Selecting the integration method

Dymosim provides a number of different integration methods for the simulation of dynamic systems. In this section the major characteristics of these methods are discussed, and rules of thumb are given which method should be selected for a problem at hand. Note however, that one should not rely on just one integration method for simulation experiments. Instead, some selected results should be checked by two or three other (different) integration methods.

First of all some important issues of integrators are explained, in order to classify the available integration methods in Dymosim. If you are familiar with these issues, just skip the next section.

Integrator properties

Relative and absolute error tolerances

Relative tolerances have to be assigned to tell the integration methods how accurately the solution $x(t)$ should be computed. The tolerance must be greater than zero and is used in a local error test for each component $i$ of the state vector $x_i$, roughly requiring at each step that

$$|\text{local error}| < \text{tolrel} \cdot |x_i| + \text{tolabs}$$

The relative tolerance $\text{tolrel}$ approximately defines the number of expected true digits in the solution. That is, if 3-4 true digits are required in the solution, the default value of $10^{-4}$ should be used. If an element of $x$ is exactly zero or near to zero, the relative tolerance is without meaning and the absolute tolerance $\text{tolabs}$ approximately defines an upper (absolute) limit on the local error. Since user's often have difficulties to see the difference between the relative and the absolute tolerance, in Dymosim $\text{tolabs} = \text{tolrel}$. This is not a good choice, if signals are permanently very small.

In the Dymosim input file it is possible to define a scaling value for every component of the state vector. The integrators effectively solve for the scaled state variables $x(i)/xscale(i)$. If some state variables are very small or if there are big differences in the magnitudes of elements of $x$, the scaling vector should be assigned appropriate elements. E.g., if $x(1)$ is always near $10^{-5}$ and $x(2)$ is always near $10^5$, the scales $xscale(1) = 10^{-5}$, $xscale(2) = 10^5$ should be used. For the scaled equations it makes again sense to use the relative tolerance as an absolute tolerance, too.

Global error

The global error is the difference between the true solution of the initial value problem and the computed approximation. Practically, all present-day codes, including the ones used in

1. More specifically, a root-mean-square norm is used to measure the size of vectors, and the error test uses the magnitude of the solution at the beginning of a step.
Dymosim, control the local error at each step and do not even attempt to control the global error directly. Usually, but not always, the accuracy of the computed state variables $x$ is comparable to the relative error tolerances. The methods will usually, but not always, deliver a more accurate solution if the tolerances are reduced. By comparing two solutions with different tolerances, one can get a fairly good idea of the true error at the bigger tolerances.

**Variable step-size, dense output**

Most integration methods available in Dymosim have a variable step-size algorithm. At every step, a method estimates the local error. The integration step-size is chosen in such a way, that the local error is smaller than the desired maximum local error, defined via the relative and absolute tolerances. This implies, that usually smaller step-sizes are used, if smaller tolerances are defined.

There is one important difference between integrators with respect to the step-size algorithm: The step-sizes of some integrators are not only influenced by the required tolerances but also by the communication time grid. The communication time grid is defined by the StartTime, the StopTime and a communication grid size and determines the points, at which results must be stored. The mentioned integrators just proceed from one grid point to the next one, i.e. the maximum possible step-size of these integrators is limited by the distance of two successive communication points. The step-size is always chosen in such a way, that the integrator meets the grid points exactly. For such methods, the communication grid must be defined carefully. In order to handle stability problems it is possible to define a smaller fixed step-size, and in this case the communication step-size should be a multiple of this smaller fixed step-size.

On the other hand there are integration methods, called dense output methods, which treat communication points differently. The step-size of such integrators is not influenced by the communication grid. The step-size is only chosen according to the required tolerances and the estimated local error. Such methods integrate past the desired communication points and determine the values of the state variables $x$ at the communication points by interpolation, which involves no evaluation of the differential equation. This is advantageous, since the choice of the communication grid, even if a dense grid is used, has nearly no influence on the efficiency of the integration. In Dymosim, the maximum allowed step-size for dense output integrators can be explicitly restricted by parameter `method(hmax)` in the input file, additionally it is possible to turn off dense output.

