODE stands for Generic Bi-rate Ordinary Differential Equation solver and is highly configurable supporting the following features:

Single-rate Mode of GBODE:

- Generic implementation for any (implicit, explicit) Runge-Kutta scheme
- Different methods for step size control
- Support of different extrapolation schemes
- Reliable event handling
- Efficient solving of non-linear equations (incl. sparse matrix handling)

GBODE

The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

GBODE stands for Generic Bi-rate Ordinary Differential Equation solver and is highly configurable supporting the following features:

Single-rate Mode of GBODE:

- Generic implementation for any (implicit, explicit) Runge-Kutta scheme
- Different methods for step size control
- Support of different extrapolation schemes
- Reliable event handling
- Efficient solving of non-linear equations (incl. sparse matrix handling)

Bi-rate Mode of GBODE (prototype):

- Same options as for the single-rate mode
- Seperation of fast and slow states (inner/outer integration)
- Step size control for each mode
- Efficient event handling

GBODE

The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

GBODE stands for Generic Bi-rate Ordinary Differential Equation solver and is highly configurable supporting the following features:

Single-rate Mode of GBODE:

- Generic implementation for any (implicit, explicit) Runge-Kutta scheme
- Different methods for step size control
- Support of different extrapolation schemes
- Reliable event handling
- Efficient solving of non-linear equations (incl. sparse matrix handling)

Bi-rate Mode of GBODE (prototype):

- Same options as for the single-rate mode
- Seperation of fast and slow states (inner/outer integration)
- Step size control for each mode
- Efficient event handling

GBODE stands for Generic Bi-rate Ordinary Differential Equation solver and is highly configurable supporting the following features:

Single-rate Mode of GBODE:

- Generic implementation for any (implicit, explicit) Runge-Kutta scheme
- Different methods for step size control
- Support of different extrapolation schemes
- Reliable event handling
- Efficient solving of non-linear equations (incl. sparse matrix handling)

Bi-rate Mode of GBODE (prototype):

- Same options as for the single-rate mode
- Seperation of fast and slow states (inner/outer integration)
- Step size control for each mode
- Efficient event handling

GBODE

The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

GBODE stands for Generic Bi-rate Ordinary Differential Equation solver and is highly configurable supporting the following features:

Single-rate Mode of GBODE:

- Generic implementation for any (implicit, explicit) Runge-Kutta scheme
- Different methods for step size control
- Support of different extrapolation schemes
- Reliable event handling
- Efficient solving of non-linear equations (incl. sparse matrix handling)

Bi-rate Mode of GBODE (prototype):

- Same options as for the single-rate mode
- Seperation of fast and slow states (inner/outer integration)
- Step size control for each mode
- Efficient event handling

Contents

Single-Rate Mode of GBODE

Status February 2022

Complete list of integration methods before GBODE development:

-s=value ***or*** **-s value** Value specifies the integration method. For additional information see the *User's Guide*

- euler - Euler - explicit, fixed step size, order 1
- heun - Heun's method - explicit, fixed step, order 2
- rungekutta - classical Runge-Kutta - explicit, fixed step, order 4
- impeuler - Euler - implicit, fixed step size, order 1
- trapezoid - trapezoidal rule - implicit, fixed step size, order 2
- imprungekutta - Runge-Kutta methods based on Radau and Lobatto IIA - implicit, fixed step size, order 1-6(selected manually by flag -impRKOrder)
- irksco - own developed Runge-Kutta solver - implicit, step size control, order 1-2
- dassl - default solver - BDF method - implicit, step size control, order 1-5
- ida - SUNDIALS IDA solver - BDF method with sparse linear solver - implicit, step size control, order 1-5
- cvode - experimental implementation of SUNDIALS CVODE solver - BDF or Adams-Moulton method - step size control, order 1-12
- rungekuttaSsc - Runge-Kutta based on Novikov (2016) - explicit, step size control, order 4-5 [experimental]
- symSolver - symbolic inline Solver [compiler flag +symSolver needed] - fixed step size, order 1
- symSolverSsc - symbolic implicit Euler with step size control [compiler flag +symSolver needed] - step size control, order 1
- qss - A QSS solver [experimental]
- optimization - Special solver for dynamic optimization

Single-Rate Mode of GBODE

Status February 2022

Situation before the GBODE development:

- Different implementation, but, with different quality
- Some methods are rarely used by the OpenModelica community
- Impossible to maintain the program code
- Many experimental features
- Only a few support all Modelica language features

Necessary capabilities of an efficient integration method :

- Realiable step size control
- Efficient event handling
- Capability to utilize sparse matrix solvers
- Highly configurable implementation
- Robust default settings

⇒ GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

Single-Rate Mode of GBODE

Status February 2022

Situation before the GBODE development:

- Different implementation, but, with different quality
- Some methods are rarely used by the OpenModelica community
- Impossible to maintain the program code
- Many experimental features
- Only a few support all Modelica language features

Necessary capabilities of an efficient integration method :

- Realiable step size control
- Efficient event handling
- Capability to utilize sparse matrix solvers
- Highly configurable implementation
- Robust default settings

$\Rightarrow$ GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

Single-Rate Mode of GBODE

Status February 2022

Situation before the GBODE development:

- Different implementation, but, with different quality
- Some methods are rarely used by the OpenModelica community
- Impossible to maintain the program code
- Many experimental features
- Only a few support all Modelica language features

Necessary capabilities of an efficient integration method :

- Realiable step size control
- Efficient event handling
- Capability to utilize sparse matrix solvers
- Highly configurable implementation
- Robust default settings

⇒ GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

Single-Rate Mode of GBODE

Status February 2022

Situation before the GBODE development:

- Different implementation, but, with different quality
- Some methods are rarely used by the OpenModelica community
- Impossible to maintain the program code
- Many experimental features
- Only a few support all Modelica language features

Necessary capabilities of an efficient integration method :

- Realiable step size control
- Efficient event handling
- Capability to utilize sparse matrix solvers
- Highly configurable implementation
- Robust default settings

⇒ GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

Single-Rate Mode of GBODE

Status February 2022

Situation before the GBODE development:

- Different implementation, but, with different quality
- Some methods are rarely used by the OpenModelica community
- Impossible to maintain the program code
- Many experimental features
- Only a few support all Modelica language features

Necessary capabilities of an efficient integration method :

- Realiable step size control
- Efficient event handling
- Capability to utilize sparse matrix solvers
- Highly configurable implementation
- Robust default settings

⇒ GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

Single-Rate Mode of GBODE

Status February 2022

Situation before the GBODE development:

- Different implementation, but, with different quality
- Some methods are rarely used by the OpenModelica community
- Impossible to maintain the program code
- Many experimental features
- Only a few support all Modelica language features

Necessary capabilities of an efficient integration method :

- Realiable step size control
- Efficient event handling
- Capability to utilize sparse matrix solvers
- Highly configurable implementation
- Robust default settings

⇒ GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

Single-Rate Mode of GBODE

Status February 2022

Situation before the GBODE development:

- Different implementation, but, with different quality
- Some methods are rarely used by the OpenModelica community
- Impossible to maintain the program code
- Many experimental features
- Only a few support all Modelica language features

Necessary capabilities of an efficient integration method :

- Realiable step size control
- Efficient event handling
- Capability to utilize sparse matrix solvers
- Highly configurable implementation
- Robust default settings

⇒ GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

Single-Rate Mode of GBODE

Status February 2022

Situation before the GBODE development:

- Different implementation, but, with different quality
- Some methods are rarely used by the OpenModelica community
- Impossible to maintain the program code
- Many experimental features
- Only a few support all Modelica language features

Necessary capabilities of an efficient integration method :

- Realiable step size control
- Efficient event handling
- Capability to utilize sparse matrix solvers
- Highly configurable implementation
- Robust default settings

⇒ GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

Single-Rate Mode of GBODE

Status February 2022

Situation before the GBODE development:

- Different implementation, but, with different quality
- Some methods are rarely used by the OpenModelica community
- Impossible to maintain the program code
- Many experimental features
- Only a few support all Modelica language features

Necessary capabilities of an efficient integration method :

- Realiable step size control
- Efficient event handling
- Capability to utilize sparse matrix solvers
- Highly configurable implementation
- Robust default settings

$\Rightarrow$ GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

Single-Rate Mode of GBODE

Status February 2022

Situation before the GBODE development:

- Different implementation, but, with different quality
- Some methods are rarely used by the OpenModelica community
- Impossible to maintain the program code
- Many experimental features
- Only a few support all Modelica language features

Necessary capabilities of an efficient integration method :

- Realiable step size control
- Efficient event handling
- Capability to utilize sparse matrix solvers
- Highly configurable implementation
- Robust default settings

$\Rightarrow$ GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

Single-Rate Mode of GBODE

Status February 2022

Situation before the GBODE development:

- Different implementation, but, with different quality
- Some methods are rarely used by the OpenModelica community
- Impossible to maintain the program code
- Many experimental features
- Only a few support all Modelica language features

Necessary capabilities of an efficient integration method :

- Realiable step size control
- Efficient event handling
- Capability to utilize sparse matrix solvers
- Highly configurable implementation
- Robust default settings

⇒ GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

Single-Rate Mode of GBODE

Status February 2022

Situation before the GBODE development:

- Different implementation, but, with different quality
- Some methods are rarely used by the OpenModelica community
- Impossible to maintain the program code
- Many experimental features
- Only a few support all Modelica language features

Necessary capabilities of an efficient integration method :

- Realiable step size control
- Efficient event handling
- Capability to utilize sparse matrix solvers
- Highly configurable implementation
- Robust default settings

$\Rightarrow$ GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

Single-Rate Mode of GBODE

Status February 2022

Situation before the GBODE development:

- Different implementation, but, with different quality
- Some methods are rarely used by the OpenModelica community
- Impossible to maintain the program code
- Many experimental features
- Only a few support all Modelica language features

Necessary capabilities of an efficient integration method :

- Realiable step size control
- Efficient event handling
- Capability to utilize sparse matrix solvers
- Highly configurable implementation
- Robust default settings

$\Rightarrow$ GBODE - The Generic Bi-Rate Ordinary Differential Equation Solver in OpenModelica

Single-Rate Mode of GBODE

The Classic Runge-Kutta Method (stages 4)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Single-Rate Mode of GBODE

The Classic Runge-Kutta Method (stages 4)

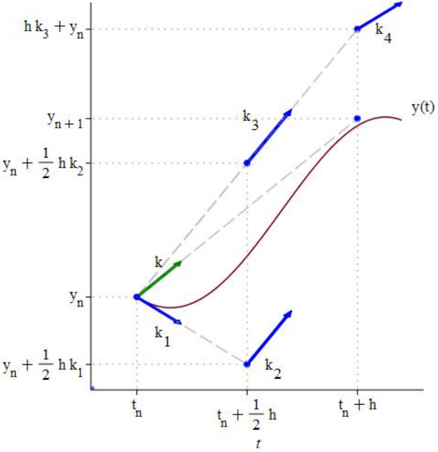
Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$k_1 = f\left(t_n, y_n\right),$$



Single-Rate Mode of GBODE

The Classic Runge-Kutta Method (stages 4)

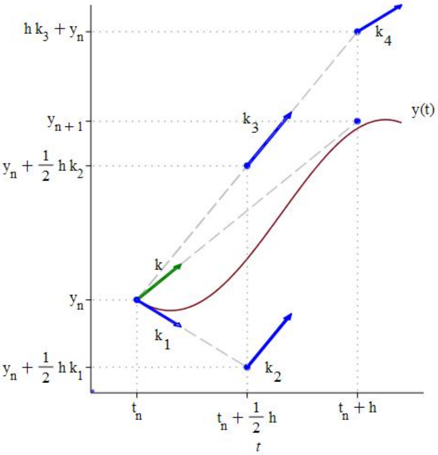
Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$k_1 = f\left(t_n, y_n\right),$$
$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_1\right),$$



Single-Rate Mode of GBODE

The Classic Runge-Kutta Method (stages 4)

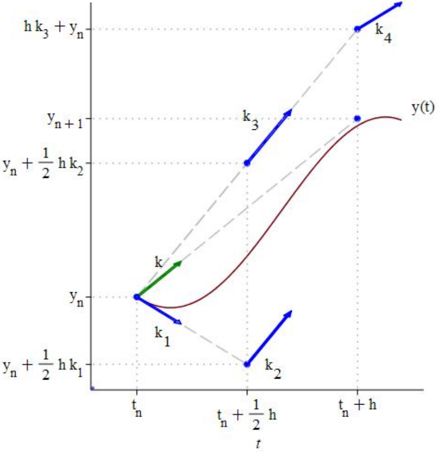
Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned} k_1 &= f\left(t_n, y_n\right), \\ k_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_1\right), \\ k_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_2\right), \end{aligned}$$



