# Package 'cgraph'

November 5, 2018

**Type** Package

**Title** Computational Graphs

**Version** 3.0.1

**Author** Ron Triepels

**Maintainer** Ron Triepels <dev@cgraph.org>

**URL** https://cgraph.org/

**BugReports** https://github.com/triepels/cgraph/issues

**Description** Allows to create, evaluate, and differentiate computational graphs in R. A computational graph is a graph representation of a multivariate function decomposed by its (elementary) operations. Nodes in the graph represent arrays while edges represent dependencies among the arrays. An advantage of expressing a function as a computational graph is that this enables to differentiate the function by automatic differentiation. The 'cgraph' package supports various operations including basic arithmetic, trigonometry operations, and linear algebra operations. It differentiates computational graphs by reverse automatic differentiation. The flexible architecture of the package makes it applicable to solve a variety of problems including local sensitivity analysis, gradient-based optimization, and machine learning.

**License** Apache License 2.0

**Encoding** UTF-8

**LazyData** true

**Imports** R6

**Suggests** Rgraphviz, testthat

**RoxygenNote** 6.1.0

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2018-11-05 09:20:10 UTC

# R topics documented:

---

`cgraph` *Computational Graph*

---

## Description

The `cgraph` class facilitates the construction, evaluation, and differentiation of computaiontal graphs in R.

## Usage

```
x <- cgraph$new()
```

## Members

**$nodes** named list, symbols of the nodes.

**$values** environment, values of the nodes.

## Methods

**$initialize** initialize a computational graph, see cg_initialize.

**$const** add a constant node to the graph, see cg_const.

**$input** add an input node to the graph, see cg_input.

**$parm** add a parameter node to the graph, see cg_parm.

**$get_parms** list all parameters and their values, see cg_get_parms.

**$add_parms** add parameters to the graph, see cg_add_parms.

**$opr** add an operation node to the graph, see cg_opr.

**$active** set the graph to be the active graph, see cg_active.

**$val** get the value of a node in the graph, see cg_val.

**$set** set the value of a node in the graph, see cg_set.

**$run** evaluate a node in the graph, see cg_run.

**$gradients** differentiate the graph by reverse automatic differentiation, see cg_gradients.

**$adj_mat** retrieve the adjacency matrix of the graph, see cg_adj_mat.

**$plot** plot the topology of the graph, see cg_plot.

## Note

Some of the methods listed above have a wrapper function that calls the method on the current active graph. For example, a parameter can be added to the current active graph by calling parm instead of calling cg_parm on the currently active cgraph object. Similarly, nodes can be evaluated or changed by calling val or set instead of calling method cg_val or cg_set respectively.

## Author(s)

Ron Triepels

## Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add an input with name 'a' to the graph.
a <- input(name = "a")

# Add a parameter with value 4 and name 'b' to the graph.
b <- parm(4, name = "b")

# Perform some operations (i.e. c = exp(a * b)).
c <- cg_exp(a * b, name = "c")
```

```
# Evaluate c at a = 2.
values <- run(c, list(a = 2))

# Retrieve the value of c.
values$c

# Differentiate the graph with respect to c.
grads <- gradients(c, values)

# Retrieve the gradient of c with respect to b.
grads$b
```

---

cg_abs *Absolute Value*

---

## Description

Calculate `abs(x)`.

## Usage

```
cg_abs(x, name = NULL)
```

## Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

## Value

cg.node, node of the operation.

## Author(s)

Ron Triepels

## See Also

abs

## cg_acos *Inverse Cosine*

### Description

Calculate `acos(x)`.

### Usage

```
cg_acos(x, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

acos

## cg_acosh *Inverse Hyperbolic Cosine*

### Description

Calculate `acosh(x)`.

### Usage

```
cg_acosh(x, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

acosh

---

`cg_active` *Change Active Graph*

---

### Description

Set the graph to be the active graph.

### Details

`$active()`

### Value

none.

### Note

Any nodes that are created are automatically added to the active graph. This also applies to operations that are created by overloaded S3 functions that do not follow the cg_<name> naming convention (such as primitive inflix functions '+' and '-').

Only one graph can be active at a time. The active graph can be changed by calling method cg_active on another cgraph object.

### Author(s)

Ron Triepels

### Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Initialize another computational graph. It becomes the current active graph.
y <- cgraph$new()

# Set graph x to be the active graph.
x$active()
```

## cg_add *Add*

### Description

Calculate `x + y`.

