16.44 ODESOLVE: Ordinary differential equations solver

The ODESOLVE package is a solver for ordinary differential equations. At the present time it has very limited capabilities. It can handle only a single scalar equation presented as an algebraic expression or equation, and it can solve only first-order equations of simple types, linear equations with constant coefficients and Euler equations. These solvable types are exactly those for which Lie symmetry techniques give no useful information. For example, the evaluation of

```
depend(y,x);
odesolve(df(y,x)=x**2+e**x,y,x);
```

yields the result

```
        X                        3
    3*E   + 3*ARBCONST(1) + X
{Y=---------------------------}
                  3
```

Main Author: Malcolm A.H. MacCallum.

Other contributors: Francis Wright, Alan Barnes.

16.44.1 Introduction

ODESolve 1+ is an experimental project to update and enhance the ordinary differential equation (ODE) solver (odesolve) that has been distributed as a standard component of REDUCE [2, 4, 3] for about 10 years. ODESolve 1+ is intended to provide a strict superset of the facilities provided by odesolve. This document describes a substantial re-implementation of previous versions of ODESolve 1+ that now includes almost none of the original odesolve code. This version is targeted at REDUCE 3.7 or later, and will not run in earlier versions. This project is being conducted partly under the auspices of the European CATHODE project [1]. Various test files, including three versions based on a published review of ODE solvers [7], are included in the ODESolve 1+ distribution. For further background see [10], which describes version 1.03. See also [11].

ODESolve 1+ is intended to implement some solution techniques itself (i.e. most of the simple and well known techniques [12]) and to provide an automatic interface to other more sophisticated solvers, such as PSODE [5, 6, 8] and CRACK [9], to handle cases where simple techniques fail. It is also intended to provide a unified interface to other special solvers, such as Laplace transforms, series solutions and numerical methods, under user request. Although none of these extensions

is explicitly implemented yet, a general extension interface is implemented (see §16.44.6).

The main motivation behind ODESolve 1+ is pragmatic. It is intended to meet user expectations, to have an easy user interface that normally does the right thing automatically, and to return solutions in the form that the user wants and expects. Quite a lot of development effort has been directed toward this aim. Hence, ODESolve 1+ solves common text-book special cases in preference to esoteric pathological special cases, and it endeavours to simplify solutions into convenient forms.

16.44.2 Installation

The file odesolve.in inputs the full set of source files that are required to implement ODESolve 1+ *assuming that the current directory is the ODESolve 1+ source directory*. Hence, ODESolve 1+ can be run without compiling it in any implementation of REDUCE 3.7 by starting REDUCE in the ODESolve 1+ source directory and entering the statement

```
1: in "odesolve.in"$
```

However, the recommended procedure is to compile it by starting REDUCE in the ODESolve 1+ source directory and entering the statements

```
1: faslout odesolve;
2: in "odesolve.in"$
3: faslend;
```

In CSL-REDUCE, this will work only if you have write access to the REDUCE image file (reduce.img), so you may need to set up a private copy first. In PSL-REDUCE, you may need to move the compiled image file odesolve.b to a directory in your PSL load path, such as the main fasl directory. Please refer to the documentation for your implementation of REDUCE for details. Once a compiled version of ODESolve 1+ has been correctly installed, it can be loaded by entering the REDUCE statement

```
1: load_package odesolve;
```

A string describing the current version of ODESolve 1+ is assigned to the algebraic-mode variable odesolve_version, which can be evaluated to check what version is actually in use.

In versions of REDUCE derived from the development source after 22 September 2000, use of the normal algebraic-mode odesolve operator causes the package to

autoload. However, the ODESolve 1+ global switches are not declared, and the symbolic mode interface provided for backward compatibility with the previous version is not defined, until after the package has loaded. The former is not a huge problem because all ODESolve switches can be accessed as optional arguments, and the backward compatibility interface should probably not be used in new code anyway.

16.44.3 User interface

The principal interface is via the operator odesolve. (It also has a synonym called dsolve to make porting of examples from Maple easier, but it does not accept general Maple syntax!) For purposes of description I will refer to the dependent variable as "y" and the independent variable as "x", but of course the names are arbitrary. The general input syntax is

```
odesolve(ode, y, x, conditions, options);
```

All arguments except the first are optional. This is possible because, if necessary, ODESolve 1+ attempts to deduce the dependent and independent variables used and to make any necessary DEPEND declarations. Messages are output to indicate any assumptions or dependence declarations that are made. Here is an example of what is probably the shortest possible valid input:

```
odesolve(df(y,x));

*** Dependent var(s) assumed to be y

*** Independent var assumed to be x

*** depend y , x

{y=arbconst(1)}
```

Output of ODESolve 1+ messages is controlled by the standard REDUCE switch msg.