Single-Rate Mode of GBODE

The Classic Runge-Kutta Method (stages 4)

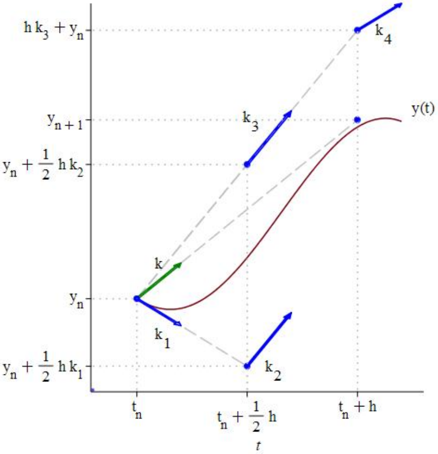
Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 &= f\left(t_n, y_n\right), \\
k_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_1\right), \\
k_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_2\right), \\
k_4 &= f\left(t_n + h, y_n + h\, k_3\right).
\end{aligned}$$



Single-Rate Mode of GBODE

The Classic Runge-Kutta Method (stages 4)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

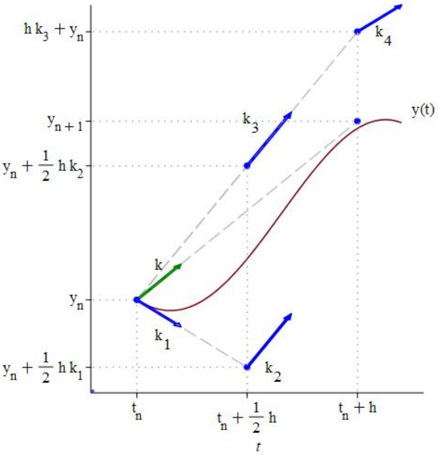
Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned} k_1 =& f\left(t_n, y_n\right), \\ k_2 =& f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_1\right), \\ k_3 =& f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_2\right), \\ k_4 =& f\left(t_n + h, y_n + h\, k_3\right). \end{aligned}$$

Perform time step:

$$\begin{aligned} t_{n+1} =& t_n + h, \\ y_{n+1} =& y_n + h \underbrace{\frac{1}{6}(k_1 + 2\,k_2 + 2\,k_3 + k_4)}_{=k} \end{aligned}$$



Single-Rate Mode of GBODE

The Classic Runge-Kutta Method (stages 4)

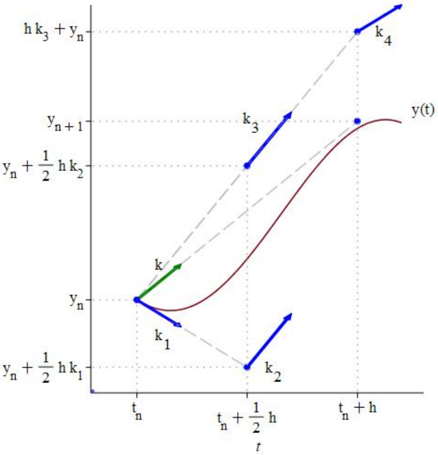
Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 &= f\left(t_n, y_n\right), \\
k_2 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_1\right), \\
k_3 &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} k_2\right), \\
k_4 &= f\left(t_n + h, y_n + h\, k_3\right).
\end{aligned}$$



Perform time step:

$$\begin{aligned}
t_{n+1} &= t_n + h, \\
y_{n+1} &= y_n + h \underbrace{\frac{1}{6}(k_1 + 2\, k_2 + 2\, k_3 + k_4)}_{=k} = y_n + \Phi(y_n, t_n, h, f).
\end{aligned}$$

Single-Rate Mode of GBODE

General Explicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

Single-Rate Mode of GBODE

General Explicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$k_1 = f\left(t_n, y_n\right),$$

Single-Rate Mode of GBODE

General Explicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned} k_1 &= f\left(t_n, y_n\right), \\ k_2 &= f\left(t_n + c_2 h, y_n + h a_{21} k_1\right), \end{aligned}$$

Single-Rate Mode of GBODE

General Explicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n, y_n\right), \\
k_2 =& f\left(t_n + c_2 h, y_n + h a_{21} k_1\right), \\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2\right),
\end{aligned}$$

Single-Rate Mode of GBODE

General Explicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 &= f\left(t_n, y_n\right), \\
k_2 &= f\left(t_n + c_2 h, y_n + h a_{21} k_1\right), \\
k_3 &= f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2\right), \\
&\vdots \\
k_s &= f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s-1} k_{s-1})\right).
\end{aligned}$$

Single-Rate Mode of GBODE

General Explicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n, y_n\right), \\
k_2 =& f\left(t_n + c_2 h, y_n + h a_{21} k_1\right), \\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2\right), \\
&\vdots \\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s-1} k_{s-1})\right).
\end{aligned}$$

Perform time step:

$$\begin{aligned}
t_{n+1} =& t_n + h, \\
y_{n+1} =& y_n + h \sum_{i=1}^{s} b_i \, k_i
\end{aligned}$$

Single-Rate Mode of GBODE

General Explicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n, y_n\right), \\
k_2 =& f\left(t_n + c_2 h, y_n + h a_{21} k_1\right), \\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2\right), \\
&\vdots \\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s-1} k_{s-1})\right).
\end{aligned}$$

Perform time step:

$$\begin{aligned}
t_{n+1} =& t_n + h, \\
y_{n+1} =& y_n + h \sum_{i=1}^{s} b_i\, k_i = y_n + \Phi(y_n, t_n, h, f).
\end{aligned}$$

Single-Rate Mode of GBODE

General Explicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 &= f\left(t_n, y_n\right), \\
k_2 &= f\left(t_n + c_2 h, y_n + h a_{21} k_1\right), \\
k_3 &= f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2\right), \\
&\vdots \\
k_s &= f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s-1} k_{s-1})\right).
\end{aligned}$$

Perform time step:

$$\begin{aligned}
t_{n+1} &= t_n + h, \\
y_{n+1} &= y_n + h \sum_{i=1}^{s} b_i \, k_i = y_n + \Phi(y_n, t_n, h, f).
\end{aligned}$$

Butcher tableau:

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & \cdots & 0 \\
c_2 & a_{21} & 0 & 0 & \cdots & 0 \\
c_3 & a_{31} & a_{32} & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & 0 \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}$$

Single-Rate Mode of GBODE

General Explicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n, y_n\right),\\
k_2 =& f\left(t_n + c_2 h, y_n + h a_{21} k_1\right),\\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2\right),\\
&\vdots\\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s-1} k_{s-1})\right).
\end{aligned}$$

Perform time step:

$$\begin{aligned}
t_{n+1} =& t_n + h,\\
y_{n+1} =& y_n + h \sum_{i=1}^{s} b_i \, k_i = y_n + \Phi(y_n, t_n, h, f).
\end{aligned}$$

Butcher tableau:

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & \cdots & 0\\
c_2 & a_{21} & 0 & 0 & \cdots & 0\\
c_3 & a_{31} & a_{32} & 0 & \cdots & 0\\
\vdots & \vdots & \vdots & \ddots & \ddots & \vdots\\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & 0\\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}$$

Necessary conditions (order 1):

$$\begin{aligned}
\sum_{i=1}^{s} b_i =& 1\\
\sum_{j=1}^{i-1} a_{ij} =& c_i, \quad \text{für} \quad i = 2, \ldots, s.
\end{aligned}$$

Single-Rate Mode of GBODE

General Implicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

Single-Rate Mode of GBODE

General Implicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$k_1 = f\left(t_n + c_1 h, y_n + h(a_{11} k_1 + a_{12} k_2 + \ldots + a_{1,s} k_s)\right),$$

Single-Rate Mode of GBODE

General Implicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n + c_1 h, y_n + h(a_{11} k_1 + a_{12} k_2 + \ldots + a_{1,s} k_s)\right), \\
k_2 =& f\left(t_n + c_2 h, y_n + h(a_{21} k_1 + a_{22} k_2 + \ldots + a_{2,s} k_s)\right), \\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2 + \ldots + a_{3,s} k_s)\right), \\
&\vdots \\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s} k_s)\right).
\end{aligned}$$

Single-Rate Mode of GBODE

General Implicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n + c_1 h, y_n + h(a_{11} k_1 + a_{12} k_2 + \ldots + a_{1,s} k_s)\right), \\
k_2 =& f\left(t_n + c_2 h, y_n + h(a_{21} k_1 + a_{22} k_2 + \ldots + a_{2,s} k_s)\right), \\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2 + \ldots + a_{3,s} k_s)\right), \\
&\vdots \\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s} k_s)\right).
\end{aligned}$$

Perform time step:

$$\begin{aligned}
t_{n+1} =& t_n + h, \\
y_{n+1} =& y_n + h \sum_{i=1}^{s} b_i \, k_i
\end{aligned}$$

Single-Rate Mode of GBODE

General Implicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n + c_1 h, y_n + h(a_{11} k_1 + a_{12} k_2 + \ldots + a_{1,s} k_s)\right),\\
k_2 =& f\left(t_n + c_2 h, y_n + h(a_{21} k_1 + a_{22} k_2 + \ldots + a_{2,s} k_s)\right),\\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2 + \ldots + a_{3,s} k_s)\right),\\
&\vdots\\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s} k_s)\right).
\end{aligned}$$

Perform time step:

$$\begin{aligned}
t_{n+1} =& t_n + h,\\
y_{n+1} =& y_n + h \sum_{i=1}^{s} b_i\, k_i = y_n + \Phi(y_n, t_n, h, f).
\end{aligned}$$

Single-Rate Mode of GBODE

General Implicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n + c_1 h, y_n + h(a_{11} k_1 + a_{12} k_2 + \ldots + a_{1,s} k_s)\right), \\
k_2 =& f\left(t_n + c_2 h, y_n + h(a_{21} k_1 + a_{22} k_2 + \ldots + a_{2,s} k_s)\right), \\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2 + \ldots + a_{3,s} k_s)\right), \\
&\vdots \\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s} k_s)\right).
\end{aligned}$$

Perform time step:

$$\begin{aligned}
t_{n+1} =& t_n + h, \\
y_{n+1} =& y_n + h \sum_{i=1}^{s} b_i \, k_i = y_n + \Phi(y_n, t_n, h, f).
\end{aligned}$$

Butcher tableau:

$$\begin{array}{c|ccccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1,s-1} & a_{1,s} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2,s-1} & a_{2,s} \\
c_3 & a_{31} & a_{32} & \cdots & a_{3,s-1} & a_{3,s} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & a_{s,s} \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}$$

Single-Rate Mode of GBODE

General Implicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n + c_1 h, y_n + h(a_{11}k_1 + a_{12}k_2 + \ldots + a_{1,s}k_s)\right),\\
k_2 =& f\left(t_n + c_2 h, y_n + h(a_{21}k_1 + a_{22}k_2 + \ldots + a_{2,s}k_s)\right),\\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31}k_1 + a_{32}k_2 + \ldots + a_{3,s}k_s)\right),\\
&\vdots\\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1}k_1 + a_{s2}k_2 + \ldots + a_{s,s}k_s)\right).
\end{aligned}$$

Perform time step:

$$\begin{aligned}
t_{n+1} =& t_n + h,\\
y_{n+1} =& y_n + h\sum_{i=1}^{s} b_i\, k_i = y_n + \Phi(y_n, t_n, h, f).
\end{aligned}$$

Butcher tableau:

$$\begin{array}{c|cccccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1,s-1} & a_{1,s}\\
c_2 & a_{21} & a_{22} & \cdots & a_{2,s-1} & a_{2,s}\\
c_3 & a_{31} & a_{32} & \cdots & a_{3,s-1} & a_{3,s}\\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & a_{s,s}\\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}$$

Necessary conditions (order 1):

$$\begin{aligned}
\sum_{i=1}^{s} b_i =& 1\\
\sum_{j=1}^{s} a_{ij} =& c_i, \quad \text{für} \quad i = 1, \ldots, s.
\end{aligned}$$