**Variable order**

Integration methods approximate the solution $x(t)$ internally by a polynomial of order $k_{ord}$. Some methods use a fixed order, other methods vary the order during the simulation. The integration step-size can be usually chosen larger (for the same maximum local error), if the order is bigger, which in turn implies a greater efficiency. The step-size and error control of the integrators are based on the assumption, that the solution $x(t)$ can be differentiated at least $k_{ord}+1$ times. Therefore if it is known for example that the result of a system is not very smooth, a low order method should be chosen. In Dymosim, the maximum order of a variable order method can be explicitly set in the input file via `method(ordmax)`. If the
maximum order is set to one, the variable order integration methods reduce to the simple (explicit or implicit) Euler formula.

**Stiff systems**

“Usual” integrators get in trouble, if “fast” and “slow” signals are present in the solution \( x(t) \). For linear differential equations this corresponds to problems where the system matrix has eigenvalues whose real part is negative and large in magnitude, compared to the reciprocal of the time span of interest. Such systems are called stiff. There exist different algorithms for the integration of stiff and non-stiff systems. However note, that problems with (undamped) highly oscillating signals are not called stiff here. At present there exists no production code to cope with this problem satisfactorily (the step-size is limited by the frequency of the highly oscillating components and therefore the simulation is slow).

The step-size of an integration method is always limited by a method specific stability boundary. The integration only remains stable and produces reliable results, if the step-size is lower than this boundary. If a system is integrated with a non-stiff integration method, and the integrator step-size is limited by the stability boundary and not by the maximum local error, the system is stiff. This usually means, that the step-size chosen from the non-stiff method becomes very small and the efficiency of the integration degrades considerably. However note, that the stiffness depends on the chosen error tolerances. If the error tolerances are decreased, the system may become non-stiff, since the step-size is now limited by the maximum local error and no longer by the stability boundary.

**Dymosim integrators**

At present, Dymosim provides ten different variable step-size integration methods. The most important characteristics of these methods are given in the following table:

<table>
<thead>
<tr>
<th>alg.</th>
<th>method</th>
<th>model type</th>
<th>order</th>
<th>stiff</th>
<th>dense output</th>
<th>root finder</th>
<th>author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DEABM</td>
<td>ODE</td>
<td>1-12</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>Shampine, Gordon, Watts</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(Sham75, Sham80)</td>
</tr>
<tr>
<td>2</td>
<td>LSODE1</td>
<td>ODE</td>
<td>1-12</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>Hindmarsh (Hind80)</td>
</tr>
<tr>
<td>3</td>
<td>LSODE2</td>
<td>ODE</td>
<td>1-5</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>Hindmarsh (Hind80)</td>
</tr>
<tr>
<td>4</td>
<td>LSODAR</td>
<td>ODE</td>
<td>1-12, 1-5</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>Petzold, Hindmarsh (Hind83)</td>
</tr>
<tr>
<td>5</td>
<td>DOPRI5</td>
<td>ODE</td>
<td>5</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>Kraft (Hair87)</td>
</tr>
<tr>
<td>6</td>
<td>DOPRI8</td>
<td>ODE</td>
<td>8</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>Kraft (Hair87)</td>
</tr>
<tr>
<td>7</td>
<td>GRK4T</td>
<td>ODE</td>
<td>4</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>Arnold (Kaps79)</td>
</tr>
<tr>
<td>8</td>
<td>DASSL</td>
<td>DAE</td>
<td>1-5</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>Petzold (Petz82, Bren89)</td>
</tr>
<tr>
<td>9</td>
<td>CDASSL</td>
<td>ODAE</td>
<td>1-5</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>Fuhrer (Fueh88)</td>
</tr>
<tr>
<td>10</td>
<td>HELIX</td>
<td>ODAE</td>
<td>2-24</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>Lubich (Lubi91)</td>
</tr>
</tbody>
</table>

There are also four different fixed step-size integration methods, intended for real-time sim-
The three supported model types are abbreviated as:

ODE Ordinary Differential Equations.
DAE Differential Algebraic Equations (of perturbation index 1).
ODAE Overdetermined Differential Algebraic Equations.

In the following table, some hints are given when to use a specific integration method:

<table>
<thead>
<tr>
<th>alg.</th>
<th>method</th>
<th>model type</th>
<th>order</th>
<th>stiff</th>
<th>dense output</th>
<th>root finder</th>
<th>author(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Euler</td>
<td>ODE</td>
<td>1</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>Dynasim</td>
</tr>
<tr>
<td>12</td>
<td>Rkfix2</td>
<td>ODE</td>
<td>2</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>Rkfix3</td>
<td>ODE</td>
<td>3</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>Rkfix4</td>
<td>ODE</td>
<td>4</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>-</td>
</tr>
</tbody>
</table>

DEABM
If you don’t know much about your problem or you don’t want to think much about it, use method DEABM. DEABM is a robust code with conservative heuristics. Other codes like LSODE1 may be faster in some situations, however DEABM will often be more reliable. Furthermore DEABM is the only code in Dynasim, which will give you a warning if your problem appears to be stiff.

