OdePkg

A package for solving differential equations with Octave

This document currently is under development

by Thomas Treichl

Table of Contents

1 Beginner's Guide

The "Beginner's Guide" is intended for new users who want to solve differential equations with the higher level language Octave 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 the section Section 1.3 [Installation and deinstallation], page 2 it is explained how OdePkg can be installed and in Section 1.5 [First tests and demos], page 2 the first examples are 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 also included. OdePkg formerly was initiated to solve explicitly formulated 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 main target, to make a package that is mostly compatible to commercial solver products.

1.2 OdePkg history and roadmap

OdePkg Version 0.0.1	The initial release was already a modification of the old "ode package" that was hosted at Octave–Forge and that was written by Marc Compere some when 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`, etc.). Since version 0.1.3 there is also 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 making 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 with help texts and tests (eg. `ode2r`, `ode5r`, `oders` etc.). Six testsuite functions have been added for performance tests of the different solvers (`odepkg_testsuite_chemakzo`, `odepkg_testsuite_oregonator`, `odepkg_testsuite_transistor`, etc.).

(current) Version 0.3.x	Ongoing work with this manual. Jeff Cash released his solvers under the GPL – first tests are done to include these solvers within OdePkg. With the beginning of version 0.3.2 new interface functions are created based on Octave's C++DLD interface to achieve a more higher performance. The first IDE–solver '`mebdfi.f`' appears and that is interfaced by '`odepkg_octsolver.mebdfi.cc`'.
(future) Version 0.4.x	Ongoing work with this manual. Fetching and adding the DASRT IDE–solver from Netlib. Porting the mex–file solvers to Octave's C++DLD interface.
(future) Version 0.5.x	(Maybe) A lot of compatibility tests.
(future) Version 0.6.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 of Octave. For this get into the directory where the current release of OdePkg can be found, 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-x.x.x.tar.gz
```

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 then please report this on the mailing–list of Octave–Forge using the email address `octave-dev@lists.sourceforge.net` .

1.4 Reporting Bugs

If you encounter problems while using OdePkg or if you find bugs in the source codes then please report 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` .

1.5 First tests 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 now be written as an Octave 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 is 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 followed by a integrated state 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 4). 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 the Octave interpreter window

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

A figure window opens and it can be seen how this ODEs are solved from $t_0 = 0.0$ to $t_e = 5.0$. If opening the figure window is unwanted then output arguments have to be used to catch the results of the solving process and to not pass the results to the window plotter, eg.

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

Results can also be obtained as 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` is 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 about the constant values anymore, 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);
```

The OdePkg can do much more while solving ODEs and DAEs, eg. setting up other output functions instead of the function `odeplot`. So 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);
```

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 (read the help files for the other solvers if there may be changes in the future). The options that can be set are described in Section 2.3 [ODE/DAE/IDE options], page 7.

Further examples have also been added to the OdePkg. These example files and functions are of the form `odepkg_equations_*`. Different testsuite examples have been added to OdePkg that are stored in files with filenames `odepkg_testsuite_*`. Before continuing reading the next chapter note that nearly every function that comes with OdePkg has its own help description and 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 then type

```
help fcnname
```

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

```
demo fcnname
```

in the Octave command window where `fcnname` is the name of the function of the demo to run. Last but not least write

```
doc odepkg
```

for opening this manual in the texinfo reader of the octave command window.

2 User's Guide

The "User's Guide" is intended for trained users who already do know in principal how to solve differential equations with the higher level language Octave and OdePkg. In this chapter it will be explained which solvers can be used for the different kind of problems in Section 2.2 [Solver families], page 4 and which options can be set for the optimization of the solving process in Section 2.3 [ODE/DAE/IDE options], page 7.

2.1 Differential Equation Problems

In this section the different kind of differential equation problems are explained that can be solved with OdePkg. The formulation of ordinary differential equations is described in section Section 2.1.1 [ODE problems], page 4 followed by the description of explicetly formulated differential algebraic equations in section Section 2.1.2 [DAE problems], page 4 and implicetely formulated differential algebraic equations Section 2.1.3 [IDE problems], page 4.

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 state variables. 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 ordinary differential equations.

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 state variables. 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 ordinary differential equations. The variable $M(t, y)$ is the squared mass matrix that may depend on y and t.

2.1.3 IDE problems

2.2 Solver families

In this section different kind of solvers are explained that have been implemented in OdePkg. This section starts with the standard M–file Runge–Kutta solvers in section Section 2.2.1 [M–file Runge–Kutta solvers], page 4 and is continued with the Mex–file Hairer–Wanner solvers in section Section 2.2.2 [Mex–file Hairer–Wanner solvers], page 5. 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 6.

2.2.1 M–file Runge–Kutta solvers

The M–file Runge–Kutta solvers are written in the Octave interpreter language and are stored '`*.m`' files. There have been implemented four different solvers of similiar structure and types, 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

[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".

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 $e = hd$, which is h times an $O(h^p)$ term, or rather $O(h^{(p+1)})$.

`ode23`
: Integrates a system of ordinary differential equations 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 ordinary differential equations 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`
: The Fehlberg 4(5) of the `ode45` pair is established and works well, however, the Dormand–Prince 4(5) 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 ordinary differential equations 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**
Non–stiff	DOPRI5	'`ode5d.m`'	'`odepkg_mexsolver_dopri5.c`'	'`dopri5.f`'
Non–stiff	DOP853	'`ode8d.m`'	'`odepkg_mexsolver_dop853.c`'	'`dop853.f`'
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 Other solvers

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 odeset can be displayed if odeset is called without any input argument and one output argument argument, eg. the following way

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

'RelTol' The option 'RelTol' is used to set the relative error tolerance for the error estimation of the solver 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). The definite error estimation equation also depends on the solver that is used, but generalized it may be of the form $e(t) = max(RelTol^T y(t), AbsTol)$. Run