16.44.3.1 Specifying the ODE and its variables

The first argument (ode) is *required*, and must be either an ODE or a variable (or expression) that evaluates to an ODE. Automatic dependence declaration works *only* when the ODE is input *directly* as an argument to the odesolve operator. Here, "ODE" means an equation or expression containing one or more derivatives of y with respect to x. Derivatives of y with respect to other variables are not

allowed because ODESolve 1+ does not solve *partial* differential equations, and symbolic derivatives of variables other than y are treated as symbolic constants. An expression is implicitly equated to zero, as is usual in equation solvers.

The independent variable may be either an operator that explicitly depends on the independent variable, e.g. $y(x)$ (as required in Maple), or a simple variable that is declared (by the user or automatically by ODESolve 1+) to depend on the independent variable. If the independent variable is an operator then it may depend on parameters as well as the independent variable. Variables may be simple identifiers or, more generally, REDUCE "kernels", e.g.

```
operator x, y;
odesolve(df(y(x(a),b),x(a)) = 0);

*** Dependent var(s) assumed to be y(x(a),b)

*** Independent var assumed to be x(a)

{y(x(a),b)=arbconst(1)}
```

The order in which arguments are given must be preserved, but arguments may be omitted, except that if x is specified then y must also be specified, although an empty list { } can be used as a "place-holder" to represent "no specified argument". Variables are distinguished from options by requiring that if a variable is specified then it must appear in the ODE, otherwise it is assumed to be an option.

Generally in REDUCE it is not recommended to use the identifier t as a variable, since it is reserved in Lisp. However, it is very common practice in applied mathematics to use it as a variable to represent time, and for that reason ODESolve 1+ provides special support to allow it as either the independent or a dependent variable. But, of course, its use may still cause trouble in other parts of REDUCE!

16.44.3.2 Specifying conditions

If specified, the "conditions" argument must take the form of an (unordered) list of (unordered lists of) equations with either y, x, or a derivative of y on the left. A single list of conditions need not be contained within an outer list. Combinations of conditions are allowed. Conditions within one (inner) list all relate to the same x value. For example:

Boundary conditions:
{{y=y0, x=x0}, {y=y1, x=x1}, ...}

Initial conditions:
{x=x0, y=y0, df(y,x)=dy0, ...}

Combined conditions:
{{y=y0, x=x0}, {df(y,x)=dy1, x=x1}, {df(y,x)=dy2, y=y2, x=x2}, ...}

Here is an example of boundary conditions:

```
odesolve(df(y,x,2) = y, y, x, {{x = 0, y = A}, {x = 1, y = B}});
```

```
         2*x       2*x           2
      - e    *a + e    *b*e + a*e  - b*e
{y=-----------------------------------}
                  x  2    x
                 e *e  - e
```

Here is an example of initial conditions:

```
odesolve(df(y,x,2) = y, y, x, {x = 0, y = A, df(y,x) = B});
```

```
      2*x       2*x
     e    *a + e    *b + a - b
{y=-------------------------}
                  x
               2*e
```

Here is an example of combined conditions:

```
odesolve(df(y,x,2) = y, y, x, {{x=0, y=A}, {x=1, df(y,x)=B}});
```

```
      2*x       2*x           2
     e    *a + e    *b*e + a*e  - b*e
{y=-------------------------------}
                  x  2    x
                 e *e  + e
```

Boundary conditions on the values of y at various values of x may also be specified by replacing the variables by equations with single values or matching lists of values on the right, of the form

y = y0, x = x0

or

y = {y0, y1, ...}, x = {x0, x2, ...}

For example

```
odesolve(df(y,x) = y, y = A, x = 0);

    x
{y=e *a}

odesolve(df(y,x,2) = y, y = {A, B}, x = {0, 1});

         2*x       2*x           2
      - e    *a + e    *b*e + a*e  - b*e
{y=-----------------------------------}
                 x  2    x
                e *e  - e
```

16.44.3.3 Specifying options and defaults

The final arguments may be one or more of the option identifiers listed in the table below, which take precedence over the default settings. All options can also be specified on the right of equations with the identifier "output" on the left, e.g. "output = basis". This facility if provided mainly for compatibility with other systems such as Maple, although it also allows options to be distinguished from variables in case of ambiguity. Some options can be specified on the left of equations that assign special values to the option. Currently, only "trode" and its synonyms can be assigned the value 1 to give an increased level of tracing.

