OdePkg

A package for solving differential equations with Octave

OdePkg and this document currently are under development

by Thomas Treichl

Table of Contents

1 Beginner's Guide 1

1.1 About OdePkg 1
1.2 OdePkg history and roadmap 1
1.3 Installation and deinstallation 2
1.4 Reporting Bugs 2
1.5 Help is wanted 2
1.6 First example and demos 2

2 User's Guide 5

2.1 Differential Equation Problems 5
2.1.1 ODE problems 5
2.1.2 DAE problems 5
2.1.3 IDE problems 5
2.2 Solver families 6
2.2.1 M–file Runge–Kutta solvers 6
2.2.2 Mex–file Hairer–Wanner solvers 7
2.2.3 Oct–File Cash BDF solvers 8
2.2.4 ODE solver performances 8
2.3 ODE/DAE/IDE options 9
2.4 M–File Function Reference 14
2.5 Oct–File Function Reference 25

3 Programmer's Guide 26

3.1 General description 26
3.2 C++ Function Reference 26
3.2.1 Source file '`odepkg_auxiliary_functions.cc`' 26
3.2.2 Source File '`odepkg_octsolver_mebdfi.cc`' 28

Appendix A 30

A.1 GNU Free Documentation License 30

Function Index 37

Index 38

1 Beginner's Guide

The "Beginner's Guide" is intended for users who are new to OdePkg and who want to solve differential equations with the Octave language and the package OdePkg. In this section it will be explained what OdePkg is about in Section 1.1 [About OdePkg], page 1 and how OdePkg grew up from the beginning in Section 1.2 [OdePkg history and roadmap], page 1. In Section 1.3 [Installation and deinstallation], page 2 it is explained how OdePkg can be installed in Octave and how it can later be removed from Octave. If you encounter problems while using OdePkg then have a look at Section 1.4 [Reporting Bugs], page 2 how these bugs can be reported. In the Section 1.6 [First example and demos], page 2 a first example is explained.

1.1 About OdePkg

OdePkg is part of the **GNU Octave Repository** (resp. the Octave–Forge project) that was initiated by Paul Kienzle in the year 2000 and that is hosted at `http://octave.sourceforge.net`. The package includes commands for setting up various options, output functions etc. before solving a set of differential equations with the solver functions that are included.

OdePkg formerly was initiated to solve ordinary differential equations (ODEs) only but there are already improvements so that differential algebraic equations (DAEs) in explicit form and in implicit form (IDEs) can also be solved. At this time OdePkg is under development with the goal to make a package that is mostly compatible to proprietary solver products.

1.2 OdePkg history and roadmap

OdePkg Version 0.0.1
: The initial release was a modification of the old "ode package" that is hosted at Octave–Forge and that was written by Marc Compere somewhen between 2000 and 2001. The four variable step–size Runge–Kutta algorithms in three solver files and the three fixed step–size solvers have been merged. It was possible to set some options for these solvers. The four output–functions (`odeprint`, `odeplot`, `odephas2` and `odephas3`) have been added along with other examples that initially have not been there.

OdePkg Version 0.1.x
: The major milestone along versions 0.1.x was that four stable solvers have been implemented (ie. `ode23`, `ode45`, `ode54` and `ode78`) supporting all options that can be set for these kind of solvers and also all necessary functions for setting their options (eg. `odeset`, `odepkg_structure_check`, `odepkg_event_handle`). Since version 0.1.3 there is also source code available that interfaces the Fortran solver '`dopri5.f`' (that is written by Ernst Hairer and Gerhard Wanner, cf. '`odepkg_mexsolver_dopri5.c`' and the helper files '`odepkgext.c`' and '`odepkgmex.c`').

OdePkg Version 0.2.x
: The main work along version 0.2.x was to make the interface functions for the non–stiff and stiff solvers from Ernst Hairer and Gerhard Wanner enough stable so that they could be compiled and installed by default. Wrapper functions have been added to the package containing a help text and test functions (eg. `ode2r`, `ode5r`, `oders`). Six testsuite functions have been added to check the performance of the different solvers (eg. `odepkg_testsuite_chemakzo`, `odepkg_testsuite_oregonator`).

(current) Version 0.3.x
: Fixed some minor bugs along version 0.3.x. Thanks to Jeff Cash, who released his Fortran `mebdfX` solvers under the GNU GPL V2 after some discussion. The first IDE solver `odebdi` appeared that is an interface function for Cash's `mebdfi` Fortran core solver. With version 0.3.5 of OdePkg a first new interface function was created based on Octave's C++ `DEFUN_DLD` interface to achieve the highest performance available. Added more examples and testsuite functions (eg. `odepkg_equations_ilorenz`, `odepkg_testsuite_implrober`). Porting all at this time present Mex–file solvers to Octave's C++ `DEFUN_DLD` interface. Ongoing work with this manual.

(future) Version 0.4.x
: (Maybe) Fetching and adding the DASRT IDE solver from Netlib. Ongoing work with this manual.

(future) Version 0.5.x
: (Maybe) Adding a last type of solvers (DDE) to OdePkg.

(future) Version 0.6.x
: (Maybe) A lot of compatibility tests.

(future) Version 0.7.x
: (Maybe) Final release before version 1.0.0.

(future) Version 1.0.0
: Completed OdePkg release 1.0.0 with M–solvers and DLD–solvers.

1.3 Installation and deinstallation

OdePkg can be installed easily using the `pkg` command in Octave. To install OdePkg download the latest release of OdePkg from the Octave–Forge download site, then get into that directory where the downloaded release of OdePkg has been saved, start Octave and type

```
pkg install odepkg-x.x.x.tar.gz
```

where '`x.x.x`' in the name of the '`*.tar.gz`' file is the current release number of OdePkg that is available. If you want to deinstall resp. remove OdePkg then simply type

```
pkg uninstall odepkg
```

and make sure that OdePkg has been removed completely and does not appear in the list of installed packages anymore with the following command

```
pkg list
```

1.4 Reporting Bugs

If you encounter problems during the installation process of OdePkg with the `pkg` command or if you have an OdePkg that seems to be broken or if you encounter problems while using OdePkg or if you find bugs in the source codes then please report all of that via email at the Octave–Forge mailing–list using the email address `octave-dev@lists.sourceforge.net` and directly send a copy to the email address `treichl@users.sourceforge.net` . Not only bugs are welcome but also any kind of comments are welcome (eg. if you think that OdePkg is absolutely useful or even unnecessary).

1.5 Help is wanted

-*- TODO -*-

1.6 First example and demos

Have a look at the first ordinary differential equation with the name "`foo`". The `foo` equation of second order may be of the form $y''(t) + C_1 y'(t) + C_2 y(t) = C_3$. With the substitutions $y_1(t) = y(t)$ and $y_2(t) = y'(t)$ this differential equation of second order can be split into two differential equations of first order, ie. $y_1'(t) = y_2(t)$ and $y_2'(t) = -C_1 y_2(t) - C_2 y_1(t) + C_3$. Next the numerical values for the constants need to be defined, ie. $C_1 = 2.0$, $C_2 = 5.0$, $C_3 = 10.0$. This set of ordinary differential equations can then be written as an Octave M–file function like

```
function vdy = foo (vt, vy, varargin)
  vdy(1,1) = vy(2);
  vdy(2,1) = - 2.0 * vy(2) - 5.0 * vy(1) + 10.0;
endfunction
```

It can be seen that this ODEs do not depend on time, nevertheless the first input argument of this function needs to be defined as the time argument *vt* followed by a solution array argument `vy` as the second input argument and a variable size input argument `varargin` that can be used to set up user defined constants or control variables.

As it is known that `foo` is a set of *ordinary* differential equations we can choose one of the four M–file Runge–Kutta solvers (cf. Section 2.2 [Solver families], page 6). It is also known that the time period of interest may be between $t_0 = 0.0$ and $t_e = 5.0$ as well as that the initial values of the ODEs are $y_1(t = 0) = 0.0$ and $y_2(t = 0) = 0.0$. Solving this set of ODEs can be done by typing the following commands in Octave

```
ode45 (@foo, [0 5], [0 0]);
```

A figure window opens and it can be seen how this ODEs are solved over time. For some of the solvers that come with OdePkg it is possible to define exact time stamps for which an solution is required. Then the example can be called eg.

```
ode45 (@foo, [0:0.1:5], [0 0]);
```

If it is not wanted that a figure window is opened while solving then output arguments have to be used to catch the results of the solving process and to not pass the results to the figure window, eg.

```
[t, y] = ode45 (@foo, [0 5], [0 0]);
```

Results can also be obtained in form of an Octave structure if one output argument is used like in the following example. Then the results are stored in the fields `S.x` and `S.y`.

```
S = ode45 (@foo, [0 5], [0 0]);
```

As noticed before, a function for the ordinary differential equations must not be rewritten all the time if some of the parameters are going to change. That's what the input argument `varargin` can be used for. So rewrite the function `foo` into `newfoo` the following way

```
function vdy = newfoo (vt, vy, varargin)
  vdy(1,1) = vy(2);
  vdy(2,1) = -varargin{1}*vy(2)-varargin{2}*vy(1)+varargin{3};
endfunction
```

There is nothing said anymore about the constant values but if using the following caller routine in the Octave interpreter window then the same results can be obtained with the new function `newfoo` as before with the function `foo` (ie. the parameters are directly feed through from the caller routine `ode45` to the function `newfoo`)

```
ode45 (@newfoo, [0 5], [0 0], 2.0, 5.0, 10.0);
```

OdePkg can do much more while solving differential equation problems, eg. setting up other output functions instead of the function `odeplot` or setting up other tolerances for the solving process etc. As a last example in this beginning chapter it is shown how this can be done, ie. with the command `odeset`

```
A = odeset ('OutputFcn', @odeprint);
ode45 (@newfoo, [0 5], [0 0], A, 2.0, 5.0, 10.0);
```

or

```
A = odeset ('OutputFcn', @odeprint, 'AbsTol', 1e-5);
ode45 (@newfoo, [0 5], [0 0], A, 2.0, 5.0, 10.0);
```

The options structure `A` that can be set up with with the command `odeset` must always be the fourth input argument when using the ODE solvers and the DAE solvers but if you are using an IDE solver then `A` must be the fifth input argument (cf. Section 2.2 [Solver families], page 6). The various options that can be set with the command `odeset` are described in Section 2.3 [ODE/DAE/IDE options], page 9.

Further examples have also been implemented within OdePkg. These example files and functions are of the form `odepkg_equations_*`. Different testsuite examples have been added that are stored in files with filenames `odepkg_testsuite_*`.

