Package 'redcas'

November 3, 2026

Type Package

Depends R (>= 4.1.0), methods

Title An Interface to the Computer Algebra System 'REDUCE'

Version 0.2.1

Date 2026-10-28

Author Martin Gregory [aut, cre]

Maintainer Martin Gregory <mairtin.macghreagoir@gmail.com>

Description 'REDUCE' is a portable general-purpose computer algebra system supporting scalar, vector, matrix and tensor algebra, symbolic differential and integral calculus, arbitrary precision numerical calculations and output in 'LaTeX' format. 'REDUCE' is based on 'Lisp' and is available on the two dialects 'Portable Standard Lisp' ('PSL') and 'Codemist Standard Lisp' ('CSL'). The 'redcas' package provides an interface for executing arbitrary 'REDUCE' code interactively from 'R', returning output as character vectors. 'R' code and 'REDUCE' code can be interspersed. It also provides specialized functions for calling 'REDUCE' functions, for example, solving simultaneous or differential equations, differentiation and factorization of polynomials and integers returning the output as an appropriate 'R' object. A further specialized function uses 'REDUCE' features to generate 'LaTeX' output and post-processes this for direct use in 'LaTeX' documents using, e.g., 'Sweave'.

License GPL-3

SystemRequirements 'REDUCE' <https://reduce-algebra.sourceforge.io>.
At least one of the 'CSL' or 'PSL' dialects must be installed
in order to use the package. Tested with 'REDUCE' packaged
releases 7335 and 6860 and a non-packaged release 7131 for both
'CSL' and 'PSL' on Linux, and packaged releases 6866and 6658
for 'CSL' on MacOS Intel and MacOS arm M5.

NeedsCompilation no

OS_type unix

Repository CRAN

Date/Publication 2026-11-03 16:00:09 UTC

Contents

redcas-package . . . 2
factorize . . . 4
odesolve . . . 7
print . . . 11
R2latex . . . 13
red2Rarray . . . 15
red2Rfn . . . 17
red2Rlist . . . 19
redcas.ode . . . 22
redcas.solve . . . 23
redClose . . . 24
redCodeDir . . . 25
redDf . . . 26
redDropOut . . . 30
redExec . . . 31
redExpand . . . 33
redFunc . . . 34
redin . . . 37
redLatex . . . 38
redSolve . . . 43
redSplitOut . . . 46
redStart . . . 48
sessionGet . . . 50
showSessions . . . 51
swset . . . 52

Index **55**

`redcas-package` *An Interface to the Computer Algebra System 'REDUCE'*

Description

'REDUCE' is a portable general-purpose computer algebra system supporting scalar, vector, matrix and tensor algebra, symbolic differential and integral calculus, arbitrary precision numerical calculations and output in 'LaTeX' format. 'REDUCE' is based on 'Lisp' and is available on the two dialects 'Portable Standard Lisp' ('PSL') and 'Codemist Standard Lisp' ('CSL'). The 'redcas' package provides an interface for executing arbitrary 'REDUCE' code interactively from 'R', returning output as character vectors. 'R' code and 'REDUCE' code can be interspersed. It also provides specialized functions for calling 'REDUCE' functions, for example, solving simultaneous or differential equations, differentiation and factorization of polynomials and integers returning the output as an appropriate 'R' object. A further specialized function uses 'REDUCE' features to generate 'LaTeX' output and post-processes this for direct use in 'LaTeX' documents using, e.g., 'Sweave'.

Details

The package allows you to start and stop a REDUCE session using `redStart` and `redClose`. There are three different ways to interact with the session:

1. `redExec` executes a vector of arbitrary REDUCE code and returns its output. The code to execute may contain any element of the language and so requires a certain level of familiarity with REDUCE programming. Output is returned as a list of character vectors containing the command output, the commands and the combined commands and output. Command output can be plain text or LaTeX;
2. `redFunc` executes a vector of REDUCE functions including those defined by the user. This requires only knowledge of the arguments of the functions to be executed. Output is returned as a character vector which can be plain text or LaTeX;
3. **Special Interface functions** provide access to specific REDUCE functions, have arguments tailored to the specific REDUCE function and construct the necessary REDUCE calls. They also provide some level of validation of the inputs. Output may be returned as plain text or LaTeX or as R functions or expressions. In this version there are interfaces to the REDUCE functions for solving algebraic equations, differentiation, solving ordinary differential equations and factorization of integers and polynomials.

Functions for converting REDUCE objects to R equivalents are available: for example `red2Rarray` and `red2Rlist` convert to arrays and lists while `red2Rfn` converts REDUCE expressions to R functions. Both REDUCE objects and some types of R objects can be rendered as LaTeX using `redLatex` and `R2latex`, respectively.

In addition to the function documentation, the vignette Using redcas provides further examples of use.

REDUCE uses a different terminology to R. Some translations:

REDUCE	**R**
command:	statement
log:	transcript
operator (prefix):	function
operator (infix):	operator
procedure:	function
switch:	(global) option

Author(s)

Martin Gregory [aut, cre]

Maintainer: Martin Gregory <mairtin.macghreagoir@gmail.com>

References

Details of the REDUCE system including extensive documentation is available at the REDUCE web site.

Examples

```
## start a session
id <- redStart()

## can only run code if session was successfully started
if (is.numeric(id)) {
   ## define vectors of REDUCE code for submission
   x1 <- "solve((x+1)^2, x);"
   x2 <- "b:=factorize(45056);"
   x3 <- c("operator v; ",
    "la:=-m/(v(2)^2 + v(3)^2)^(1/2) ;")

   ## execute x1 and display on the output of the commands
   y1 <- redExec(id, x1)
   writeLines(y1$out)

   ## execute x2 and display first the output and then the commands
   y2 <- redExec(id, x2)
   writeLines(c(y2$out, y2$cmd))

   ## execute x3 and display the log (commands and output interspersed)
   writeLines(y3 <- redExec(id, x3, split=FALSE))

   ## close the session
   redClose(id)
   }
```

`factorize` *factorize numbers or polynomials*

Description

Call the REDUCE *factorize* operator to factorize an integer, rational or complex number or univariate and multivariate polynomials with such coefficients.

Usage

```
factorize(id, x, prime=integer(0), outfmt="text", eol=FALSE,
          switches=character(0), verbose=FALSE, ...)
```

Arguments

`id`	the session identifier returned by `redStart`. Required with no default.
`x`	the numbers or polynomials to factorize as a numeric, character or expression vector or a list containing values which are either numeric, character or expression. Required.

prime	Integer. The factorizer works by first reducing multivariate problems to univariate ones and then solving the univariate ones modulo small primes. The primes are selected using a random number generator. This argument allows one to specify a prime known to be effective in factoring the polynomial. Details in section 9.2 of the user manual.
outfmt	Character. Specifies how the result is to be return. Allowed values are "text", "latex" or "expr" which causes plain text, LaTeX or expressions to be returned. Default "text".
eol	Logical. When output is "latex" and eol is true, append LaTeX newline characters to all but the last element of each vector in the output.
switches	Character vector of switches to be applied while factorize is executing. See details for information on switches influencing the output and swset for details of the switch argument. Note that neither ECHO, NAT nor FANCY are allowed. Optional.
verbose	Logical. Controls whether informational message are printed. Default is FALSE.
...	Arguments which can be passed to redLatex (usermap, mathenv) or redExec, (notify, timeout).

Details

Numbers to be factorized may also be provided as character.

Arguments vec, split, drop.blank.lines and use.names are not allowed in dots (...).

Switches and other settings:

1. **complex**: Default: off. This switch must be on in order for factorization to find complex roots. In certain cases using imaginary coefficients will find complex roots but no all. For example, factorizing `x^3 + i*x^2 + x + i` with complex off returns the roots `(x-i)` and `(x+1)^2` while `x^2 + 2*i*x - 1` returns the expression itself.
2. **ifactor**: Default: off. If a polynomial has a numeric factor turning *ifactor* on will cause the numeric factor itself to be factorized, provided the switch factor is not on.
3. **trfac**: Default: off. When on, information on the algorithm used will be displayed provided the factorization is complex enough, for example for the polynomial

   ```
   p1:=(x-1)*(x-3)*(x-7)*(x-1)*(x+y**2)
   ```

   it will be displayed but not for

   ```
   15*x^2*y -3*x**2*y**2 + 6*x*y**3
   ```
4. **trallfac**: Default: off. This causes more detailed information to be displayed

Performance: factorize uses red2Rlist to convert the REDUCE lists produced by *factorize* to R lists. Since both LaTex and REDUCE lists use braces, conversion must be performed on plain output. If LaTeX output is requested this is performed after factorization by R2latex which means there are two calls to REDUCE. However, the bulk of the time spent is due to REDUCE'S *factorize* which can be computationally expensive.

Value

The result of factoring for a single number or polynomial element is a two-level list where each sublist containing a factor and exponent pair. For example:

```
80: factorize(x**4 - 2*x**3*y + 2*x*y**3 - y**4);
    { {x + y, 1}, {x - y, 3} }
```

If multiple numbers/polynomials are specified in x a list of such two-level lists.

If the input is prime, there is only one pairlist consisting of the input itself and an exponent of 1.

If input is a rational or decimal number, or a polynomial with such coefficients, a fraction is factored out to make the coefficients integers. This fraction is returned as the first element of the list. For example:

```
82: factorize((2.5*x)**2 - (7*y/2)**2);
    { 1/4, {5*x + 7*y, 1}, {5*x - 7*y, 1} }
83: factorize(2310.1);
    { 1/10, {13, 1}, {1777, 1} }
```

If outfmt is not "latex" numbers are numeric or complex. Polynomial factors involving variables are expressions if outfmt="expr" or character if outfmt="text".

If outfmt="latex" then all values are character. In addition, instead of lists of factor/exponent pairs, the exponents are applied if not equal to one, e.g. if the input is

```
1536,
    8*x**5 + 60*x**4 + 214*x**3 + 605*x**2 + 1200*x + 1000,
    y^2+2*y+1
```

then the REDUCE output is

```
{{{2, 9}, {3, 1}},{{x**2+8, 1},{2*x+5, 2}},{{y+1, 2}}}
```

and the output with outfmt="latex" is

```
 list(c("2^9", "3"), c("x^2+8" "(2x+5)^3"), c("(y+1)^2"))
```

If eol=TRUE then the value would be

```
 list(c("2^9\\", "3"), c("x^2+8\\" "(2x+5)^3"), c("(y+1)^2"))
```

Conversion of numbers to numeric or complex can handle arbitrary mathematical expressions produced by REDUCE involving products, quotients, exponents, sums, differences and parentheses.

Author(s)

Martin Gregory

See Also

redLatex and R2latex

Examples

```
s1 <- redStart()
## can only run code if session was successfully started
if (is.numeric(s1)) {

    writeLines("=== Example 1: numbers ===")
    str(factorize(s1, c(2310, 2310.1, 49/2)))

    writeLines("=== Example 2:polynomial with complex coefficients ===")
    str(factorize(s1, "a^2-(i*b)^2", switches=c(on="complex")))

    writeLines("=== Example 3: numbers and polynomials input using list ===")
    a <- list("x**2 + (2/5)*x - 1", 1536,
              "x**3 + x**2*y**2 - 10*x**2 - 10*x*y**2 + 21*x + 21*y**2")
    str(factorize(s1, a))

    redClose(s1);
 }
```

---

odesolve *function to solve ordinary differential equations using the REDUCE* odesolve *operator*

Description

odesolve generates code for the REDUCE *odesolve* operator, executes it and converts the result to appropriate R objects if possible.