The following switches set default options – they are all off by default. Options set locally using option arguments override the defaults set by switches.

Switch	Option	Effect on solution
odesolve_explicit	explicit	fully explicit
odesolve_expand	expand	expand roots of unity
odesolve_full	full	fully explicit and expanded
odesolve_implicit	implicit	implicit instead of parametric
	algint	turn on algint
odesolve_noint	noint	turn off selected integrations
odesolve_verbose	verbose	display ODE and conditions
odesolve_basis	basis	output basis solution for linear ODE
	trode	
trode	trace	turn on algorithm tracing
	tracing	
odesolve_fast	fast	turn off heuristics
odesolve_check	check	turn on solution checking

An "explicit" solution is an equation with y isolated on the left whereas an "implicit" solution is an equation that determines y as one or more of its solutions. A

"parametric" solution expresses both x and y in terms of some additional parameter. Some solution techniques naturally produce an explicit solution, but some produce either an implicit or a parametric solution. The "explicit" option causes `ODESolve 1+` to attempt to convert solutions to explicit form, whereas the "implicit" option causes `ODESolve 1+` to attempt to convert parametric solutions (only) to implicit form (by eliminating the parameter). These solution conversions may be slow or may fail in complicated cases.

`ODESolve 1+` introduces two operators used in solutions: `root_of_unity` and `plus_or_minus`, the latter being a special case of the former, i.e. a second root of unity. These operators carry a tag that associates the same root of unity when it appears in more than one place in a solution (cf. the standard `root_of` operator). The "expand" option expands a single solution expressed in terms of these operators into a set of solutions that do not involve them. `ODESolve 1+` also introduces two operators `expand_roots_of_unity` [which should perhaps be named `expand_root_of_unity`] and `expand_plus_or_minus`, that are used internally to perform the expansion described above, and can be used explicitly.

The "algint" option turns on "algebraic integration" locally only within `ODESolve 1+`. It also loads the `algint` package if necessary. Algint allows `ODESolve 1+` to solve some ODEs for which the standard REDUCE integrator hangs (i.e. takes an extremely long time to return). If the resulting solution contains unevaluated integrals then the algint switch should be turned on outside `ODESolve 1+` before the solution is re-evaluated, otherwise the standard integrator may well hang again! For some ODEs, the algint option leads to better solutions than the standard REDUCE integrator.

Alternatively, the "noint" option prevents REDUCE from attempting to evaluate the integrals that arise in some solution techniques. If `ODESolve 1+` takes too long to return a result then you might try adding this option to see if it helps solve this particular ODE, as illustrated in the test files. This option is provided to speed up the computation of solutions that contain integrals that cannot be evaluated, because in some cases REDUCE can spend a long time trying to evaluate such integrals before returning them unevaluated. This only affects integrals evaluated *within* the `ODESolve 1+` operator. If a solution containing an unevaluated integral that was returned using the "noint" option is re-evaluated, it may again take REDUCE a very long time to fail to evaluate the integral, so considerable caution is recommended! (A global switch called "noint" is also installed when `ODESolve 1+` is loaded, and can be turned on to prevent REDUCE from attempting to evaluate *any* integrals. But this effect may be very confusing, so this switch should be used only with extreme care. If you turn it on and then forget, you may wonder why REDUCE seems unable to evaluate even trivial integrals!)

The "verbose" option causes `ODESolve 1+` to display the ODE, variables and conditions as it sees them internally, after pre-processing. This is intended for use

in demonstrations and possibly for debugging, and not really for general users.

The "basis" option causes `ODESolve 1+` to output the general solutions of linear ODEs in basis format (explained below). Special solutions (of ODEs with conditions) and solutions of nonlinear ODEs are not affected.

The "trode" (or "trace" or "tracing") option turns on tracing of the algorithms used by `ODESolve 1+`. It reports its classification of the ODE and any intermediate results that it computes, such as a chain of progressively simpler (in some sense) ODEs that finally leads to a solution. Tracing can produce a lot of output, e.g. see the test log file "`zimmer.rlg`". The option "`trode = 1`" or the global assignment "`!*trode := 1`" causes `ODESolve 1+` to report every test that it tries in its classification process, producing even more tracing output. This is probably most useful for debugging, but it may give the curious user further insight into the operation of `ODESolve 1+`.