Before reading the next chapter note that nearly every function that comes with OdePkg has its own help text and its own demos. Look for yourself how the different functions, options and combinations can be used. If you want to have a look at the help description of a special function then type

```
help fcnname
```

in the Octave window where `fcnname` is the name of the function for the help text to be viewed. Type

```
demo fcnname
```

in the Octave window where `fcnname` is the name of the function of the demo to run. Least but not last write

```
doc odepkg
```

for opening this manual in the texinfo reader of the Octave window.

2 User's Guide

The "User's Guide" is intended for trained users who already know in principal how to solve differential equations with the Octave language and OdePkg. In this chapter it will be explained which problems can be solved with OdePkg in Section 2.1 [Differential Equation Problems], page 5. It will be explained which solvers can be used for the different kind of problems in Section 2.2 [Solver families], page 6 and which options can be set for the optimization of the solving process in Section 2.3 [ODE/DAE/IDE options], page 9. The help text of all M–file functions and all Oct-file functions have been extracted and are displayed in the sections Section 2.4 [M–File Function Reference], page 14 and Section 2.5 [Oct–File Function Reference], page 25.

2.1 Differential Equation Problems

In this section the different kind of differential equation problems that can be solved with OdePkg are explained. The formulation of ordinary differential equations is described in section Section 2.1.1 [ODE problems], page 5 followed by the description of explicetly formulated differential algebraic equations in section Section 2.1.2 [DAE problems], page 5 and implicetely formulated differential algebraic equations Section 2.1.3 [IDE problems], page 5.

2.1.1 ODE problems

ODE problems in general are of the form $y'(t) = f(t, y)$ where $y'(t)$ may be a scalar or vector of derivatives. The variable t always is a scalar describing one point of time and the variable $y(t)$ is a scalar or vector of solutions from the last time step of the set of ordinary differential equations. If the problem is non–stiff then the Section 2.2.1 [M–file Runge–Kutta solvers], page 6 can be used to solve such kind of differential equation problems but if the problem is stiff then it is recommended to use the Section 2.2.2 [Mex–file Hairer–Wanner solvers], page 7. An ODE problem definition in Octave must look like

```
function [dy] = ODEproblem (t, y, varargin)
```

2.1.2 DAE problems

DAE problems in general are of the form $M(t, y) \cdot y'(t) = f(t, y)$ where $y'(t)$ may be a scalar or vector of derivatives. The variable t always is a scalar describing one point of time and the variable $y(t)$ is a scalar or vector of solutions from the set of differential algebraic equations. The variable $M(t, y)$ is the squared *singular* mass matrix that may depend on y and t. If $M(t, y)$ is not *singular* then the set of equations from above can normally also be written as an ODE problem. If it does not depend on time then it can be defined as a constant matrix or a function. If it does depend on time then it must be defined as a function. Use the command `odeset` to pass the mass matrix information to the solver function (cf. Section 2.3 [ODE/DAE/IDE options], page 9). If the problem is non–stiff then the Section 2.2.1 [M–file Runge–Kutta solvers], page 6 can be used to solve such kind of differential equation problems but if the problem is stiff then it is recommended to use the Section 2.2.2 [Mex–file Hairer–Wanner solvers], page 7. A DAE problem definition in Octave must look like

```
function [dy] = DAEproblem (t, y, varargin)
```

and the mass matrix definition can either be a constant mass matrix or a valid function handle to a mass matrix calculation function that can be set with the command `odeset` (cf. option `Mass` of section Section 2.3 [ODE/DAE/IDE options], page 9).

2.1.3 IDE problems

IDE problems in general are of the form $y'(t) + f(t, y) = 0$ where $y'(t)$ may be a scalar or vector of derivatives. The variable t always is a scalar describing one point of time and the variable $y(t)$

is a scalar or vector of solutions from the set of implicit differential equations. Only IDE solvers can be used to solve such kind of differential equation problems. A DAE problem definition in Octave must look like

```
function [residual] = IDEproblem (t, y, yd, varargin)
```

2.2 Solver families

In this section the different kind of solvers are introduced that have been implemented in OdePkg. This section starts with the basic M–file Runge–Kutta solvers in section Section 2.2.1 [M–file Runge–Kutta solvers], page 6 and is continued with the Mex–file Hairer–Wanner solvers in section Section 2.2.2 [Mex–file Hairer–Wanner solvers], page 7. Performance tests have also been added to the OdePkg. Some of these performance results have been added to section Section 2.2.4 [ODE solver performances], page 8.

2.2.1 M–file Runge–Kutta solvers

The M–file Runge–Kutta solvers are written in the Octave interpreter language and that are stored as '`*.m`'–files. There have been implemented four different solvers with a very similiar structure, ie. `ode23`, `ode45`, `ode54` and `ode78`[1].

The order of all of the following Runge–Kutta methods is the order of the local truncation error, which is the principle error term in the portion of the Taylor series expansion that gets dropped, or intentionally truncated. This is different from the local error which is the difference between the estimated solution and the actual, or true solution. The local error is used in stepsize selection and may be approximated by the difference between two estimates of different order, $l(h) = x(O(h+1)) - x(O(h))$. With this definition, the local error will be as large as the error in the lower order method. The local truncation error is within the group of terms that gets multipled by h when solving for a solution from the general Runge–Kutta method. Therefore, the order–p solution created by the Runge–Kunge method will be roughly accurate to $O(h^{(p+1)})$ since the local truncation error shows up in the solution as $e = h \cdot d$ which is h–times an $O(h^p)$–term, or rather $O(h^{(p+1)})$.

`ode23` Integrates a system of non–stiff ordinary differential equations (non-stiff ODEs) using second and third order Runge–Kutta formulas. This particular third order method reduces to Simpson's 1/3 rule and uses the third order estimation for the output solutions. Third order accurate Runge–Kutta methods have local and global errors of $O(h^4)$ and $O(h^3)$ respectively and yield exact results when the solution is a cubic (the variable h is the step size from one integration step to another integration step). This solver requires three function evaluations per integration step.

`ode45` Integrates a system of non–stiff ordinary differential equations (non-stiff ODEs) using fourth and fifth order embedded formulas from Fehlberg. This is a fourth–order accurate integrator therefore the local error normally expected is $O(h^5)$. However, because this particular implementation uses the fifth–order estimate for x_{out} (ie. local extrapolation) moving forward with the fifth–order estimate should yield local error of $O(h^6)$. This solver requires six function evaluations per integration step.

`ode54` Integrates a system of non–stiff ordinary differential equations (non-stiff ODEs) using fifth and fourth order Runge–Kutta formulas. The Fehlberg 4(5) of the `ode45`

[1] The descriptions for these Runge–Kutta solvers have been taken from the help texts of the initial M–file Runge–Kutta solvers that were written by Marc Compere, he also pointed out that "a relevant discussion on step size choice can be found on page 90ff in U.M. Ascher, L.R. Petzold, Computer Methods for Ordinary Differential Equations and Differential–Agebraic Equations, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1998".

pair is established and works well, however, the Dormand–Prince 5(4) pair minimizes the local truncation error in the fifth–order estimate which is what is used to step forward (local extrapolation). Generally it produces more accurate results and costs roughly the same computationally. This solver requires seven function evaluations per integration step.

`ode78` Integrates a system of non–stiff ordinary differential equations (non-stiff ODEs) using seventh and eighth order Runge–Kutta formulas. This is a seventh–order accurate integrator therefore the local error normally expected is $O(h^8)$. However, because this particular implementation uses the eighth–order estimate for x_{out} moving forward with the eighth–order estimate will yield errors on the order of $O(h^9)$. This solver requires thirteen function evaluations per integration step.

2.2.2 Mex–file Hairer–Wanner solvers

The Mex–file Hairer–Wanner solvers are written in Fortran (hosted at `http://www.unige.ch/~hairer`) and have been added to the OdePkg as a compressed file with the name '`hairer.tgz`'. The licence of these solvers is a modified BSD license (without advertising clause) and can be found as '`licence.txt`' file in the '`hairer.tgz`' package and therefore the Fortran files are GPL compatible. Papers and other details about these solvers can be found at the host adress.

Interface functions for these solvers have been created and have been added to the OdePkg. Their names are '`odepkg_mexsolver_xxx.c`' where '`xxx`' is the name of the Fortran file that is interfaced. The corresponding '`odepkg_mexsolver_xxx.mex`' files are created automatically when installing OdePkg with the `pkg` command, but can also be build manually with the instructions given as a preamble of every '`odepkg_mexsolver_xxx.c`' file.

To provide a shorter name to access these solver functions also wrapper functions have been added that do link to the interface functions, eg. the command `oderd` links to the interface functions `odepkg_mexsolver_radau` and should do exactly the same. Another reason of adding wrapper functions was that help texts, demos and tests cannot be added to the '`odepkg_mexsolver_xxx.c`' files. For accessing the help texts, demos and tests for one of these solvers you should therefore always use the name of the wrapper function, eg. `help oderd`.

The Mex–file Hairer–Wanner solvers have been added to the OdePkg to also solve stiff ordinary differential equations that cannot be solved with one of the M–file Runge–Kutta solvers. The following table gives an overview about which solver can be used for the different kind of problems.

ODE Problem	**Solver name**	**Wrapper file**	**Interface file**	**Fortran file**
(**deprecated**) Non–stiff	DOPRI5	'`ode5d.m`'	'`odepkg_mexsolver_dopri5.c`'	'`dopri5.f`'
(**deprecated**) Non–stiff	DOP853	'`ode8d.m`'	'`odepkg_mexsolver_dop853.c`'	'`dop853.f`'
(**deprecated**) Non–stiff	ODEX	'`odeox.m`'	'`odepkg_mexsolver_odex.c`'	'`odex.f`'
Stiff	RADAU	'`ode2r.m`'	'`odepkg_mexsolver_radau.c`'	'`radau.f`'
Stiff	RADAU5	'`ode5r.m`'	'`odepkg_mexsolver_radau5.c`'	'`radau5.f`'
Stiff	RODAS	'`oders.m`'	'`odepkg_mexsolver_rodas.c`'	'`rodas.f`'
Stiff	SEULEX	'`odesx.m`'	'`odepkg_mexsolver_seulex.c`'	'`seulex.f`'

Overview about Fortran, Interface and Wrapper files for Hairer–Wanner solvers.