```
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);
```

to see the effect of using different values for the option 'RelTol'.

'AbsTol' The option 'AbsTol' is used to set the absolute error tolerance for the error estimation of the solver 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). The definite error estimation equation also depends on the solver that is used, but generalized it may be of the form $e(t) = max(RelTol^T y(t), AbsTol)$. Run

```
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);
```

to see the effect of using different values for the option 'AbsTol'.

'NormControl'

The option 'NormControl' is used to set the type of error tolerance calculation of the solver while solving. It can either be the string 'on' or 'off'. At the time the solver starts the initialization procedure a warning message may be displayed if the solver will ignore the 'on' setting of this option because of an unhandled resp. missing implementation. The definite error estimation equation if set 'on' also depends on the solver that is used, but generalized it may be of the form $e(t) = max(RelTol^T max(norm(y(t), Inf)), AbsTol)$. Run

```
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);
```

to see the effect of using different values for the option '`NormControl`'.

'`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 be different when using different solvers. Run

```
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);
```

to see the effect of using different values for the option '`MaxStep`'.

'`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. Run

```
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);
```

to see the effect of using different values for the option '`InitialStep`'.

'`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. Run

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

to see the effect of using an output function with the option '`OutputFcn`'. 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. Run

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

to see an example.

'`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 $0 <= Refine <= 5$. Run

```
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);
```

to see the effect of using different values for the option '`Refine`'.

'`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. Run

```
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);
```

to see the effect of using different values for the option '`OutputSel`'.

'`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'`. Run

```
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);
```

to see the effect of using different values for the option '`Stats`'. 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 output structure in field '`stats`'. Run

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

to see an example.

'`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. Run

```
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);
```

to see the effect of using an Jacobian function with the option '`Jacobian`'. User defined Jacobian functions must have the form as described before (ie. '`function vr = fjac (vt, vy, varargin)`'.

'`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. Run

```
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);
```

to see the effect of using a Mass function with the option '`Mass`'. User defined Mass functions must have the form as described before (ie. '`function vmas = fmas (vt, vy, varargin)`'.

'`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'`. Run

```
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);
```

to see the effect of using a Mass function with the option '`MStateDependence`'. 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 single 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. Run

```
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);
```

to see the effect of not using the option '`NonNegative`' in the upper subplot and if using the option '`NonNegative`' in the lower suplot.

'`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. Run

```
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));
```

to see the effect of using an Events function with the option '`Events`'.

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

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

3 Coder's Guide

3.1 C Mex Function Reference

`void mexFixMsgTxt` (*const char * vmsg*) [Function]

vmsg: The string that has to be displayed

Displays the string `vmsg` in the octave window as "FIXME: ..." and continues.

`void mexUsgMsgTxt` (*const char * vmsg*) [Function]

vmsg: The string that has to be displayed

Displays the string `vmsg` in the octave window as "usage: ..." and stops computation because of an empty error message.

`bool mxIsEqual` (*const mxArray * vone, const mxArray * vtwo*) [Function]

vone: The first mxArray variable

vtwo: The second mxArray variable

Compares the two mxArrays `vone` and `vtwo` and returns a boolean value that is either *true* if both mxArrays are the same or *false* if the two mxArrays are different.

Return value: The constant *true* or *false*.

`bool mxIsVector` (*const mxArray * vmat*) [Function]

vmat: The numerical mxArray

Returns a boolean value that is either *true* if `vmat` is a vector or *false* if `vmat` is no vector.

Return value: The constant *true* or *false*.

`bool mxIsColumnVector` (*const mxArray * vmat*) [Function]

vmat: The numerical mxArray

Returns a boolean value that is either *true* if `vmat` is a column vector or *false* if `vmat` is no column vector.

Return value: The constant *true* or *false*.

`bool mxIsRowVector` (*const mxArray * vmat*) [Function]

vmat: The numerical mxArray

Returns a boolean value that is either *true* if `vmat` is a row vector or *false* if `vmat` is no row vector.

Return value: The constant *true* or *false*.

`bool mxIsMatrix` (*const mxArray * vmat*) [Function]

vmat: The numerical mxArray

Returns a boolean value that is either *true* if `vmat` is a numerical matrix or *false* if `vmat` is no matrix.

Return value: The constant *true* or *false*.

`mxArray * mxGetMatrixRow` (*mxArray * vmat, unsigned int vind*) [Function]

vmat: The numerical mxArray

Returns a newly allocated numerical mxArray with one row of elements from the matrix or vector `vmat`.

Return value: An newly allocated mxArray.

`mxArray * mxGetMatrixColumn` (*mxArray * `vmat`, unsigned int `vind`*) [Function]

vmat: The numerical mxArray

Returns a newly allocated numerical mxArray with one column of elements from the matrix or vector `vmat`.

Return value: An newly allocated mxArray.

`mxArray * mxTransposeMatrix` (*mxArray * `vmat`*) [Function]

vmat: The numerical mxArray

Returns a newly allocated numerical mxArray matrix that is the non-conjugate transposed matrix of `vmat`.

Return value: An newly allocated mxArray.

Appendix A

A.1 GNU Free Documentation License

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

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.