### Usage

```
cg_add(x, y, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `y` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

add

## cg_add_parms *Add Parameters*

### Description

Add parameters to the graph.

### Arguments

| | |
|---|---|
| `...` | numeric vectors or arrays, the values of the parameters. Is ignored when `parms` is not `NULL`. |
| `parms` | named list, the parameters that are to be added to the graph. |

### Details

```
$add_parms(..., parms = NULL)
```

### Value

nothing.

### Note

Parameters can be named by providing named arguments to ... or by naming the elements of argument parms. In case no names are provided, parameters are tried to be added to the graph under an auto-generated name. No default value is set for parameters with value NULL.

### Author(s)

Ron Triepels

### Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add some parameters.
x$add_parms(prm1 = 1, prm2 = 2, prm3 = 3)

# List the parameters.
x$get_parms()
```

---

## cg_adj_mat *Adjacency Matrix*

---

### Description

Retrieve the adjacency matrix of the graph.

### Details

$adj.mat()

### Value

numeric matrix, the adjacency matrix of the graph.

### Author(s)

Ron Triepels

## cg_asin *Inverse Sine*

### Description

Calculate `asin(x)`.

### Usage

```
cg_asin(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

asin

## cg_asinh *Inverse Hyperbolic Sine*

### Description

Calculate `asinh(x)`.

### Usage

```
cg_asinh(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

asinh

---

`cg_as_double` *Coerce to a Numeric Vector*

---

### Description

Coerce `x` to a one-dimensional numeric vector.

### Usage

```
cg_as_double(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Note

This function is identical to `cg.as.numeric`.

### Author(s)

Ron Triepels

### See Also

as.double

## cg_as_numeric *Coerce to a Numeric Vector*

### Description

Coerce x to a one-dimensional numeric vector.

### Usage

```
cg_as_numeric(x, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Note

This function is identical to cg.as.double.

### Author(s)

Ron Triepels

### See Also

as.numeric

## cg_atan *Inverse Tangent*

### Description

Calculate atan(x).

### Usage

```
cg_atan(x, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

atan

---

## cg_atanh *Inverse Hyperbolic Tangent*

---

### Description

Calculate `atanh(x)`.

### Usage

```
cg_atanh(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

atanh

## cg_colMeans *Column Means*

### Description

Calculate `colMeans(x)`.

### Usage

```
cg_colMeans(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Note

Function colMeans is called without setting argument `na.rm` and `dims`.

### Author(s)

Ron Triepels

### See Also

colMeans

## cg_colSums *Column Sums*

### Description

Calculate `colSums(x)`.

### Usage

```
cg_colSums(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Note

Function colSums is called without setting argument `na.rm` and `dims`.

### Author(s)

Ron Triepels

### See Also

colSums

---

## `cg_const` *Add Constant*

---

### Description

Add a constant node to the graph.

### Arguments

| | |
|---|---|
| `value` | numeric vector or array, value of the node (optional). |
| `name` | character scalar, name of the node (optional). In case argument `name` is missing, the node is tried to be added to the graph under an auto-generated name. |

### Details

```
$const(value = NULL, name = NULL)
```

### Value

cg_node, constant.

### Note

Constants are ignored when differentiating a graph. The intended use of constants is that they are given a fixed value. However, it is still possible to change the value of constants when evaluating or differentiating a graph (see cg_run and cg_gradients for more details).

There is a wrapper function const that calls this method on the current active graph.

### Author(s)

Ron Triepels

## Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add a constant with value 1 and name 'c' to the graph.
x$const(1, name = "c")
```

---

cg_cos *Cosine*

---

## Description

Calculate `cos(x)`.

## Usage

```
cg_cos(x, name = NULL)
```

## Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

## Value

cg.node, node of the operation.

## Author(s)

Ron Triepels

## See Also

cos

## cg_cosh *Hyperbolic Cosine*

### Description

Calculate `cosh(x)`.

### Usage

```
cg_cosh(x, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

cosh

## cg_crossprod *Matrix Crossproduct*

### Description

Calculate `crossprod(x, y)`.