2.2.3 Oct–File Cash BDF solvers

The backward differentiation algorithm solvers have been written by Jeff Cash in the Fortran language and that are hosted at `http://pitagora.dm.uniba.it/~testset`. They have been added to the OdePkg as a compressed file with the name '`cash.tgz`'. The license of these solvers is a General Public License V2 that can be found as a preamble of each Fortran solver source file. Papers and other details about these solvers can be found at the host adress given before and also at Jeff Cash's homepage at `http://www.ma.ic.ac.uk/~jcash`.

Interface functions for these solvers have been created and that have been added to the OdePkg. Their names are '`odepkg_octsolver_xxx.cc`' where '`xxx`' is the name of the Fortran file that is interfaced. There is created one '`*.oct`'–file with the name '`odepkg_dldsolver_functions.oct`' that includes all solver functions that are accessible in Octave. This file is created automatically when OdePkg is installed with the `pkg` command. Each solver '`odepkg_octsolver_xxx.cc`' file can also be compiled manually with the instructions given as a preamble of every '`odepkg_octsolver_xxx.cc`' file.

2.2.4 ODE solver performances

```
>> odepkg ('odepkg_performance_mathires');
-----------------------------------------------------------------------------------------
 Solver  RelTol  AbsTol   Init   Mescd    Scd  Steps  Accept  FEval  JEval  LUdec    Time
-----------------------------------------------------------------------------------------
 ode113  1e-007  1e-007  1e-009   7.57   5.37  24317   21442  45760                11.697
  ode23  1e-007  1e-007  1e-009   7.23   5.03  13876   13862  41629                 2.634
  ode45  1e-007  1e-007  1e-009   7.91   5.70  11017   10412  66103                 2.994
 ode15s  1e-007  1e-007  1e-009   7.15   4.95    290     273    534      8     59   0.070
 ode23s  1e-007  1e-007  1e-009   6.24   4.03    702     702   2107    702    702   0.161
 ode23t  1e-007  1e-007  1e-009   6.00   3.79    892     886   1103      5     72   0.180
ode23tb  1e-007  1e-007  1e-009   5.85   3.65    735     731   2011      5     66   0.230
-----------------------------------------------------------------------------------------

octave:1> odepkg ('odepkg_performance_octavehires');
-----------------------------------------------------------------------------------------
 Solver  RelTol  AbsTol   Init   Mescd    Scd  Steps  Accept  FEval  JEval  LUdec    Time
-----------------------------------------------------------------------------------------
  ode23   1e-07   1e-07   1e-09   7.95   5.53  16179   13646  48534               168.182
  ode45   1e-07   1e-07   1e-09   8.06   5.64   9401    9398  56400               134.011
  ode54   1e-07   1e-07   1e-09   8.31   5.89   8854    7697  61971               127.261
  ode78   1e-07   1e-07   1e-09   9.06   6.64   7287    6613  94718               168.769
  odeox   1e-07   1e-07   1e-09   6.67   4.25  10969    8881 194129               226.890
  ode5d   1e-07   1e-07   1e-09   0.14  -2.28   1014    1014   6086                 6.775
  ode8d   1e-07   1e-07   1e-09   0.16  -2.26   1046    1030  15385                17.602
  ode2r   1e-07   1e-07   1e-09   7.69   5.27     59      59    849     50     59   1.231
  ode5r   1e-07   1e-07   1e-09   7.55   5.13     81      81    671     71     81   1.380
  odesx   1e-07   1e-07   1e-09   6.63   4.21     39      37   1135     27    190   1.782
  oders   1e-07   1e-07   1e-09   7.08   4.66    138     138    828    138    138   2.071
-----------------------------------------------------------------------------------------


>> odepkg ('odepkg_performance_matchemakzo');
-----------------------------------------------------------------------------------------
 Solver  RelTol  AbsTol   Init   Mescd    Scd  Steps  Accept  FEval  JEval  LUdec    Time
-----------------------------------------------------------------------------------------
 ode113  1e-007  1e-007  1e-007    NaN    Inf      -       -      -      -      -       -
  ode23  1e-007  1e-007  1e-007    NaN    Inf     15      15     47                 0.431
  ode45  1e-007  1e-007  1e-007    NaN    Inf     15      15     92                 0.170
 ode15s  1e-007  1e-007  1e-007   7.04   6.20    161     154             4     35   0.521
 ode23s  1e-007  1e-007  1e-007   7.61   6.77   1676    1676   5029   1676   1677   2.704
 ode23t  1e-007  1e-007  1e-007   5.95   5.11    406     404             3     39   0.611
ode23tb  1e-007  1e-007  1e-007    NaN    Inf    607           3036      1    608   6.730
-----------------------------------------------------------------------------------------

octave:1> odepkg ('odepkg_performance_octavechemakzo');
-----------------------------------------------------------------------------------------
```

Solver	RelTol	AbsTol	Init	Mescd	Scd	Steps	Accept	FEval	JEval	LUdec	Time
ode23	1e-07	1e-07	1e-07	0.45	-0.43	432	385	1293			2.926
ode45	1e-07	1e-07	1e-07	0.45	-0.43	277	238	1656			3.087
ode54	1e-07	1e-07	1e-07	0.45	-0.43	216	214	1505			2.769
ode78	1e-07	1e-07	1e-07	0.45	-0.43	210	170	2717			4.700
ode78	1e-07	1e-07	1e-07	2.94	2.05	193	160	4815			6.150
ode5d	1e-07	1e-07	1e-07	2.95	2.06	234	234	1406			1.499
ode8d	1e-07	1e-07	1e-07	2.95	2.06	161	142	2056			2.149
ode2r	1e-07	1e-07	1e-07	8.50	7.57	43	43	372	39	43	0.486
ode5r	1e-07	1e-07	1e-07	8.50	7.57	43	43	372	39	43	0.491
odesx	1e-07	1e-07	1e-07	7.46	6.53	22	22	502	19	96	0.597
oders	1e-07	1e-07	1e-07	7.92	7.04	68	67	401	66	67	0.642

2.3 ODE/DAE/IDE options

The default values of an OdePkg options structure can be displayed with the command `odeset`. If `odeset` is called without any input argument and one output argument then a OdePkg options structure with default values is created, eg.

```
A = odeset ();
disp (A);
```

Other values than default values can also be set with the command `odeset`. The function description of the command `odeset` can be found in the Section 2.4 [M–File Function Reference], page 14. The values that can be set with this command are

'`RelTol`' The option '`RelTol`' is used to set the relative error tolerance for the error estimation of the solver that is used while solving. It can either be a positive scalar or a vector with every element of the vector being a positive scalar (this depends on the solver that is used if both variants are supported). The definite error estimation equation also depends on the solver that is used but generalized it may be of the form $e(t) = max(r_{tol}^T y(t), a_{tol})$. Evaluate the following example for the visualization of the effect if this option is set

```
A = odeset ('RelTol', 1, 'OutputFcn', @odeplot);
ode78 (@odepkg_equations_vanderpol, [0 20], [2 0], A);
B = odeset ('RelTol', 1e-10, 'OutputFcn', @odeplot);
ode78 (@odepkg_equations_vanderpol, [0 20], [2 0], B);
```

'`AbsTol`' The option '`AbsTol`' is used to set the absolute error tolerance for the error estimation of the solver that is used while solving. It can either be a positive scalar or a vector with every element of the vector being a positive scalar (it depends on the solver that is used if both variants are supported). The definite error estimation equation also depends on the solver that is used but generalized it may be of the form $e(t) = max(r_{tol}^T y(t), a_{tol})$. Evaluate the following example for the visualization of the effect if this option is set

```
A = odeset ('AbsTol', 1e-3, 'OutputFcn', @odeplot);
ode78 (@odepkg_equations_vanderpol, [0 20], [2 0], A);
B = odeset ('AbsTol', 1e-10, 'OutputFcn', @odeplot);
ode78 (@odepkg_equations_vanderpol, [0 20], [2 0], B);
```

'`NormControl`'

The option '`NormControl`' is used to set the type of error tolerance calculation of the solver that is used while solving. It can either be the string `'on'` or `'off'`. At the time the solver starts solving a warning message may be displayed if the solver will ignore the `'on'` setting of this option because of an unhandled resp. missing implementation. If set `'on'` then the definite error estimation equation also depends on the solver that is used but generalized it may be of the form $e(t) = max(r_{tol}^T max(norm(y(t), \infty)), a_{tol})$. Evaluate the following example for the visualization of the effect if this option is set

```
A = odeset ('NormControl', 'on', 'OutputFcn', @odeplot);
ode78 (@odepkg_equations_vanderpol, [0 20], [2 0], A);
B = odeset ('NormControl', 'off', 'OutputFcn', @odeplot);
ode78 (@odepkg_equations_vanderpol, [0 20], [2 0], B);
```

'MaxStep' The option 'MaxStep' is used to set the maximum step size for the solver that is used while solving. It can only be a positive scalar. By default this value is set internally by every solver and also may differ when using different solvers. Evaluate the following example for the visualization of the effect if this option is set

```
A = odeset ('MaxStep', 10, 'OutputFcn', @odeprint);
ode78 (@odepkg_equations_vanderpol, [0 20], [2 0], A);
B = odeset ('MaxStep', 1e-1, 'OutputFcn', @odeprint);
ode78 (@odepkg_equations_vanderpol, [0 20], [2 0], B);
```

'InitialStep'
The option 'InitialStep' is used to set the initial first step size for the solver. It can only be a positive scalar. By default this value is set internally by every solver and also may be different when using different solvers. Evaluate the following example for the visualization of the effect if this option is set

```
A = odeset ('InitialStep', 1, 'OutputFcn', @odeprint);
ode78 (@odepkg_equations_vanderpol, [0 1], [2 0], A);
B = odeset ('InitialStep', 1e-5, 'OutputFcn', @odeprint);
ode78 (@odepkg_equations_vanderpol, [0 1], [2 0], B);
```

'InitialSlope'
The option 'InitialSlope' is not handled by any of the solvers by now.