If the calculation of the right hand side of the differential equation is relatively expensive and your problem is non-stiff or moderately stiff and/or you want a great many output points, use DEABM or LSODE1.

LSODE1
LSODE1 is similar to DEABM. It is also a multi-step method using the Adams/Bashforth/Moulton formula. However some important details are realized differently. You can use LSODE1 in the same situations as DEABM.

LSODE2
If your system is stiff, use method LSODE2. This integrator uses a multi-step method with the BDF-formula, the so-called Gear method.

LSODAR
If you need a root finder to handle state events or if your system is stiff on some time intervals and non-stiff on other time intervals, use method LSODAR. LSODAR is a combination of LSODE1 and LSODE2. It starts with the algorithm of LSODE1 and switches between the algorithms of LSODE1 and LSODE2 depending on the stiffness of the system. If your system is non-stiff over the integration period you should get nearly the same results with respect to the number of function evaluations and computing time as with method LSODE1.

DOPRI5
If the calculation of the right hand side of the differential equation is relatively cheap and your problem is non-stiff, use method DOPRI5. This method works well for moderate relative error tolerance requirements of about $\text{tolrel}=10^{-4}$. For higher precision computations a higher order method like DOPRI8 might be more effective with respect to computing time. If you need a great many output points, DOPRI5 and DOPRI8 are not well suited. Use instead a dense output integrator, for example DEABM.

Integrator DOPRI5 uses a Runge-Kutta method with the order-comparing step-size formulae of Prince and Dormand.
DOPRI8

Method DOPRI8 is similar to DOPRI5. The main difference lies in the different orders of the methods. DOPRI8 uses a Runge-Kutta method of order 8. DOPRI8 may be used in the same situations as DOPRI5 if a higher precision is required (say $\text{tolrel}=10^{-5}$).

GRK4T

If the calculation of the right hand side of the differential equation is relatively cheap and your problem is stiff and/or contains highly oscillating solution components, use method GRK4T. GRK4T is a $A(89.3)$-stable linearly-implicit Rosenbrock type method of order four, i.e., the stability region of the integrator is nearly the whole left half plane. GRK4T can be also used for non-stiff problems without any problems (but with a higher amount of computing time than a non-stiff solver as DOPRI5 or DOPRI8).

DASSL

If you have differential algebraic equations (DAE), you have to use DASSL, since this is the only DAE-solver available in Dymosim. DASSL is designed to integrate stiff systems using a BDF-method. Method DASSL calls integrator DASSL, if no indicator functions are defined (no step events can appear) and calls integrator DASSLRT otherwise. DASSLRT is a modified version of DASSL for which a root finder was added. Method DASSL may also be used to integrate (stiff) ordinary differential equations. Dymosim makes the necessary conversions of the right hand side.

ODASSL

ODASSL solves higher index differential algebraic equations or overdetermined differential algebraic equations. It is a modified version of DASSL. Therefore if you integrate an ODE system with ODASSL you will get nearly the same results with respect to the number of function evaluations and computing time as with method DASSL.

MEXX

MEXX is a variant of the MEXX family of codes to solve special index-2 differential algebraic equations with an optional integral invariant. It uses a half-explicit extrapolation algorithm of orders 2 through 24 and is designed to integrate non-stiff systems. Presently, it is not possible to use MEXX for the integration of ordinary differential equations.

---

**Dymosim reference**

**Model functions for Dymosim**

When user defined C-functions are called from model equations a reference to the source code file for these functions needs to be put in a file userfunc.c in the working directory.

```
#include "myfunc.c"
```

In addition to the Modelica functions, the functions min( , ), max( , ) are supported. PackShape and PackMaterial are two functions provided to support visual objects. The following functions are also provided for inclusion in Dymola models when Dymosim is used as simulator.

```
status = LogVariable(x)
```

LogVariable is a function for helping tracking run-time problems in models. An equation like

```
status = LogVariable(x)
```

It will give a message in the event log of the form