### Usage

```
cg_crossprod(x, y = NULL, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric matrix. |
| y | cg.node, placeholder for a numeric matrix (optional). |
| name | character scalar, name of the operation (optional). |

## Value

cg.node, node of the operation.

## Author(s)

Ron Triepels

## See Also

crossprod

---

`cg_default_library` *Default Function Library*

---

## Description

This is the default function library used by a computational graph. It implements all graph operators by base R functions.

## Usage

```
cg_default_library
```

## Format

An object of class `environment` of length 126.

---

`cg_div` *Divide*

---

## Description

Calculate `x / y`.

## Usage

```
cg_div(x, y, name = NULL)
```

## Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `y` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

divide

---

`cg_equal` *Equal*

---

### Description

Calculate `x == y`.

### Usage

```
cg_equal(x, y, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `y` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

equal

## cg_exp *Exponential Function*

### Description

Calculate `exp(x)`.

### Usage

```
cg_exp(x, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

exp

## cg_get_parms *Get Parameters*

### Description

List all parameters and their values.

### Details

`$get_parms()`

### Value

named list, parameters of the graph.

### Author(s)

Ron Triepels

## Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add some parameters.
x$add_parms(prm1 = 1, prm2 = 2, prm3 = 3)

# List the parameters.
x$get_parms()
```

---

`cg_gradients` *Calculate Gradients*

---

## Description

Differentiate the graph with respect to node `name` by reverse automatic differentiation.

## Arguments

| | |
|---|---|
| `name` | character scalar, name of the node that is differentiated. |
| `values` | named list or environment, values that are subsituted for the nodes in the graph. |
| `index` | numeric scalar, index of the target node that is differentiated. Defaults to the first element. |

## Details

```
$gradients(name, values = new.env(), index = 1)
```

## Value

environment, the gradients of all nodes with respect to target node `name`.

## Note

All nodes required to compute node `name` must have a value, or their value must be able to be computed at run-time. The values of nodes can be obtained by first evaluating node `name` in the graph using function cg_run. The values obtained by this function for the nodes can then be supplied to argument `values`.

Currently, the cgraph package can only differentiate scalar target nodes. In case the value of target node `name` is a vector or an array, argument `index` can be used to specify which element of the vector or array is to be differentiated.

The gradients of all ancestors or `name` are returned. Constant nodes are not differentiated and their gradients are not returned. The gradients have the same shape as the values of the nodes.

There is a wrapper function gradients that calls this method on the current active graph.

### Author(s)

Ron Triepels

### Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add some parameters.
a <- x$parm(2, name = "a")
b <- x$parm(4, name = "b")

# Perform some operations on the parameters.
c <- cg_sin(a) + cg_cos(b) - cg_tan(a)

# Differentiate the graph with respect to c.
grads <- x$gradients(c, x$run(c))

# Retrieve the gradient of c with respect to a.
grads$a
```

---

## cg_greater *Greater*

---

### Description

Calculate `x > y`.

### Usage

```
cg_greater(x, y, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| y | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

greater

## cg_greater_equal *Greater or Equal*

### Description

Calculate `x >= y`.

### Usage

```
cg_greater_equal(x, y, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `y` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

greater or equal

## cg_initialize *Computational Graph*

### Description

Initialize a computational graph.

### Arguments

| | |
|---|---|
| `library` | environment, function library used by the graph. Defaults to cg_default_library. |

### Details

```
$new(library = cgraph::cg_default_library)
```

## Value

cgraph object.

## Note

The cgraph object is set to be the active graph.

## Author(s)

Ron Triepels

## Examples

```
# Initialize a new computational graph.
x <- cgraph$new()
```

---

`cg_input` *Add Input*

---

## Description

Add an input node to the graph.

## Arguments

| | |
|---|---|
| `value` | numeric vector or array, value of the node (optional). |
| `name` | character scalar, name of the node (optional). In case argument `name` is missing, the node is tried to be added to the graph under an auto-generated name. |

## Details

```
$input(value = NULL, name = NULL)
```

## Value

cg_node, input.

## Note

The intended use of inputs is that they are not given a fixed value but behave as placeholders. Values can be supplied for inputs when evaluating or differentiating a graph (see cg_run and cg_gradients for more details).

There is a wrapper function input that calls this method on the current active graph.

## Author(s)

Ron Triepels

## Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add an input with name 'x' to the graph.
x$input(name = "x")
```

---

cg_less *Less*

---

## Description

Calculate `x < y`.