'OutputFcn'
The option 'OutputFcn' can be used to set up an output function for displaying the results of the solver while solving. It must be a function handle to a valid function. There are four predefined output functions available with OdePkg. `odeprint` prints the actual time values and results in the Octave window while solving, `odeplot` plots the results over time in a new figure window while solving, `odephas2` plots the first result over the second result as a two–dimensional plot while solving and `odephas3` plots the first result over the second result over the third result as a three–dimensional plot while solving. Evaluate the following example for the visualization of the effect if this option is set

```
A = odeset ('OutputFcn', @odeprint);
ode78 (@odepkg_equations_vanderpol, [0 2], [2 0], A);
```

User defined output functions can also be used. A typical framework for a self–made output function may then be of the form

```
function [vret] = odeoutput (vt, vy, vdeci, varargin)
  switch vdeci
    case 'init'
      ## Do everything needed to intialize output function
    case 'calc'
      ## Do everything needed to create output
    case 'done'
      ## Do everything needed to clean up output function
```

```
  endswitch
endfunction
```

The output function `odeplot` is also set automatically if the solver calculation routine is called without any output argument. Evaluate the following example for the visualization of the effect if this option is not set and no output argument is given

```
ode78 (@odepkg_equations_vanderpol, [0 20], [2 0]);
```

'`Refine`' The option '`Refine`' is used to set the interpolation factor that is used to increase the quality for the output values if an output function is also set with the option '`OutputFcn`'. It can only be a integer value $0 <= Refine <= 5$. Evaluate the following example for the visualization of the effect if this option is set

```
A = odeset ('Refine', 0, 'OutputFcn', @odeplot);
ode78 (@odepkg_equations_vanderpol, [0 20], [2 0], A);
B = odeset ('Refine', 3, 'OutputFcn', @odeplot);
ode78 (@odepkg_equations_vanderpol, [0 20], [2 0], B);
```

'`OutputSel`'
The option '`OutputSel`' is used to set the components for which output has to be performed if an output function is also set with the option '`OutputFcn`'. It can only be a vector of integer values. Evaluate the following example for the visualization of the effect if this option is set

```
A = odeset ('OutputSel', [1, 2], 'OutputFcn', @odeplot);
ode78 (@odepkg_equations_vanderpol, [0 20], [2 0], A);
B = odeset ('OutputSel', [2], 'OutputFcn', @odeplot);
ode78 (@odepkg_equations_vanderpol, [0 20], [2 0], B);
```

'`Stats`' The option '`Stats`' is used to print cost statistics about the solving process after solving has been finished. It can either be the string `'on'` or `'off'`. Evaluate the following example for the visualization of the effect if this option is set

```
A = odeset ('Stats', 'off');
[a, b] = ode78 (@odepkg_equations_vanderpol, [0 2], [2 0], A);
B = odeset ('Stats', 'on');
[c, d] = ode78 (@odepkg_equations_vanderpol, [0 2], [2 0], B);
```

The cost statistics can also be obtained if the solver calculation routine is called with one output argument. The cost statistics then are in the field '`stats`' of the output arguemnt structure. Evaluate the following example for the visualization of the effect if this option is set

```
A = odeset ('Stats', 'on');
B = ode78 (@odepkg_equations_vanderpol, [0 2], [2 0], A);
disp (B);
```

'`Jacobian`'
The option '`Jacobian`' can be used to set up an external Jacobian function or Jacobian matrix for DAE solvers to achieve faster and better results (ODE Runge–Kutta solvers do not need to handle a Jacobian function handle or Jacobian matrix). It must either be a function handle to a valid function or a full constant matrix of size squared the dimension of the set of differential equations. User defined Jacobian functions must have the form '`function [vjac] = fjac (vt, vy, varargin)`'. Evaluate the following example for the visualization of the effect if this option is set

```
function vdy = fpol (vt, vy, varargin)
  vdy = [vy(2); (1 - vy(1)^2) * vy(2) - vy(1)];
endfunction
```

```
function vr = fjac (vt, vy, varargin)
  vr = [0, 1; ...
        -1-2*vy(1)*vy(2), 1-vy(1)^2];
endfunction

A = odeset ('Stats', 'on');
B = odepkg_mexsolver_radau (@fpol, [0 20], [2 0], A);
C = odeset ('Jacobian', @fjac, 'Stats', 'on');
D = odepkg_mexsolver_radau (@fpol, [0 20], [2 0], C);
```

'JPattern'
: The option 'JPattern' is not handled by any of the solvers by now.

'Vectorized'
: The option 'Vectorized' is not handled by any of the solvers by now.

'Mass'
: The option 'Mass' can be used to set up an external Mass function or Mass matrix for solving DAE problems. It depends on the solver that is used if 'Mass' is supported or not. It must either be a function handle to a valid function or a full constant matrix of size squared the dimension of the set of differential equations. User defined Jacobian functions must have the form '`function vmas = fmas (vt, vy, varargin)`'. Evaluate the following example for the visualization of the effect if this option is set

```
function vdy = frob (t, y, varargin)
  vdy(1,1) = -0.04*y(1)+1e4*y(2)*y(3);
  vdy(2,1) =  0.04*y(1)-1e4*y(2)*y(3)-3e7*y(2)^2;
  vdy(3,1) =  y(1)+y(2)+y(3)-1;
endfunction

function vmas = fmas (vt, vy, varargin)
  vmas =  [1, 0, 0; 0, 1, 0; 0, 0, 0];
endfunction

A = odeset ('Mass', @fmas);
B = oderd (@frob, [0 1e8], [1 0 0], A);
```

'MStateDependence'
: The option 'MStateDependence' can be used to set up the type of the external Mass function for solving DAE problems if a Mass function handle is set with the option 'Mass'. It depends on the solver that is used if 'MStateDependence' is supported or not. It must be a string of the form 'none', 'weak' or 'strong'. Evaluate the following example for the visualization of the effect if this option is set

```
function vdy = frob (vt, vy, varargin)
  vdy(1,1) = -0.04*vy(1)+1e4*vy(2)*vy(3);
  vdy(2,1) =  0.04*vy(1)-1e4*vy(2)*vy(3)-3e7*vy(2)^2;
  vdy(3,1) =  vy(1)+vy(2)+vy(3)-1;
endfunction

function vmas = fmas (vt, varargin)
  vmas =  [1, 0, 0; 0, 1, 0; 0, 0, 0];
endfunction
```

```
A = odeset ('Mass', @fmas, 'MStateDependence', 'none');
B = oderd (@frob, [0 1e8], [1 0 0], A);
```

User defined Mass functions must have the form as described before (ie. '`function vmas = fmas (vt, varargin)`' if the option '`MStateDependence`' was set to `'none'`, otherwise the user defined Mass function must have the form '`function vmas = fmas (vt, vy, varargin)`' if the option '`MStateDependence`' was set to either `'weak'` or `'strong'`.

'`MvPattern`'
: The option '`MvPattern`' is not handled by any of the solvers by now.

'`MassSingular`'
: The option '`MassSingular`' is not handled by any of the solvers by now.

'`NonNegative`'
: The option '`NonNegative`' can be used to set solution variables to zero even if their real solution would be a negative value. It must be a vector describing the positions in the solution vector for which the option '`NonNegative`' should be used. Evaluate the following example for the visualization of the effect if this option is set

```
vfun = @(vt,vy) -abs(vy);
vopt = odeset ('NonNegative', [1]);

[vt1, vy1] = ode78 (vfun, [0 100], [1]);
[vt2, vy2] = ode78 (vfun, [0 100], [1], vopt);

subplot (2,1,1); plot (vt1, vy1);
subplot (2,1,2); plot (vt2, vy2);
```

'`Events`'
: The option '`Events`' can be used to set up an Event function, ie. the Event function can be used to find zero crossings in one of the results. It must either be a function handle to a valid function. Evaluate the following example for the visualization of the effect if this option is set

```
function vdy = fbal (vt, vy, varargin)
  vdy(1,1) =  vy(2)+3;
  vdy(2,1) = -9.81; %# m/s
endfunction

function [veve, vterm, vdir] = feve (vt, vy, varargin)
  veve  = vy(1); %# Which event component should be tread
  vterm =     1; %# Terminate if an event is found
  vdir  =    -1; %# In which direction, -1 for falling
endfunction

A = odeset ('Events', @feve);
B = ode78 (@fbal, [0 1.5], [1 3], A);
plot (B.x, B.y(:,1));
```

'`MaxOrder`'
: The option '`MaxOrder`' can be used to set the maximum order of the backward differentiation algorithm of the `odebdi` solver. It must be a scalar integer value

between 1 and 7. Evaluate the following example for the visualization of the effect if this option is set

```
function res = fwei (t, y, yp, varargin)
  res = t*y^2*yp^3 - y^3*yp^2 + t*yp*(t^2 + 1) - t^2*y;
endfunction

function [dy, dyp] = fjac (t, y, yp, varargin)
  dy  = 2*t*y*yp^3 - 3*y^2*yp^2 - t^2;
  dyp = 3*t*y^2*yp^2 - 2*y^3*yp + t*(t^2 + 1);
endfunction

A = odeset ('AbsTol', 1e-6, 'RelTol', 1e-6, 'Jacobian', @fjac, ...
      'Stats', 'on', 'MaxOrder', 1, 'BDF', 'on')
B = odeset (A, 'MaxOrder', 5)
C = odebdi (@fwei, [1 10], 1.2257, 0.8165, A);
D = odebdi (@fwei, [1 10], 1.2257, 0.8165, B);
plot (C.x, C.y, 'bo-', D.x, D.y, 'rx:');
```

'`BDF`' The option '`BDF`' is only supported by the `odebdX` solvers. Using these solvers the option '`BDF`' will automatically be set `'on'` (even if it was set `'off'` before) because the `odebdX` solvers all use the backward differentiation algorithm to solve the different kind of problems.

2.4 M–File Function Reference

`[] = ode23 (@`*fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]`)` [Function File]
`[`*sol*`] = ode23 (@`*fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]`)` [Command]
`[`*t*, *y*, `[`*xe*, *ye*, *ie*`]] = ode23 (@`*fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]`)` [Command]

This function file can be used to solve a set of non–stiff ordinary differential equations (non–stiff ODEs) or non–stiff differential algebraic equations (non–stiff DAEs) with the well known explicit Runge–Kutta method of order (2,3).

If this function is called with no return argument then plot the solution over time in a figure window while solving the set of ODEs that are defined in a function and specified by the function handle `@`*fun*. The second input argument *slot* is a double vector that defines the time slot, *init* is a double vector that defines the initial values of the states, *opt* can optionally be a structure array that keeps the options created with the command `odeset` and *par1*, *par2*, ... can optionally be other input arguments of any type that have to be passed to the function defined by `@`*fun*.

If this function is called with one return argument then return the solution *sol* of type structure array after solving the set of ODEs. The solution *sol* has the fields *x* of type double column vector for the steps chosen by the solver, *y* of type double column vector for the solutions at each time step of *x*, *solver* of type string for the solver name and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector that keep the informations of the event function if an event function handle is set in the option argument *opt*.

If this function is called with more than one return argument then return the time stamps *t*, the solution values *y* and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector.

