Comparison of Numerical Integration Methods in OpenModelica
Status and Plans on Integration methods

Willi Braun      Bernhard Bachmann

FH Bielefeld
University of
Applied Sciences

University of Applied Sciences Bielefeld
Bielefeld, Germany

February 6, 2017
Motivation

Basic criteria

Stability vs. Performance.
Motivation

Basic criteria

Stability vs. Performance.
Motivation

Basic criteria

Stability vs. Performance.
Motivation

We are 2 times slower, but we want to get 3 times faster.

Rüdiger

Outline:
- Overview of the current available solver
- Comparision of IDA and DASSL
- Improved Symbolic Inline Solver
- Comparison of DAEMode vs. ODEMode
Motivation

We are 2 times slower, but we want to get 3 times faster.

Rüdiger

Outline:
- Overview of the current available solver
- Comparision of IDA and DASSL
- Improved Symbolic Inline Solver
- Comparison of DAEMode vs. ODEMode
Solver in OpenModelica

\[ 0 = f(x(t), \dot{x}(t), y(t), u(t), t) \]
\[ \downarrow \]
\[ 0 = f(x(t), z(t), u(t), t), z(t) = \left( \begin{array}{c} \dot{x}(t) \\ y(t) \end{array} \right) \]
\[ \downarrow \]
\[ z(t) = \left( \begin{array}{c} \dot{x}(t) \\ y(t) \end{array} \right) = g(x(t), u(t), p, t) \]
\[ \downarrow \]
\[ \dot{x}(t) = h(x(t), u(t), p, t) \]
\[ y(t) = k(x(t), u(t), p, t) \]

General Characteristic
- explicit vs. implicit
- higher order
- with step size control
- multi-step methods
Solver in OpenModelica

General Characteristic

- explicit vs. implicit
- higher order
- step size control
- multi-step methods
Solver in OpenModelica

General Characteristic:
- explicit vs. implicit
- higher order
- step size control
- multi-step methods

Explicit Euler

\[ \dot{x} \approx \frac{x(t_{n+1}) - x(t_n)}{h_n} \]

\[ x(t_{n+1}) = x(t_n) + h_n \cdot f(t_n, x(t_n)) \]

- very cheap
- poor stability region

solver name: euler
Solver in OpenModelica

General Characteristic:
- explicit vs. implicit
- higher order
- step size control
- multi-step methods

Implicit Euler

\[ \dot{x} \approx \frac{x(t_n) - x(t_{n-1})}{h_n} \]

\[ x(t_n) = x(t_{n-1}) + h_n \cdot f(t_n, x(t_n)) \]

- very stable
- quite expensive
- non-linear loop solved by KINSOL

solver name: impeuler
Explicit Runge-Kutta Methods

Butcher tableau:

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

Solver in OpenModelica

General Characteristic:

- explicit vs. implicit
- higher order
- step size control
- multi-step methods

solver name: heun, rungekutta
Solver in OpenModelica

General Characteristic:
- explicit vs. implicit
- higher order
- step size control
- multi-step methods

Explicit Runge-Kutta Methods:
- orders 2 and 4
- good performance
- still small stability region

solver name: heun, rungekutta
Solver in OpenModelica

General Characteristic:
- explicit vs. implicit
- higher order
- step size control
- multi-step methods

implicit Runge-Kutta methods

Butcher tableau:

\[
\begin{array}{c|ccc}
\frac{1}{3} & \frac{5}{12} & -\frac{1}{12} \\
1 & \frac{3}{4} & \frac{1}{4} \\
\frac{3}{4} & \frac{1}{4} & \\
\end{array}
\]

solver name: impeuler, trapzoide, imprungekutta
Solver in OpenModelica

General Characteristic:

- explicit vs. implicit
- higher order
- step size control
- multi-step methods

implicit Runge-Kutta methods

- order 1-6 (-impRKOrder=X)
- very stable
- quite expensive
- non-linear loop solved by KINSOL

solver name: impeuler, trapzoide, imprungekutta
Solver in OpenModelica

General Characteristic

- explicit vs. implicit
- higher order
- step size control
- multi-step methods

Explicit Runge-Kutta Step Size Control

Butcher tableau:

| c_1 | 0   | 0 | 0 | ... | 0 | 0 |
| c_2 | a_{21} | 0 | 0 | ... | 0 | 0 |
| c_3 | a_{31} | a_{32} | 0 | ... | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... |
| c_n | a_{n1} | a_{n2} | a_{n3} | ... | a_{n(s-1)} | 0 |
| b_1 | \hat{b}_1 | b_2 | \hat{b}_2 | b_3 | \hat{b}_3 | ... | b_{s-1} | \hat{b}_{s-1} | b_s |
| \hat{b}_1 | \hat{b}_2 | \hat{b}_3 | ... | \hat{b}_{s-1} | \hat{b}_s |

- embedded Runge-Kutta formulas
- quite fast
- better stability region
- Current status: experimental

solver name: rungekuttaSsc
Solver in OpenModelica

General Characteristic

- explicit vs. implicit
- higher order
- step size control
- multi-step methods

Implicit Runge-Kutta Step Size Control

Butcher tableau:

<table>
<thead>
<tr>
<th>$c_1$</th>
<th>$a_{11}$</th>
<th>$a_{12}$</th>
<th>...</th>
<th>$a_{1s}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_2$</td>
<td>$a_{21}$</td>
<td>$a_{22}$</td>
<td>...</td>
<td>$a_{2s}$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$c_n$</td>
<td>$a_{n1}$</td>
<td>$a_{n2}$</td>
<td>...</td>
<td>$a_{ns}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$b_1$</th>
<th>$b_2$</th>
<th>...</th>
<th>$b_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{b}_1$</td>
<td>$\hat{b}_2$</td>
<td>...</td>
<td>$\hat{b}_s$</td>
</tr>
</tbody>
</table>

- Own implementation
- For now order 1-2
- Using own newton solver
- Current status: experimental

solver name: irksco
Solver in OpenModelica
General Characteristic

General Characteristic:
- explicit vs. implicit
- higher order
- step size control
- multi-step methods

Multi-Step BDF method: DASSL

- implicit
- order control
- step size control

solver name: dassl, ida
Solver in OpenModelica

General Characteristic

**General Characteristic:**
- explicit vs. implicit
- higher order
- step size control
- multi-step methods

**SUNDIALS IDA solver**
- DASSL re-implementation in C
- Interface to fast linear solver (KLU)
- usable for large-scale models

solver name: dassl, ida
## Selected compared models

<table>
<thead>
<tr>
<th>model</th>
<th>solver</th>
<th>steps</th>
<th>evalF</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullRobot</td>
<td>dassl</td>
<td>5475</td>
<td>19363</td>
<td>3.114</td>
</tr>
<tr>
<td></td>
<td>ida</td>
<td>5659</td>
<td>19533</td>
<td>3.154</td>
</tr>
<tr>
<td>HeatExhanger</td>
<td>dassl</td>
<td>158</td>
<td>1334</td>
<td>5.972</td>
</tr>
<tr>
<td></td>
<td>ida</td>
<td>161</td>
<td>1374</td>
<td>6.181</td>
</tr>
<tr>
<td>EngineV6</td>
<td>dassl</td>
<td>15179</td>
<td>35622</td>
<td>15.0516</td>
</tr>
<tr>
<td></td>
<td>ida</td>
<td>15509</td>
<td>35667</td>
<td>14.9201</td>
</tr>
<tr>
<td>Themal.Motor</td>
<td>dassl</td>
<td>896</td>
<td>722167</td>
<td>2.44322</td>
</tr>
<tr>
<td></td>
<td>ida</td>
<td>920</td>
<td>722167</td>
<td>2.79349</td>
</tr>
</tbody>
</table>

## ScaleableTestSuite DASSL vs. IDA

Get your own impression:

**DASSL (2017-01-18)** vs. **IDA (2017-01-21)**
Symbolic Inline Integration

Symbolic Inline

Replaces \( \text{der}(\text{states}) \) by forward difference quotient:
\[-\text{symSolver}=\text{expEuler} \]
or by backward difference quotient:
\[-\text{symSolver}=\text{impEuler} \]

Symbolical Implications

- Result is a pure algebraic system
- Apply OpenModelica Backend (e.g. Tearing, symbolic simplification)
- Basic step size control available
- Current status: experimental