The "fast" option disables all non-deterministic solution techniques (including most of those for nonlinear ODEs of order > 1). It may be most useful if `ODESolve 1+` is used as a subroutine, including calling it recursively in a hook. It makes `ODESolve 1+` behave like the `odesolve` distributed with REDUCE versions up to and including 3.7, and so does not affect the `odesolve.tst` file. The "fast" option causes `ODESolve 1+` to return no solution fast in cases where, by default, if would return either a solution or no solution (perhaps much) more slowly. Solution of sufficiently simple "deterministically-solvable" ODEs is unaffected.

The "check" option turns on checking of the solution. This checking is performed by code that is largely independent of the solver, so as to perform a genuinely independent check. It is not turned on by default so as to avoid the computational overhead, which is currently of the order of 30%. A check is made that each component solution satisfies the ODE and that a general solution contains at least enough arbitrary constants, or equivalently that a basis solution contains enough basis functions. Otherwise, warning messages are output. It is possible that `ODESolve 1+` may fail to verify a solution because the automatic simplification fails, which indicates a failure in the checker rather than in the solver. This option is not yet well tested; please report any checking failures to me (FJW).

In some cases, in particular when an implicit solution contains an unevaluated integral, the checker may need to differentiate an integral with respect to a variable other than the integration variable. In order to do this, it turns on the differentiator switch "allowdfint" globally. [I hope that this setting will eventually become the default.] In some cases, in particular in symbolic solutions of Clairaut ODEs, the checker may need to differentiate a composition of operators using the chain rule. In order to do this, it turns on the differentiator switch "expanddf" locally only. Although the code to support both these differentiator facilities has been in REDUCE for a while, they both require patches that are currently only applied when

`ODESolve 1+` is loaded. [I hope that these patches will eventually become part of REDUCE itself.]

16.44.4 Output syntax

If `ODESolve 1+` is successful it outputs a list of sub-solutions that together represent the solution of the input ODE. Each sub-solution is either an equation that defines a branch of the solution, explicitly or implicitly, or it is a list of equations that define a branch of the solution parametrically in the form $\{y = G(p), x = F(p), p\}$. Here p is the parameter, which is actually represented in terms of an operator called `arbparam` which has an integer argument to distinguish it from other unrelated parameters, as usual for arbitrary values in REDUCE.

A general solution will contain a number of arbitrary constants represented by an operator called `arbconst` with an integer argument to distinguish it from other unrelated arbitrary constants. A special solution resulting from applying conditions will contain fewer (usually no) arbitrary constants.

The general solution of a linear ODE in basis format is a list consisting of a list of basis functions for the solution space of the reduced ODE followed by a particular solution if the input ODE had a y-independent "driver" term, i.e. was not reduced (which is sometimes ambiguously called "homogeneous"). The particular solution is normally omitted if it is zero. The dependent variable y does not appear in a basis solution. The linear solver uses basis solutions internally.

Currently, there are cases where `ODESolve 1+` cannot solve a linear ODE using its linear solution techniques, in which case it will try nonlinear techniques. These may generate a solution that is not (obviously) a linear combination of basis solutions. In this case, if a basis solution has been requested, `ODESolve 1+` will report that it cannot separate the nonlinear combination, which it will return as the default linear combination solution.

If `ODESolve 1+` fails to solve the ODE then it will return a list containing the input ODE (always in the form of a differential expression equated to 0). At present, `ODESolve 1+` does not return partial solutions. If it fails to solve any part of the problem then it regards this as complete failure. (You can probably see if this has happened by turning on algorithm tracing.)

16.44.5 Solution techniques

The `ODESolve 1+` interface module pre-processes the problem and applies any conditions to the solution. The other modules deal with the actual solution.

`ODESolve 1+` first classifies the input ODE according to whether it is linear or nonlinear and calls the appropriate solver. An ODE that consists of a product of

linear factors is regarded as nonlinear. The second main classification is based on whether the input ODE is of first or higher degree.

Solution proceeds essentially by trying to reduce nonlinear ODEs to linear ones, and to reduce higher order ODEs to first order ODEs. Only simple linear ODEs and simple first-order nonlinear ODEs can be solved directly. This approach involves considerable recursion within `ODESolve 1+`.

If all solution techniques fail then `ODESolve 1+` attempts to factorize the derivative of the whole ODE, which sometimes leads to a solution.

16.44.5.1 Linear solution techniques

`ODESolve 1+` splits every linear ODE into a "reduced ODE" and a "driver" term. The driver is the component of the ODE that is independent of y, the reduced ODE is the component of the ODE that depends on y, and the sign convention is such that the ODE can be written in the form "reduced ODE = driver". The reduced ODE is then split into a list of "ODE coefficients".