Run examples with the command

```
demo ode23
```

`[] = ode2r (@fun, slot, init, [opt], [par1, par2, ...])` [Function File]
`[sol] = ode2r (@fun, slot, init, [opt], [par1, par2, ...])` [Command]
`[t, y, [xe, ye, ie]] = ode2r (@fun, slot, init, [opt], [par1, par2, ...])` [Command]

This function file can be used to solve a set of non–stiff ordinary differential equations (non–stiff ODEs) or non–stiff differential algebraic equations (non–stiff DAEs). This function file is a wrapper to '`odepkg_mexsolver_radau.c`' that uses Hairer's and Wanner's Fortran solver '`radau.f`'.

If this function is called with no return argument then plot the solution over time in a figure window while solving the set of ODEs that are defined in a function and specified by the function handle `@`*fun*. The second input argument *slot* is a double vector that defines the time slot, *init* is a double vector that defines the initial values of the states, *opt* can optionally be a structure array that keeps the options created with the command `odeset` and *par1*, *par2*, ... can optionally be other input arguments of any type that have to be passed to the function defined by `@`*fun*.

If this function is called with one return argument then return the solution *sol* of type structure array after solving the set of ODEs. The solution *sol* has the fields *x* of type double column vector for the steps chosen by the solver, *y* of type double column vector for the solutions at each time step of *x*, *solver* of type string for the solver name and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector that keep the informations of the event function if an event function handle is set in the option argument *opt*.

If this function is called with more than one return argument then return the time stamps *t*, the solution values *y* and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector.

Run examples with the command

```
demo ode2r
```

`[] = ode45 (@fun, slot, init, [opt], [par1, par2, ...])` [Function File]
`[sol] = ode45 (@fun, slot, init, [opt], [par1, par2, ...])` [Command]
`[t, y, [xe, ye, ie]] = ode45 (@fun, slot, init, [opt], [par1, par2, ...])` [Command]

This function file can be used to solve a set of non–stiff ordinary differential equations (non–stiff ODEs) or non–stiff differential algebraic equations (non–stiff DAEs) with the well known explicit Runge–Kutta method of order (4,5).

If this function is called with no return argument then plot the solution over time in a figure window while solving the set of ODEs that are defined in a function and specified by the function handle `@`*fun*. The second input argument *slot* is a double vector that defines the time slot, *init* is a double vector that defines the initial values of the states, *opt* can optionally be a structure array that keeps the options created with the command `odeset` and *par1*, *par2*, ... can optionally be other input arguments of any type that have to be passed to the function defined by `@`*fun*.

If this function is called with one return argument then return the solution *sol* of type structure array after solving the set of ODEs. The solution *sol* has the fields *x* of type double column vector for the steps chosen by the solver, *y* of type double column vector for the solutions at each time step of *x*, *solver* of type string for the solver name and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector that keep the informations of the event function if an event function handle is set in the option argument *opt*.

If this function is called with more than one return argument then return the time stamps *t*, the solution values *y* and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector.

Run examples with the command

```
demo ode45
```

`[] = ode54 (@fun, slot, init, [opt], [par1, par2, ...])` [Function File]
`[sol] = ode54 (@fun, slot, init, [opt], [par1, par2, ...])` [Command]
`[t, y, [xe, ye, ie]] = ode54 (@fun, slot, init, [opt], [par1, par2, ...])` [Command]

This function file can be used to solve a set of non–stiff ordinary differential equations (non–stiff ODEs) or non–stiff differential algebraic equations (non–stiff DAEs) with the well known explicit Runge–Kutta method of order (5,4).

If this function is called with no return argument then plot the solution over time in a figure window while solving the set of ODEs that are defined in a function and specified by the function handle `@`*fun*. The second input argument *slot* is a double vector that defines the time slot, *init* is a double vector that defines the initial values of the states, *opt* can optionally be a structure array that keeps the options created with the command `odeset` and *par1*, *par2*, ... can optionally be other input arguments of any type that have to be passed to the function defined by `@`*fun*.

If this function is called with one return argument then return the solution *sol* of type structure array after solving the set of ODEs. The solution *sol* has the fields *x* of type double column vector for the steps chosen by the solver, *y* of type double column vector for the solutions at each time step of *x*, *solver* of type string for the solver name and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector that keep the informations of the event function if an event function handle is set in the option argument *opt*.

If this function is called with more than one return argument then return the time stamps *t*, the solution values *y* and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector.

Run examples with the command

```
demo ode54
```

`[] = ode5d (@fun, slot, init, [opt], [par1, par2, ...])` [Function File]
`[sol] = ode5d (@fun, slot, init, [opt], [par1, par2, ...])` [Command]
`[t, y, [xe, ye, ie]] = ode5d (@fun, slot, init, [opt], [par1, par2, ...])` [Command]

This function file can be used to solve a set of non–stiff ordinary differential equations (non–stiff ODEs) or non–stiff differential algebraic equations (non–stiff DAEs) with the well known explicit Runge–Kutta method of order (5,4).

Note: The function files '`odepkg_mexsolver_dopri5`' and '`ode5d`' will be removed when version 0.4.0 of OdePkg will be released. A similiar solver method is '`ode54`', please use the '`ode54`' solver instead.

`[] = ode5r (@fun, slot, init, [opt], [par1, par2, ...])` [Function File]
`[sol] = ode5r (@fun, slot, init, [opt], [par1, par2, ...])` [Command]

`[t, y, [xe, ye, ie]] = ode5r (@fun, slot, init, [opt], [par1, par2, ...])` [Command]

This function file can be used to solve a set of non–stiff ordinary differential equations (non–stiff ODEs) and non-stiff differential algebraic equations (non-stiff DAEs). This function file is a wrapper to '`odepkg_mexsolver_radau5.c`' that uses Hairer's and Wanner's Fortran solver '`radau5.f`'.

If this function is called with no return argument then plot the solution over time in a figure window while solving the set of ODEs that are defined in a function and specified by the function handle `@`*fun*. The second input argument *slot* is a double vector that defines the time slot, *init* is a double vector that defines the initial values of the states, *opt* can optionally be a structure array that keeps the options created with the command `odeset` and *par1*, *par2*, ... can optionally be other input arguments of any type that have to be passed to the function defined by `@`*fun*.

If this function is called with one return argument then return the solution *sol* of type structure array after solving the set of ODEs. The solution *sol* has the fields *x* of type double column vector for the steps chosen by the solver, *y* of type double column vector for the solutions at each time step of *x*, *solver* of type string for the solver name and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector that keep the informations of the event function if an event function handle is set in the option argument *opt*.

If this function is called with more than one return argument then return the time stamps *t*, the solution values *y* and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector.

Run examples with the command

```
demo ode5r
```

`[] = ode78 (@fun, slot, init, [opt], [par1, par2, ...])` [Function File]
`[sol] = ode78 (@fun, slot, init, [opt], [par1, par2, ...])` [Command]
`[t, y, [xe, ye, ie]] = ode78 (@fun, slot, init, [opt], [par1, par2, ...])` [Command]

This function file can be used to solve a set of non–stiff ordinary differential equations (non–stiff ODEs) or non–stiff differential algebraic equations (non–stiff DAEs) with the well known explicit Runge–Kutta method of order (7,8).

If this function is called with no return argument then plot the solution over time in a figure window while solving the set of ODEs that are defined in a function and specified by the function handle `@`*fun*. The second input argument *slot* is a double vector that defines the time slot, *init* is a double vector that defines the initial values of the states, *opt* can optionally be a structure array that keeps the options created with the command `odeset` and *par1*, *par2*, ... can optionally be other input arguments of any type that have to be passed to the function defined by `@`*fun*.

If this function is called with one return argument then return the solution *sol* of type structure array after solving the set of ODEs. The solution *sol* has the fields *x* of type double column vector for the steps chosen by the solver, *y* of type double column vector for the solutions at each time step of *x*, *solver* of type string for the solver name and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector that keep the informations of the event function if an event function handle is set in the option argument *opt*.

If this function is called with more than one return argument then return the time stamps *t*, the solution values *y* and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector.

Run examples with the command

```
demo ode78
```

[] = ode8d (@*fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]) [Function File]
[*sol*] = ode8d (@*fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]) [Command]
[*t*, *y*, [*xe*, *ye*, *ie*]] = ode8d (@*fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]) [Command]

This function file can be used to solve a set of non–stiff ordinary differential equations (non–stiff ODEs) or non–stiff differential algebraic equations (non–stiff DAEs) with the well known explicit Runge–Kutta method of order (8,5,3).

Note: The function files '`odepkg_mexsolver_dop853`' and '`ode8d`' will be removed when version 0.4.0 of OdePkg will be released. A similiar solver method is '`ode78`', please use the '`ode78`' solver instead.

[*value*] = odeget (*odestruct*, *option*, [*default*]) [Function File]
[*values*] = odeget (*odestruct*, {*opt1*, *opt2*, ...}, [{*def1*, *def2*, ...}]) [Command]

If this function is called with two input arguments and the first input argument *odestruct* is of type structure array and the second input argument *option* is of type string then return the option value *value* that is specified by the option name *option* in the OdePkg option structure *odestruct*. Optionally if this function is called with a third input argument then return the default value *default* if *option* is not set in the structure *odestruct*.

If this function is called with two input arguments and the first input argument *odestruct* is of type structure array and the second input argument *option* is of type cell array of strings then return the option values *values* that are specified by the option names *opt1*, *opt2*, ... in the OdePkg option structure *odestruct*. Optionally if this function is called with a third input argument of type cell array then return the default value *def1* if *opt1* is not set in the structure *odestruct*, *def2* if *opt2* is not set in the structure *odestruct*, ...

Run examples with the command

```
demo odeget
```

[] = odeox (@*fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]) [Function File]
[*sol*] = odeox (@*fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]) [Command]
[*t*, *y*, [*xe*, *ye*, *ie*]] = odeox (@*fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]) [Command]

This function file can be used to solve a set of non–stiff ordinary differential equations (non–stiff ODEs) and non–stiff differential algebraic equations (non–stiff DAEs).

Note: The function files '`odepkg_mexsolver_odex`' and '`odeox`' will be removed when version 0.4.0 of OdePkg will be released. A similiar solver method does not exist in OdePkg but you can use '`ode23, ode45, ode54`' or '`ode78`' instead.