Single-Rate Mode of GBODE

General Implicit Runge-Kutta Method (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n + c_1 h, y_n + h(a_{11}k_1 + a_{12}k_2 + \ldots + a_{1,s}k_s)\right), \\
k_2 =& f\left(t_n + c_2 h, y_n + h(a_{21}k_1 + a_{22}k_2 + \ldots + a_{2,s}k_s)\right), \\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31}k_1 + a_{32}k_2 + \ldots + a_{3,s}k_s)\right), \\
&\vdots \\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1}k_1 + a_{s2}k_2 + \ldots + a_{s,s}k_s)\right).
\end{aligned}$$

Perform time step:

$$\begin{aligned}
t_{n+1} =& t_n + h, \\
y_{n+1} =& y_n + h\sum_{i=1}^{s} b_i\, k_i = y_n + \Phi(y_n, t_n, h, f).
\end{aligned}$$

Butcher tableau:

$$\begin{array}{c|cccccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1,s-1} & a_{1,s} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2,s-1} & a_{2,s} \\
c_3 & a_{31} & a_{32} & \cdots & a_{3,s-1} & a_{3,s} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & a_{s,s} \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}$$

Necessary conditions (order 1):

$$\begin{aligned}
\sum_{i=1}^{s} b_i =& 1 \\
\sum_{j=1}^{s} a_{ij} =& c_i, \quad \text{für} \quad i = 1, \ldots, s.
\end{aligned}$$

Compact Butcher tableau:

$$\begin{array}{c|c}
c & A \\
\hline
 & b
\end{array}$$

Single-Rate Mode of GBODE

Singly-Diagonal Implicit Runge-Kutta Method - SDIRK (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$k_1 = f\left(t_n + c_1 h, y_n + h\gamma k_1\right),$$

Single-Rate Mode of GBODE

Singly-Diagonal Implicit Runge-Kutta Method - SDIRK (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n + c_1 h, y_n + h\gamma k_1\right), \\
k_2 =& f\left(t_n + c_2 h, y_n + h(a_{21} k_1 + \gamma k_2\right),
\end{aligned}$$

Single-Rate Mode of GBODE

Singly-Diagonal Implicit Runge-Kutta Method - SDIRK (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n + c_1 h, y_n + h\gamma k_1\right), \\
k_2 =& f\left(t_n + c_2 h, y_n + h(a_{21} k_1 + \gamma k_2\right), \\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2 + \gamma k_3)\right),
\end{aligned}$$

Single-Rate Mode of GBODE

Singly-Diagonal Implicit Runge-Kutta Method - SDIRK (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n + c_1 h, y_n + h\gamma k_1\right), \\
k_2 =& f\left(t_n + c_2 h, y_n + h(a_{21} k_1 + \gamma k_2\right), \\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2 + \gamma k_3)\right), \\
&\vdots \\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s-1} k_{s-1} + \gamma k_s)\right).
\end{aligned}$$

Single-Rate Mode of GBODE

Singly-Diagonal Implicit Runge-Kutta Method - SDIRK (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n + c_1 h, y_n + h\gamma k_1\right),\\
k_2 =& f\left(t_n + c_2 h, y_n + h(a_{21} k_1 + \gamma k_2\right),\\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2 + \gamma k_3)\right),\\
&\vdots\\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s-1} k_{s-1} + \gamma k_s)\right).
\end{aligned}$$

Perform time step:

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

Single-Rate Mode of GBODE

Singly-Diagonal Implicit Runge-Kutta Method - SDIRK (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n + c_1 h, y_n + h\gamma k_1\right), \\
k_2 =& f\left(t_n + c_2 h, y_n + h(a_{21} k_1 + \gamma k_2\right), \\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2 + \gamma k_3)\right), \\
&\vdots \\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s-1} k_{s-1} + \gamma k_s)\right).
\end{aligned}$$

Perform time step:

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i\, k_i = y_n + \Phi(y_n, t_n, h, f).$$

Single-Rate Mode of GBODE

Singly-Diagonal Implicit Runge-Kutta Method - SDIRK (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n + c_1 h, y_n + h\gamma k_1\right),\\
k_2 =& f\left(t_n + c_2 h, y_n + h(a_{21} k_1 + \gamma k_2\right),\\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2 + \gamma k_3)\right),\\
&\vdots\\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s-1} k_{s-1} + \gamma k_s)\right).
\end{aligned}$$

Perform time step:

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i\, k_i = y_n + \Phi(y_n, t_n, h, f).$$

Butcher tableau:

$$\begin{array}{c|ccccc}
c_1 & \gamma & 0 & 0 & \cdots & 0\\
c_2 & a_{21} & \gamma & 0 & \cdots & 0\\
c_3 & a_{31} & a_{32} & \gamma & \cdots & 0\\
\vdots & \vdots & \vdots & & \ddots & \vdots\\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \gamma\\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}$$

Single-Rate Mode of GBODE

Singly-Diagonal Implicit Runge-Kutta Method - SDIRK (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n + c_1 h, y_n + h\gamma k_1\right), \\
k_2 =& f\left(t_n + c_2 h, y_n + h(a_{21} k_1 + \gamma k_2\right), \\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2 + \gamma k_3)\right), \\
& \vdots \\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s-1} k_{s-1} + \gamma k_s)\right).
\end{aligned}$$

Perform time step:

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i\, k_i = y_n + \Phi(y_n, t_n, h, f).$$

Butcher tableau:

$$\begin{array}{c|ccccc}
c_1 & \gamma & 0 & 0 & \cdots & 0 \\
c_2 & a_{21} & \gamma & 0 & \cdots & 0 \\
c_3 & a_{31} & a_{32} & \gamma & \cdots & 0 \\
\vdots & \vdots & \vdots & & \ddots & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \gamma \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}$$

Compact Butcher tableau:

$$\begin{array}{c|c}
c & A \\
\hline
 & b
\end{array}$$

Single-Rate Mode of GBODE

Explicit Singly-Diagonal Implicit Runge-Kutta Method - ESDIRK (stages s)

Initial value problem:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

Get approximation of y_{n+1} at t_{n+1} (step size h):

Calculate derivatives:

$$\begin{aligned}
k_1 =& f\left(t_n, y_n\right), \\
k_2 =& f\left(t_n + c_2 h, y_n + h(a_{21} k_1 + \gamma k_2\right), \\
k_3 =& f\left(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2 + \gamma k_3)\right), \\
&\vdots \\
k_s =& f\left(t_n + c_s h, y_n + h(a_{s1} k_1 + a_{s2} k_2 + \ldots + a_{s,s-1} k_{s-1} + \gamma k_s)\right).
\end{aligned}$$

Perform time step:

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i \, k_i = y_n + \Phi(y_n, t_n, h, f).$$

Butcher tableau:

$$\begin{array}{c|ccccc}
0 & 0 & 0 & 0 & \cdots & 0 \\
c_2 & a_{21} & \gamma & 0 & \cdots & 0 \\
c_3 & a_{31} & a_{32} & \gamma & \cdots & 0 \\
\vdots & \vdots & \vdots & & \ddots & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \gamma \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s
\end{array}$$

Compact Butcher tableau:

$$\begin{array}{c|c}
c & A \\
\hline
 & b
\end{array}$$

Single-Rate Mode of GBODE

Requirements for Step Size Control

Calculate two approximations $(y_{n+1}, \hat{y}_{n+1})$ to the solution at time $t_n + h$ with different error order $(p, \hat{p})$:

Single-Rate Mode of GBODE

Requirements for Step Size Control

Calculate two approximations $(y_{n+1}, \hat{y}_{n+1})$ to the solution at time $t_n + h$ with different error order $(p, \hat{p})$:

Richardson extrapolation:
Given:
Integration method of order p.

$$y_{n+1} = y_n + \Phi(y_n, t_n, h, f).$$

Single-Rate Mode of GBODE

Requirements for Step Size Control

Calculate two approximations $(y_{n+1}, \hat{y}_{n+1})$ to the solution at time $t_n + h$ with different error order $(p, \hat{p})$:

Richardson extrapolation:
Given:
Integration method of order p.

$$y_{n+1} = y_n + \Phi(y_n, t_n, h, f).$$

Perform one integration step of length h and two integration steps of length $\frac{h}{2}$ and construct an approximation $\hat{y}_{n+1}$ of order $\hat{p} = p + 1$

$$\begin{aligned}
y_{n+1} =& \Phi(y_n, t_n, h, f), \\
y_{n+\frac{1}{2}} =& \Phi(y_n, t_n, \frac{h}{2}, f), \\
\tilde{y}_{n+1} =& \Phi(y_{n+\frac{1}{2}}, t_n + \frac{h}{2}, \frac{h}{2}, f), \\
\hat{y}_{n+1} =& \frac{2^p \, \tilde{y}_{n+1} - y_{n+1}}{2^p - 1}.
\end{aligned}$$

Single-Rate Mode of GBODE

Requirements for Step Size Control

Calculate two approximations $(y_{n+1}, \hat{y}_{n+1})$ to the solution at time $t_n + h$ with different error order $(p, \hat{p})$:

Richardson extrapolation:
Given:
Integration method of order p.

$$y_{n+1} = y_n + \Phi(y_n, t_n, h, f).$$

Perform one integration step of length h and two integration steps of length $\frac{h}{2}$ and construct an approximation $\hat{y}_{n+1}$ of order $\hat{p} = p + 1$

$$\begin{aligned} y_{n+1} &= \Phi(y_n, t_n, h, f), \\ y_{n+\frac{1}{2}} &= \Phi(y_n, t_n, \frac{h}{2}, f), \\ \tilde{y}_{n+1} &= \Phi(y_{n+\frac{1}{2}}, t_n + \frac{h}{2}, \frac{h}{2}, f), \\ \hat{y}_{n+1} &= \frac{2^p\, \tilde{y}_{n+1} - y_{n+1}}{2^p - 1}. \end{aligned}$$

Embedded Runge-Kutta methods: Butcher tableau:

$$\begin{array}{c|ccccc} c_1 & a_{11} & a_{12} & \cdots & a_{1,s-1} & a_{1,s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2,s-1} & a_{2,s} \\ c_3 & a_{31} & a_{32} & \cdots & a_{3,s-1} & a_{3,s} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & a_{s,s} \\ \hline & b_1 & b_2 & \cdots & b_{s-1} & b_s \\ & \hat{b}_1 & \hat{b}_2 & \cdots & \hat{b}_{s-1} & \hat{b}_s \end{array}$$

Single-Rate Mode of GBODE

Requirements for Step Size Control

Calculate two approximations $(y_{n+1}, \hat{y}_{n+1})$ to the solution at time $t_n + h$ with different error order $(p, \hat{p})$:

Richardson extrapolation:
Given:
Integration method of order p.

$$y_{n+1} = y_n + \Phi(y_n, t_n, h, f).$$

Perform one integration step of length h and two integration steps of length $\frac{h}{2}$ and construct an approximation $\hat{y}_{n+1}$ of order $\hat{p} = p + 1$

$$\begin{aligned}
y_{n+1} =& \Phi(y_n, t_n, h, f),\\
y_{n+\frac{1}{2}} =& \Phi(y_n, t_n, \frac{h}{2}, f),\\
\tilde{y}_{n+1} =& \Phi(y_{n+\frac{1}{2}}, t_n + \frac{h}{2}, \frac{h}{2}, f),\\
\hat{y}_{n+1} =& \frac{2^p\,\tilde{y}_{n+1} - y_{n+1}}{2^p - 1}.
\end{aligned}$$

Embedded Runge-Kutta methods: Butcher tableau:

$$\begin{array}{c|ccccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1,s-1} & a_{1,s}\\
c_2 & a_{21} & a_{22} & \cdots & a_{2,s-1} & a_{2,s}\\
c_3 & a_{31} & a_{32} & \cdots & a_{3,s-1} & a_{3,s}\\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots\\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & a_{s,s}\\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s\\
 & \hat{b}_1 & \hat{b}_2 & \cdots & \hat{b}_{s-1} & \hat{b}_s
\end{array}$$

$$\begin{aligned}
y_{n+1} =& y_n + h\sum_{i=1}^{s} b_i\,k_i\\
\hat{y}_{n+1} =& y_n + h\sum_{i=1}^{s} \hat{b}_i\,k_i
\end{aligned}$$

Single-Rate Mode of GBODE

Requirements for Step Size Control

Calculate two approximations $(y_{n+1}, \hat{y}_{n+1})$ to the solution at time $t_n + h$ with different error order $(p, \hat{p})$:

Richardson extrapolation:
Given:
Integration method of order p.

$$y_{n+1} = y_n + \Phi(y_n, t_n, h, f).$$

Perform one integration step of length h and two integration steps of length $\frac{h}{2}$ and construct an approximation $\hat{y}_{n+1}$ of order $\hat{p} = p + 1$

$$\begin{aligned}
y_{n+1} =& \Phi(y_n, t_n, h, f), \\
y_{n+\frac{1}{2}} =& \Phi(y_n, t_n, \frac{h}{2}, f), \\
\tilde{y}_{n+1} =& \Phi(y_{n+\frac{1}{2}}, t_n + \frac{h}{2}, \frac{h}{2}, f), \\
\hat{y}_{n+1} =& \frac{2^p \, \tilde{y}_{n+1} - y_{n+1}}{2^p - 1}.
\end{aligned}$$

Embedded Runge-Kutta methods: Butcher tableau:

$$\begin{array}{c|ccccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1,s-1} & a_{1,s} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2,s-1} & a_{2,s} \\
c_3 & a_{31} & a_{32} & \cdots & a_{3,s-1} & a_{3,s} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & a_{s,s} \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & b_s \\
 & \hat{b}_1 & \hat{b}_2 & \cdots & \hat{b}_{s-1} & \hat{b}_s
\end{array}$$

$$\begin{aligned}
y_{n+1} =& y_n + h \sum_{i=1}^{s} b_i \, k_i \\
\hat{y}_{n+1} =& y_n + h \sum_{i=1}^{s} \hat{b}_i \, k_i
\end{aligned}$$

$\Rightarrow \quad |y_{n+1} - \hat{y}_{n+1}| \approx O(h^q)$ with $q = \min(p, \hat{p})$ is an estimate to the approximation error.