Usage

```
odesolve(id, eqn, depvar=character(0), indvar=character(0),
       icond=character(0), bcond=character(0), opt=character(0),
       outfmt="text", vec=FALSE, eol=TRUE,
       switches=character(0), verbose=FALSE, ...)
```

Arguments

id	the session identifier returned by redStart. Required with no default
eqn	Character. A single equation to be solved. If an equation is reduced the =0 may be omitted. Required. No default.
depvar	A single character string specifying the dependent variable. Optional. No default. If not specified, REDUCE tries to determine it from the equation.
indvar	A single character string specifying the independent variable. Optional. No default. If not specified, REDUCE tries to determine it from the equation.

icond	A character vector specifying initial conditions. Optional. See details.
bcond	A character vector specifying boundary conditions. Optional. See details.
opt	A character vector of options to pass to *odesolve*. Optional. See details.
outfmt	Character. "text" or "latex". Controls whether the solution and, if present, the general solution is output as plain text (default) or LaTeX.
vec	Logical. Only relevant if latex is TRUE. If TRUE a list is first collapsed to a vector. Default: FALSE
eol	Logical. Only relevant if latex is TRUE. If TRUE append the LaTeX line end command to all but the last element of each character vector with more than one element. Default: TRUE
switches	a character vector of switches to be applied while *odesolve* is executing. Note that neither ECHO, NAT nor FANCY are allowed. See Details below. Optional.
verbose	Logical. Controls whether informational message are printed. Default is FALSE.
...	Arguments which can be passed to `R2latex`, `ltx.append.eol` (eolch) or `redLatex` and `redExec` (use.names, usermap, mathenv, notify, timeout). Do not pass the `redExec` `split` option as otherwise the entire REDUCE raw output will be converted to LaTeX.

Details

odesolve is a REDUCE operator for solving differential equations, both linear and non-linear.

If you specify dependent or independent variables which do not appear in the equation, reduce will interpret these as options and produce an error message.

Conditions:

For convenience we will use `x` and `y` to refer to the independent and dependent variables, respectively. Conditions are equations with either `y`, `x`, or a derivative of `y` on the left and with an expression on the right.

Initial conditions may be specified as a single, comma-separated character string of equations. The first item specifies the value of `x`, and subsequent items the values for `y` or derivatives thereof, for example:

```
icond="x=x0, y=y0, df(y,x)=y1"
```

They may also be specified as a vector with one element for each equation:

```
icond=c("x=x0", "y=y0", "df(y,x)=y1")
```

Boundary conditions are specified as a sequence of paired equations, the first element of which specifies the `x` value while the second is a value of `y` or one of its derivatives. As for initial conditions, the boundary conditions may be specified as a single, comma-separated character string or as a vector, for example

```
bcond="x=x0, y=y0, x=x1, y=y1, x=x2, df(y,x)=y2"
bcond=c("x=x0", "y=y0", "x=x1", "y=y1", "x=x2", "df(y,x)=y2")
```

Note that for boundary conditions there must be an even number of equations.

Options:

A series of switches are available to control how *odesolve* operates. For each switch there is a corresponding option which applies the effect of the switch for the duration of execution as follows:

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

If the solution contains either of the operators *roots_of_unity* or *plus_or_minus*, these can be expanded by including the *expand* option. In this case the solution list will most likely have more elements.

If the option *basis* is used to generate a set of basis solutions for a linear ODE, the solution consists of a list of two elements. The first is a list of the basis equations and the second is a specific solution. If conditions are applied, *basis* is ignored.

REDUCE may produce a parametric solution consisting of a list of three elements. The first and second are expressions of an independent parameter for the dependent and independent variables, respectively. The third names the parameter. There is no option for requesting this.

If the option *trode* is used in conjunction with conditions, the general solution is also provided in the `gen.solution` slot, even if there no specific solution satisfying any conditions specified. The *trode* option also provides information about what method was used to solve the equation.

Note that some combinations of the explicit, implicit, full and basis options for some equations may produce the following error message:

```
+++ Error attempt to take cdr of an atom: !~indefinite!-value!~
```

See the REDUCE user-contributed packages manual for further details of options and of the methods used for solving equations.

Switches:

The `switches` argument takes input using the format defined for `swset`, i.e. a character vector with two named elements, on and off, each of which is a comma separated string of switches. These switches are set prior to execution of odesolve and restored to their previous values afterwards. This argument allows for setting switches other than those set by the `opt` argument.

Note that some switches can affect whether *odesolve* finds a solution. As an example, consider the Lagrange equation

```
y = 2x*df(y,x) - a*df(y,x)^3
```

If *DIV* and *INTSTR* are off and *ALLFAC* is on, which are the defaults for these switches, no solution is found. Toggling these switches does produce a solution.

Note also that the **redcas** `odesolve` turns off the NAT switch as otherwise parsing the results would be very difficult. `odesolve` will ensure that *NAT* is off for the duration of its execution. Turning on the *NAT* switch using the `switches` parameter is ignored. If verbose is true, a message is printed.

Dependence:

If you do not declare the dependent and independent variables, *odesolve* will do so for you by issuing a *depend* call. Note that this dependence is not removed when *odesolve* exits. In this case REDUCE generates a relevant message.

LaTeX:

Since REDUCE *odesolve* returns a list, `R2latex` is used to convert the plain text to LaTeX. The plain text output of an `odesolve` call can also be converted using a direct call to `R2latex`.

Value

`odesolve` returns an object of class `"redcas.ode"` which contains the following elements:

ode the ordinary differential equation.

solution a list containing the solution. The exact structure depends on the odesolve options supplied. See above.

gen.solution list. If the *trode* option and conditions were specified, this slot will contain the general solution.

msg character. Messages return by REDUCE.

vars character. The dependent and independent variables passed to *odesolve*

initial character. The initial conditions passed to *odesolve*

boundary character. The boundary conditions passed to *odesolve*

options character. The options passed.

switches character. The vector of switches passed to *odesolve*

odeCall character. The full odesolve call passed to REDUCE.

raw character. The raw output returned by REDUCE.

The `redcas.ode` object may be printed using the `print` method.

Author(s)

martin gregory

See Also

`print` to print a `redcas.ode` object, REDUCE user contributed packages documentation for details of the REDUCE *odesolve* operator and `redStart` for creating a REDUCE session.

Examples

```
## Open a CSL session:
s1 <- redStart()

## can only run code if session was successfully started
if (is.numeric(s1)) {

    ## solve a Lagrange equation with default switch settings
    lagrange1 <- odesolve(s1, "y = 2x*df(y,x) - a*df(y,x)^3", "y", "x")
    print(lagrange1)

    writeLines("\n\n")

    ## now with div and intstr on, allfac off
    lagrange2 <- odesolve(s1, "y = 2x*df(y,x) - a*df(y,x)^3", "y", "x",
                     switches=c(on="div,intstr", off="allfac"))
    print(lagrange2)

    writeLines("\n\n")

    ## check that the switches have been return to original state
    chk <- redExec(s1, c('swlist:={div, intstr, allfac};',
                         'for each el in swlist do write el, ": ", swget(el);'))
    writeLines(chk$out)

   ## close session:
   redClose(s1)
}
```

`print` *redcas methods for function* `print`

Description

Print methods for redcas.solve and redcas.ode objects

Usage

```
## S4 method for signature 'redcas.solve'
print(x, ...)

## S4 method for signature 'redcas.ode'
print(x, ...)
```

Arguments

`x`	an object of class `redcas.solve` or `redcas.ode`.
`...`	other arguments passed to print.

Details

`print.redcas.solve` displays the equations, solutions if any and the list of unknowns and switches. The solutions displayed are those returned from REDUCE without any transformation to R objects.

The layout depends on the length of the individual solution values and the `width` option. If there is sufficient space, the solutions for each unknown are printed in separate columns on the same line, for example with a width of 80, the output of the example below will be:

```
Equations:
  x+y+z = 0
  x^2 + y^2 + z^2 = 9
  x^2 + y^2 = z^2

Number of solutions: 4

Solutions:
               x               y               z
   2.12132034356               0  -2.12132034356
  -2.12132034356               0   2.12132034356
               0   2.12132034356  -2.12132034356
               0  -2.12132034356   2.12132034356

Unknowns: x,y,z
Switches:  rounded
```

If all values for a solution with not fit on a line, they are printed below each other. With a width of 30, the solutions section would be

```
Solution 1:
x:    2.12132034356
y:                0
z:   -2.12132034356

Solution 2:
x:   -2.12132034356
y:                0
z:    2.12132034356

Solution 3:
x:                0
y:    2.12132034356
z:   -2.12132034356

Solution 4:
x:                0
y:   -2.12132034356
z:    2.12132034356
```

`print.redcas.ode` always displays the ODE, the solution or "None" if no solution is found, the dependent and independent variables and the exact odesolve call passed to REDUCE. If there are

messages from the REDUCE odesolve operator these are also displayed. If initial conditions, boundary conditions, options or switches were specified in the call to odesolve these will also be displayed. print.redcas.ode does not display the raw slot.

Value

Both print.redcas.solve and print.redcas.ode are called for their side-effect and returns NULL.

Examples

```
## start the session
r0 <- redStart()

## can only run code if session was successfully started
if (is.numeric(r0)) {
   rsobj <- redSolve(id=r0,
       eqns=c("x+y+z = 0", "x^2 + y^2 + z^2 = 9", "x^2 + y^2 = z^2"),
       unknowns=c("x", "y", "z"),
       switch=c("on rounded;") )
   print(rsobj)

   writeLines("\n\n")

   ode <- odesolve(id=r0, "df(y,x)=exp(x)", "y", "x")
   print(ode)

   redClose(r0)
}
```

R2latex *functions to convert an arbitrary R list or character vector to LaTeX.*

Description

R2latex converts the contents of a list to LaTeX either maintaining the list structure or collapsing it to a vector. The possibly nested list may contain objects of type numeric, character, expression, data frame. Objects of other types are returned unchanged.

ltx.append.eol appends the LaTeX end of line command to all but the last element of a character vector for use in align, array, eqnarray and similar environments. If passed a list, ltx.append.eol processes all character vectors in the list. May be called by R2latex.

Usage

```
R2latex(id, x, vec=FALSE, eol=FALSE, eolch="\\\\", switches=character(0),
        verbose=FALSE, ...)
ltx.append.eol(x, eolch="\\\\", ...)
```

Arguments

id	the session identifier returned by redStart. Required with no default
x	a vector of type numeric, character, expression, data frame or a possibly nested list of such types. Required. No default.
vec	Logical. If TRUE a list is collapsed to a vector prior to conversion. Default: FALSE
eol	Logical. If TRUE append the LaTeX line end command to all but the last element of each character vector with more than one element. Default: FALSE
eolch	Character. The line end command, default \\
switches	a character vector of switches to be applied while R2latex is executing. Note that neither ECHO, FANCY nor NAT are allowed. See swset for details of the format. Optional.
verbose	Logical. Controls whether informational message are printed. Default is FALSE.
...	Arguments which can be passed to ltx.append.eol, redLatex or redExec (use.names, usermap, mathenv, notify, timeout). Do not pass the redExec split option as otherwise the entire REDUCE raw output will be converted to LaTeX.