## Usage

```
cg_less(x, y, name = NULL)
```

## Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| y | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

## Value

cg.node, node of the operation.

## Author(s)

Ron Triepels

## See Also

less

## cg_less_equal *Less or Equal*

### Description

Calculate `x <= y`.

### Usage

```
cg_less_equal(x, y, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| y | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

less or equal

## cg_linear *Linear Transformation*

### Description

Calculate `linear(x, y, z)`.

### Usage

```
cg_linear(x, y, z, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric matrix. |
| y | cg.node, placeholder for a numeric matrix. |
| z | cg.node, placeholder for a numeric vector. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Note

This function is equivalent to `cg.matmul(x, y) + cg.as.numeric(z)`.

### Author(s)

Ron Triepels

### See Also

linear

---

## cg_ln *Natural Logarithm*

---

### Description

Calculate `log(x)`.

### Usage

```
cg_ln(x, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

log

## cg_log10 *Logarithmic Base 10*

### Description

Calculate `log10(x)`.

### Usage

```
cg_log10(x, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

log10

## cg_log2 *Logarithm Base 2*

### Description

Calculate `log2(x)`.

### Usage

```
cg_log2(x, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

log

---

`cg_matmul` *Matrix Multiplication*

---

### Description

Calculate `x %*% y`.

### Usage

```
cg_matmul(x, y, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric matrix. |
| `y` | cg.node, placeholder for a numeric matrix. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

matmult

## cg_max *Maxima*

### Description

Calculate `max(x)`.

### Usage

```
cg_max(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Note

Function max is called without setting argument `na.rm`.

### Author(s)

Ron Triepels

### See Also

max

## cg_mean *Arithmetic Mean*

### Description

Calculate `mean(x)`.

### Usage

```
cg_mean(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Note

Function mean is called without setting argument `trim` and `na.rm`.

### Author(s)

Ron Triepels

### See Also

mean

---

`cg_min` *Minima*

---

### Description

Calculate `min(x)`.

### Usage

```
cg_min(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Note

Function min is called without setting argument `na.rm`.

### Author(s)

Ron Triepels

### See Also

min

## cg_mul *Multiply*

### Description

Calculate `x * y` element-wise.

### Usage

```
cg_mul(x, y, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| y | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

multiply

## cg_neg *Negative*

### Description

Calculate `-x`.

### Usage

```
cg_neg(x, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

negative

---

`cg_not` *Not*

---

### Description

Calculate `!x`.

### Usage

```
cg_not(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

not

## cg_not_equal *Not equal*

### Description

Calculate `x != y`.

### Usage

```
cg_not_equal(x, y, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `y` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

not equal

## cg_opr *Add Operation*

### Description

Add an operation node to the graph.

### Arguments

| | |
|---|---|
| `call` | symbol, operation performed by the node. |
| `grads` | list of symbols, gradients functions of the input nodes that are consumed by the operation in argument `call`. |
| `args` | list of cg_node objects, the nodes that are consumed by the operation in argument `call`. |
| `name` | character scalar, name of the node (optional). In case argument `name` is missing, the node is tried to be added to the graph under an auto-generated name. |

### Details

```
$opr(call, grads, binding, name = NULL)
```

### Value

cg_node, operation.

### Note

The operation to be performed by the node should be provided as a symbol to argument `call`. If this operation consumes any other nodes in the graph, then the gradient function of the operation with respect to these input nodes should be supplied as a symbol to argument `gradients`. These gradients must be a function of each input's gradient. A gradient function must be provided for each input node as specified by argument `args`.

There is a wrapper function opr that calls this method on the current active graph.

### Author(s)

Ron Triepels

---

## cg_parm *Add Parameter*

---

### Description

Add a parameter node to the graph.

### Arguments

| | |
|---|---|
| `value` | numeric vector or array, value of the node (optional). |
| `name` | character scalar, name of the node (optional). In case `name` is missing, the node is tried to be added to the graph under an auto-generated name. |

### Details

```
$parm(value = NULL, name = NULL)
```

### Value

cg_node, parameter.

### Note

Parameters are assumed to be subject to some optimization process. Hence, their value might change over time.

There is a wrapper function parm that calls this method on the current active graph.

### Author(s)

Ron Triepels

### Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add a parameter with value 1 and name 'p' to the graph.
x$parm(1, name = "p")
```

---

## cg_plot *Plot*

---

### Description

Plot the topology of the graph.