Compact Butcher tableau:

$$\begin{array}{c|c} c & A \\ \hline & b \end{array}$$

Compact Butcher tableau:

$$\begin{array}{c|c} c & A \\ \hline & b \end{array}$$

Error propagation can be controlled via the stability function:

$$R(z) = 1 + zb^T \left(I - zA\right)^{-1} e,$$

e stands for the vector of ones.

Compact Butcher tableau:

$$\begin{array}{c|c} c & A \\ \hline & b \end{array}$$

Error propagation can be controlled via the stability function:

$$R(z) = 1 + zb^T \left(I - zA\right)^{-1} e,$$

e stands for the vector of ones.
If λ is an eigenvalue of the Jacobian J_f of f, then

$$\lambda h \in S := \{z \in \mathbb{C} : |R(z)| < 1\}$$

Compact Butcher tableau:

$$\begin{array}{c|c} c & A \\ \hline & b \end{array}$$

Error propagation can be controlled via the stability function:

$$R(z) = 1 + zb^T (I - zA)^{-1} e,$$

e stands for the vector of ones.
If λ is an eigenvalue of the Jacobian J_f of f, then

$$\lambda h \in S := \{z \in \mathbb{C} : |R(z)| < 1\}$$

Example (explicit Euler):

Butcher tableau:

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

$$\Rightarrow R(z) = 1 + z \Rightarrow S = \{z \in \mathbb{C} : |z + 1| < 1\}$$

Compact Butcher tableau:

$$\begin{array}{c|c} c & A \\ \hline & b \end{array}$$

Error propagation can be controlled via the stability function:

$$R(z) = 1 + zb^T (I - zA)^{-1} e,$$

e stands for the vector of ones.
If λ is an eigenvalue of the Jacobian J_f of f, then

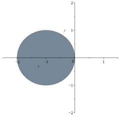
$$\lambda h \in S := \{z \in \mathbb{C} : |R(z)| < 1\}$$

Example (explicit Euler):

Butcher tableau:

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

Stability region:



$$\Rightarrow R(z) = 1 + z \Rightarrow S = \{z \in \mathbb{C} : |z + 1| < 1\}$$

Compact Butcher tableau:

$$\begin{array}{c|c} c & A \\ \hline & b \end{array}$$

Error propagation can be controlled via the stability function:

$$R(z) = 1 + zb^T (I - zA)^{-1} e,$$

e stands for the vector of ones.
If λ is an eigenvalue of the Jacobian J_f of f, then

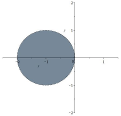
$$\lambda h \in S := \{z \in \mathbb{C} : |R(z)| < 1\}$$

Example (explicit Euler):

Butcher tableau:

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

Stability region:



$$\Rightarrow R(z) = 1 + z \Rightarrow S = \{z \in \mathbb{C} : |z + 1| < 1\}$$

```
model CurtissHirschfelder
  Real y(start=0);
  Real z;
  parameter Real a=50;
  parameter Real y0 = a^2/(a^2+1);
  parameter Real b(fixed=false);
 initial equation
  y=z;
equation
  der(y) = -a*(y-cos(time));
  z = a/(a^2+1)*(a*cos(time)+
       sin(time))+b*exp(-a*time);
end CurtissHirschfelder;
```

Compact Butcher tableau:

$$\begin{array}{c|c} c & A \\ \hline & b \end{array}$$

Error propagation can be controlled via the stability function:

$$R(z) = 1 + zb^T (I - zA)^{-1} e,$$

e stands for the vector of ones.
If λ is an eigenvalue of the Jacobian J_f of f, then

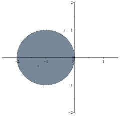
$$\lambda h \in S := \{z \in \mathbb{C} : |R(z)| < 1\}$$

Example (explicit Euler):

Butcher tableau:

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

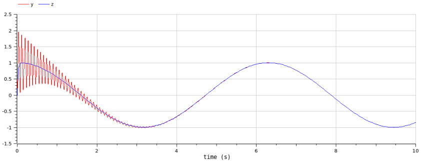
Stability region:



$$\Rightarrow R(z) = 1 + z \Rightarrow S = \{z \in \mathbb{C} : |z + 1| < 1\}$$

```
model CurtissHirschfelder
  Real y(start=0);
  Real z;
  parameter Real a=50;
  parameter Real y0 = a^2/(a^2+1);
  parameter Real b(fixed=false);
 initial equation
  y=z;
equation
  der(y) = -a*(y-cos(time));
  z = a/(a^2+1)*(a*cos(time)+
       sin(time))+b*exp(-a*time);
end CurtissHirschfelder ;
```

Step size $h = 0.039$ ($\lambda = -50, -2 < \lambda h < 0$):



Compact Butcher tableau:

$$\begin{array}{c|c} c & A \\ \hline & b \end{array}$$

Error propagation can be controlled via the stability function:

$$R(z) = 1 + zb^T \left(I - zA\right)^{-1} e,$$

e stands for the vector of ones.
If λ is an eigenvalue of the Jacobian J_f of f, then

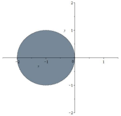
$$\lambda h \in S := \{z \in \mathbb{C} : |R(z)| < 1\}$$

Example (explicit Euler):

Butcher tableau:

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

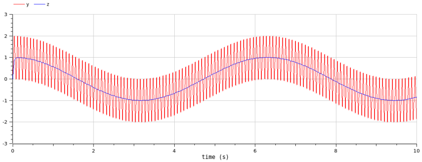
Stability region:



$$\Rightarrow R(z) = 1 + z \Rightarrow S = \{z \in \mathbb{C} : |z + 1| < 1\}$$

```
model CurtissHirschfelder
  Real y(start=0);
  Real z;
  parameter Real a=50;
  parameter Real y0 = a^2/(a^2+1);
  parameter Real b(fixed=false);
 initial equation
  y=z;
equation
  der(y) = -a*(y-cos(time));
  z = a/(a^2+1)*(a*cos(time)+
        sin(time))+b*exp(-a*time);
end CurtissHirschfelder;
```

Step size $h = 0.04$ ($\lambda = -50, -2 < \lambda h < 0$):



Compact Butcher tableau:

$$\begin{array}{c|c} c & A \\ \hline & b \end{array}$$

Error propagation can be controlled via the stability function:

$$R(z) = 1 + zb^T (I - zA)^{-1} e,$$

e stands for the vector of ones.
If λ is an eigenvalue of the Jacobian J_f of f, then

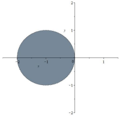
$$\lambda h \in S := \{z \in \mathbb{C} : |R(z)| < 1\}$$

Example (explicit Euler):

Butcher tableau:

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

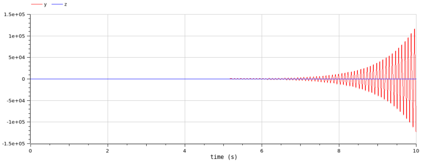
Stability region:



$$\Rightarrow R(z) = 1 + z \Rightarrow S = \{z \in \mathbb{C} : |z + 1| < 1\}$$

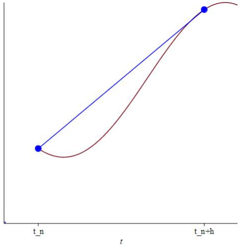
```
model CurtissHirschfelder
  Real y( start =0);
  Real z;
  parameter Real a=50;
  parameter Real y0 = a^2/(a^2+1);
  parameter Real b( fixed =false);
 initial equation
  y=z;
equation
  der(y) = -a*(y-cos(time));
  z = a/(a^2+1)*(a*cos(time)+
       sin(time))+b*exp(-a*time);
end CurtissHirschfelder ;
```

Step size $h = 0.041$ $(\lambda = -50, -2 < \lambda h < 0)$:



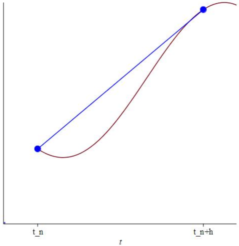
Interpolation between integrator steps

Linear interpolation:

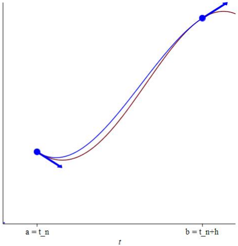


Interpolation between integrator steps

Linear interpolation:

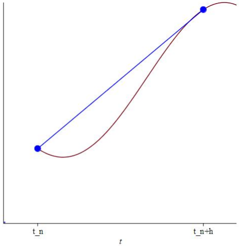


Hermite interpolation:

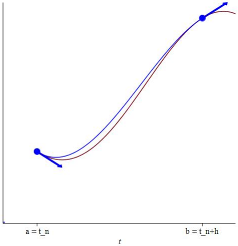


Interpolation between integrator steps

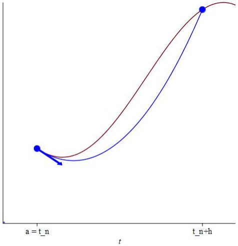
Linear interpolation:



Hermite interpolation:



Hermite interpolation a:

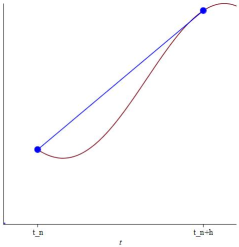


Single-Rate Mode of GBODE

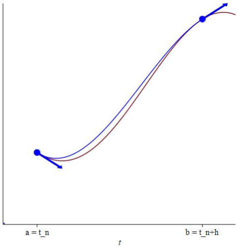
Interpolation and Dense Output

Interpolation between integrator steps

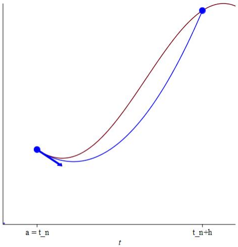
Linear interpolation:



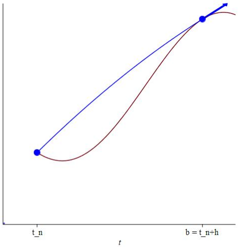
Hermite interpolation:



Hermite interpolation a:

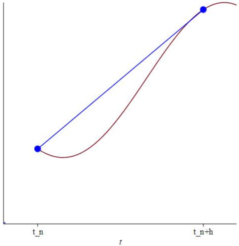


Hermite interpolation b:

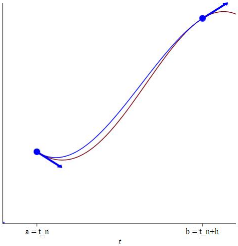


Interpolation between integrator steps

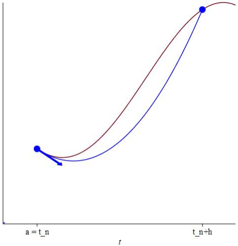
Linear interpolation:



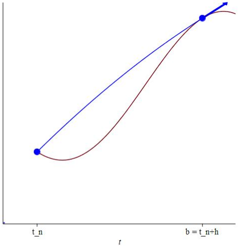
Hermite interpolation:



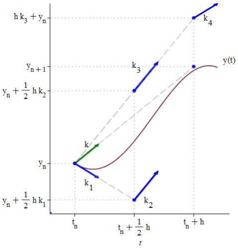
Hermite interpolation a:



Hermite interpolation b:



Dense output:

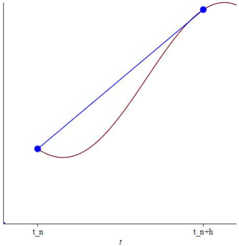


Single-Rate Mode of GBODE

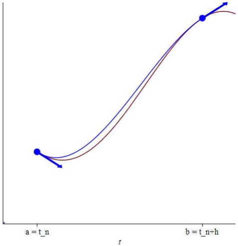
Interpolation and Dense Output

Interpolation between integrator steps

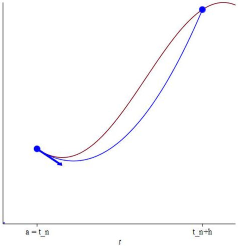
Linear interpolation:



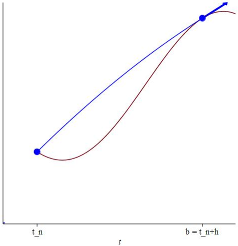
Hermite interpolation:



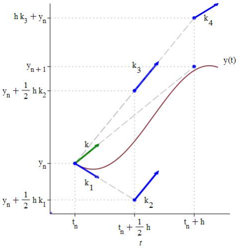
Hermite interpolation a:



Hermite interpolation b:



Dense output:



Calculation using the already computed derivative information:

$$\theta \in [0, 1]$$

$$y(t_n + \theta h) \approx y_n + h \sum_{i=1}^{s} b_i(\theta) k_i$$

$b_1(\theta), \ldots, b_s(\theta)$ are polynomials depending on the RK method. Dense output gives the same approximation order as the underlying RK method.