Details

R2latex converts character, numeric or expression vectors, data frames and lists, possibly nested, containing these types. If a list contains other objects, these are returned unchanged. It calls redLatex to perform the conversion and optionally calls ltx.append.eol to append line end commands for use in multi-line math environments.

For a list this means recursing down the list and calling redLatex for each character vector. In turn this means sending commands to the REDUCE session for each vector which could have a significant effect on performance. Testing a list of depth 3 with 12 nodes showed a seven-fold increase in elapsed time compared to using the default vec=TRUE while a depth 4 list with 16 nodes showed a ten-fold increase.

If input contains a non-supported type and verbose is TRUE, R2latex writes an informational message.

Value

Depending on the value of vec, R2latex returns a character vector (TRUE) or a list with the same structure as the input (FALSE).

ltx.append.eol returns an object of the same type and structure as the input with the end of line command appended to all but the last element.

Numeric values are returned unchanged but are subject to the ROUNDED switch. If this is off they are returned as quotients, otherwise as decimals. Note that passing some expressions may result in a number being returned, for example exp(i*pi).

Author(s)

martin gregory

See Also

redLatex for details of conversion to LaTeX and the permitted arguments.

Examples

```
## Open a CSL session:
s1 <- redStart()

## can only run code if session was successfully started
if (is.numeric(s1)) {
   ## solving the ODE df(y(x),x,2) - 2/x^2*y(x) = 7x^4 + 3*x^3 with the
   ## basis option produces the solution:
   sol <- list(list("x**(-1)", "x**2"), "1/4*x**6 + 1/6*x**5")

   writeLines(c("using the default options produces a list with the same structure ",
                "as the input and no end of line command."))
   ltx1 <- R2latex(s1, sol)
   utils::str(ltx1)
   writeLines("\n\n")

   writeLines(c("using eol=TRUE produces a list with the same structure as the",
                "input with one end of line command in the first sublist."))
   ltx2 <- R2latex(s1, sol, eol=TRUE)
   utils::str(ltx2)
   writeLines("\n\n")

   writeLines(c("using vec=TRUE and eol=TRUE produces a vector with two",
                "end of line commands. "))
   ltx3 <- R2latex(s1, sol, vec=TRUE, eol=TRUE)
   utils::str(ltx3)

   ## close session:
   redClose(s1)
}
```

---

red2Rarray *Converting REDUCE arrays to R*

Description

Convert REDUCE arrays to R arrays containing a character version of the originals.

Usage

```
red2Rarray(id, x, outfmt="text", switches=character(0), verbose=FALSE)
```

Arguments

id	session identification returned by redStart. Required.
x	Character. A vector naming the arrays to convert. The arrays must already exist in the current REDUCE session. Items named in the vector which do not exist or are not arrays are ignored. Elements which are not valid REDUCE names cause an error. No default.
outfmt	Character. "text" or "latex". Controls whether the solution and, if present, the general solution is output as plain text (default) or LaTeX.
switches	a character vector with one or both of the following named elements **on** a comma-separated string of switches which should be on; **off** a comma-separated string of switches which should be off. specifying the desired state of switches for the duration of execution of red2Rarray. Note that the switches ECHO, NAT, NERO and FANCY are not allowed.
verbose	Logical. Controls whether an informational message about toggling the *NAT* and *NERO* switches is printed.

Details

REDUCE array dimensions are indexed starting with zero, so an array declaration *array a(3,4);* will result in an R array with dimensions c(4,5).

A further difference is that REDUCE arrays are global in scope, no matter where they are defined although this has no effect on the converted array.

red2Rarray ensures that the *NAT* switch is off if outfmt is "text" as otherwise the results will not be parsed correctly. It also unconditionally ensures that the NERO switch is off as otherwise any zeros in the REDUCE array would not be returned and the conversion to an R array would fail. After conversion, both NAT and NERO are restored to their previous state.

Specifying either *NAT* or *NERO* using the switches parameter will be ignored and an appropriate message issued.

Value

red2Rarray returns a list with one element for each array named in the input. Since REDUCE arrays may contain symbolic expressions, red2Rarray always returns characters.

Author(s)

Martin Gregory

See Also

redExec for details of REDUCE output, swset for controlling switches and red2Rfn for converting the array elements to R functions.

Examples

```
## start the session
s1 <- redStart()

## can only run code if session was successfully started
if (is.numeric(s1)) {
   ## create the array
   redcode <- c("array g(2,2) ;",
  "operator x;",
  "g(0,0) := -u^(-2);",
  "g(1,1) := (u*x(3))^2;",
  "g(2,2) := g(1,1) * (sin(x(1)))^2;",
  "on nero ;"
  )

   o2 <- redExec(s1, redcode)

   ## convert the array
   print(gltx <- red2Rarray(s1, "g", switches=c(off="exp"), outfmt="latex"))

   ## close the session
   redClose(s1)
}
```

`red2Rfn` *create R functions from REDUCE expressions*

Description

Convert REDUCE expressions to R functions which evaluate the expression in R. The expressions are the output of any **redcas** function which executes REDUCE code, for example, `redFunc`, `redSolve` or `odesolve`.

Usage

```
red2Rfn(x, name=character(0))
```

Arguments

`x`	A character vector containing the REDUCE expressions, for example the `solutions` slot of `redSolve` or the `solution` slot of `odesolve`. Required.
`name`	A character vector of corresponding function names. Optional. If not specified, names must be included in `x` or the vector must be named unless `x` contains only a single element. See details.

Details

Expressions returned from REDUCE functions may be character vectors or lists. Depending on the function, lists can be nested. If input to be processed is a list, you must either use `lapply` or first convert it to a vector using `unlist`.

Names: Names can be assigned in three ways in the following order of priority:

1. using the `name` argument;
2. using names embedded in the `x` argument, for example as returned by `redSolve`
   ```
   red2Rfn(c("e1=x+y", "e2=x-y"))
   ```
3. using `names(x)` if these are present

If none of the three ways provide names `red2Rfn` returns FALSE. An exception is made for `x` containing only a single element because in this case `red2Rfn` returns a single function instead of a list.

If relying on `x` containing embedded names, all elements must contain an embedded name.

The `solutions` slot of `redSolve` is a list with one named vector per solution. To convert these results to functions, first use `unlist`. If this will result in a vector with duplicate names, `red2Rfn` modifies the duplicates by adding an integer suffix which makes the duplicate unique.

The `solution` slot of `odesolve` is a possibly nested list of equations which may have the form `variable = expression` or simply of an expression without a name. If there are unnamed expression, for example when a parametric solution is returned, you must either use the `name` argument or insert names into the expressions before calling `red2Rfn`.

Function arguments: When parsing, `red2Rfn` assumes that names follow REDUCE conventions, i.e. the regular expression `[a-zA-Z][a-zA-Z0-9_]*`. If input to `red2Rfn` comes from REDUCE this will generally be the case. There are, however, some caveats:

1. REDUCE allows characters not matching the regular expression above in names if the characters are escaped, e.g. `!-a` is a valid name.
2. some REDUCE mathematical functions may differ in their arguments from R: in both languages, `log` is the natural logarithm but in R it has a second argument to specify the base. This defaults to `exp(1)` so in most cases this is not an issue.
3. core R does not have the trigonometric functions sec, csc, cot, their inverses, or their hyperbolic counterparts although these are provided by the package **pracma**.
   These situations are not handled by `red2Rfn` in the current version.

Value

if `x` contains only a single expression, `red2Rfn` returns the function, otherwise a named list of the same length as `x`, each of which is the function for calculating the corresponding expression.

Author(s)

martin gregory

See Also

`redSolve`, `odesolve`, `redFunc`

Examples

```
## a single expression as input returns the function directly
f1 <- red2Rfn("y=(exp(x) - 1)/(exp(x) + x^2)")
curve(f1, 0, 20)

## vector input produces a list and renames the second y to y0
eqns <- c("y=(12*a2 + 12*a1*x + x**4)/12",
          "y=(x+1)^2",
          "q=p^2 + 1 + exp(-r)",
          "p=cos(-r)")
f2 <- red2Rfn(eqns)
names(f2)
f2$y0(5)
```

`red2Rlist` *Extracting REDUCE lists and functions from* `redExec` *output*

Description

These functions support converting REDUCE lists contained in character vectors to R lists and extracting REDUCE function calls.

Usage

```
red2Rlist(x, tochar=FALSE, verbose=FALSE)
redlsExtract(x, level=1)
fnMatch(x, subch=";")
dlmMatch(x, delim="{", verbose=FALSE)
sublist2char(l)
```

Arguments

`x`	Character. A vector containing REDUCE output which possibly contains REDUCE lists. No default.
`tochar`	Logical. Controls whether the function `sublist2char` is applied to the converted list. See Details.
`verbose`	Logical. Controls whether informational messages are printed. Default is not to print.
`level`	Integer. The level of nesting from which to return lists. All lists within this level are returned. Default is 1, the top level.
`subch`	Character. The character to be used when substituting for commas in function calls. Default is the semi-colon. See details.
`delim`	Character. The opening delimiter of the pair to use for matching. Default is "{".
`l`	List. An arbitrary R list

Details

REDUCE output may contain lists as many functions return lists. Some examples from the core REDUCE are *changevar, coeff, decompose, factorize, i_solve, length, mateigen, num_min, odesolve, r_solve, rad2dms, solve* and *trigfactorize*. Input to functions may also be lists. Lists can also be nested: *solve*, for example, returns a list containing a list for each solution. User contributed packages may also return lists. A REDUCE list is a comma separated sequence of expressions enclosed in braces, for example:

```
{a, b, c}
{{a}, {{b, c=Binomial(n,m)}, d-f}, e}
```

While `redSolve`, for example, parses the REDUCE list and returns R lists, `redExec` returns only character vectors. `red2Rlist` can be applied to the output of `redExec` to create an R list.

`red2Rlist` calls `redlsExtract` to extract the REDUCE lists from `x` which may contain output other than lists. Because lists may contain function calls and because list elements and function arguments both use comma as a separator, `red2Rlist` also calls `fnMatch` to identify functions and, by temporarily replacing the commas in the function, avoids splitting function arguments into list elements. Finally, if the argument `tochar` is TRUE, `red2Rlist` calls `sublist2char` to convert sublists containing only elements of type character to character vectors in order to facilitate displayed output. This option is needed if the list is to be processed by `ltx.append.eol` as otherwise the latter has no effect.

`redlsExtract` calls `dlmMatch` to determine the coordinates (starting row/column, ending row/column) of lists of the specified level and uses this information to extract the lists.

`fnMatch` calls `dlmMatch` to determine the coordinates of function calls and uses this to extract the calls.

`dlmMatch` finds matching delimiter pairs (brace, bracket, parenthesis) to enable extracting the contents within the matching delimiters.

Value

`red2Rlist` returns a list with a structure matching the REDUCE list(s). Elements of the lowest level lists are characters to avoid accidental evaluation due to matching variables in the environment. If `x` contains a single REDUCE list, `red2Rlist` returns a single R list containing character representations of the REDUCE list, otherwise a list containing the converted lists. These can be converted to expressions and evaluated as required.

Note: to convert the output of `red2Rlist` to LaTeX use `R2latex`.

`redlsExtract` returns a character vector with one element for each list of the specified level present in `x`. The returned list's syntax is the REDUCE format. Any non-list content is ignored. Note that this argument is currently not available for `red2Rlist`.

`fnMatch` returns a list with two elements

comma a character vector of the function calls

semicolon a character vector of the function calls with commas replaced by `subch`

These can be used in `gsub` with `fixed=TRUE` to replace or restore the commas.