[*ret*] = odephas2 (*t*, *y*, *flag*) [Function File]

Open a new figure window and plot the first result from the variable *y* that is of type double column vector over the second result from the variable *y* while solving. The types and the values of the input parameter *t* and the output parameter *ret* depend on the input value *flag* that is of type string. If *flag* is

'`"init"`' then *t* must be a double column vector of length 2 with the first and the last time step and nothing is returned from this function,

'""' then *t* must be a double scalar specifying the actual time step and the return value is true (resp. value 1),

'"done"' then *t* must be a double scalar specifying the last time step and nothing is returned from this function.

This function is called by a OdePkg solver function if it was specified in an OdePkg options structure with the `odeset`. This function is an OdePkg internal helper function therefore it should never be necessary that this function is called directly by a user. There is only little error detection implemented in this function file to achieve the highest performance.

Run examples with the command

```
demo odephas2
```

[*ret*] = **odephas3** (*t*, *y*, *flag*) [Function File]

Open a new figure window and plot the first result from the variable *y* that is of type double column vector over the second and the third result from the variable *y* while solving. The types and the values of the input parameter *t* and the output parameter *ret* depend on the input value *flag* that is of type string. If *flag* is

'"init"' then *t* must be a double column vector of length 2 with the first and the last time step and nothing is returned from this function,

'""' then *t* must be a double scalar specifying the actual time step and the return value is true (resp. value 1),

'"done"' then *t* must be a double scalar specifying the last time step and nothing is returned from this function.

This function is called by a OdePkg solver function if it was specified in an OdePkg options structure with the `odeset`. This function is an OdePkg internal helper function therefore it should never be necessary that this function is called directly by a user. There is only little error detection implemented in this function file to achieve the highest performance.

Run examples with the command

```
demo odephas3
```

[] = **odepkg** () [Function File]

OdePkg is part of the GNU Octave Repository (resp. the Octave–Forge project). The package includes commands for setting up various options, output functions etc. before solving a set of differential equations with the solver functions that are also included. At this time OdePkg is under development with the main target to make a package that is mostly compatible to proprietary solver products.

If this function is called without any input argument then open the OdePkg tutorial in the Octave window. The tutorial can also be opened with the following command

```
doc odepkg
```

[*res*] = **odepkg_equations_ilorenz** (*t*, *y*, *varyd*) [Function File]

Return three residuals of the implicit ordinary differential equations (IDEs) from the "Lorenz attractor" implementation, cf. `http://en.wikipedia.org/wiki/Lorenz_equation` for further details. The output argument *res* is a column vector and contains the residuals, *y* is a column vector that contains the integration results from the previous integration step, *yd* is a column vector that contains the derivatives of the last integration step and *t* is a scalar value with the actual time stamp. There is no error handling implemented in this function to achieve the highest performance available.

Run examples with the command

```
demo odepkg_equations_ilorenz
```

`[ydot] = odepkg_equations_lorenz (t, y)` [Function File]

Return three derivatives of the non–stiff ordinary differential equations (non–stiff ODEs) from the "Lorenz attractor" implementation, cf. `http://en.wikipedia.org/wiki/Lorenz_equation` for further details. The output argument *ydot* is a column vector and contains the derivatives, the input argument *y* also is a column vector that contains the integration results from the previous integration step and *t* is a double scalar that keeps the actual time stamp. There is no error handling implemented in this function to achieve the highest performance available.

Run examples with the command

```
demo odepkg_equations_lorenz
```

`[ydot] = odepkg_equations_pendulous (t, y)` [Function File]

Return two derivatives of the non–stiff ordinary differential equations (non–stiff ODEs) from a pendulum implementation, ie. the motion of a simple pendulum with damping, cf. `http://en.wikipedia.org/wiki/Pendulum` for further details. The output argument *ydot* is a column vector and contains the derivatives, the input argument *y* also is a column vector that contains the integration results from the previous integration step and *t* is a double scalar that keeps the actual time stamp. There is no error handling implemented in this function to achieve the highest performance available.

Run examples with the command

```
demo odepkg_equations_pendulous
```

`[ydot] = odepkg_equations_roessler (t, y)` [Function File]

Return the three derivatives of the non–stiff ordinary differential equations (non–stiff ODEs) from the Roessler attractor implementation, cf. `http://en.wikipedia.org/wiki/R%C3%B6ssler_attractor` for further details. The output argument *ydot* is a column vector and contains the derivatives, the input argument *y* also is a column vector that contains the integration results from the previous integration step and *t* is a double scalar that keeps the actual time stamp. There is no error handling implemented in this function to achieve the highest performance available.

Run examples with the command

```
demo odepkg_equations_roessler
```

`[ydot] = odepkg_equations_secondorderlag (t, y, [u, K, T1, T2])` [Function File]

Return two derivatives of the non-stiff ordinary differential equations (non-stiff ODEs) from the second order lag implementation, cf. `http://en.wikipedia.org/wiki/Category:Control_theory` for further details. The output argument *ydot* is a column vector and contains the derivatives, the input argument *y* also is a column vector that contains the integration results from the previous integration step and *t* is a double scalar that keeps the actual time stamp. There is no error handling implemented in this function to achieve the highest performance available.

Run examples with the command

```
demo odepkg_equations_secondorderlag
```

`[ydot] = odepkg_equations_vanderpol (t, y, [mu])` [Function File]

Return the two derivatives of the non-stiff ordinary differential equations (non-stiff ODEs) from the "Van der Pol" implementation, cf. `http://en.wikipedia.org/wiki/Van_der_Pol_oscillator` for further details. The output argument *ydot* is a column vector and contains the derivatives, the input argument *y* also is a column vector that contains the integration results from the previous integration step and *t* is a double scalar that keeps the actual time stamp. There is no error handling implemented in this function to achieve the highest performance available.

Run examples with the command

```
demo odepkg_equations_vanderpol
```

[*sol*] = odepkg_event_handle (*@fun*, *time*, *y*, *flag*, [*par1*, *par2*, ...]) [Function File]

Return the solution of the event function that is specified as the first input argument *@fun* in form of a function handle. The second input argument *time* is of type double scalar and specifies the time of the event evaluation, the third input argument *y* is of type double column vector and specifies the solutions, the third input argument *flag* is of type string and can be of the form

'"init"' then initialize internal persistent variables of the function `odepkg_event_handle` and return an empty cell array of size 4,

'"calc"' then do the evaluation of the event function and return the solution *sol* as type cell array of size 4,

'"done"' then cleanup internal variables of the function `odepkg_event_handle` and return an empty cell array of size 4.

Optionally if further input arguments *par1*, *par2*, ... of any type are given then pass these parameters through `odepkg_event_handle` to the event function.

This function is an OdePkg internal helper function therefore it should never be necessary that this function is called directly by a user. There is only little error detection implemented in this function file to achieve the highest performance.

[*newstruct*] = odepkg_structure_check (*oldstruct*, ["*solver*"]) [Function File]

If this function is called with one input argument of type structure array then check the field names and the field values of the OdePkg structure *oldstruct* and return the structure as *newstruct* if no error is found. Optionally if this function is called with a second input argument "*solver*" of type string taht specifies the name of a valid OdePkg solver then a higher level error detection is performed. The function does not modify any of the field names or field values but terminates with an error if an invalid option or value is found.

This function is an OdePkg internal helper function therefore it should never be necessary that this function is called directly by a user. There is only little error detection implemented in this function file to achieve the highest performance.

Run examples with the command

```
demo odepkg_structure_check
```

[*mescd*] = odepkg_testsuite_calcmescd (*solution*, *reference*, *abstol*, *reltol*) [Function File]

If this function is called with four input arguments of type double scalar or column vector then return a normalized value for the minimum number of correct digits *mescd* that is calculated from the solution at the end of an integration interval *solution* and a set of reference values *reference*. The input arguments *abstol* and *reltol* are used to calculate a reference solution that depends on the relative and absolute error tolerances.

Run examples with the command

```
demo odepkg_testsuite_calcmescd
```

[*scd*] = odepkg_testsuite_calcscd (*solution*, *reference*, *abstol*, *reltol*) [Function File]

If this function is called with four input arguments of type double scalar or column vector then return a normalized value for the minimum number of correct digits *scd* that is calculated from the solution at the end of an integration interval *solution* and a set of reference values *reference*. The input arguments *abstol* and *reltol* are unused but present because of compatibility to the function `odepkg_testsuite_calcmescd`.

Run examples with the command

```
demo odepkg_testsuite_calcscd
```

[*solution*] = **odepkg_testsuite_chemakzo** (@*solver*, *reltol*) [Function File]

If this function is called with two input arguments and the first input argument `@`*solver* is a function handle describing an OdePkg solver and the second input argument *reltol* is a double scalar describing the relative error tolerance then return a cell array *solution* with performance informations about the chemical AKZO Nobel testsuite of differential algebraic equations after solving (DAE–test).

Run examples with the command

```
demo odepkg_testsuite_chemakzo
```

[*solution*] = **odepkg_testsuite_hires** (@*solver*, *reltol*) [Function File]

If this function is called with two input arguments and the first input argument `@`*solver* is a function handle describing an OdePkg solver and the second input argument *reltol* is a double scalar describing the relative error tolerance then return a cell array *solution* with performance informations about the HIRES testsuite of ordinary differential equations after solving (ODE–test).

Run examples with the command

```
demo odepkg_testsuite_hires
```

[*solution*] = **odepkg_testsuite_implakzo** (@*solver*, *reltol*) [Function File]

If this function is called with two input arguments and the first input argument `@`*solver* is a function handle describing an OdePkg solver and the second input argument *reltol* is a double scalar describing the relative error tolerance then return a cell array *solution* with performance informations about the chemical AKZO Nobel testsuite of implicit differential algebraic equations after solving (IDE–test).

Run examples with the command

```
demo odepkg_testsuite_implakzo
```

[*solution*] = **odepkg_testsuite_implrober** (@*solver*, *reltol*) [Function File]

If this function is called with two input arguments and the first input argument `@`*solver* is a function handle describing an OdePkg solver and the second input argument *reltol* is a double scalar describing the relative error tolerance then return a cell array *solution* with performance informations about the implicit form of the modified ROBERTSON testsuite of implicit differential algebraic equations after solving (IDE–test).

Run examples with the command

```
demo odepkg_testsuite_implrober
```

[*solution*] = **odepkg_testsuite_oregonator** (@*solver*, *reltol*) [Function File]

If this function is called with two input arguments and the first input argument `@`*solver* is a function handle describing an OdePkg solver and the second input argument *reltol* is a double scalar describing the relative error tolerance then return a cell array *solution* with performance informations about the OREGONATOR testsuite of ordinary differential equations after solving (ODE–test).