Single-Rate Mode of GBODE

Explicit Euler method (order 1(2), stages 1(2))

Butcher-Tableaus: (Constant step size/Richardson extrapolation vs. variable step size)

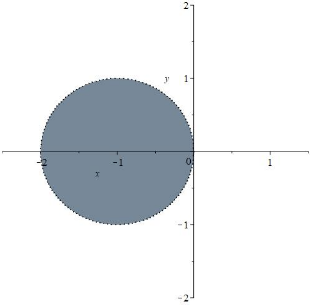
$$\begin{array}{c|c} 0.0 & 0.0 \\ \hline & 1.0 \end{array}$$

$$\begin{array}{c|cc} 0.0 & 0.0 & 0.0 \\ 0.5 & 0.5 & 0.0 \\ \hline & 1.0 & 0.0 \\ & 0.0 & 1.0 \end{array}$$

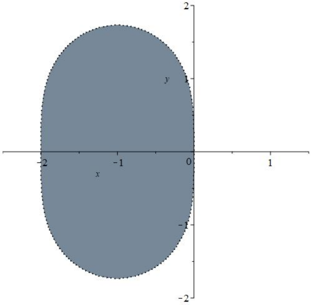
Properties:

1. error order: 1 (2)
2. stability regions
3. real axes limit: -2
4. hermite interpolation
5. gbode flag:`-gbm=expl_euler`

main RK method



embedded RK method



Single-Rate Mode of GBODE

Original Dormand–Prince method (order 5 (4), stages 7)

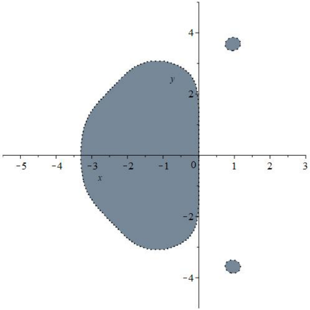
Butcher-Tableau:

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.200	0.200	0.0	0.0	0.0	0.0	0.0	0.0
0.300	0.0750	0.225	0.0	0.0	0.0	0.0	0.0
0.800	0.978	−3.73	3.56	0.0	0.0	0.0	0.0
0.889	2.95	−11.6	9.82	−0.291	0.0	0.0	0.0
1.0	2.85	−10.8	8.91	0.278	−0.274	0.0	0.0
1.0	0.0911	0.0	0.449	0.651	−0.322	0.131	0.0
	0.0911	0.0	0.449	0.651	−0.322	0.131	0.0
	0.0899	0.0	0.453	0.614	−0.272	0.0890	0.0250

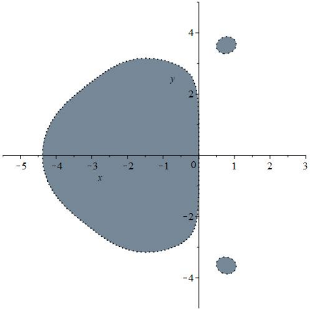
Properties:

1. error order: 5 (4)
2. stability regions
3. real axes limit: -4.39
4. dense output
5. gbode flag:`-gbm=dopri45`

main RK method



embedded RK method



Single-Rate Mode of GBODE

Dormand–Prince method with strong stability region (order 1 (5), stages 7)

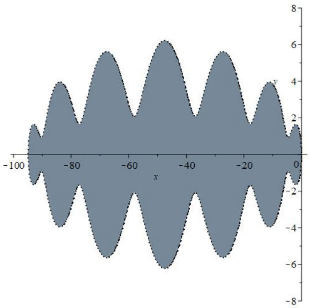
Butcher-Tableau:

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.200	0.200	0.0	0.0	0.0	0.0	0.0	0.0
0.300	0.0750	0.225	0.0	0.0	0.0	0.0	0.0
0.800	0.978	-3.73	3.56	0.0	0.0	0.0	0.0
0.889	2.95	-11.6	9.82	-0.291	0.0	0.0	0.0
1.0	2.85	-10.8	8.91	0.278	-0.274	0.0	0.0
1.0	0.0911	0.0	0.449	0.651	-0.322	0.131	0.0
	0.279	0.499	0.220	0.00222	-0.000109	0.00000291	0.0000000679
	0.0911	0.0	0.449	0.651	-0.322	0.131	0.0

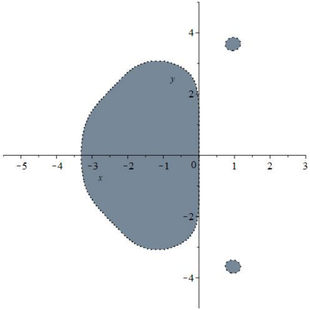
Properties:

1. error order: 1 (5)
2. stability regions
3. real axes limit: -94.82
4. dense output
5. gbode flag:`-gbm=dopriSsc1`

main RK method



embedded RK method



Single-Rate Mode of GBODE

Dormand–Prince method with strong stability region (order 2 (5), stages 7)

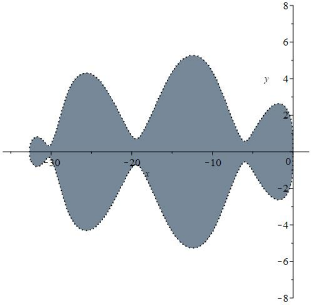
Butcher-Tableau:

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.200	0.200	0.0	0.0	0.0	0.0	0.0	0.0
0.300	0.0750	0.225	0.0	0.0	0.0	0.0	0.0
0.800	0.978	−3.73	3.56	0.0	0.0	0.0	0.0
0.889	2.95	−11.6	9.82	−0.291	0.0	0.0	0.0
1.0	2.85	−10.8	8.91	0.278	−0.274	0.0	0.0
1.0	0.0911	0.0	0.449	0.651	−0.322	0.131	0.0
	−0.486	−0.235	1.66	0.0709	−0.00905	0.000668	0.0000480
	0.0911	0.0	0.449	0.651	−0.322	0.131	0.0

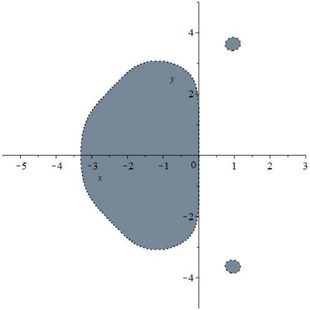
Properties:

1. error order: 2 (5)
2. stability regions
3. real axes limit: -32.65
4. dense output
5. gbode flag:`-gbm=dopriSsc2`

main RK method



embedded RK method



Single-Rate Mode of GBODE

Fehlberg method (order 8(7), stages 13)

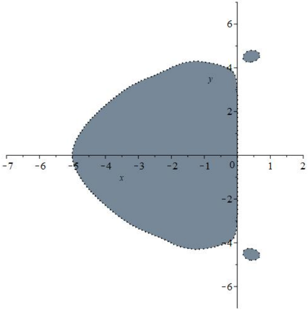
Butcher-Tableau:

0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0741	0.0741	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.111	0.0278	0.0833	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.167	0.0417	0.0	0.125	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.417	0.417	0.0	−1.56	1.56	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.500	0.0500	0.0	0.0	0.250	0.200	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.833	−0.231	0.0	0.0	1.16	−2.41	2.31	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.167	0.103	0.0	0.0	0.0	0.271	−0.222	0.0144	0.0	0.0	0.0	0.0	0.0	0.0
0.667	2.0	0.0	0.0	−8.83	15.6	−11.9	0.744	3.0	0.0	0.0	0.0	0.0	0.0
0.333	−0.843	0.0	0.0	0.213	−7.23	5.76	−0.317	2.83	−0.0833	0.0	0.0	0.0	0.0
1.0	0.581	0.0	0.0	−2.08	4.39	−3.67	0.520	0.549	0.274	0.439	0.0	0.0	0.0
0.0	0.0146	0.0	0.0	0.0	0.0	−0.146	−0.0146	−0.0732	0.0732	0.146	0.0	0.0	0.0
1.0	−0.433	0.0	0.0	−2.08	4.39	−3.52	0.535	0.622	0.201	0.293	0.0	1.0	0.0
	0.0	0.0	0.0	0.0	0.0	0.324	0.257	0.257	0.0321	0.0321	0.0	0.0488	0.0488
	0.0488	0.0	0.0	0.0	0.0	0.324	0.257	0.257	0.0321	0.0321	0.0488	0.0	0.0

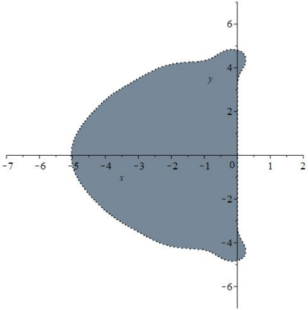
Properties:

1. error order: 8 (7)
2. stability regions
3. real axes limit: -5.1
4. hermite interpolation
5. gbode flag:`-gbm=fehlberg78`

main RK method



embedded RK method



Single-Rate Mode of GBODE

Euler method (order 1(2), stages 1(2), implicit)

Butcher-Tableaus: (Constant step size/Richardson extrapolation vs. variable step size)

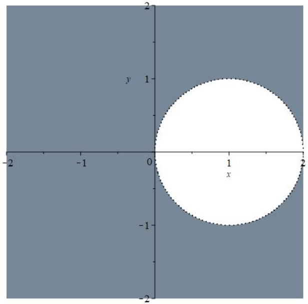
$$\begin{array}{c|c} 1.0 & 1.0 \\ \hline & 1.0 \end{array}$$

$$\begin{array}{c|cc} 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 1.0 \\ \hline & 0.0 & 1.0 \\ & 0.5 & 0.5 \end{array}$$

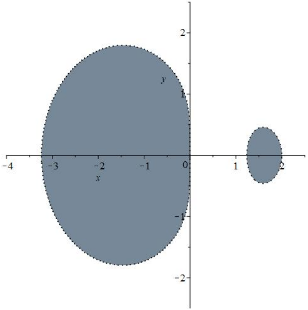
Properties:

1. error order: 1 (2)
2. stability regions
3. real axes limit: $-\infty$
4. hermite interpolation
5. gbode flag:`-gbm=impl_euler`

main RK method



embedded RK method



Single-Rate Mode of GBODE

Gauss3 method (order 6(2), stages 3, implicit)

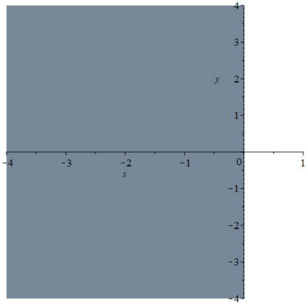
Butcher-Tableau:

$$\begin{array}{c|ccc}
0.113 & 0.139 & -0.0360 & 0.00979 \\
0.500 & 0.300 & 0.222 & -0.0225 \\
0.887 & 0.268 & 0.480 & 0.139 \\
\hline
 & 0.278 & 0.444 & 0.278 \\
 & -0.833 & 2.67 & -0.833
\end{array}$$