`dlmMatch` returns a data frame with the following columns

row0 element index of delimiter

col0 column of delimiter in row0

row1 element index of matching delimiter

col1 column of matching delimiter in row1

paren opening delimiter character

level nesting level, 1 being the top level

`sublist2char` returns a list with the same structure as the input but with character vectors replacing sublists containing only elements of type character.

Note

These functions use only R code and do not require an active REDUCE session. They can also be used on output files created directly by REDUCE.

Author(s)

Martin Gregory

See Also

`redExec` for details of REDUCE output.

Examples

```
## a simple list
str(red2Rlist("{a, b, c}"))

## simulated output from REDUCE with two lists
redout <- c("Some output from REDUCE with a simple list",
            "{a,b,c}",
            "and a more complex example including a function call",
            "{{a}, {{b, c=Binomial(n,m)}, d-f}, e}")

## create the R version of the REDUCE lists
str(red2Rlist(redout,verbose=FALSE))

## within red2Rlist the following functions are called
## get the coordinates of the lists
dlmMatch(redout)

## extract the lists
redlsExtract(redout)

## extract the functions
as.data.frame(fnMatch(redout))
```

redcas.ode *Class* "redcas.ode"

Description

Contains the inputs and outputs of a call to the function odesolve which executes the REDUCE *odesolve* operator to solve ordinary differential equations.

Objects from the Class

Objects are created by calls to the function odesolve and can be used as input to the print method.

Slots

.Data: Object of class "list" containing the actual data.

ode: Object of class "character" containing the equation to be solved.

solution: Object of class "list" containing the solution of the ODE. In general there will be only a single element. Multiple elements arise when a parametric solution is provided or when certain options are specified. This is effectively a copy of the list returned by the REDUCE *odesolve* operator.

gen.solution: Object of class "list" containing a general solution. These arise when initial or boundary conditions are specified and the trode option is used. In this case REDUCE provides both the solution including the conditions and the general one. It is also possible that no solution satisfies the initial conditions but a general solution exists.

msg: Object of class "character". Any messages returned by the REDUCE *odesolve* operator.

vars: Object of class "character". The dependent and independent variables

icond: character. The initial conditions passed to *odesolve*

bcond: character. The boundary conditions passed to *odesolve*

opt: character. The options passed to *odesolve*.

switches: Object of class "character". Any REDUCE switches for the duration of executing the REDUCE *odesolve* function.

odeCall: character. The full odesolve call passed to REDUCE.

raw: character. The raw output returned by REDUCE.

Extends

Class "list", from data part. Class "vector", by class "list", distance 2.

Methods

A print method, print, is defined.

Author(s)

Martin Gregory

See Also

odesolve is currently the only function using this class.

Examples

```
showClass("redcas.ode")
```

---

redcas.solve *Class* "redcas.solve"

Description

Contains the inputs and outputs of a call to the function redSolve which executes the REDUCE solve operator to determine solutions to a system of algebraic equations.

Objects from the Class

Objects are created by calls to the function redSolve and can be used as input to the print method.

Slots

.Data: Object of class "list" containing the actual data.

solutions: Object of class "list" containing a character vector for each solution of the system of equations. The character vectors have a named element for each unknown with the name being the unknown and the value being the value of the unknown for the solution. This is essentially a copy of the list returned by the REDUCE solve operator.

rsolutions: Object of class "list" containing a list for each solution converted to appropriate R types where possible. A vector will not suffice because the values of the unknowns may be of different types: integer, real, complex or expression. Explicit real and complex numbers are always converted. If the REDUCE switch ROUNDED is off, real and complex numbers may be expressions and may result in errors or NaN if conversion were attempted.

nsolutions: Object of class "numeric". The number of solutions

rc: Object of class "numeric". Return code from the redSolve call: 0 is success and 1 failure.

root_of: Object of class "complex". A solution may express one or more unknowns in terms of the roots of another equation. In this case the other equation is enclosed in root_of(). This slot is a complex vector with the real part identifying the solution contain root_of() and the imaginary part the index of the unknown in that solution.

eqns: Object of class "character". The system of equations provided to redSolve

unknowns: Object of class "character". The unknowns provided to redSolve

switches: Object of class "character". Any REDUCE switches set prior to executing the REDUCE solve function.

Extends

Class "list", from data part. Class "vector", by class "list", distance 2.

Methods

A print method, `print`, is defined.

Author(s)

Martin Gregory

See Also

`redSolve` is currently the only function using this class.

Examples

```
showClass("redcas.solve")
```

---

redClose *function to close a REDUCE session and optionally save the log file*

Description

`redClose` closes the connection to the REDUCE session thereby ending the session. If requested the log file is copied from the temporary location to one specified by the `log` argument.

Usage

```
redClose(id, log)
```

Arguments

`id`	the session identifier returned by `redStart`. Required with no default
`log`	the path to the location in which to save the session's log file. Optional.

Details

The session log returned by `redClose` includes the submit block markers set by `redExec`.

In addition to closing the connection, redClose removes the session's entry from the session registry.

Value

`redClose` returns TRUE if the session is closed, FALSE if it does not exist.

Author(s)

martin gregory

See Also

`redStart` for creating a REDUCE session and `showSessions` to display the session registry.

Examples

```
## Open a CSL session:
s1 <- redStart()
## can only run code if session was successfully started
if (is.numeric(s1)) {
   ## show session details:
   print(showSessions())
   ## close session:
   redClose(s1)
}
```

`redCodeDir` *Retrieve the directory containing the REDUCE code which is part of* `redcas`.

Description

`redCodeDir` retrieves the full path to the directory containing the REDUCE code which is part of `redcas`.

Usage

```
redCodeDir()
```

Details

In addition to returning the REDUCE code directory, `redCodeDir` assigns three environment variables:

REDCODEDIR the full path to the directory containing the REDUCE code which is the `reduce` sub-directory of the `redcas` package;

REDCAS_CODE_PATH the full path to the program `redcas.red` which defines REDUCE procedures used by `redcas`;

TMPRINT_PSL_PATH the full path to the program `tmprint-psl.red` which makes a modified version of the REDUCE package TMPRINT which implements LaTeX rendering for the PSL dialect. The change is minor and is designed to ensure that the TeXmacs markup is identical to that of CSL.

You can use the two programs directly in REDUCE. You need only load `redcas.red` as it loads `tmprint-psl.red` if executing in PSL. In order to have `tmprint-psl.red` loaded, you must assign the environment variable `TMPRINT_PSL_PATH` before starting REDUCE.

Value

`redCodeDir` returns the full path to the directory containing the REDUCE code which is part of `redcas`.

Author(s)

martin gregory

Examples

```
redCodeDir()
Sys.getenv("REDCAS_CODE_PATH")
Sys.getenv("TMPRINT_PSL_PATH")
```

`redDf` *Differentiate expressions by one or more variables specifying the degree for each variable.*

Description

Call the REDUCE *df* operator to differentiate one or more expressions by one or more variables specifying the degree for each variable and returning the output as either LaTeX or R functions.

Usage

```
redDf(id, expr, depend=character(0), style=character(0), outfmt="func",
      switches=character(0), name=character(0), verbose=FALSE, ...)
```

Arguments

`id`	the session identifier returned by `redStart`. Required with no default.
`expr`	A character vector containing the arguments for the REDUCE operator DF. Required. Each element of `expr` is string of the form `x, v(1) [,d(1)] [,v2 [,d(2)]]...[,v(n) [,d(n)]]` where `x` is the expression to be differentiated, v(i) are the variables to differentiate by and d(i) is the degree of differentiation by v(i). If d(i) is 1 it may be omitted.
`depend`	A character vector of dependency declarations. This is only relevant if an item in the differentiand depends on one or more of the v(i). Optional. Applies for all elements of `expr`. See details.
`style`	A single character value specifying how the "d" in `dy/dx` in the result are to be written. Values are `partial`, `total` or `indexed`. Applies only to LaTeX output. Optional. Note that in REDUCE, the default value partial. Note also that the same style applies to all elements in `expr`.
`outfmt`	Character. Controls the type of output. Possible values are "func", "latex" or "text" which return an R function, LaTeX or plain text, respectively. Default is "func".

switches	a character vector of switches to be applied while redDf is executing. If outfmt is "func", the NAT switch is not allowed. The ECHO and FANCY switches are never allowed. See swset for details. Optional.
name	A character vector of corresponding function names. Optional. Only used if functions will be returned. See details.
verbose	Logical. Controls whether informational message are printed. Default is FALSE.
...	Arguments which can be passed to redLatex (usermap, mathenv) or redExec, (notify, timeout).

Details

Dependencies: If the differentiand contains variables other than those differentiated by, these variables are treated as constant. For example, in differentiating exp(x)*z w.r.t. x dz/dx is zero and so the result is exp(x)*z. If z actually depends on x the statement

```
depend z,x;
```

declares this dependency and the derivative is then exp(x)*z + exp(x)*dz/dx. The depend argument allows the *depend* declaration be executed before *df* is executed. The declaration is of the form

```
depvar, indvar1 [, indvar2] [... ,indvarN]
```

and a separate element is required for each dependent variable. There order of the elements does not have to match that of expr and can have a different length.

In addition, if an element is preceded by ~, the dependency will be removed after *df* execution. For example,

```
c("y, x, w", "~z, x")
```

will revoke the dependency of z after *df* execution but not that of y.

Switches influencing differentiation:

1. **commutedf/nocommutedf/simpnoncomdf**: Default: off/off/off. These switches control the commutativity of derivation when an expression is differentiated w.r.t. multiple variables. Details can be found in section 7.7.1 of the REDUCE manual. Note that turning on one of the pair nocommutedf/commutedf turns off the other;
2. **expanddf**: Default: off. enables use of the chain rule;
3. **allowdfint**: Default: off. enables use of differentiation under the integral sign if this results in a simplification;
4. **dfint**: Default: off. If both this and *allowdfint* are turned on, differentiation under the integral sign is enabled unconditionally.

Switches affecting display:

If the switch *dfprint* is turned on (it is off by default) then derivatives are displayed using subscripts. In plain output this looks like

```
 2
d f
----  ==>  f
dxdy        x,y
```

but only when the switch *nat* is on.

Global REDUCE variables affecting display: The argument `style` sets the global variable *fancy_print_df* which controls the display of derivatives when `outfmt` is "latex". As with `depend`, prefixing the value with a tilde restores the variable to the the value it had before execution.

Rules:
The operator DF is capable of differentiating mathematical functions provided by REDUCE, including those in the special functions (SPECFN) package. For user-defined functions the LET declaration can be used to define rules for relations among functions and variables, including derivatives. Here is an example of defining the derivative of a function *p* without explicitly defining the function:

```
1: operator p, q ;
2: for all x let df(p(x),x) = q(x)^2 ;
3: df(p(z), z) ;

    2
q(z)

4: df(3*p(f*g), g);

         2 3*q(f*g) *f
```

The syntax *for all x ...* may be abbreviated as

```
let df(p(~x),x) = q(x)^2 ;
```

Details can be found in section 11.3 of the user manual.

Names:
If functions are to be returned (`outfmt` is "func"), names for the output of `red2Rfn` are required. If these are not specified using the `name` argument, names are assigned using the pattern "fn*i*" where *i* is the element number of `expr`. For elements with errors or warnings, the pattern "msg*i*" is used.