Run examples with the command

```
demo odepkg_testsuite_oregonator
```

`[solution] = odepkg_testsuite_pollution (@solver, reltol)` [Function File]

If this function is called with two input arguments and the first input argument `@solver` is a function handle describing an OdePkg solver and the second input argument *reltol* is a double scalar describing the relative error tolerance then return the cell array *solution* with performance informations about the POLLUTION testsuite of ordinary differential equations after solving (ODE–test).

Run examples with the command

```
demo odepkg_testsuite_pollution
```

`[solution] = odepkg_testsuite_robertson (@solver, reltol)` [Function File]

If this function is called with two input arguments and the first input argument `@solver` is a function handle describing an OdePkg solver and the second input argument *reltol* is a double scalar describing the relative error tolerance then return a cell array *solution* with performance informations about the modified ROBERTSON testsuite of differential algebraic equations after solving (DAE–test).

Run examples with the command

```
demo odepkg_testsuite_robertson
```

`[solution] = odepkg_testsuite_transistor (@solver, reltol)` [Function File]

If this function is called with two input arguments and the first input argument `@solver` is a function handle describing an OdePkg solver and the second input argument *reltol* is a double scalar describing the relative error tolerance then return the cell array *solution* with performance informations about the TRANSISTOR testsuite of differential algebraic equations after solving (DAE–test).

Run examples with the command

```
demo odepkg_testsuite_transistor
```

`[ret] = odeplot (t, y, flag)` [Function File]

Open a new figure window and plot the results from the variable *y* of type column vector over time while solving. The types and the values of the input parameter *t* and the output parameter *ret* depend on the input value *flag* that is of type string. If *flag* is

'`"init"`' then *t* must be a double column vector of length 2 with the first and the last time step and nothing is returned from this function,

'`""`' then *t* must be a double scalar specifying the actual time step and the return value is true (resp. value 1),

'`"done"`' then *t* must be a double scalar specifying the last time step and nothing is returned from this function.

This function is called by a OdePkg solver function if it was specified in an OdePkg options structure with the `odeset`. This function is an OdePkg internal helper function therefore it should never be necessary that this function is called directly by a user. There is only little error detection implemented in this function file to achieve the highest performance.

Run examples with the command

```
demo odeplot
```

`[ret] = odeprint (t, y, flag)` [Function File]

Display the results of the set of differential equations in the Octave window while solving. The first column of the screen output shows the actual time stamp that is given with the input arguemtn *t*, the following columns show the results from the function evaluation that are given by the column vector *y*. The types and the values of the input parameter *t* and the output parameter *ret* depend on the input value *flag* that is of type string. If *flag* is '"init"' then t must be a double column vector of length 2 with the first and the last time step and nothing is returned from this function,

'""' then t must be a double scalar specifying the actual time step and the return value is true (resp. value 1),

'"done"' then t must be a double scalar specifying the last time step and nothing is returned from this function.

This function is called by a OdePkg solver function if it was specified in an OdePkg options structure with the `odeset`. This function is an OdePkg internal helper function therefore it should never be necessary that this function is called directly by a user. There is only little error detection implemented in this function file to achieve the highest performance.

Run examples with the command

```
demo odeprint
```

[] = oders (*@fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]) [Function File]
[*sol*] = oders (*@fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]) [Command]
[*t*, *y*, [*xe*, *ye*, *ie*]] = oders (*@fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]) [Command]

This function file can be used to solve a set of non–stiff ordinary differential equations (non–stiff ODEs) and non-stiff differential algebraic equations (non-stiff DAEs). This function file is a wrapper to '`odepkg_mexsolver_rodas.c`' that uses Hairer's and Wanner's Fortran solver '`rodas.f`'.

If this function is called with no return argument then plot the solution over time in a figure window while solving the set of ODEs that are defined in a function and specified by the function handle `@`*fun*. The second input argument *slot* is a double vector that defines the time slot, *init* is a double vector that defines the initial values of the states, *opt* can optionally be a structure array that keeps the options created with the command `odeset` and *par1*, *par2*, ... can optionally be other input arguments of any type that have to be passed to the function defined by `@`*fun*.

If this function is called with one return argument then return the solution *sol* of type structure array after solving the set of ODEs. The solution *sol* has the fields *x* of type double column vector for the steps chosen by the solver, *y* of type double column vector for the solutions at each time step of *x*, *solver* of type string for the solver name and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector that keep the informations of the event function if an event function handle is set in the option argument *opt*.

If this function is called with more than one return argument then return the time stamps *t*, the solution values *y* and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector.

Run examples with the command

```
demo oders
```

[*odestruct*] = odeset () [Function File]
[*odestruct*] = odeset ("*field1*", *value1*, "*field2*", *value2*, ...) [Command]
[*odestruct*] = odeset (*oldstruct*, "*field1*", *value1*, "*field2*", *value2*, ...) [Command]
[*odestruct*] = odeset (*oldstruct*, *newstruct*) [Command]

If this function is called without an input argument then return a new OdePkg options structure array that contains all the necessary fields and sets the values of all fields to default values.

If this function is called with string input arguments "*field1*", "*field2*", ... identifying valid OdePkg options then return a new OdePkg options structure with all necessary fields and set the values of the fields "*field1*", "*field2*", ... to the values *value1*, *value2*, ...

If this function is called with a first input argument *oldstruct* of type structure array then overwrite all values of the options "*field1*", "*field2*", ... of the structure *oldstruct* with new values *value1*, *value2*, ... and return the modified structure array.

If this function is called with two input argumnets *oldstruct* and *newstruct* of type structure array then overwrite all values in the fields from the structure *oldstruct* with new values of the fields from the structure *newstruct*. Empty values of *newstruct* will not overwrite values in *oldstruct*.

For a detailed explanation about valid fields and field values in an OdePkg structure aaray have a look at the '`odepkg.pdf`', Section 'ODE/DAE/IDE options' or run the command `doc odepkg` to open the tutorial.

Run examples with the command

```
demo odeset
```

`[] = odesx (@`*fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]) [Function File]
`[`*sol*`] = odesx (@`*fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]) [Command]
`[`*t*, *y*, [*xe*, *ye*, *ie*]`] = odesx (@`*fun*, *slot*, *init*, [*opt*], [*par1*, *par2*, ...]) [Command]

This function file can be used to solve a set of non–stiff ordinary differential equations (non–stiff ODEs) and non-stiff differential algebraic equations (non-stiff DAEs). This function file is a wrapper to '`odepkg_mexsolver_seulex.c`' that uses Hairer's and Wanner's Fortran solver '`seulex.f`'.

If this function is called with no return argument then plot the solution over time in a figure window while solving the set of ODEs that are defined in a function and specified by the function handle `@`*fun*. The second input argument *slot* is a double vector that defines the time slot, *init* is a double vector that defines the initial values of the states, *opt* can optionally be a structure array that keeps the options created with the command `odeset` and *par1*, *par2*, ... can optionally be other input arguments of any type that have to be passed to the function defined by `@`*fun*.

If this function is called with one return argument then return the solution *sol* of type structure array after solving the set of ODEs. The solution *sol* has the fields *x* of type double column vector for the steps chosen by the solver, *y* of type double column vector for the solutions at each time step of *x*, *solver* of type string for the solver name and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector that keep the informations of the event function if an event function handle is set in the option argument *opt*.

If this function is called with more than one return argument then return the time stamps *t*, the solution values *y* and optionally the extended time stamp information *xe*, the extended solution information *ye* and the extended index information *ie* all of type double column vector.

Run examples with the command

```
demo odesx
```

2.5 Oct–File Function Reference

TODO

3 Programmer's Guide

3.1 General description

TODO

3.2 C++ Function Reference

3.2.1 Source file 'odepkg_auxiliary_functions.cc'

`octave_value` **`odepkg_auxiliary_getmapvalue`** (*std::string vnam, Octave_map vmap*) [Function]

Return the `octave_value` from the field that is identified by the string *vnam* of the `Octave_map` that is given by *vmap*. The input arguments of this function are

- *vnam*: The name of the field whose value is returned
- *vmap*: The map that is checked for the presence of the field

`octave_idx_type` **`odepkg_auxiliary_isvector`** (*octave_value vval*) [Function]

Return the constant `true` if the value of the input argument *vval* is a valid numerical vector of `length > 1` or return the constant `false` otherwise. The input argument of this function is

- *vval*: The `octave_value` that is checked for being a valid numerical vector

`octave_value_list` **`odepkg_auxiliary_evaleventfun`** (*octave_value veve, octave_value vt, octave_value vy, octave_value_list vextarg, octave_idx_type vdeci*) [Function]

Return the values that come from the evaluation of the `Events` user function. The return arguments depend on the call to this function, ie. if *vdeci* is `0` then initilaization of the `Events` function is performed. If *vdeci* is `1` then a normal evaluation of the `Events` function is performed and the information from the `Events` evaluation is returned (cf. '`odepkg_event_handle.m`' for further details). If *vdeci* is `2` then cleanup of the `Events` function is performed and nothing is returned. The input arguments of this function are

- *veve*: The `Events` function that is evaluated
- *vt*: The time stamp at which the events function is called
- *vy*: The solutions of the set of ODEs at time *vt*
- *vextarg*: Extra arguments that are feed through to the `Events` function
- *vdeci*: A decision flag that describes what evaluation should be done

`octave_idx_type` **`odepkg_auxiliary_evalplotfun`** (*octave_value vplt, octave_value vsel, octave_value vt, octave_value vy, octave_value_list vextarg, octave_idx_type vdeci*) [Function]

Return a constant that comes from the evaluation of the `OutputFcn` function. The return argument depends on the call to this function, ie. if *vdeci* is `0` then initilaization of the `OutputFcn` function is performed and nothing is returned. If *vdeci* is `1` then a normal evaluation of the `OutputFcn` function is performed and either the constant `true` is returned if solving should be stopped or `false` is returned if solving should be continued (cf. '`odeplot.m`' for further details). If *vdeci* is `2` then cleanup of the `OutputFcn` function is performed and nothing is returned. The input arguments of this function are

- *vplt*: The `OutputFcn` function that is evaluated
- *vsel*: The output selection vector for which values should be treated

- *vt*: The time stamp at which the events function is called
- *vy*: The solutions of the set of ODEs at time *vt*
- *vextarg*: Extra arguments that are feed through to the `OutputFcn` function
- *vdeci*: A decision flag that describes what evaluation should be done

`octave_value_list odepkg_auxiliary_evaljacide` (*octave_value vjac, octave_value vt, octave_value vy, octave_value vdy, octave_value_list vextarg*) [Function]

Return two matrices that come from the evaluation of the `Jacobian` function. The input arguments of this function are

- *vjac*: The `Jacobian` function that is evaluated
- *vt*: The time stamp at which the events function is called
- *vy*: The solutions of the set of IDEs at time *vt*
- *vdy*: The derivatives of the set of IDEs at time *vt*
- *vextarg*: Extra arguments that are feed through to the `Jacobian` function

Note: This function can only be used for IDE problem solvers.