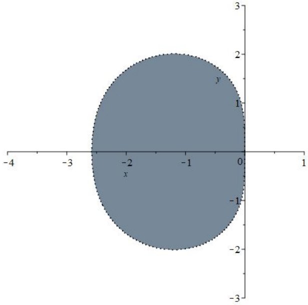
Properties:

1. error order: 6 (2)
2. stability regions
3. real axes limit: $-\infty$
4. hermite interpolation
5. gbode flag:`-gbm=gauss3`

main RK method



embedded RK method



Single-Rate Mode of GBODE

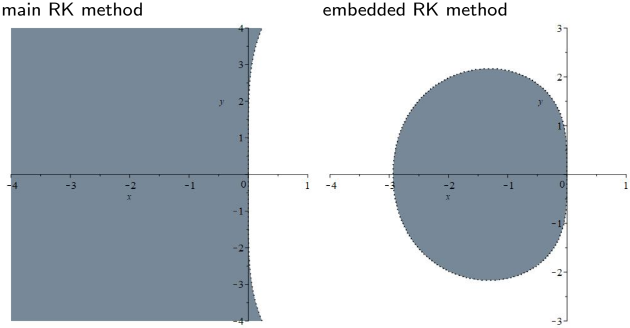
RadauIA3 method (order 5(2), stages 3, implicit)

Butcher-Tableau:

$$
\begin{array}{l|lll}
0.0 & 0.111 & -0.192 & 0.0805 \\
0.355 & 0.111 & 0.292 & -0.0481 \\
0.845 & 0.111 & 0.537 & 0.197 \\
\hline
 & 0.111 & 0.512 & 0.376 \\
 & -1.0 & 2.43 & -0.429
\end{array}
$$

Properties:

1. error order: 5 (2)
2. stability regions
3. real axes limit: $-\infty$
4. hermite interpolation
5. gbode flag:`-gbm=radauIA3`



Single-Rate Mode of GBODE

LobattoIIIC3 method (order 4(2), stages 3, implicit)

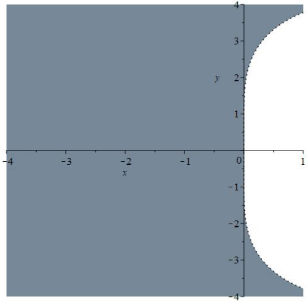
Butcher-Tableau:

$$
\begin{array}{l|lll}
0.0 & 0.167 & -0.333 & 0.167 \\
0.500 & 0.167 & 0.417 & -0.0833 \\
1.0 & 0.167 & 0.667 & 0.167 \\
\hline
 & 0.167 & 0.667 & 0.167 \\
 & -0.500 & 2.0 & -0.500
\end{array}
$$

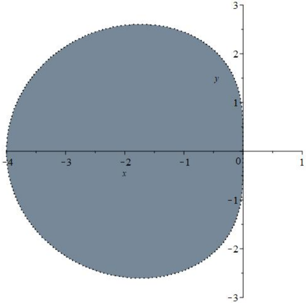
Properties:

1. error order: 4 (2)
2. stability regions
3. real axes limit: $-\infty$
4. hermite interpolation
5. gbode flag:`-gbm=lobattoIIIC3`

main RK method



embedded RK method



Single-Rate Mode of GBODE

SDIRK3 method (order 3(2), stages 3, implicit)

Butcher-Tableau:

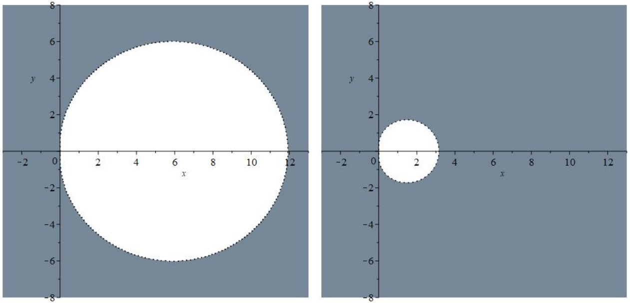
$$
\begin{array}{c|ccc}
0.789 & 0.789 & 0.0 & 0.0 \\
0.210 & -0.576 & 0.789 & 0.0 \\
1.0 & 0.0 & 0.211 & 0.789 \\
\hline
 & 0.500 & 0.500 & 0.0 \\
 & -3.52 & 1.58 & 2.94
\end{array}
$$

Properties:

1. error order: 3 (2)
2. stability regions
3. real axes limit: $-\infty$
4. hermite interpolation
5. gbode flag:`-gbm=sdirk3`

main RK method

embedded RK method



Single-Rate Mode of GBODE

ESDIRK4 method (order 4(3), stages 4, implicit) - default

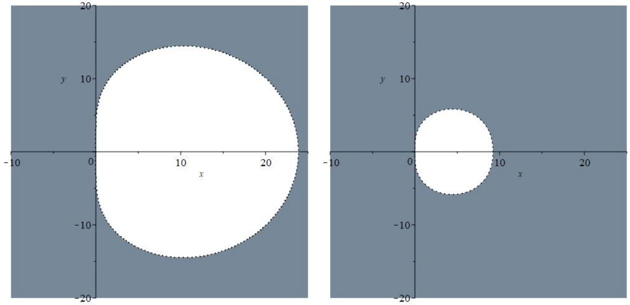
Butcher-Tableau:

$$\begin{array}{c|cccccc}
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.500 & 0.250 & 0.250 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.148 & -0.051 & -0.051 & 0.250 & 0.0 & 0.0 & 0.0 \\
0.625 & -0.0759 & -0.0759 & 0.528 & 0.250 & 0.0 & 0.0 \\
1.04 & -0.725 & -0.725 & 1.59 & 0.658 & 0.250 & 0.0 \\
1.0 & -0.0153 & -0.0153 & 0.387 & 0.502 & -0.108 & 0.250 \\
\hline
 & -0.0153 & -0.0153 & 0.387 & 0.502 & -0.108 & 0.250 \\
 & -2.66 & -2.62 & 4.76 & 1.11 & 0.732 & -0.32
\end{array}$$

Properties:

1. error order: 4 (3)
2. stability regions
3. real axes limit: $-\infty$
4. dense output
5. gbode flag:`-gbm=esdirk4`

main RK method — embedded RK method



Single-Rate Mode of GBODE

Available Runge-Kutta Methods in GBODE

Simulation flag: `-gbm =` (fast state integration method `-gbfm =`)

- Explicit Runge-Kutta method:
  - Standard methods:
    `expl_euler` (order 1),
    `heun` (order 2),
    `merson` (order 4),
    `rungekutta` (order 4),
    `dopri45` (order 5),
    `tsit5` (order 5),
    `fehlberg12` (order 2), `fehlberg45` (order 5), `fehlberg78` (order 8)
  - High order methods:
    `rk810` (order 10), `rk1012` (order 12), `rk1214` (order 14)
  - Strong stability methods:
    `dopriSsc1` (order 1), `dopriSsc2` (order 2),
    `mersonSsc1` (order 1), `mersonSsc2` (order 2),
    `fehlbergSsc1` (order 1), `fehlbergSsc2` (order 2),
    `rungekuttaSsc` (order 1)

Single-Rate Mode of GBODE

Available Runge-Kutta Methods in GBODE

Simulation flag: `-gbm =` (fast state integration method `-gbfm =`)

- Explicit Runge-Kutta method:
  - Standard methods:
    `expl_euler` (order 1),
    `heun` (order 2),
    `merson` (order 4),
    `rungekutta` (order 4),
    `dopri45` (order 5),
    `tsit5` (order 5),
    `fehlberg12` (order 2), `fehlberg45` (order 5), `fehlberg78` (order 8)
  - High order methods:
    `rk810` (order 10), `rk1012` (order 12), `rk1214` (order 14)
  - Strong stability methods:
    `dopriSsc1` (order 1), `dopriSsc2` (order 2),
    `mersonSsc1` (order 1), `mersonSsc2` (order 2),
    `fehlbergSsc1` (order 1), `fehlbergSsc2` (order 2),
    `rungekuttaSsc` (order 1)

Single-Rate Mode of GBODE

Available Runge-Kutta Methods in GBODE

Simulation flag: `-gbm =` (fast state integration method `-gbfm =`)

- Explicit Runge-Kutta method:
  - Standard methods:
    `expl_euler` (order 1),
    `heun` (order 2),
    `merson` (order 4),
    `rungekutta` (order 4),
    `dopri45` (order 5),
    `tsit5` (order 5),
    `fehlberg12` (order 2), `fehlberg45` (order 5), `fehlberg78` (order 8)
  - High order methods:
    `rk810` (order 10), `rk1012` (order 12), `rk1214` (order 14)
  - Strong stability methods:
    `dopriSsc1` (order 1), `dopriSsc2` (order 2),
    `mersonSsc1` (order 1), `mersonSsc2` (order 2),
    `fehlbergSsc1` (order 1), `fehlbergSsc2` (order 2),
    `rungekuttaSsc` (order 1)

Single-Rate Mode of GBODE

Available Runge-Kutta Methods in GBODE

Simulation flag: `-gbm =` (fast state integration method `-gbfm =`)

- Implicit Runge-Kutta method:
  - Standard methods:
    `impl_euler` (order 1), `trapezoid` (order 2)
  - (Explicit) singly-diagonal methods:
    `sdirk2` (order 2), `sdirk3` (order 3), `esdirk2` (order 2), `esdirk3` (order 3), `esdirk4` (order 4)
  - Gaussian methods:
    `gauss2` (order 4), `gauss3` (order 6), `gauss4` (order 8), `gauss5` (order 10), `gauss6` (order 12)
  - Radau methods:
    `radauIA2` (order 3), `radauIA3` (order 5), `radauIA4` (order 7), `radauIIA2` (order 3), `radauIIA3` (order 5), `radauIIA4` (order 7)
  - Lobatto methods:
    `lobattoIIIA2` (order 2), `lobattoIIIA3` (order 4), `lobattoIIIA4` (order 6), `lobattoIIIB2` (order 2), `lobattoIIIB3` (order 4), `lobattoIIIB4` (order 6), `lobattoIIIC2` (order 2), `lobattoIIIC3` (order 4), `lobattoIIIC4` (order 6)

Single-Rate Mode of GBODE

Available Runge-Kutta Methods in GBODE

Simulation flag: `-gbm =` (fast state integration method `-gbfm =`)

- Implicit Runge-Kutta method:
  - Standard methods:
    `impl_euler` (order 1), `trapezoid` (order 2)
  - (Explicit) singly-diagonal methods:
    `sdirk2` (order 2), `sdirk3` (order 3), `esdirk2` (order 2), `esdirk3` (order 3), `esdirk4` (order 4)
  - Gaussian methods:
    `gauss2` (order 4), `gauss3` (order 6), `gauss4` (order 8), `gauss5` (order 10), `gauss6` (order 12)
  - Radau methods:
    `radauIA2` (order 3), `radauIA3` (order 5), `radauIA4` (order 7), `radauIIA2` (order 3), `radauIIA3` (order 5), `radauIIA4` (order 7)
  - Lobatto methods:
    `lobattoIIIA2` (order 2), `lobattoIIIA3` (order 4), `lobattoIIIA4` (order 6), `lobattoIIIB2` (order 2), `lobattoIIIB3` (order 4), `lobattoIIIB4` (order 6), `lobattoIIIC2` (order 2), `lobattoIIIC3` (order 4), `lobattoIIIC4` (order 6)

Single-Rate Mode of GBODE

Available Runge-Kutta Methods in GBODE

Simulation flag: `-gbm =` (fast state integration method `-gbfm =`)

- Implicit Runge-Kutta method:
  - Standard methods:
    `impl_euler` (order 1), `trapezoid` (order 2)
  - (Explicit) singly-diagonal methods:
    `sdirk2` (order 2), `sdirk3` (order 3), `esdirk2` (order 2), `esdirk3` (order 3), `esdirk4` (order 4)
  - Gaussian methods:[a]
    `gauss2` (order 4), `gauss3` (order 6), `gauss4` (order 8), `gauss5` (order 10), `gauss6` (order 12)
  - Radau methods:
    `radauIA2` (order 3), `radauIA3` (order 5), `radauIA4` (order 7), `radauIIA2` (order 3), `radauIIA3` (order 5), `radauIIA4` (order 7)
  - Lobatto methods:
    `lobattoIIIA2` (order 2), `lobattoIIIA3` (order 4), `lobattoIIIA4` (order 6), `lobattoIIIB2` (order 2), `lobattoIIIB3` (order 4), `lobattoIIIB4` (order 6), `lobattoIIIC2` (order 2), `lobattoIIIC3` (order 4), `lobattoIIIC4` (order 6)

[a] Current Restriction: Fully implicit (Gauss, Radau, Lobatto) RK methods are not yet supported for fast state integration.