### Arguments

| | |
|---|---|
| ... | additional arguments that can be passed on to the plot function of the Rgraphiz package. |

### Details

```
$plot(...)
```

### Value

none.

### Note

A visual representation of the graph might be usefull for debugging purposes. This functions requires the Rgraphviz package.

### Author(s)

Ron Triepels

## cg_pmax *Parallel Maxima*

### Description

Calculate `pmax(x, y)`.

### Usage

```
cg_pmax(x, y, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `y` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Note

Function pmax is called without setting argument `na.rm`.

### Author(s)

Ron Triepels

### See Also

pmax

## cg_pmin *Parallel Minima*

### Description

Calculate `pmin(x, y)`.

### Usage

```
cg_pmin(x, y, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| y | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Note

Function pmin is called without setting argument na.rm.

### Author(s)

Ron Triepels

### See Also

pmin

---

## cg_pos *Positive*

---

### Description

Calculate x.

### Usage

```
cg_pos(x, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

positive

## cg_pow *Power*

### Description

Calculate `x ^ y`.

### Usage

```
cg_pow(x, y, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| y | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

power

## cg_prod *Product of Vector Elements*

### Description

Calculate `prod(x)`.

### Usage

```
cg_prod(x, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Note

In contrast to the base prod function, this function only accepts a single vector or array.

Function prod is called without setting argument `na.rm`.

### Author(s)

Ron Triepels

### See Also

prod

---

`cg_reshape` *Reshape Array Dimensions*

---

### Description

Change the dimensions of array `x` to `dim`.

### Usage

```
cg_reshape(x, dim, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `dim` | cg.node, placeholder for a numeric scalar or vector, the dimensions of the new array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Note

The elements of `x` are re-arranged column-wise by base function array.

### Author(s)

Ron Triepels

### See Also

array

## cg_rowMeans *Row Means*

### Description

Calculate `rowMeans(x)`.

### Usage

```
cg_rowMeans(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Note

Function rowMeans is called without setting argument `na.rm` and `dims`.

### Author(s)

Ron Triepels

### See Also

rowMeans

## cg_rowSums *Row Sums*

### Description

Calculate `rowSums(x)`.

### Usage

```
cg_rowSums(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Note

Function rowSums is called without setting argument `na.rm` and `dims`.

### Author(s)

Ron Triepels

### See Also

rowSums

---

`cg_run` *Evaluate the Graph*

---

### Description

Evaluate node `name` in the graph.

### Arguments

| | |
|---|---|
| `name` | character scalar, name of the node that is evaluated. |
| `values` | named list or environment, values that are subsituted for the nodes in the graph. |

### Details

```
$run(name, values = new.env())
```

### Value

environment, the value of node `name` including the value of all ancestors of `name`.

### Note

All nodes required to compute node `name` must have a value or their value must be able to be computed at run-time. Nodes can be assigned a value when they are created or by calling method cg_set. Alternatively, argument `values` can be used to substitute values for nodes that do not have a value (e.g. inputs) or to fix their values.

Only those nodes needed to compute node `name` are evaluated and their values are returned. Values of operation nodes that are cached by function cg_val are ignored and re-computed.

There is a wrapper function run that calls this method on the current active graph.

### Author(s)

Ron Triepels

### Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add an input.
a <- x$input(name = "a")

# Square the input (i.e. b = a^2).
b <- cg_pow(a, x$const(2), name = "b")

# Evaluate b at a = 2.
values <- x$run(b, list(a = 2))

# Retrieve the value of b.
values$b
```

---

## cg_set *Change the Value of a Node in the Graph*

---

### Description

Change the value of node `name` in the graph.

### Arguments

| | |
|---|---|
| `name` | character scalar, name of the node that is changed. |
| `value` | R object, new value of the node. |

### Details

```
$set(name, value)
```

### Value

nothing.

### Note

The cached value of all nodes that directly or indirectly dependend on node `name` is removed. The value of these nodes will be re-computed the next time cg_val is called.

### Author(s)

Ron Triepels

## Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add a parameter
a <- x$parm(20, name = "a")

# Change value of a
x$set(a, 40)

# Evaluate a
x$val(a)
```

---

`cg_sigmoid` *Sigmoid*

---

## Description

Calculate `sigmoid(x)`.