Handling errors and warnings:
If `expr` has multiple elements and REDUCE produces errors or warnings for some elements, the output for those elements will be character and will include the messages. `redDf` continues to process subsequent elements after an error or warning so elements without errors or warnings are not affected by the errors or warnings.

Differences between R and REDUCE:
`redDf` assumes that names follow REDUCE conventions, i.e. the regular expression `[a-zA-Z][a-zA-Z0-9_]*`. There are some caveats:

1. REDUCE allows characters not matching the regular expression above in names if the characters are escaped, e.g. *!-a* is a valid name.
2. some REDUCE mathematical functions may differ in their arguments from R: in both languages, `log` is the natural logarithm and has only one argument but in R it has a second argument to specify the base.
3. core R does not have the trigonometric functions sec, csc, cot, their inverses, or their hyperbolic counterparts although these are provided by the package **pracma**.
   These situations are not handled by `redDf` in the current version.

Value

If `expr` contains only a single expression, `redDf` returns

- a character vector with one element if `outfmt` is "text" and NAT is off;
- a character vector with one or more elements if `outfmt` is "text" and NAT is on;
- a character vector with one element if `outfmt` is "latex";
- if `outfmt` is "func", a function for calculating the corresponding expression or, if the result can be converted to a number, a number.

If `expr` contains more than one expression, a list of the same length as `expr` containing objects as just defined.

If REDUCE produces an error or warning for an element of `expr`, the corresponding output element is character and contains the error or warning message.

The caveats in the section "Differences between R and REDUCE" above also apply to the output.

Author(s)

martin gregory

See Also

`red2Rfn` which is used to convert REDUCE expressions to R functions.

Examples

```
s1 <- redStart()
## can only run code if session was successfully started
if (is.numeric(s1)) {
    ## differentiate two expressions specifying temporary dependency for
    ## the first and a permanent one for the second. Return character
    writeLines("=== Example 1  ===")
    print(redDf(s1, c("v*cos(w), w", "(a+b)^2, a"),
          depend=c("~v, w", "b, a"), outfmt="text"))

    ## repeat the call without dependencies and return functions. Note
    ## that the result for the first expression is different because the
    ## dependency v ~ w is no longer in effect
    writeLines("=== Example 2  ===")
    print(redDf(s1, c("v*cos(w), w", "(a+b)^2, a"), outfmt="func"))

    ## effect of the switch DIV turned on (off is default)
    writeLines("=== Example 3  ===")
    print(redDf(s1, "log(x)*(x+y)^2, y,x", switches=(c(on="div"))))


    redClose(s1);
 }
```

redDropOut *Removes certain elements from the REDUCE output.*

Description

Removes blank lines or the submit block end marker from the REDUCE output.

Usage

```
redDropOut(x, what)
```

Arguments

x	a character vector containing REDUCE output
what	a character string describing what to drop. what='blank' removes blank lines, i.e. those matching regular expression "^$" while what='marker' removes the end of block marker

Details

redDropOut is called by redExec if the option drop.blank.lines is TRUE, so you generally do not need to call it explicitly. If you have forgotten to set the option, you can use this function to remove blank lines.

The purpose of what='marker' is to remove the end of block marker described in the details section of redExec. This may be useful if you need to process a REDUCE transcript not produced by redcas.

When removing blank lines and the output is not LaTeX or the switch *NAT* is on, output may be difficult to read and difficult to parse.

Value

The input vector with the corresponding elements removed.

Author(s)

Martin Gregory

See Also

redExec

Examples

```
x <- c("first", "", "last")
## this will return elements 1 and 3:
redDropOut(x, "blank")
```

redExec *execute REDUCE commands from R*

Description

sends a vector of commands to the REDUCE process for execution and retrieves and formats the output.

Usage

```
redExec(id, x, split = TRUE, drop.blank.lines = FALSE, notify=0, timeout=0)
```

Arguments

id	session identification returned by redStart. Required.
x	A character vector contain either the REDUCE commands to execute, the names of REDUCE command files or a mixture of both. Required.
split	Logical. Should the output be split into separate vectors for echoed commands and for output. Default TRUE.
drop.blank.lines	Logical. Should blank lines be removed from the output? Default FALSE.
notify	while waiting for the REDUCE commands to complete, write a note to the console every notify seconds. Default is zero which suppresses the message.
timeout	number of seconds after which to terminate the function if it is still waiting for output from the REDUCE process. Default is 0 which will never initiate termination.

Details

Before submitting the vector x, redExec modifies it as follows:

- inserts a ";" terminator at the start of x. This is required to ensure that the first command in the next submit block has a number. Numbers are required to distinguish commands from command output;
- replaces any file names in x with their contents. This means that the contents of the files are executed directly and not via an *IN* or *IN_TEX* command. Files are specified by preceding the filename with file:. A terminator is not required and, if present, is ignored;
- appends a command to x in order to set an end of block marker. This is required so that redExec can wait until all the commands submitted have been executed. This is especially important for commands which run for a long time.

When reading the output from the submitted commands, redExec removes the end of block command and the marker it sets.

Care must be taken when using the drop.blank.lines option. In particular, if using the REDUCE *NAT* switch, which is the default, and output is not formatted for LaTeX, it may be difficult to read the output.

If REDUCE prompts for input during a submit block, `redExec` will write an informative message, return the output and close the session, writing the log to the transcript. This most likely happens if you use a variable as an operator but did not declare it as such. One way to avoid this is to turn off the switch INT which causes REDUCE to declare any undeclared operators it encounters. If you do this, be sure to turn on INT before you call `redClose` as otherwise the REDUCE session will hang. You should use the code *on int;;*. Note that both semicolons are required. Since REDUCE is run via a pipe, INT is on by default.

If the last statement (element of `x`) is missing a terminator (semi-colon or dollar sign) `redExec` checks whether the last non-blank, non-comment element of the input vector has a terminator. If not it stops with an appropriate message and returns FALSE. This reduces the likelihood that `redExec` fails to return.

We recommend ensuring that ECHO is on during execution. When OFF, some REDUCE code may place output on numbered lines resulting in a missing first element of output. `redStart` defaults to setting ECHO on.

Value

Like R, the REDUCE log contains both commands and outputs. If split is TRUE a list containing the following elements is returned:

out the output of the executed commands;

cmd the executed commands

raw the interspersed commands and output.

If split is FALSE, `redExec` returns a character vector with the output from the REDUCE commands.

Author(s)

Martin Gregory

See Also

`redStart` for starting a REDUCE session, `redSplitOut` and `redDropOut` for post-processing the output if the `split` or `drop.blank.lines` are not used. `redin` can simplify the entry of REDUCE command at the R console or in an R program.

Examples

```
s1 <- redStart()

## can only run code if session was successfully started
if (is.numeric(s1)) {
   ## submit with neither split nor drop.blank.lines
   x1 <- c("solve((x+1)^2, x);",
    "b:=factorize(45056);",
    "operator v; ",
    "la:=-m/(v(2)^2 + v(3)^2)^(1/2) ;")
   writeLines("## submit with neither split nor drop.blank.lines")
   writeLines(y1 <- redExec(s1, x1, split=FALSE))
```

```
  ## submit with split=TRUE and drop.blank.lines=TRUE
  writeLines("\n## submit with split=TRUE and drop.blank.lines=TRUE")
  writeLines(redExec(s1, x1, drop.blank.lines=TRUE)[["out"]])

  ## submit with file name in input
  tfile <- tempfile('reduce')
  writeLines(c(paste0('write "code from ', tfile, '";'),
 "b:=factorize(54056);"),
      tfile)
  x3 <- c("la:=-m/(v(2)^2 + v(3)^2)^(1/2) ;",
   paste0("file:",tfile))
  y3 <- redExec(s1, x3, split=TRUE)
  writeLines("\n## submit with file name in input")
  writeLines("## commands")
  writeLines(y3$cmd)
  writeLines("\n## output")
  writeLines(y3$out)
  redClose(s1)
}
```

---

redExpand *Insert the content of a file into a vector*

Description

A vector of REDUCE commands may contain the name of a file containing REDUCE commands. redExpand returns a vector containing the commands up to the element containing the file name, the contents of the file and the elements following the file name. Multiple files may be specified in the input vector.

Usage

```
redExpand(x)
```

Arguments

x	vector of commands to submit to REDUCE, possibly including names of files of commands to submit.

Details

redExpand is called by redExec before submitting commands so it is generally not necessary to call it explicitly. It can be used if you wish do the expansion yourself. If the input vector elements x_j, x_k, where $j < k$, are file references, then the output vector will consist of

- $x_1 \ldots x_{j-1}$
- the contents of x_j
- $x_{j+1} \ldots x_{k-1}$
- the contents of x_k
- $x_{k+1} \ldots x_n$

Value

a character vector containing the commands to submit with any file names replaced by their contents.

Author(s)

Martin Gregory

See Also

redExec

---

redFunc *execute any REDUCE prefix operator or procedure directly from R*

Description

While **redcas** provides direct interfaces to a small subset of REDUCE prefix operators (*df, factorize, solve, odesolve*) and returns appropriate R objects, redFunc can execute any REDUCE prefix operator or user-defined procedure directly returning the results as character. To simplify the text we use the term procedure to cover both procedures and prefix operators.

Usage

```
redFunc(id, func, param, switches=character(0), pkg=character(0), outfmt="text",
        verbose=FALSE, ...)
```

Arguments

id	session identification returned by redStart. Required.
func	A character or expression vector containing the names of the procedures to be executed. Required.
param	List. Each element is a character, numeric or expression vector providing the arguments for the corresponding procedure. Required. If only one procedure is specified, this can be a vector.
switches	a character vector with one or both of the following named elements **on** a comma-separated string of switches which should be on; **off** a comma-separated string of switches which should be off. specifying the desired state of switches for the duration of execution of all procedures named in func. The ECHO and FANCY switches are not allowed. See swset for details.
pkg	Character. User contributed REDUCE packages to load before executing the procedures named in func. Optional. See details.
outfmt	Character. "text" or "latex". Controls whether the result is returned as plain text (default) or LaTeX.

verbose	Logical. Controls whether an informational messages are printed. Default is FALSE.
...	Arguments which can be passed to redLatex or redExec. Optional. The arguments split, drop.blank.lines and use.names may not be passed.

Details

Making procedures available:

There are three sources of REDUCE procedures: those defined in the core system which are always available, those defined in user contributed packages and those defined by the user in-line or in sourced programs.

When calling procedures from some user contributed packages, the package is automatically loaded. For those which do not autoload, the pkg option can be used. Packages ELLIPFN, ODESOLVE and SPECFN autoload for most of their procedures. For further details see the documentation in the User Contributed Packages section of the REDUCE manual.

In-line or sourced procedures must be defined before use.

If a package is not autoloaded, REDUCE responds with a request to declare the procedure as an operator. For example, calling the *setring* procedure from the CALI package does not autoload the package resulting in the following message:

```
REDUCE is waiting for a prompt:
Declare setring operator ?  (Y or N)

REDUCE session log:
...
The session will now be terminated.
```

See redExec for an explanation of why the session is terminated.

Argument checking:

No checking of the procedure names or the type (in REDUCE) or values of the arguments are done - these are left to REDUCE. If errors occur, appropriate messages are written. One exception is the package name "module". While there is no such package, using this name causes REDUCE to hang.

Before execution redFunc checks that the length of func and param are equal, that func is character and that the elements of param are character or numeric. If any of these checks fail, the function stops and returns FALSE.