Single-Rate Mode of GBODE

Available Runge-Kutta Methods in GBODE

Simulation flag: `-gbm =` (fast state integration method `-gbfm =`)

- Implicit Runge-Kutta method:
  - Standard methods:
    `impl_euler` (order 1), `trapezoid` (order 2)
  - (Explicit) singly-diagonal methods:
    `sdirk2` (order 2), `sdirk3` (order 3), `esdirk2` (order 2), `esdirk3` (order 3), `esdirk4` (order 4)
  - Gaussian methods:[a]
    `gauss2` (order 4), `gauss3` (order 6), `gauss4` (order 8), `gauss5` (order 10), `gauss6` (order 12)
  - Radau methods:
    `radauIA2` (order 3), `radauIA3` (order 5), `radauIA4` (order 7), `radauIIA2` (order 3), `radauIIA3` (order 5), `radauIIA4` (order 7)
  - Lobatto methods:
    `lobattoIIIA2` (order 2), `lobattoIIIA3` (order 4), `lobattoIIIA4` (order 6), `lobattoIIIB2` (order 2), `lobattoIIIB3` (order 4), `lobattoIIIB4` (order 6), `lobattoIIIC2` (order 2), `lobattoIIIC3` (order 4), `lobattoIIIC4` (order 6)

[a] Current Restriction: Fully implicit (Gauss, Radau, Lobatto) RK methods are not yet supported for fast state integration.

Single-Rate Mode of GBODE

Available Runge-Kutta Methods in GBODE

Simulation flag: `-gbm =` (fast state integration method `-gbfm =`)

- Implicit Runge-Kutta method:
  - Standard methods:
    `impl_euler` (order 1), `trapezoid` (order 2)
  - (Explicit) singly-diagonal methods:
    `sdirk2` (order 2), `sdirk3` (order 3), `esdirk2` (order 2), `esdirk3` (order 3), `esdirk4` (order 4)
  - Gaussian methods:[a]
    `gauss2` (order 4), `gauss3` (order 6), `gauss4` (order 8), `gauss5` (order 10), `gauss6` (order 12)
  - Radau methods:
    `radauIA2` (order 3), `radauIA3` (order 5), `radauIA4` (order 7), `radauIIA2` (order 3), `radauIIA3` (order 5), `radauIIA4` (order 7)
  - Lobatto methods:
    `lobattoIIIA2` (order 2), `lobattoIIIA3` (order 4), `lobattoIIIA4` (order 6), `lobattoIIIB2` (order 2), `lobattoIIIB3` (order 4), `lobattoIIIB4` (order 6), `lobattoIIIC2` (order 2), `lobattoIIIC3` (order 4), `lobattoIIIC4` (order 6)

[a] Current Restriction: Fully implicit (Gauss, Radau, Lobatto) RK methods are not yet supported for fast state integration.

Single-Rate Mode of GBODE

Recommendation for the choice of integration methods

Simulation flag: `-gbm =` (fast state integration method `-gbfm =`)

- **Explicit** Runge-Kutta method are well suited for **non-stiff equations**:
  `rungekutta` (order 4),
  `dopri45` (order 5, dense output),
  `tsit5` (order 5, dense output),
  `fehlberg78` (order 8).
- Runge-Kutta methods with **strong stability regions** are well suited for **mildly stiff equations**, where the eigenvalues of the system are on the negative real axis:
  `dopriSsc1` (order 1, dense output),
  `dopriSsc2` (order 2, dense output).
- **Implicit** Runge-Kutta method are well suited for **stiff equations**:
  `esdirk2` (order 2, dense output),
  `esdirk3` (order 3, dense output),
  `esdirk4` (order 4, dense output).

Single-Rate Mode of GBODE

Recommendation for the choice of integration methods

Simulation flag: `-gbm =` (fast state integration method `-gbfm =`)

- **Explicit** Runge-Kutta method are well suited for **non-stiff equations**:
  `rungekutta` (order 4),
  `dopri45` (order 5, dense output),
  `tsit5` (order 5, dense output),
  `fehlberg78` (order 8).
- Runge-Kutta methods with **strong stability regions** are well suited for **mildly stiff equations**, where the eigenvalues of the system are on the negative real axis:
  `dopriSsc1` (order 1, dense output),
  `dopriSsc2` (order 2, dense output).
- **Implicit** Runge-Kutta method are well suited for **stiff equations**:
  `esdirk2` (order 2, dense output),
  `esdirk3` (order 3, dense output),
  `esdirk4` (order 4, dense output).

Single-Rate Mode of GBODE

Recommendation for the choice of integration methods

Simulation flag: `-gbm =` (fast state integration method `-gbfm =`)

- **Explicit** Runge-Kutta method are well suited for **non-stiff equations**:
  `rungekutta` (order 4),
  `dopri45` (order 5, dense output),
  `tsit5` (order 5, dense output),
  `fehlberg78` (order 8).
- Runge-Kutta methods with **strong stability regions** are well suited for **mildly stiff equations**, where the eigenvalues of the system are on the negative real axis:
  `dopriSsc1` (order 1, dense output),
  `dopriSsc2` (order 2, dense output).
- **Implicit** Runge-Kutta method are well suited for **stiff equations**:
  `esdirk2` (order 2, dense output),
  `esdirk3` (order 3, dense output),
  `esdirk4` (order 4, dense output).

Single-Rate Mode of GBODE

Configuration (Simulation) Flags of GBODE

General naming convention:

`-gb...`:	flags for single-rate integration method or outer integration (slow states)
`-gbf...`:	flags for bi-rate integration method or inner integration (fast states)

Single-Rate Mode of GBODE

Configuration (Simulation) Flags of GBODE

General naming convention:

`-gb...`:	flags for single-rate integration method or outer integration (slow states)
`-gbf...`:	flags for bi-rate integration method or inner integration (fast states)

Step size control:

`-gbctrl` (`-gbfctrl`) =

- `const`: Constant step size
- `i`: I controller for step size (default)
- `pi`: PI controller for step size
- ...

Single-Rate Mode of GBODE

Configuration (Simulation) Flags of GBODE

General naming convention:

`-gb...`:	flags for single-rate integration method or outer integration (slow states)
`-gbf...`:	flags for bi-rate integration method or inner integration (fast states)

Step size control:
`-gbctrl` (`-gbfctrl`) =

- `const`: Constant step size
- `i`: I controller for step size (default)
- `pi`: PI controller for step size
- ...

Non-linear solver method:
`-gbnls` (`-gbfnls`) =

- `newton`: Newton method, dense solver
- `kinsol`: SUNDIALS KINSOL: Inexact Newton, sparse solver (default)

Single-Rate Mode of GBODE

Configuration (Simulation) Flags of GBODE

General naming convention:

`-gb...`:	flags for single-rate integration method or outer integration (slow states)
`-gbf...`:	flags for bi-rate integration method or inner integration (fast states)

Step size control:
`-gbctrl` (`-gbfctrl`) =

- `const`: Constant step size
- `i`: I controller for step size (default)
- `pi`: PI controller for step size
- ...

Non-linear solver method:
`-gbnls` (`-gbfnls`) =

- `newton`: Newton method, dense solver
- `kinsol`: SUNDIALS KINSOL: Inexact Newton, sparse solver (default)

Extrapolation vs. Embedded RK-method:
`-gberr` (`-gbferr`) =

- 1: Richardson extrapolation
- 0: Embedded RK-method (default)

Single-Rate Mode of GBODE

Configuration (Simulation) Flags of GBODE

General naming convention:

`-gb...`: flags for single-rate integration method or outer integration (slow states)
`-gbf...`: flags for bi-rate integration method or inner integration (fast states)

Step size control:
`-gbctrl` (`-gbfctrl`) =

- `const`: Constant step size
- `i`: I controller for step size (default)
- `pi`: PI controller for step size
- ...

Non-linear solver method:
`-gbnls` (`-gbfnls`) =

- `newton`: Newton method, dense solver
- `kinsol`: SUNDIALS KINSOL: Inexact Newton, sparse solver (default)

Extrapolation vs. Embedded RK-method:
`-gberr` (`-gbferr`) =

- `1`: Richardson extrapolation
- `0`: Embedded RK-method (default)

Interpolation method with error control:
`-gbint` (`-gbfint`) =

- linear: Linear interpolation (1st order)
- hermite: Hermite interpolation (3rd order)
- hermite_a: Hermite interpolation (only for left hand side)
- hermite_b: Hermite interpolation (only for right hand side)
- hermite_errctrl: Hermite interpolation with error control
- dense_output: use dense output formula for interpolation
- dense_output_errctrl: use dense output fomular with error control

Single-Rate Mode of GBODE

GBODE configuration to replace old solvers

old:	`-s=euler`
new:	`-s=gbode -gbm=expl_euler -gbctrl=const`
old:	`-s=heun`
new:	`-s=gbode -gbm=heun -gbctrl=const`
old:	`-s=rungekutta`
new:	`-s=gbode -gbm=rungekutta -gbctrl=const`
old:	`-s=impeuler`
new:	`-s=gbode -gbm=impl_euler -gbctrl=const`
old:	`-s=trapezoid`
new:	`-s=gbode -gbm=trapezoid -gbctrl=const`
old:	`-s=imprungekutta`
new	`-s=gbode -gbm=...`[a] `-gbctrl=const`
old:	`-s=irksco`
new:	`-s=gbode -gbm=trapezoid`
old:	`-s=rungekuttaSsc`
new:	`-s=gbode -gbm=rungekuttaSsc`

[a] is one of the lobatto or radau or gauss Runge Kutta methods

Single-Rate Mode of GBODE

Regression Tests on Some Libraries (solvers: gbode, cvode, master)

Buildings_latest

gbode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	1499	1472	1472	1472	1472	1472	1369	1243

cvode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	1499	1472	1472	1472	1472	1472	1400	1281

master	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	1503	1503	1502	1502	1502	1502	1441	1282

Single-Rate Mode of GBODE

Regression Tests on Some Libraries (solvers: gbode, cvode, master)

ModelicaTest_trunk

gbode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	597	596	592	592	592	589	570	525

cvode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	597	596	592	592	592	589	580	536

master	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	597	596	592	592	592	589	580	537

Single-Rate Mode of GBODE

Regression Tests on Some Libraries (solvers: gbode, cvode, master)

PNlib

gbode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	92	92	92	92	92	92	92	92

cvode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	92	92	92	92	92	92	92	91

master	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	92	92	92	92	92	92	92	92

Single-Rate Mode of GBODE

Regression Tests on Some Libraries (solvers: gbode, cvode, master)

ThermofluidStream

gbode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	75	75	75	75	75	75	59	54

cvode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	75	75	75	75	75	75	65	63

master	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	75	75	75	75	75	75	73	72

Single-Rate Mode of GBODE

Regression Tests on Some Libraries (solvers: gbode, cvode, master)

ClaRa_dev

gbode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	146	145	145	145	145	145	87	44

cvode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	146	145	145	145	145	145	94	49

master	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	146	145	145	145	145	145	96	49

Single-Rate Mode of GBODE

Regression Tests on Some Libraries (solvers: gbode, cvode, master)

Modelica_trunk

gbode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	534	528	526	526	526	526	513	470

cvode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	534	528	526	526	526	526	501	463

master	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	534	528	526	526	526	526	522	477

PowerSystems_latest

gbode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	128	128	121	121	115	115	103	94

cvode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	128	128	121	121	115	115	104	90

master	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	128	128	121	121	115	115	106	95

Single-Rate Mode of GBODE

Regression Tests on Some Libraries (solvers: gbode, cvode, master)

ScalableTestSuite

gbode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	263	259	259	259	259	259	232	218

cvode	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	263	259	259	259	259	259	211	203

master	Total	Frontend	Backend	SimCode	Templates	Compilation	Simulation	Verification
	263	259	259	259	259	259	248	242

Bi-Rate Mode of GBODE

How to use the prototype?