The linear solver now determines the order of the ODE. If it is 1 then the ODE is immediately solved using an integrating factor (if necessary). For a higher order linear ODE, `ODESolve 1+` considers a sequence of progressively more complicated solution techniques. For most purposes, the ODE is made "monic" by dividing through by the coefficient of the highest order derivative. This puts the ODE into a standard form and effectively deals with arbitrary overall algebraic factors that would otherwise confuse the solution process. (Hence, there is no need to perform explicit algebraic factorization on linear ODEs.) The only situation in which the original non-monic form of the ODE is considered is when checking for exactness, which may depend critically on otherwise irrelevant overall factors.

If the ODE has constant coefficients then it can (in principle) be solved using elementary "D-operator" techniques in terms of exponentials via an auxiliary equation. However, this works only if the polynomial auxiliary equation can be solved. Assuming that it can and there is a driver term, `ODESolve 1+` tries to use a method based on inverse "D-operator" techniques that involves repeated integration of products of the solutions of the reduced ODE with the driver. Experience (by Malcolm MacCallum) suggests that this normally gives the most satisfactory form of solution if the integrals can be evaluated. If any integral fails to evaluate, the more general method of "variation of parameters", based on the Wronskian of the solution set of the reduced ODE, is used instead. This involves only a single integral and so can never lead to nested unevaluated integrals.

If the ODE has non-constant coefficients then it may be of Euler (sometimes ambiguously called "homogeneous") type, which can be trivially reduced to an ODE with constant coefficients. A shift in x is accommodated in this process. Next it is tested for exactness, which leads to a first integral that is an ODE of order one

lower. After that it is tested for the explicit absence of y and low order derivatives, which allows trivial order reduction. Then the monic ODE is tested for exactness, and if that fails and the original ODE was non-monic then the original form is tested for exactness.

Finally, pattern matching is used to seek a solution involving special functions, such as Bessel functions. Currently, this is implemented only for second-order ODEs satisfied by Bessel and Airy-integral functions. It could easily be extended to other orders and other special functions. Shifts in x could also be accommodated in the pattern matching. [Work to enhance this component of `ODESolve 1+` is currently in progress.]

If all linear techniques fail then `ODESolve 1+` currently calls the variable interchange routine (described below), which takes it into the nonlinear solver. Occasionally, this is successful in producing some, although not necessarily the best, solution of a linear ODE.

16.44.5.2 Nonlinear solution techniques

In order to handle trivial nonlinearity, `ODESolve 1+` first factorizes the ODE algebraically, solves each factor that depends on y and then merges the resulting solutions. Other factors are ignored, but a warning is output unless they are purely numerical.

If all attempts at solution fail then `ODESolve 1+` checks whether the original (unfactored) ODE was exact, because factorization could destroy exactness. Currently, `ODESolve 1+` handles only first and second order nonlinear exact ODEs.

A version of the main solver applied to each algebraic factor branches depending on whether the ODE factor is linear or nonlinear, and the nonlinear solver branches depending on whether the order is 1 or higher and calls one of the solvers described in the next two sections. If that solver fails, `ODESolve 1+` checks for exactness (of the factor). If that fails, it checks whether only a single order derivative is involved and tries to solve algebraically for that. If successful, this decomposes the ODE into components that are, in some sense, simpler and may be solvable. (However, in some cases these components are algebraically very complicated examples of simple types of ODE that the integrator cannot in practice handle, and it can take a very long time before returning an unevaluated integral.)

If all else fails, `ODESolve 1+` interchanges the dependent and independent variables and calls the top-level solver recursively. It keeps a list of all ODEs that have entered the top-level solver in order to break infinite loops that could arise if the solution of the variable-interchanged ODE fails.

First-order nonlinear solution techniques If the ODE is a first-degree polynomial in the derivative then `ODESolve 1+` represents it in terms of the "gradient", which is a function of x and y such that the ODE can be written as "$dy/dx = \textit{gradient}$". It then checks *in sequence* for the following special types of ODE, each of which it can (in principle) solve:

Separable The gradient has the form $f(x)g(y)$, leading immediately to a solution by quadrature, i.e. the solution can be immediately written in terms of indefinite integrals. (This is considered to be a solution of the ODE, regardless of whether the integrals can be evaluated.) The solver recognises both explicit and implicit dependence when detecting separable form.

Quasi-separable The gradient has the form $f(y + kx)$, which is (trivially) separable after a linear transformation. It arises as a special case of the "quasi-homogeneous" case below, but is better treated earlier as a case in its own right.