Solver name: symSolver, symSolverSsc
Symbolic Inline Integration

Symbolic Inline
Replaces \texttt{der}(states) by forward difference quotient:
\begin{verbatim}
--symSolver=expEuler
\end{verbatim}
or by backward difference quotient:
\begin{verbatim}
--symSolver=impEuler
\end{verbatim}

Symbolical Implications

- Result is a pure algebraic system
- Apply OpenModelica Backend (e.g. Tearing, symbolic simplification)
- Basic step size control available
- Current status: experimental

solver name: symSolver, symSolverSsc
DAE Integration

\[ 0 = f(x(t), \dot{x}(t), y(t), u(t), t) \]

\[ \downarrow \]

\[ 0 = f(x(t), z(t), u(t), t), z(t) = \begin{pmatrix} \dot{x}(t) \\ y(t) \end{pmatrix} \]

\[ z(t) = \begin{pmatrix} \dot{x}(t) \\ y(t) \end{pmatrix} = g(x(t), u(t), p, t) \]

\[ \downarrow \]

\[ \dot{x}(t) = h(x(t), u(t), p, t) \]

\[ y(t) = k(x(t), u(t), p, t) \]

⇒ typical ODE transformation
DAE Integration

\[ 0 = f(x(t), \dot{x}(t), y(t), u(t), t) \]
\[ \downarrow \]
\[ 0 = f(x(t), z(t), u(t), t), \quad z(t) = \left( \begin{array}{c} \dot{x}(t) \\ y(t) \end{array} \right) \]
\[ z(t) = \left( \begin{array}{c} \dot{x}(t) \\ y(t) \end{array} \right) = g(x(t), u(t), p, t) \]
\[ \downarrow \]
\[ \dot{x}(t) = h(x(t), u(t), p, t) \]
\[ y(t) = k(x(t), u(t), p, t) \]

⇒ typical ODE transformation

Skip Matching and Sorting
DAE Integration

\[ 0 = f(x(t), \dot{x}(t), y(t), u(t), t) \]
\[ \downarrow \]
\[ 0 = f(x(t), z(t), u(t), t), z(t) = \left( \begin{array}{c} \dot{x}(t) \\ y(t) \end{array} \right) \]
\[ z(t) = \left( \begin{array}{c} \dot{x}(t) \\ y(t) \end{array} \right) = g(x(t), u(t), p, t) \]
\[ \downarrow \]
\[ \dot{x}(t) = h(x(t), u(t), p, t) \]
\[ y(t) = k(x(t), u(t), p, t) \]

⇒ typical ODE transformation

Current Status

- additional DAE code is generated (simflags="-daeMode")
- Event handling and initialization require matching and sorting
- Two options:
  --daeMode=[dynamic|all]
**Selected compared models**

<table>
<thead>
<tr>
<th>model</th>
<th>solver</th>
<th>steps</th>
<th>evalF</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CascadedFirstOrder_N_6400</td>
<td>dae</td>
<td>2510</td>
<td>2766</td>
<td>3.00101</td>
</tr>
<tr>
<td></td>
<td>ode</td>
<td>2512</td>
<td>3268</td>
<td>5.78234</td>
</tr>
<tr>
<td>DistributionSystemLinear_N_10_M_10</td>
<td>dae</td>
<td>53</td>
<td>149</td>
<td>0.0759903</td>
</tr>
<tr>
<td></td>
<td>ode</td>
<td>73</td>
<td>2493</td>
<td>5.01925</td>
</tr>
</tbody>
</table>

**ScaleableTestSuite DAE vs. ODE**

Get your own impression: **ODE mode (2017-01-12) vs. DAE mode (2017-01-13)**
Plans and Outlook

- Further improvements on the DAEMode
- Develop OSI (based on FMI) for the OM runtimes
- Include the available methods to FMI/CS
- Adding CVODE integrator from SUNDIALS suite
- Further development on irksco and symSolver
Plans and Outlook

Questions

- Further improvements on the DAEMode
- Develop OSI (based on FMI) for the OM runtimes
- Include the available methods to FMI/CS
- Adding CVODE integrator from SUNDIALS suite
- Further development on irksco and symSolver