Checking of the argument switches is as in other functions where these appear.

Specifying procedure arguments:

Variable names, expressions and numbers may be passed as R character type. Numbers may also be passed as numeric if param is specified as a list or if only one procedure is being called and this procedure expects only numeric input.

Lists must be passed as character using the REDUCE syntax:

```
"{element-1, ..., element-N}"
```

where elements may be numeric, character or list.

If a REDUCE procedure argument requires a literal string, i.e. is neither a number, variable nor expression, you must include double quotes in the value either by escaping or by using single quotes for the quoting the argument to `redFunc`. Alternively, an expression may be used.

Passing arguments to redLatex or redExec:

Passing any of the arguments `split`, `drop.blank.lines` and `use.names` will cause redFunc to terminate because specific settings of these arguments are required. Allowed are `usermap`, `mathenv`, `notify` or `timeout`. If specified, these apply to all procedures named in `func`.

Converting output to R objects:

Depending on the type of result, the functions `red2Rarray`, `red2Rfn`, `red2Rlist` or `as.numeric` can be used to convert to corresponding R objects.

Doing these conversions requires `outfmt` to be `text` and to set NAT off using the `switch` argument.

Handling errors and warnings:

If `func` has multiple elements and REDUCE produces errors or warnings occur for some elements, the output for those elements will be character and will include the messages. `redFunc` continues to process subsequent elements after an error or warning so elements without errors or warnings are processed as directed.

Commands and declarations not supported:

Only procedures which may be used in expressions are supported. In particular, passing a command or declaration will result in the REDUCE warning message

```
*** asltx called with 1 instead of 4 arguments
```

followed by an error message similar to the following

```
***** String  invalid as identifier
***** Syntax error: , invalid after a
*****  invalid as operator
*****  invalid as Array declaration
```

Commands include `clear`, `in`, `on`, `off`, `out`, `shut` and `show` while declarations include `array`, `define`, `depend`, `korder`, `let`, `matrix`, `operator` and `order`.

Value

If `func` contains only a single element, the result of the procedure call is a vector.

If `func` contains more than one element, a list of named elements containing the result of each procedure call is returned. The name of each element is the procedure name. If one or more procedures are present multiple times in `func`, the names are made unique by adding an integer suffix *n* for the nth duplicate.

The result of the procedure call is character and may contain numbers, expressions or REDUCE lists.

Author(s)

Martin Gregory

See Also

red2Rarray, red2Rfn or red2Rlist for converting the result to a corresponding R object.

Examples

```
s1 <- redStart()

## can only run code if session was successfully started
if (is.numeric(s1)) {
   writeLines(ex1 <- redFunc(s1, "factorize", list(c("2310"))))
   redFunc(s1, c("factorize", "sin"), list("x^2-1", "pi/2"))
   redFunc(s1, "df", list(c("x*exp(x)", "x", "2")), switches=c(off="nat"))

   ## LaTeX output
   redFunc(s1, "df", list(c("x*exp(x)", "x", "2")), outfmt="latex", mathenv="equation")

   ## convert factorize output to R list
   utils::str(red2Rlist(ex1))
   redClose(s1)
}
```

---

redin *Create a vector of REDUCE commands by reading data inline.*

Description

Allows entering REDUCE commands line by line as if writing a reduce program, i.e. the individual commands do not need to be quoted and double quotes in a REDUCE command do not need to be escaped.

Usage

```
redin(x)
```

Arguments

x	a single, possibly multi-line literal string enclosed in single quotes.

Details

This is a convenience function which simply calls

```
unlist(strsplit(x, "\n"))
```

to convert the input to a vector with each element being the corresponding line in the input.

Value

A character vector containing the commands entered.

Note

An alternative which additionally dispenses with the single quotes is scan

```
> cmd <- scan(what=character(0), sep="\")
1: factorize 9997 ;
2: write "hello;";
3: q:=(x+y)^3 ;
4: df(q, x) ;
5:
Read 4 items
```

but this has the disadvantage in comparison to redin that it only works at the top level so can't be used in a function or in a Sweave input file.

Author(s)

martin gregory

Examples

```
## create a vector of REDUCE commands
cmd <- redin('factorize 9997 ;
write "this is some quoted text";
q:=(x+y)^3 ;
df(q, x) ;')

## view the commands
writeLines(cmd)
```

---

redLatex *Convert REDUCE output to LaTeX*

Description

Send mathematical expressions or the names of one or more arrays or expressions to the REDUCE process for display as LaTeX or plain text. If the object is an array, each element is printed. The LaTeX produced by REDUCE is then post-processed to translate object names and to convert array arguments to indices according to maps specified by the user, and, optionally, to enclose the result in a math environment specified by the user.

Usage

```
redLatex(id, x, usermap, mathenv="", outfmt="latex", use.names=FALSE,
  switches=character(0), debug=FALSE, ...)
asltx(id, x, usermap, mathenv="", mode="fancy", use.names=TRUE,
      notify=0, timeout=0, debug=FALSE)
```

Arguments

id	session identification returned by redStart. Required.
x	is a vector of type character, numeric or expression. If character elements are either the name of an existing REDUCE object or a mathematical expression to display. These are not statements so no terminator is required.
usermap	A list containing one or more of the three named items listed below. In each case the item is a named character vector. The transformations are done by gsub. The name of an element is the regular expression to match and the value is the replacement: **ident** a named character vector specifying translation of REDUCE identifiers to LaTeX names. **index** a named character vector specifying the conversion of arguments to indices. The name specifies which identifier to process. Since this mapping is done after the identifier mapping, you need to use the LaTeX name for any name you have mapped. The value is a sequence of ^ and _ characters specifying whether the index is contravariant (superscript) or covariant (subscript), respectively. **arb** a named character vector specifying arbitrary transformations. This argument is optional. Default is no user-specified transformation. See Details and the vignette Using redcas for further information and examples.
mathenv	Character. The name of a LaTeX math environment in which to enclose an expression or each element of an array. Defaults to no environment. This is useful for displaying the array elements in a subordinate math environment, for example \dmath* within a dgroup environment.
outfmt	Character. Specifies the output format which can be one of "latex" or "text". The default is "latex". If "text" then the current setting of the NAT switch is used.
mode	Character. Whether to display x as LaTeX (*FANCY*) or not (*NAT*). Default is *FANCY*. "" is treated as *NAT*.
use.names	Logical. Whether to include the object's name followed by = in the output. Default is FALSE for redLatex but TRUE for asltx.
notify	while waiting for the REDUCE commands to complete, write a note to the console every notify seconds. Default is zero which suppresses the message.
timeout	number of seconds after which to terminate the function if it is still waiting for output from the REDUCE process. Default is 0 which will never initiate termination.
switches	a character vector of switches to be applied while redLatex is executing. Note that neither ECHO, NAT nor FANCY are allowed because redLatex requires specific states of these switches. See swset for details. Optional.
debug	Boolean. When TRUE the mappings are printed to enable debugging since regular expressions can be tricky. Default is FALSE.
...	Arguments which can be passed to redExec, for example notify, timeout

.

Details

redLatex is a replacement for asltx in order to allow finer control over the output type. With outfmt="latex", output is LaTeX. If outfmt is "text", the current setting of *NAT* determines whether plain text uses multiple lines to display an expression (ON NAT) or only a single line (OFF NAT). asltx is kept for backward compatibility but is now simply a wrapper for redLatex which should be used in preference.

If redLatex is passed the name of an array, the individual elements are printed sequentially as, unlike R, REDUCE does not provide a print method for arrays. In order to identify the elements, set use.names=TRUE in order to print the array name and indices. Arrays are printed in row order which is important if names are not included. Furthermore if a mathematics environment is specified, each element will be enclosed in an environment. In this case one may wish to use a sub-environment such as dmath from the LaTeX breqn package. An alternative is to use red2Rarray to create an R array, optionally as LaTeX. This may be more useful if the name is not required and can use the R array print method.

For scalar expressions, lists, matrices and equations REDUCE does have a print method.

For matrices, when outfmt is "latex" these are displayed using the LaTeX matrix environment. When outfmt is "text" and NAT is on matrices are printed as a single object and not element by element as is the case for arrays. When NAT is off matrices are printed in row order as:

```
mat((row1), (row2), ...,
  (rowN))
```

If use.names is true and outfmt is "text", the matrix name is printed on a separate line before the matrix.

If use.names=TRUE is specified, any explicit expressions are printed without a name to avoid having expression=expression as output. Note that this does not currently take account of escaped "invalid" characters in a REDUCE identifier.

redLatex uses redExec to display the object x as desired by:

1. constructing a call to the REDUCE function redLatex with appropriate quoting of the arguments;
2. executing this call using redExec and specifying split=TRUE;
3. applying a standard set of transformations to remove some non-LaTeX markup and perform some conversions on the result. See below;
4. applying a set of transformations specified by the usermap argument.

Features of REDUCE conversion to LaTeX: REDUCE provides three different methods for converting output to LaTeX, the packages TMPRINT, RLFI and TRI. **redcas** uses the TMPRINT package which was designed for use with the TeXmacs editor (hence TM) and triggers conversion using the *fancy* and *fancy_tex* switches. TMPRINT was chosen because it can be easily applied to fragments, supports using the \frac command, produces LaTeX output and is well supported. In contrast RLFI is designed to produce complete documents using LaTeX 2.09 syntax and does not support the \frac command. TRI produces plain TeX output.

TMPRINT supports

1. inserting \left and \right in nested parentheses, braces and brackets;

2. converting variables whose names are those of Greek letters or common mathematical functions to the corresponding LaTeX command. Capitalized Greek letter names are mapped to the command for the capital. For example, `psi` is mapped to `\psi` while `Psi` is mapped to `\Psi`. There are three exceptions; `epsilon` and `kappa` are mapped to `\varepsilon` and `\varkappa`, respectively. If you need `\epsilon` or `\kappa` you can use the `usermap` argument. `Gamma` is mapped to `\gamma`. To get `\Gamma` use `!Gamma` as the variable name.
   There is one further issue: `omicron` generates the error `***** unknown print item`, and any subsequent LaTeX output produces this error. The only solution is to avoid using `omicron`.
3. the following names are mapped to special symbols:

infinity	\infty	union	\cup
partial!-df	\partial	member	\in
empty!-set	\emptyset	and	\wedge
not	\neg	or	\vee
leq	\leq	when	|
geq	\geq	!*wcomma!*	,\,
neq	\neq	replaceby	\Rightarrow
intersection	\cap	!~	\forall

   Note that the exclamation mark is an escape character which allows any character to be used in a symbol. REDUCE inherits this ability from the underlying LISP.

Transformations: In addition to the transformations done by TMPRINT, `redLatex` performs some standard transformations to clean up some aspects of the LaTeX. It also provides the `usermap` argument to allow specification of three different types of optional transformation:

1. renaming identifiers, i.e. variable names;
2. converting function arguments to indices;
3. arbitrary substitutions.

The order in which transformations are done is:

1. standard transformations except for replacing `\*` with space;
2. user renaming of identifiers;
3. conversion of function arguments to indices;
4. arbitrary user transformations;
5. replacement `\*` with space.

The difference between arbitrary and identifier transformations is that the latter are anchored at word boundaries to avoid unintentional renaming when the name of an identifier is a substring of another identifier.

The reason for replacing `\*` with space last is to allow an arbitrary transformation to replace it with, for example, the LaTeX `\thinspace` or `\,` command.