Homogeneous The gradient has the form $f(y/x)$, which is algebraically homogeneous. A substitution of the form "$y = vx$" leads to a first-order linear ODE that is (in principle) immediately solvable.

Quasi-homogeneous The gradient has the form $f(\frac{a_1x+b_1y+c_1}{a_2x+b_2y+c_2})$, which is homogeneous after a linear transformation.

Bernoulli The gradient has the form $P(x)y + Q(x)y^n$, in which case the ODE is a first-order linear ODE for y^{1-n}.

Riccati The gradient has the form $a(x)y^2 + b(x)y + c(x)$, in which case the ODE can be transformed into a *linear* second-order ODE that may be solvable.

If the ODE is not first-degree then it may be linear in either x or y. Solving by taking advantage of this leads to a parametric solution of the original ODE, in which the parameter corresponds to y'. It may then be possible to eliminate the parameter to give either an implicit or explicit solution.

An ODE is "solvable for y" if it can be put into the form $y = f(x, y')$. Differentiating with respect to x leads to a first-order ODE for $y'(x)$, which may be easier to solve than the original ODE. The special case that $y = xF(y') + G(y')$ is called a Lagrange (or d'Alembert) ODE. Differentiating with respect to x leads to a first-order linear ODE for $x(y')$. The even more special case that $y = xy' + G(y')$, which may arise in the equivalent implicit form $F(xy' - y) = G(y')$, is called a Clairaut ODE. The general solution is given by replacing y' by an arbitrary constant, and it may be possible to obtain a singular solution by differentiating and solving the resulting factors simultaneously with the original ODE.

An ODE is "solvable for x" if it can be put into the form $x = f(y, y')$. Differentiating with respect to y leads to a first-order ODE for $y'(y)$, which may be easier to solve than the original ODE.

Currently, `ODESolve 1+` recognises the above forms only if the ODE manifestly has the specified form and does not try very hard to actually solve for x or y, which perhaps it should!

Higher-order nonlinear solution techniques The techniques used here are all special cases of Lie symmetry analysis, which is not yet applied in any general way.

Higher-order nonlinear ODEs are passed through a number of "simplifier" filters that are applied in succession, regardless of whether the previous filter simplifies the ODE or not. Currently, the first filter tests for the explicit absence of y and low order derivatives, which allows trivial order reduction. The second filter tests whether the ODE manifestly depends on $x + k$ for some constant k, in which case it shifts x to remove k.

After that, `ODESolve 1+` tests for each of the following special forms in sequence. The sequence used here is important, because the classification is not unique, so it is important to try the most useful classification first.

Autonomous An ODE is autonomous if it does not depend explicitly on x, in which case it can be reduced to an ODE in y' of order one lower.

Scale invariant or equidimensional in x An ODE is scale invariant if it is invariant under the transformation $x \to ax, y \to a^p y$, where a is an arbitrary indeterminate and p is a constant to be determined. It can be reduced to an autonomous ODE, and thence to an ODE of order one lower. The special case $p = 0$ is called equidimensional in x. It is the nonlinear generalization of the (reduced) linear Euler ODE.

Equidimensional in y An ODE is equidimensional in y if it is invariant under the transformation $y \to ay$. An exponential transformation of y leads to an ODE of the same order that *may* be "more linear" and so easier to solve, but there is no guarantee of this. All (reduced) linear ODEs are trivially equidimensional in y.

The recursive nature of `ODESolve 1+`, especially the thread described in this section, can lead to complicated "arbitrary constant expressions". Arbitrary constants must be included at the point where an ODE is solved by quadrature. Further processing of such a solution, as may happen when a recursive solution stack is unwound, can lead to arbitrary constant expressions that should be re-written as simple arbitrary constants. There is some simple code included to perform this arbitrary constant simplification, but it is rudimentary and not entirely successful.

16.44.6 Extension interface

The idea is that the ODESolve extension interface allows any user to add solution techniques without needing to edit and recompile the main source code, and (in principle) without needing to be intimately familiar with the internal operation of `ODESolve 1+`.

The extension interface consists of a number of "hooks" at various critical places within `ODESolve 1+`. These hooks are modelled in part on the hook mechanism used to extend and customize the Emacs editor, which is a large Lisp-based system with a structure similar to that of REDUCE . Each `ODESolve 1+` hook is an identifier which can be defined to be a function (i.e. a procedure), or have assigned to it (in symbolic mode) a function name or a (symbolic mode) list of function names. The function should be written to accept the arguments specified for the particular hook, and it should return either a solution to the specified class of ODE in the specified form or nil.