`octave_value odepkg_auxiliary_evaljacode` (*octave_value vjac, octave_value vt, octave_value vy, octave_value_list vextarg*) [Function]

Return a matrix that comes from the evaluation of the `Jacobian` function. The input arguments of this function are

- *vjac*: The `Jacobian` function that is evaluated
- *vt*: The time stamp at which the events function is called
- *vy*: The solutions of the set of ODEs at time *vt*
- *vextarg*: Extra arguments that are feed through to the `Jacobian` function

Note: This function can only be used for ODE and DAE problem solvers.

`octave_value odepkg_auxiliary_evalmassode` (*octave_value vmass, octave_value vstate, octave_value vt, octave_value vy, octave_value_list vextarg*) [Function]

Return a matrix that comes from the evaluation of the `Mass` function. The input arguments of this function are

- *vmass*: The `Mass` function that is evaluated
- *vstate*: The state variable that either is the string `'none'`, `'weak'` or `'strong'`
- *vt*: The time stamp at which the events function is called
- *vy*: The solutions of the set of ODEs at time *vt*
- *vextarg*: Extra arguments that are feed through to the `Mass` function

Note: This function can only be used for ODE and DAE problem solvers.

`octave_value odepkg_auxiliary_makestats` (*octave_value_list vstats, octave_idx_type vprnt*) [Function]

Return an *octave_value* that contains fields about performance informations of a finished solving process. The input arguments of this function are

- *vstats*: The statistics informations that need to be handled
  1. hello
- *vprnt*: If `true` then the statistics information also is displayed on screen

`octave_idx_type odepkg_auxiliary_mebdfanalysis` (*octave_idx_type verr*) [Function]

TODO

`octave_idx_type odepkg_auxiliary_solstore` (*octave_value &vt, octave_value &vy, octave_value vsel, octave_idx_type vdeci*) [Function]

If *vdeci* is `0` (*vt* is a pointer to the initial time step and *vy* is a pointer to the initial values vector) then this function is initialized. Otherwise if *vdeci* is `1` (*vt* is a pointer to another time step and *vy* is a pointer to the solution vector) the values of *vt* and *vy* are added to the internal variable, if *vdeci* is `2` then the internal vectors are returned. The input arguments of this function are

- *vt*: The time stamp at which the events function is called
- *vy*: The solutions of the set of ODEs at time *vt*
- *vsel*: The selection vector for which values should be treated
- *vdeci*: A decision flag that describes what evaluation should be done

3.2.2 Source File '`odepkg_octsolver_mebdfi.cc`'

`octave_idx_type (*odepkg_mebdfi_usrtype)` [Typedef]

This `typedef` is used to define the input and output arguments of the user function for the IDE problem that is further needed by the Fortran core solver `mebdfi`. The implementation of this `typedef` is

```
typedef octave_idx_type (*odepkg_mebdfi_usrtype)
  (const octave_idx_type& N, const double& T, const double* Y,
   double* DELTA, const double* YPRIME, const octave_idx_type* IPAR,
   const double* RPAR, const octave_idx_type& IERR);
```

`octave_idx_type (*odepkg_mebdfi_jactype)` [Typedef]

This `typedef` is used to define the input and output arguments of the `Jacobian` function for the IDE problem that is further needed by the Fortran core solver `mebdfi`. The implementation of this `typedef` is

```
typedef octave_idx_type (*odepkg_mebdfi_jactype)
  (const double& T, const double* Y, double* PD, const octave_idx_type& N,
   const double* YPRIME, const octave_idx_type* MBND, const double& CON,
   const octave_idx_type* IPAR, const double* RPAR, const octave_idx_type& IERR);
```

`F77_RET_T F77_FUNC (mebdfi, MEBDFI)` (*const octave_idx_type& N, const double& T0, const double& HO, const double* Y0, const double* YPRIME, const double& TOUT, const double& TEND, const octave_idx_type& MF, octave_idx_type& IDID, const octave_idx_type& LOUT, const octave_idx_type& LWORK, const double* WORK, const octave_idx_type& LIWORK, const octave_idx_type* IWORK, const octave_idx_type* MBND, const octave_idx_type& MAXDER, const octave_idx_type& ITOL, const double* RTOL, const double* ATOL, const double* RPAR, const octave_idx_type* IPAR, odepkg_mebdfi_jactype, odepkg_mebdfi_usrtype, octave_idx_type& IERR*); [Prototype]

The prototype `F77_FUNC (mebdfi, MEBDFI)` is used to represent the information about the Fortran core solver `mebdfi` that is defined in the Fortran source file '`mebdfi.f`' (cf. the Fortran source file '`mebdfi.f`' for further details).

`static octave_value_list vmebdfiextarg` [Variable]

This static variable is used to store the extra arguments that are needed by some or by all of the `OutputFcn`, the `Jacobian` function and the `Events` function while solving the IDE problem.

`static octave_value *vmebdfiodefun` [Variable]

This static variable is used to store the value for the user function that defines the set of IDEs.

`static octave_value vmebdfijacfun` [Variable]

This static variable is used to store the value for the `Jacobian` function or the `Jacobian` matrix that is needed if Jacobian evaluation should be performed.

`octave_idx_type odepkg_mebdfi_usrfcn` (*const octave_idx_type& N, const double& T, const double* Y, double* DELTA, const double* YPRIME, GCC_ATTR_UNUSED const octave_idx_type* IPAR, GCC_ATTR_UNUSED const double* RPAR, GCC_ATTR_UNUSED const octave_idx_type& IERR*) [Function]

Return `true` if the evaluation of the user function was successful, return `false` otherwise. This function is directly called from the Fortran core solver `mebdfi`. The input arguments of this function are

- *N*: The number of equations that are defined for the IDE–problem
- *T*: The actual time stamp for the current function evaluation
- *Y*: The function values from the last successful integration step of length *N*
- *DELTA*: The residual vector that needs to be calculated of length *N*
- *YPRIME*: The derivative values from the last successful integration step of length *N*
- *IPAR*: The integer parameters that are passed to the user function (unused)
- *RPAR*: The real parameters that are passed to the user function (unused)
- *IERR*: The error flag that can be set on each evaluation (unused)

`octave_idx_type odepkg_mebdfi_jacfcn` (*const double& T, const double* Y, double* PD, const octave_idx_type& N, const double* YPRIME, GCC_ATTR_UNUSED const octave_idx_type* MBND, const double& CON, GCC_ATTR_UNUSED const octave_idx_type* IPAR, GCC_ATTR_UNUSED const double* RPAR, GCC_ATTR_UNUSED const octave_idx_type& IERR*) [Function]

Return `true` if the evaluation of the Jacobian function (that is defined for a special IDE problem in Octave) was successful, otherwise return `false`. This function is directly called from the Fortran core solver `mebdfi`. The input arguments of this function are

- *T*: The actual time stamp for the current function evaluation
- *Y*: The function values from the last successful integration step of length *N*
- *PD*: The values of partial derivatives of the Jacobian matrix of size *N*
- *N*: The number of equations that are defined for the IDE–problem
- *YPRIME*: The derivative values from the last successful integration step of length *N*
- *MBND*: A vector of size 4 describing the sizes of a banded Jacobian (unused)
- *CON*: A constant value that is set before the evaluation of the Jacobian function
- *IPAR*: The integer parameters that are passed to the user function (unused)
- *RPAR*: The real parameters that are passed to the user function (unused)
- *IERR*: The error flag that can be set on each evaluation (unused)

`DEFUN_DLD` (*odebdi, args, nargout, 'help string'*) [Function]

Return the results of the solving process of the IDE problem from the Fortran core solver `mebdfi` to the caller function (cf. `help odebdi` within Octave for further details about this function). the Argument *odebdi* is the name of the function that can be used in Octave and *'help string'* is the help text that is displayed if the command `help odebdi` is called from Octave. The input arguments of this function are

- *args*: The input arguments in form of an `octave_value_list`
- *nargout*: The number of output arguments that are required

Appendix A

A.1 GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

   The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

   This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

   We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

   This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

   A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

   A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

   The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

   The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its

Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Function Index

D

DEFUN_DLD 29

O

ode23 14
ode2r 15
ode45 15
ode54 16
ode5d 16
ode5r 16
ode78 17
ode8d 18
odeget 18
odeox 18
odephas2 18
odephas3 19
odepkg 19
odepkg_auxiliary_evaleventfun 26
odepkg_auxiliary_evaljacide 27
odepkg_auxiliary_evaljacode 27
odepkg_auxiliary_evalmassode 27
odepkg_auxiliary_evalplotfun 26
odepkg_auxiliary_getmapvalue 26
odepkg_auxiliary_isvector 26
odepkg_auxiliary_makestats 27
odepkg_auxiliary_mebdfanalysis 27
odepkg_auxiliary_solstore 28
odepkg_equations_ilorenz 19
odepkg_equations_lorenz 20
odepkg_equations_pendulous 20
odepkg_equations_roessler 20
odepkg_equations_secondorderlag 20
odepkg_equations_vanderpol 20
odepkg_event_handle 21
odepkg_mebdfi_jacfcn 29
odepkg_mebdfi_usrfcn 29
odepkg_structure_check 21
odepkg_testsuite_calcmescd 21
odepkg_testsuite_calcscd 21
odepkg_testsuite_chemakzo 22
odepkg_testsuite_hires 22
odepkg_testsuite_implakzo 22
odepkg_testsuite_implrober 22
odepkg_testsuite_oregonator 22
odepkg_testsuite_pollution 23
odepkg_testsuite_robertson 23
odepkg_testsuite_transistor 23
odeplot 23
odeprint 23
oders 24
odeset 24
odesx 25

Index

A
Appendix 30

B
bugs 2

C
C++ Function Reference 26

D
deinstallation 2

F
FDL, GNU Free Documentation License 30
First example and demos 2

G
General description 26

H
help 2
history 1

I
installation 2

M
M–File Function Reference 14

O
Oct–File Function Reference 25
ODE/DAE/IDE options 9

P
Problems 5
Programmer's Guide 26

R
roadmap 1

S
Solver 6

U
User's Guide 5