Standard transformations: The following transformations are always done by `redLatex`:

1. removal of markup used for interactive display in TeXmacs and which has no correspondence with LaTeX markup;
2. replacement of REDUCE assignments `:=` and `~:=~` with a plain equals sign;
3. reversing the unwanted conversion of numeric suffix to index. The REDUCE FANCY option assumes that a numeric suffix to a variable is a subscript index even when the variable is an array, e.g. `cs1(a,b)` is converted to `cs_{1}(a,b)`. `redLatex` undoes this conversion.

4. removal of redundant `mathrm` commands;
5. replacement of `mathit{Q}` with `mathrm{Q}` where `Q` is one of the Greek capitals A, B, E, Z, H, I, M, N, O, P. i.e. those which have the same form as a Latin capital. This transformation is required to have the same type for as the Greek-only capitals;
6. replacement of `\*`, which REDUCE inserts as multiplication, with a space to avoid the case where a LaTeX command precedes the `\*` sign.

User transformations:
The `usermap` argument allows renaming variables, converting arguments in REDUCE objects to indices and specifying whether indices are covariant or contravariant. If a LaTeX command is used in the map, four backslashes are required to get the single backslash for LaTeX because the mapping is done using regular expressions. For example suppose we have arrays `s`, `g` and `kplus` in REDUCE and want the following mapping to LaTeX

REDUCE	LaTeX
s	\Sigma
g	g
kplus	K^+

then the `ident` element of the list passed to `usermap` would be

```
ident=c("s"="\\\\Sigma", kplus="K")
```

For `kplus` we need to use the `arb` map to add the raised plus. Since we are not mapping `g` it does not need to appear here. P
To illustrate mapping arguments to indices, assume that all four arrays in the example above are two dimensional and `s` has one subscript and one superscript, `g` and `kplus` have two subscripts then the `index` element will be

```
index=c("\\\\Sigma"="_^", "g"="__", "K"="__")
```

Note that the name of the `index` elements must be the result of applying the `ident` mapping.
Finally `arb` map for raising the plus would be

```
arb=c("K"="K^\{+\}"
```

The braces must be escaped here because they have a special meaning in LaTeX.

Value

A list containing the following elements is returned:

tex the transformed LaTeX output;

out the output of the executed commands;

cmd the executed commands

raw the interspersed commands and output.

Author(s)

Martin Gregory

See Also

`R2latex` for converting lists and data frames to LaTeX, vignette Using redcas for examples and `redExec` for details of executing REDUCE code.

Examples

```
## start the session
s1 <- redStart()

## can only run code if session was successfully started
if (is.numeric(s1)) {
   ## create the arrays. Set switch NERO to ON to suppress zeroes.
   redcode <- c("array g(2,2), s(3,3), kplus(2,2) ;",
                "operator x;",
                "g(0,0) := -u^(-2);",
                "g(1,1) := (u*x(3))^2;",
                "g(2,2) := g(1,1) * (sin(x(1)))^2;",
                "s(0,0) := 0;",
                "s(1,1) := df((u*x(3))^2, x(3));",
                "s(2,2) := df(g(2,2), x(1));",
                "s(3,3) := u^2;",
                "kplus(0,0) := df(g(0,0), u);",
                "kplus(1,1) := df(g(1,1), u);",
                "kplus(2,2) := df(g(2,2), u) ;",
                "on nero ; "
                )

    o2 <- redExec(s1, redcode)
    ## define mappings. arb[1] maps K to K^{+}, arb[2] removes \left and \right
    imap <- list(ident=c(kplus="K", s="\\\\Sigma"),
                index=c(g="__", K="__", "\\Sigma"="_^"),
                arb=c("K"="K^\\{+\\}", "(\\\\left|\\\\right)"=""))
    dash <- strrep("-",70)

    ## create LaTeX output
    writeLines(c("g: no names, no map", dash,
                redLatex(s1, "g")$tex, ""))
    writeLines(c("g: names, no map", dash,
                redLatex(s1, "g", use.names=TRUE)$tex, ""))
    writeLines(c("g: names, map indices, drop \\left and \\right", dash,
                redLatex(s1, "g", use.names=TRUE, usermap=imap)$tex, ""))
    writeLines(c("Sigma: no names, plain text", dash,
                redLatex(s1, "s", outfmt="text")[["tex"]], ""))
    writeLines(c("Sigma: names, map indices, drop \\left and \\right", dash,
                redLatex(s1, "s", use.names=TRUE, usermap=imap)$tex, ""))
   writeLines(c("Kplus: names, map indices, add superscript +, drop \\left and \\right", dash,
                redLatex(s1, "kplus", use.names=TRUE, usermap=imap)$tex, ""))

    ## close the session
    s1.done <- redClose(s1)
}
```

---

redSolve *function to solve a system of equations using the REDUCE* solve *operator*

Description

redSolve generates code for the REDUCE *solve* operator, executes it and converts the result to appropriate R objects if possible.

Usage

```
redSolve(id, eqns, unknowns, outfmt="text", switches=character(0),
         verbose=FALSE, ...)
```

Arguments

id	the session identifier returned by redStart. Required with no default
eqns	the system of equations to be solved as vector of type character or expression with one element per equation. If an expression is used, equations must be in the normal form and the =0 must be omitted. Required. No default.
unknowns	the unknowns for the system of equations specified by eqns as a vector character of type character or expression with one element for each unknown. Required. No default.
outfmt	Character. "text" or "latex". Controls whether the solutions are output as plain text (default) or LaTeX.
switches	a character vector of switches to be applied while solve is executing. Note that neither ECHO, NAT nor FANCY are allowed. See Details below. Optional.
verbose	Logical. Controls whether an informational message about toggling the ECHO or NAT switch is printed if the function resets these. Default is FALSE.
...	Arguments which can be passed to redLatex or redExec (usermap, mathenv, notify, timeout). The arguments use.names, split and drop.blank.lines are not allowed.

Details

solve is a REDUCE operator for solving one or more simultaneous algebraic equations in one or more variables, both linear and non-linear. Equations may contain arbitrary constants which is why it is necessary to specify the unknowns. For example, given a quadratic equation with constants a, b and c:

```
16: solve(a*x^2 + b*x + c, x);

                      2
    sqrt( - 4*a*c + b ) - b
{x=------------------------,
              2*a

                         2
     - (sqrt( - 4*a*c + b ) + b)
 x=-----------------------------}
                2*a
```

The standard solution is returned as a REDUCE list. We can change the order of the expression to get the usual form by using the *korder* declaration before calling solve:

```
17: korder b, a;
18: solve(a*x^2 + b*x + c, x);

          2
    sqrt(b  - 4*a*c) - b
{x=----------------------,
           2*a

              2
     - (sqrt(b  - 4*a*c) + b)
 x=---------------------------}
               2*a
```

If the solve operator could not find an explicit solution, it may still be able to find one in terms of an equation for the unknown. Such a solution is identified by specifying the subsidiary equation as the argument of the operator *root_of*. If there are several unknowns, each solution will be a list of equations for the unknowns. For example,

```
solve(x^7 - x^6 + x^2 = 1, x) ;
{x=root_of(x_**6  + x_ + 1,x_,tag_1),x=1}
```

Switches: The `switches` argument take input using the format defined for `swset`, i.e. a character vector with two named elements, on and off, each of which is a comma separated string of switches. These switches are set prior to execution of solve and restored to their previous values afterwards. You can still use the format in **redcas** 0.1.1, a character vector with on and off statements, and this will be converted to the swset format.

Note that `redSolve` turns off the NAT switch as otherwise the results will not be parsed correctly. `redSolve` will ensure that NAT is off for the duration of its execution. Turning on the NAT switch using the `switches` parameter is ignored. If verbose is true, a message is printed.

There are several switches related to *solve*, in particular *arbvars, cramer, multiplicities, fullroots, trigform* and *varopt*. See the REDUCE manual section 7.17 for details of these and of the methods used for solving equations.

Setting the *arbvars* switch to off suppresses the use of arbitrary constants. If this is on and the system of equations is under-determined solutions may be presented only for some of the unknowns. These solutions will be functions of unknowns for which no solution is presented.

Value

`redSolve` returns an object of class `"redcas.solve"` which contains the following elements:

solutions a list of the solutions as returned by REDUCE. Each solution is a character vector.

rsolutions a list of the solutions, where possible converted to appropriate R objects. Because each element of a solution may be of a different type (real, complex, string), each solution is a list rather than a vector.

nsolutions numeric. The number of solutions.

rc numeric. The return code. 0 is success, 1 failure

root_of a complex vector identifying which solutions are presented in terms of the REDUCE *root_of* operator. A solution may expressed one or more unknowns in terms of the roots of another equation. In this case the other equation is enclosed in *root_of()*. This slot is a complex vector with the real part identifying the solution containing *root_of()* and the imaginary part the index of the unknown in that solution. This item is intended for use in a future release.

eqns the vector of equations passed to `redSolve`

unknowns the vector of unknowns passed to `redSolve`

switches the vector of switches passed to `redSolve`

The `redcas.solve` object may be printed using the `print` method.

Author(s)

martin gregory

See Also

`print` to print a `redcas.solve` object, REDUCE manual section 7.17 for details of the REDUCE *solve* operator and `redStart` for creating a REDUCE session.

Examples

```
## Open a CSL session:
s1 <- redStart()

## can only run code if session was successfully started
if (is.numeric(s1)) {
   sol1 <- redSolve(s1, eqns=c("x+3y=7", "y-x=1"), unknowns=c("x", "y"))
   print(sol1)

   ## because arbvars is off, solutions are returned for b and c in
   ## terms of a
   sol2  <- redSolve(id=s1, eqns=c("a^3+b", "b^2+c"), unknowns=c("a", "b", "c"),
                     switches=c(off="arbvars"))
   print(sol2)

   ## close session:
   redClose(s1)
}
```

`redSplitOut` *convert the vector returned by redExec into commands and command output.*

Description

REDUCE output contains both the commands issued and the output from the commands. Since the output is of interest, redSplitOut separates the input vector into a list containing vectors of output, commands and the input to `redSplitOut`.

Usage

```
redSplitOut(x)
```

Arguments

`x`	a character vector containing REDUCE output

Details

Although it is possible to use the REDUCE switch ECHO to prevent display of commands, this does not prevent display of the sequence numbers for the commands. `redSplitOut` can be called by `redExec` to separate the output from the commands. It can also be used after the call if the `split` option was not used or on an arbitrary output file from REDUCE, for example the full output which can be retrieved by `redClose`.

Value

A list containing three elements:

`out`	the command output
`cmd`	the commands
`raw`	the input to `redSplitOut`, i.e. the combined cmd and out vectors.

Author(s)

Martin Gregory

See Also

`redExec`

Examples

```
result <- c("5: ",
            "{x=-1}",
            "",
            "6: ",
            "b := {{2,12},{11,1}}",
            "",
            "7: ",
            "8: ",
            "                - m",
            "1a := ---------------------",
            "               2       2",
```

```
            "       sqrt(v(3)  + v(2) )",
            "",
            "9: ",
            "",
            "10: ")
split <- redSplitOut(result)
writeLines(split$out)
```

---

redStart *function to start a REDUCE session*

Description

Starts a REDUCE session using a pipe connection sending the output (log) to a temporary file.