If a hook returns a non-nil value then that value is used by `ODESolve 1+` as the solution of the ODE at that stage of the solution process. [If the ODE being solved was generated internally by `ODESolve 1+` or conditions are imposed then the solution will be re-processed before being finally returned by `ODESolve 1+`.] If a hook returns nil then it is ignored and `ODESolve 1+` proceeds as if the hook function had not been called at all. This is the same mechanism that it used internally by `ODESolve 1+` to run sub-solvers. If a hook evaluates to a list of function names then they are applied in turn to the hook arguments until a non-nil value is returned and this is the value of the hook; otherwise the hook returns nil. The same code is used to run all hooks and it checks that an identifier is the name of a function before it tries to apply it; otherwise the identifier is ignored. However, the hook code does not perform any other checks, so errors within functions run by hooks will probably terminate `ODESolve 1+` and errors in the return value will probably cause fatal errors later in `ODESolve 1+`. Such errors are user errors rather than `ODESolve 1+` errors!

Hooks are defined in pairs which are inserted before and after critical stages of the solver, which currently means the general ODE solver, the nonlinear ODE solver, and the solver for linear ODEs of order greater than one (on the grounds that solving first order linear ODEs is trivial and the standard `ODESolve 1+` code should always suffice). The precise interface definition is as follows.

A reference to an "algebraic expression" implies that the REDUCE representation is a prefix or pseudo-prefix form. A reference to a "variable" means an identifier (and never a more general kernel). The "order" of an ODE is always an explicit positive integer. The return value of a hook function must always be either nil or an algebraic-mode list (which must be represented as a prefix form). Since the input and output of hook functions uses prefix forms (and never standard quotient forms), hook functions can equally well be written in either algebraic or symbolic

mode, and in fact `ODESolve 1+` uses a mixture internally. (An algebraic-mode procedure can return nil by returning nothing. The integer zero is *not* equivalent to nil in the context of `ODESolve 1+` hooks.)

Hook names: `ODESolve_Before_Hook`, `ODESolve_After_Hook`.

Run before and after: The general ODE solver.

Arguments: 3

1. The ODE in the form of an algebraic expression with no denominator that must be made identically zero by the solution.
2. The dependent variable.
3. The independent variable.

Return value: A list of equations exactly as returned by `ODESolve 1+` itself.

Hook names: `ODESolve_Before_Non_Hook`, `ODESolve_After_Non_Hook`.

Run before and after: The nonlinear ODE solver.

Arguments: 4

1. The ODE in the form of an algebraic expression with no denominator that must be made identically zero by the solution.
2. The dependent variable.
3. The independent variable.
4. The order of the ODE.

Return value: A list of equations exactly as returned by `ODESolve 1+` itself.

Hook names: `ODESolve_Before_Lin_Hook`, `ODESolve_After_Lin_Hook`.

Run before and after: The general linear ODE solver.

Arguments: 6

1. A list of the coefficient functions of the "reduced ODE", i.e. the coefficients of the different orders (including zero) of derivatives of the dependent variable, each in the form of an algebraic expression, in low to high derivative order. [In general the ODE will not be "monic" so the leading (i.e. last) coefficient function will not be 1. Hence, the ODE may contain an essentially irrelevant overall algebraic factor.]
2. The "driver" term, i.e. the term involving only the independent variable, in the form of an algebraic expression. The sign convention is such that "reduced ODE = driver".
3. The dependent variable.
4. The independent variable.
5. The (maximum) order (> 1) of the ODE.
6. The minimum order derivative present.

Return value: A list consisting of a basis for the solution space of the reduced ODE and optionally a particular integral of the full ODE. This list does not contain any equations, and the dependent variable never appears in it. The particular integral may be omitted if it is zero. The basis is itself a list of algebraic expressions in the independent variable. (Hence the return value is always a list and its first element is also always a list.)

Hook names: `ODESolve_Before_Non1Grad_Hook`, `ODESolve_After_Non1Grad_Hook`.

Run before and after: The solver for first-order first-degree nonlinear ("gradient") ODEs, which can be expressed in the form $dy/dx = \mathrm{gradient}(y, x)$.

Arguments: 3

1. The "gradient", which is an algebraic expression involving (in general) the dependent and independent variables, to which the ODE equates the derivative.
2. The dependent variable.
3. The independent variable.

Return value: A list of equations exactly as returned by `ODESolve 1+` itself. (In this case the list should normally contain precisely one equation.)

The file `extend.tst` contains a very simple test and demonstration of the operation of the first three classes of hook.