## Usage

```
cg_sigmoid(x, name = NULL)
```

## Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

## Value

cg.node, node of the operation.

## Author(s)

Ron Triepels

## cg_sin *Sine*

### Description

Calculate `sin(x)`.

### Usage

```
cg_sin(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

sin

## cg_sinh *Hyperbolic Sine*

### Description

Calculate `sinh(x)`.

### Usage

```
cg_sinh(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

## Value

cg.node, node of the operation.

## Author(s)

Ron Triepels

## See Also

sinh

---

`cg_sqrt` *Square Root*

---

## Description

Calculate `sqrt(x)`.

## Usage

```
cg_sqrt(x, name = NULL)
```

## Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

## Value

cg.node, node of the operation.

## Author(s)

Ron Triepels

## See Also

sqrt

## cg_sub *Subtract*

### Description

Calculate `x - y`.

### Usage

```
cg_sub(x, y, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| y | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

subtract

## cg_sum *Sum of Vector Elements*

### Description

Calculate `sum(x)`.

### Usage

```
cg_sum(x, name = NULL)
```

### Arguments

| | |
|---|---|
| x | cg.node, placeholder for a numeric vector or array. |
| name | character scalar, name of the operation (optional). |

## Value

cg.node, node of the operation.

## Note

In contrast to the base sum function, this function only accepts a single vector or array.

Function sum is called without setting argument `na.rm`.

## Author(s)

Ron Triepels

## See Also

sum

---

`cg_t` *Matrix Transpose*

---

## Description

Perform `t(x)`.

## Usage

```
cg_t(x, name = NULL)
```

## Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric matrix. |
| `name` | character scalar, name of the operation (optional). |

## Value

cg.node, node of the operation.

## Author(s)

Ron Triepels

## See Also

t

## cg_tan *Tangent*

### Description

Calculate `tan(x)`.

### Usage

```
cg_tan(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

### Value

cg.node, node of the operation.

### Author(s)

Ron Triepels

### See Also

tan

## cg_tanh *Hyperbolic Tangent*

### Description

Calculate `tanh(x)`.

### Usage

```
cg_tanh(x, name = NULL)
```

### Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric vector or array. |
| `name` | character scalar, name of the operation (optional). |

## Value

cg.node, node of the operation.

## Author(s)

Ron Triepels

## See Also

tanh

---

`cg_tcrossprod` *Transpose Matrix Crossproduct*

---

## Description

Calculate `tcrossprod(x, y)`.

## Usage

```
cg_tcrossprod(x, y = NULL, name = NULL)
```

## Arguments

| | |
|---|---|
| `x` | cg.node, placeholder for a numeric matrix. |
| `y` | cg.node, placeholder for a numeric matrix (optional). |
| `name` | character scalar, name of the operation (optional). |

## Value

cg.node, node of the operation.

## Author(s)

Ron Triepels

## See Also

tcrossprod

cg_val *Evaluate a Node in the Graph*

## Description

Evaluate node name in the graph.

## Arguments

| | |
|---|---|
| name | character scalar, name of the node that is evaluated. |

## Details

$val(name)

## Value

R object, the value of the node.

## Note

The values of all nodes are cached for performance reasons. Only those nodes needed to compute node name and that have not yet been retrieved by cg_val are computed.

## Author(s)

Ron Triepels

## Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add a parameter
a <- x$parm(20, name = "a")

# Evaluate a
x$val(a)
```

`const` *Add Constant*

## Description

Add a constant node to the active graph.

## Usage

```
const(value = NULL, name = NULL)
```

## Arguments

| | |
|---|---|
| `value` | numeric vector or array, value of the node (optional). |
| `name` | character scalar, name of the node (optional). In case argument `name` is missing, the node is tried to be added to the graph under an auto-generated name. |

## Value

cg_node, constant.

## Note

Constants are ignored when differentiating a graph. The intended use of constants is that they are given a fixed value. However, it is still possible to change the value of constants when evaluating or differentiating a graph (see run and gradients for more details).

## Author(s)

Ron Triepels

## Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add a constant with value 1 and name 'c' to the graph.
const(1, name = "c")
```

`gradients` *Calculate Gradients*

## Description

Differentiate the active graph with respect to node `name` by reverse automatic differentiation.