Usage

```
redStart(dialect = "csl", dirpath, options, echo=TRUE, switches=character(0))
```

Arguments

dialect	character. The version of lisp with which the REDUCE executable was built. Allowed values are csl, the default, or psl.
dirpath	character. Path to a directory containing the REDUCE executables. Optional, see details below.
options	character. Options for the csl version. See details.
echo	boolean. Should the *ECHO* switch be turned on or off. Default is TRUE to facilitate separating output from command. If other switches are being set for the session using the switches argument, ECHO can be set there.
switches	a character vector of switches to be set after starting the session. This must have the same form as the switches argument of swset. Optional. Note that unlike other functions with this argument, the switches are not reset after redStart completes execution. Note also that FANCY is not allowed because this formats all output as LaTeX. This can cause issues for **redcas** functions when parsing output. Functions which support the latex option of the outfmt argument selectively turn on FANCY when producing output, so there should not be any need to set FANCY otherwise..

Details

redcas finds the REDUCE executables (redcsl or redpsl) in two different ways. The first is the explicit specification of the path using the dirpath argument passed to the redStart function. If this does not contain the REDUCE executable, redStart stops executing on the assumption that a specific version was intended.

The second method is used if dirpath is not specified. This determines the path at package load time by searching the following three locations in turn until the executable is found:

- the environment variable REDUCE_EXEC exists and contains the path to a directory containing an executable;
- the R option `reduce_exec` exists and contains the path to a directory containing the executable;
- the REDUCE executable is found in a directory named in the environment variable PATH.

This approach provides the flexibility to have a default version while being able to call different versions explicitly.

While it is possible to start a second REDUCE session without closing the first, garbage collection closes its connection. A future release will fix this issue.

`redStart` carries out the following actions:

- creates the REDUCE session using a pipe connection and sending the REDUCE output to a temporary file (in REDUCE terminology referred to as a log);
- registers the session, acquiring a session identifier;
- calls `redExec` to
  - load the REDUCE ASSIST package which provides, among other features, functions which are indispensable for managing lists,
  - load the REDUCE functions which are part of **redcas** and
  - set the *ECHO* switch if requested and any switches specified in the switches argument;
- returns the session identifier.

If you specify `off="echo"` in the switches argument and `echo=TRUE`, the latter takes precedence.

For CSL only, command line options may be passed to redcsl via the `options` argument. For a list of options and their meanings use the command `redcsl --help` for a GUI interface or `redcsl --help --nogui` for output at the terminal. This documentation is not available from within the CSL GUI interface. Note that the PSL executable, `redpsl` has no command line arguments.

Value

The session identifier which is an integer >0 if the session started successfully. If the session could not be started, `redStart` terminates using `stop`. If no REDUCE executable was found returns FALSE.

An invalid value for the switches argument causes `redStart` to abort the session creation and return FALSE with an appropriate message.

Author(s)

martin gregory

See Also

`showSessions` to list active sessions and `redClose` for closing a session.

Examples

```
## Open a CSL session:
id1 <- redStart()
## can only run code if session was successfully started
if (is.numeric(id1)) {
   ## show session details:
   print(showSessions())
   ## close session:
   redClose(id1)
}
```

`sessionGet` *retrieve a specific attribute of a REDUCE session.*

Description

For a session, retrieve one of the pipe connection, REDUCE command, path to log file, how many blocks were submitted and how many lines were retrieved from the log file.

Usage

```
sessionGet(id, what)
```

Arguments

`id`	The session id for which to display the details. The session id is returned by `redStart`.
`what`	The attribute to retrieve which can be one of **pcon** the pipe connection to the REDUCE session; **cmd** the full path to the REDUCE executable; **logname** the full path to the log file; **blockn** the number of blocks of REDUCE code submitted, i.e. number of calls to any function which sends code to REDUCE; **lines.read** the number of lines read from the log file for the session.

Value

For `pcon` an integer identifying the connection object, for `cmd` and `logname` a character vector with one element and for `blockn` and `lines.read` an integer.

Author(s)

martin gregory

See Also

`showSessions` for returning all attributes, `redStart` for creating a REDUCE session and `redClose` for closing a session.

Examples

```
## Open a CSL session:
id1 <- redStart()
## can only run code if session was successfully started
if (is.numeric(id1)) {
   ## retrieve the entire transcript up to now
   readLines(sessionGet(id1,'logname'))

   ## details of the connection
   getConnection(sessionGet(id1,'pcon'))

   ## close session:
   redClose(id1)
}
```

`showSessions` *function to list details for currently active REDUCE sessions.*

Description

For each active REDUCE session returns the full path to the REDUCE executable and the temporary log file, the connection identifier, the number of code blocks executed and the number of lines read from the log so far.

Usage

```
showSessions(id)
```

Arguments

`id`	The session id for which to display the details. May be omitted. If none specified, display details for all sessions. The session id is returned by `redStart`.

Value

A data frame with the following columns:

`id`	integer, the session identifier;
`pcon`	integer, the input connection for the REDUCE session;
`cmd`	character, full path to the REDUCE executable;
`logname`	character, full path to the log file;
`blockn`	integer, number of blocks of REDUCE code submitted, i.e. number of calls to any function which sends code to REDUCE, currently `redExec`, `asltx` and `redSolve` ;
`lines.read`	integer, number of lines read from the log file for the session.

Author(s)

martin gregory

See Also

redStart for creating a REDUCE session and redClose for closing a session.

Examples

```
## Open a PSL session:
id1 <- redStart('psl')
## can only run code if session was successfully started
if (is.numeric(id1)) {
   ## show session details:
   print(showSessions(id1))

   ## retrieve the entire transcript up to now
   writeLines(readLines(showSessions(id1)[,'logname'], warn=FALSE))

   ## close session:
   redClose(id1)
}
```

---

swset *Set and restore REDUCE switches*

Description

Set one or more switches to a desired state remembering which had to be changed so that these can later be restored to the original state.

Usage

```
swset(id, switches, swlist="sw!-to!-toggle", disallow=character(0),
      verbose=FALSE)
swreset(id, swlist="sw!-to!-toggle")
```

Arguments

id	The session id for which to display the details. The session id is returned by redStart.
switches	a character vector with one or both of the following named elements **on** a comma-separated string of switches which should be on; **off** a comma-separated string of switches which should be off.
swlist	a string containing the name of a REDUCE list in which to save the names of the switches which were toggled. Default is *sw!-to!-toggle*. See details below.

disallow	a character vector of the names of switches which should not be changed. This is primarily for use in **redcas** functions which accept the switches argument and require a specific state of some switches. swset always disallows FANCY. See details below.
verbose	Logical. Whether to report on the removal of disallowed switches.

Details

swset checks which specified switches are not in the desired state, toggles them and populates the REDUCE list named in swlist with the names of these switches.

swreset restores the previous state of the switches named in swlist by toggling. It then deletes the list.

If using two swset calls without resetting the first before calling the second, be sure to use a different list name and ensure that no switch is specified in both calls. Do not explicitly set a switch named in a swset call until after the corresponding swreset has been called.

Values for swlist must be valid names and should avoid names which might be present in the REDUCE session. Names may contain characters not normally allowed by escaping them using the exclamation mark as is done for the default value. This can reduce the risk of using an existing name. Furthermore REDUCE is case-insensitive in that by default it converts all upper case input, with the exception of strings, to lower case. So if we pass swlist="Upper!-list", the list is named upper-list in REDUCE. When swreset is called to reset the switches in the list, it makes the same transformation are REDUCE does to avoid a message No list called 'Upper!-sw!-list' exists in REDUCE session. To avoid the case fold, each upper case letter may be escaped by !. To avoid confusion, use only lower case letters.

swset prohibits setting of the FANCY switch because this formats all output as LaTeX. This can cause issues for **redcas** functions when parsing output. Functions which support the latex option of the outfmt argument selectively turn on FANCY when producing output, so there is no need to set FANCY otherwise.

While swset allows ECHO, all **redcas** functions require it to be on in order to facilitate separating commands from output. For this reason we recommend setting ECHO to on when calling redExec which cannot call swset.

Value

swset returns a vector containing the names of the toggled switches.

swreset invisibly returns a list of the restored state for toggled switches, for example

```
off div
off expandlogs
on  allfac
```

Author(s)

martin gregory

See Also

showSessions for returning all attributes, redStart for creating a REDUCE session and redClose for closing a session.

Examples

```
## Open a CSL session:
s1 <- redStart()
## can only run code if session was successfully started
if (is.numeric(s1)) {
   ## show current state of several switches
   state0 <-redExec(s1, c('swl := {nero, div, exp, allfac};',
                  'for each s in swl do write if swget(s) then "on " else "off", " ", s;'),
                   drop.blank.lines=TRUE)[["out"]]
   writeLines(state0)

   ## turn div off and exp on
   Rls <- swset(s1, switches=c(on="exp,nero", off="div"), swlist="redls")
   writeLines(Rls)

   ## check the state
   redExec(s1, 'for each s in swl do write if swget(s) then "on " else "off", " ", s;',
           drop.blank.lines=TRUE)[["out"]]

   ## restore the state
   restore <-swreset(s1, swlist="redls")
   writeLines(restore)

   ## close session:
   redClose(s1)
}
```

Index

∗ **classes**
redcas.ode, 22
redcas.solve, 23
∗ **methods**
print, 11
∗ **package**
redcas-package, 2

asltx, *51*
asltx *(*redLatex*)*, 38

dlmMatch *(*red2Rlist*)*, 19

factorize, 4
fnMatch *(*red2Rlist*)*, 19

gsub, *39*

list, *22, 23*
ltx.append.eol, *8*, *20*
ltx.append.eol *(*R2latex*)*, 13

odesolve, 7, *18*, *22, 23*

print, *10*, 11, *22–24*, *46*
print,redcas.ode-method *(*print*)*, 11
print,redcas.solve-method *(*print*)*, 11
print-method *(*print*)*, 11

R2latex, *3*, *7, 8*, *10*, 13, *20*, *42*
red2Rarray, *3*, 15, *37*, *40*
red2Rfn, *3*, *16*, 17, *29*, *37*
red2Rlist, *3*, 19, *37*
redcas *(*redcas-package*)*, 2
redcas-package, 2
redcas.ode, *10*, 22
redcas.ode-class *(*redcas.ode*)*, 22
redcas.solve, 23, *45*
redcas.solve-class *(*redcas.solve*)*, 23
redClose, *3*, 24, *47*, *49, 50*, *52*, *54*
redCodeDir, 25
redDf, 26
redDropOut, 30, *32*
redExec, *3*, *5*, *8*, *14*, *16*, *20, 21*, *24*, *27*, *30*, 31, *33–35*, *39, 40*, *42*, *44*, *47*, *49*, *51*
redExpand, 33
redFunc, *3*, *18*, 34
redin, *32*, 37
redLatex, *3*, *5*, *7, 8*, *14, 15*, *27*, *35*, 38, *44*
redlsExtract *(*red2Rlist*)*, 19
redSolve, *18*, *20*, *23, 24*, 43, *51*
redSplitOut, *32*, 46
redStart, *3, 4*, *7*, *10*, *14*, *16*, *24*, *26*, *31, 32*, *34*, *39*, *44*, *46*, 48, *50–52*, *54*

scan, *38*
sessionGet, 50
showSessions, *24*, *49, 50*, 51, *54*
sublist2char *(*red2Rlist*)*, 19
swreset *(*swset*)*, 52
swset, *5*, *9*, *14*, *16*, *27*, *34*, *39*, *45*, *48*, 52

unlist, *18*

vector, *22, 23*