This extension interface is experimental and subject to change. Please check the version of this document (or the source code) for the version of `ODESolve 1+` you are actually running.

16.44.7 Change log

27 February 1999 Version 1.06 frozen.

13 July 2000 Version 1.061 added an extension interface.

8 August 2000 Version 1.062 added the "fast" option.

21 September 2000 Version 1.063 added the "trace", "check" and "algint" options, the "Non1Grad" hooks, handling of implicit dependence in separable ODEs, and handling of the general class of quasi-homogeneous ODEs.

28 September 2000 Version 1.064 added support for using 't' as a variable and replaced the version identification output by the `odesolve_version` variable.

14 August 2001 Version 1.065 fixed obscure bugs in the first-order nonlinear ODE handler and the arbitrary constant simplifier, and revised some tracing messages slightly.

16.44.8 Planned developments

- Extend special-function solutions and allow shifts in x.
- Improve solution of linear ODEs, by (a) using linearity more generally to solve as "CF + PI", (b) finding at least polynomial solutions of ODEs with polynomial coefficients, (c) implementing non-trivial reduction of order.
- Improve recognition of exact ODEs, and add some support for more general use of integrating factors.
- Add a "classify" option, that turns on trode but avoids any actual solution, to report all possible (?) top-level classifications.
- Improve arbconst and arbparam simplification.
- Add more standard elementary techniques and more general techniques such as Lie symmetry, Prelle-Singer, etc.

- Improve integration support, preferably to remove the need for the "noint" option.
- Solve systems of ODEs, including under- and over-determined ODEs and systems. Link to CRACK (Wolf) and/or DiffGrob2 (Mansfield)?
- Move more of the implementation to symbolic-mode code.

Bibliography

[1] CATHODE (Computer Algebra Tools for Handling Ordinary Differential Equations) http://www-lmc.imag.fr/CATHODE/, http://www-lmc.imag.fr/CATHODE2/.

[2] A. C. Hearn and J. P. Fitch (ed.), *REDUCE User's Manual 3.6*, RAND Publication CP78 (Rev. 7/95), RAND, Santa Monica, CA 90407-2138, USA (1995).

[3] M. A. H. MacCallum, An Ordinary Differential Equation Solver for REDUCE, *Proc. ISSAC '88, ed. P. Gianni, Lecture Notes in Computer Science* **358**, Springer-Verlag (1989), 196–205.

[4] M. A. H. MacCallum, ODESOLVE, LaTeX file `reduce/doc/odesolve.tex` distributed with REDUCE 3.6. The first part of this document is included in the printed REDUCE User's Manual 3.6 [2], 345–346.

[5] Y.-K. Man, *Algorithmic Solution of ODEs and Symbolic Summation using Computer Algebra*, PhD Thesis, School of Mathematical Sciences, Queen Mary and Westfield College, University of London (July 1994).

[6] Y.-K. Man and M. A. H. MacCallum, A Rational Approach to the Prelle-Singer Algorithm, *J. Symbolic Computation*, **24** (1997), 31–43.

[7] F. Postel and P. Zimmermann, A Review of the ODE Solvers of AXIOM, DERIVE, MAPLE, MATHEMATICA, MACSYMA, MUPAD and REDUCE, *Proceedings of the 5th Rhine Workshop on Computer Algebra, April 1-3, 1996, Saint-Louis, France.* Specific references are to the version dated April 11, 1996. The latest version of this review, together with log files for each of the systems, is available from http://www.loria.fr/~zimmerma/ComputerAlgebra/.

[8] M. J. Prelle and M. F. Singer, Elementary First Integrals of Differential Equations, *Trans. AMS* **279** (1983), 215–229.

[9] T. Wolf and A. Brand, The Computer Algebra Package CRACK for Investigating PDEs, LaTeX file reduce/doc/crack.tex distributed with REDUCE 3.6. A shorter document is included in the printed REDUCE User's Manual 3.6 [2], 241–244.

[10] F. J. Wright, An Enhanced ODE Solver for REDUCE. *Programmirovanie* No 3 (1997), 5–22, in Russian, and *Programming and Computer Software* No 3 (1997), in English.

[11] F. J. Wright, Design and Implementation of ODESolve 1+ : An Enhanced REDUCE ODE Solver. CATHODE Workshop Report, Marseilles, May 1999, CATHODE (1999).
http://centaur.maths.qmw.ac.uk/Papers/Marseilles/.

[12] D. Zwillinger, *Handbook of Differential Equations*, Academic Press. (Second edition 1992.)