Principle procedure:

1. Define percentage of states for the selection of fast states (flag: `-gbratio`)
2. Outer integrator solves for all states using a selected Runge Kutta method (flag: `-gbm`)
3. Step size control of the outer integrator is based on the slow states error (flag: `-gbctrl`)
4. Inner integrator refines the fast states that do not fullfill the error tolerance using a selected Runge Kutta method (flag: `-gbfm`)
5. Step size control of the inner integrator is based on the fast states error (flag: `-gbfctrl`)
6. For the inner integrator the values of the slow states are interpolated (flag: `-gbint`)

Bi-Rate Mode of GBODE

How to use the prototype?

Principle procedure:

1. Define percentage of states for the selection of fast states (flag: `-gbratio`)
2. Outer integrator solves for all states using a selected Runge Kutta method (flag: `-gbm`)
3. Step size control of the outer integrator is based on the slow states error (flag: `-gbctrl`)
4. Inner integrator refines the fast states that do not fullfill the error tolerance using a selected Runge Kutta method (flag: `-gbfm`)
5. Step size control of the inner integrator is based on the fast states error (flag: `-gbfctrl`)
6. For the inner integrator the values of the slow states are interpolated (flag: `-gbint`)

Bi-Rate Mode of GBODE

How to use the prototype?

Principle procedure:

1. Define percentage of states for the selection of fast states (flag: `-gbratio`)
2. Outer integrator solves for all states using a selected Runge Kutta method (flag: `-gbm`)
3. Step size control of the outer integrator is based on the slow states error (flag: `-gbctrl`)
4. Inner integrator refines the fast states that do not fullfill the error tolerance using a selected Runge Kutta method (flag: `-gbfm`)
5. Step size control of the inner integrator is based on the fast states error (flag: `-gbfctrl`)
6. For the inner integrator the values of the slow states are interpolated (flag: `-gbint`)

Bi-Rate Mode of GBODE

How to use the prototype?

Principle procedure:

1. Define percentage of states for the selection of fast states (flag: `-gbratio`)
2. Outer integrator solves for all states using a selected Runge Kutta method (flag: `-gbm`)
3. Step size control of the outer integrator is based on the slow states error (flag: `-gbctrl`)
4. Inner integrator refines the fast states that do not fullfill the error tolerance using a selected Runge Kutta method (flag: `-gbfm`)
5. Step size control of the inner integrator is based on the fast states error (flag: `-gbfctrl`)
6. For the inner integrator the values of the slow states are interpolated (flag: `-gbint`)

Bi-Rate Mode of GBODE

How to use the prototype?

Principle procedure:

1. Define percentage of states for the selection of fast states (flag: `-gbratio`)
2. Outer integrator solves for all states using a selected Runge Kutta method (flag: `-gbm`)
3. Step size control of the outer integrator is based on the slow states error (flag: `-gbctrl`)
4. Inner integrator refines the fast states that do not fullfill the error tolerance using a selected Runge Kutta method (flag: `-gbfm`)
5. Step size control of the inner integrator is based on the fast states error (flag: `-gbfctrl`)
6. For the inner integrator the values of the slow states are interpolated (flag: `-gbint`)

Bi-Rate Mode of GBODE

How to use the prototype?

Principle procedure:

1. Define percentage of states for the selection of fast states (flag: `-gbratio`)
2. Outer integrator solves for all states using a selected Runge Kutta method (flag: `-gbm`)
3. Step size control of the outer integrator is based on the slow states error (flag: `-gbctrl`)
4. Inner integrator refines the fast states that do not fullfill the error tolerance using a selected Runge Kutta method (flag: `-gbfm`)
5. Step size control of the inner integrator is based on the fast states error (flag: `-gbfctrl`)
6. For the inner integrator the values of the slow states are interpolated (flag: `-gbint`)

Bi-Rate Mode of GBODE

How to use the prototype?

Principle procedure:

1. Define percentage of states for the selection of fast states (flag: `-gbratio`)
2. Outer integrator solves for all states using a selected Runge Kutta method (flag: `-gbm`)
3. Step size control of the outer integrator is based on the slow states error (flag: `-gbctrl`)
4. Inner integrator refines the fast states that do not fullfill the error tolerance using a selected Runge Kutta method (flag: `-gbfm`)
5. Step size control of the inner integrator is based on the fast states error (flag: `-gbfctrl`)
6. For the inner integrator the values of the slow states are interpolated (flag: `-gbint`)

Burgers' equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0,$$

Bi-Rate Mode of GBODE

How to use the prototype?

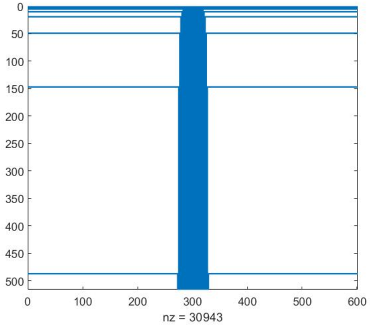
Principle procedure:

1. Define percentage of states for the selection of fast states (flag: `-gbratio`)
2. Outer integrator solves for all states using a selected Runge Kutta method (flag: `-gbm`)
3. Step size control of the outer integrator is based on the slow states error (flag: `-gbctrl`)
4. Inner integrator refines the fast states that do not fullfill the error tolerance using a selected Runge Kutta method (flag: `-gbfm`)
5. Step size control of the inner integrator is based on the fast states error (flag: `-gbfctrl`)
6. For the inner integrator the values of the slow states are interpolated (flag: `-gbint`)

Burgers' equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0,$$

Activity diagram (Burgers' equation):



Conclusions & Future Work

GBODE stands for Generic Bi-rate Ordinary Differential Equation solver and is highly configurable supporting the following features:

Single-rate Mode of GBODE:

- Generic implementation for any (implicit, explicit) Runge-Kutta scheme
- Different methods for step size control
- Support of different extrapolation schemes
- Reliable event handling
- Efficient solving of non-linear equations (incl. sparse matrix handling)

Bi-rate Mode of GBODE (prototype):

- Same options as for the single-rate mode
- Seperation of fast and slow states (inner/outer integration)
- Step size control for each mode
- Efficient event handling

Single-rate Mode of GBODE:

- Handle issues and do bug fixing
- Increase coverage of the MSL and industrial libraries
- Further improve efficiency
  - solution of nonlinear equations, code execution, event handling
- Utilize available parallelization capabilities for simulation speed-up
  - Jacobian evaluation

Single-rate Mode of GBODE:

- Handle issues and do bug fixing
- Increase coverage of the MSL and industrial libraries
- Further improve efficiency
  - solution of nonlinear equations, code execution, event handling
- Utilize available parallelization capabilities for simulation speed-up
  - Jacobian evaluation

Single-rate Mode of GBODE:

- Handle issues and do bug fixing
- Increase coverage of the MSL and industrial libraries
- Further improve efficiency
  - solution of nonlinear equations, code execution, event handling
- Utilize available parallelization capabilities for simulation speed-up
  - Jacobian evaluation

Single-rate Mode of GBODE:

- Handle issues and do bug fixing
- Increase coverage of the MSL and industrial libraries
- Further improve efficiency
  - solution of nonlinear equations, code execution, event handling
- Utilize available parallelization capabilities for simulation speed-up
  - Jacobian evaluation

Single-rate Mode of GBODE:

- Handle issues and do bug fixing
- Increase coverage of the MSL and industrial libraries
- Further improve efficiency
  - solution of nonlinear equations, code execution, event handling
- Utilize available parallelization capabilities for simulation speed-up
  - Jacobian evaluation

Conclusions & Future Work

Single-rate Mode of GBODE:

- Handle issues and do bug fixing
- Increase coverage of the MSL and industrial libraries
- Further improve efficiency
  - solution of nonlinear equations, code execution, event handling
- Utilize available parallelization capabilities for simulation speed-up
  - Jacobian evaluation

Conclusions & Future Work

Single-rate Mode of GBODE:

- Handle issues and do bug fixing
- Increase coverage of the MSL and industrial libraries
- Further improve efficiency
  - solution of nonlinear equations, code execution, event handling
- Utilize available parallelization capabilities for simulation speed-up
  - Jacobian evaluation

Bi-rate Mode of GBODE (prototype):

- Same options as for the single-rate mode
- Avoid evaluation of equations not necessary for the fast states integration
  - Generate dependency graph beforehand
- Selective equation evaluation and event iteration

Single-rate Mode of GBODE:

- Handle issues and do bug fixing
- Increase coverage of the MSL and industrial libraries
- Further improve efficiency
  - solution of nonlinear equations, code execution, event handling
- Utilize available parallelization capabilities for simulation speed-up
  - Jacobian evaluation

Bi-rate Mode of GBODE (prototype):

- Same options as for the single-rate mode
- Avoid evaluation of equations not necessary for the fast states integration
  - Generate dependency graph beforehand
- Selective equation evaluation and event iteration

Single-rate Mode of GBODE:

- Handle issues and do bug fixing
- Increase coverage of the MSL and industrial libraries
- Further improve efficiency
  - solution of nonlinear equations, code execution, event handling
- Utilize available parallelization capabilities for simulation speed-up
  - Jacobian evaluation

Bi-rate Mode of GBODE (prototype):

- Same options as for the single-rate mode
- Avoid evaluation of equations not necessary for the fast states integration
  - Generate dependency graph beforehand
- Selective equation evaluation and event iteration

Conclusions & Future Work

Single-rate Mode of GBODE:

- Handle issues and do bug fixing
- Increase coverage of the MSL and industrial libraries
- Further improve efficiency
  - solution of nonlinear equations, code execution, event handling
- Utilize available parallelization capabilities for simulation speed-up
  - Jacobian evaluation

Bi-rate Mode of GBODE (prototype):

- Same options as for the single-rate mode
- Avoid evaluation of equations not necessary for the fast states integration
  - Generate dependency graph beforehand
- Selective equation evaluation and event iteration

Special Thanks

- **Francesco Casella**, who invited me to Politecnico Milano during my sabbatical and for many fruitful discussions on GBODE. He provided application examples to test the methods and repeatedly showed the limitations of the current implementation.
- **Andreas Heuermann**, who helped me with interfacing GBODE to the nonlinear solvers of OpenModelica including sparse matrix handling.
- **Karim Abdelhak**, **Philip Hannebohm** for several discussions and bug fixing during the implementation work.
- The **OpenModelica developers** that worked hard to achieve the current available OpenModelica environment.

Special Thanks

- **Francesco Casella**, who invited me to Politecnico Milano during my sabbatical and for many fruitful discussions on GBODE. He provided application examples to test the methods and repeatedly showed the limitations of the current implementation.
- **Andreas Heuermann**, who helped me with interfacing GBODE to the nonlinear solvers of OpenModelica including sparse matrix handling.
- **Karim Abdelhak**, **Philip Hannebohm** for several discussions and bug fixing during the implementation work.
- The **OpenModelica developers** that worked hard to achieve the current available OpenModelica environment.

Special Thanks

- **Francesco Casella**, who invited me to Politecnico Milano during my sabbatical and for many fruitful discussions on GBODE. He provided application examples to test the methods and repeatedly showed the limitations of the current implementation.
- **Andreas Heuermann**, who helped me with interfacing GBODE to the nonlinear solvers of OpenModelica including sparse matrix handling.
- **Karim Abdelhak**, **Philip Hannebohm** for several discussions and bug fixing during the implementation work.
- The **OpenModelica developers** that worked hard to achieve the current available OpenModelica environment.

Special Thanks

- **Francesco Casella**, who invited me to Politecnico Milano during my sabbatical and for many fruitful discussions on GBODE. He provided application examples to test the methods and repeatedly showed the limitations of the current implementation.
- **Andreas Heuermann**, who helped me with interfacing GBODE to the nonlinear solvers of OpenModelica including sparse matrix handling.
- **Karim Abdelhak**, **Philip Hannebohm** for several discussions and bug fixing during the implementation work.
- The **OpenModelica developers** that worked hard to achieve the current available OpenModelica environment.

Special Thanks

- **Francesco Casella**, who invited me to Politecnico Milano during my sabbatical and for many fruitful discussions on GBODE. He provided application examples to test the methods and repeatedly showed the limitations of the current implementation.
- **Andreas Heuermann**, who helped me with interfacing GBODE to the nonlinear solvers of OpenModelica including sparse matrix handling.
- **Karim Abdelhak**, **Philip Hannebohm** for several discussions and bug fixing during the implementation work.
- The **OpenModelica developers** that worked hard to achieve the current available OpenModelica environment.

QUESTIONS?