## Usage

```
gradients(name, values = new.env(), index = 1)
```

## Arguments

| | |
|---|---|
| `name` | character scalar, name of the node that is differentiated. |
| `values` | named list or environment, values that are subsituted for the nodes in the graph. |
| `index` | numeric scalar, index of the target node that is differentiated. Defaults to the first element. |

## Value

environment, the gradients of all nodes with respect to target node `name`.

## Note

All nodes required to compute node `name` must have a value, or their value must be able to be computed at run-time. The values of nodes can be obtained by first evaluating node `name` in the graph using function run. The values obtained by this function for the nodes can then be supplied to argument `values`.

Currently, the cgraph package can only differentiate scalar target nodes. In case the value of target node `name` is a vector or an array, argument `index` can be used to specify which element of the vector or array is to be differentiated.

The gradients of all ancestors or `name` are returned. Constant nodes are not differentiated and their gradients are not returned. The gradients have the same shape as the values of the nodes.

## Author(s)

Ron Triepels

## Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add some parameters.
a <- parm(2, name = "a")
b <- parm(4, name = "b")
```

```
# Perform some operations on the parameters.
c <- cg_sin(a) + cg_cos(b) - cg_tan(a)

# Differentiate the graph with respect to c.
grads <- gradients(c, run(c))

# Retrieve the gradient of c with respect to a.
grads$a
```

---

`input` *Add Input*

---

## Description

Add an input node to the active graph.

## Usage

```
input(value = NULL, name = NULL)
```

## Arguments

| | |
|---|---|
| `value` | numeric vector or array, value of the node (optional). |
| `name` | character scalar, name of the node (optional). In case argument `name` is missing, the node is tried to be added to the graph under an auto-generated name. |

## Value

cg_node, input.

## Note

The intended use of inputs is that they are not given a fixed value but behave as placeholders. Values can be supplied for inputs when evaluating or differentiating a graph (see run and gradients for more details).

## Author(s)

Ron Triepels

## Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add an input with name 'x' to the graph.
input(name = "x")
```

opr *Add Operation*

## Description

Add an operation node to the active graph.

## Usage

```
opr(call, grads, args, name = NULL)
```

## Arguments

| | |
|---|---|
| call | symbol, operation performed by the node. |
| grads | list of symbols, gradients functions of the input nodes that are consumed by the operation in argument call. |
| args | list of cg_node objects, the nodes that are consumed by the operation in argument call. |
| name | character scalar, name of the node (optional). In case argument name is missing, the node is tried to be added to the graph under an auto-generated name. |

## Value

cg_node, operation.

## Note

The operation to be performed by the node should be provided as a symbol to argument call. If this operation consumes any other nodes in the graph, then the gradient function of the operation with respect to these input nodes should be supplied as a symbol to argument gradients. These gradients must be a function of each input's gradient. A gradient function must be provided for each input node as specified by argument args.

## Author(s)

Ron Triepels

parm *Add Parameter*

## Description

Add a parameter node to the active graph.

## Usage

```
parm(value = NULL, name = NULL)
```

## Arguments

| | |
|---|---|
| value | numeric vector or array, value of the node (optional). |
| name | character scalar, name of the node (optional). In case name is missing, the node is tried to be added to the graph under an auto-generated name. |

## Value

cg_node, parameter.

## Note

Parameters are assumed to be subject to some optimization process. Hence, their value might change over time.

## Author(s)

Ron Triepels

## Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add a parameter with value 1 and name 'p' to the graph.
parm(1, name = "p")
```

run *Evaluate the Graph*

## Description

Evaluate node name in the active graph.

## Usage

```
run(name, values = list())
```

## Arguments

| | |
|---|---|
| name | character scalar, name of the node that is evaluated. |
| values | named list or environment, values that are subsituted for the nodes in the graph. |

## Value

environment, the value of node name including the value of all ancestors of name.

## Note

All nodes required to compute node name must have a value or their value must be able to be computed at run-time. Nodes can be assigned a value when they are created or by calling method set. Alternatively, argument values can be used to substitute values for nodes that do not have a value (e.g. inputs) or to fix their values.

Only those nodes needed to compute node name are evaluated and their values are returned. Values of operation nodes that are cached by function val are ignored and re-computed.

## Author(s)

Ron Triepels

## Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add an input.
a <- input(name = "a")

# Square the input (i.e. b = a^2).
b <- cg_pow(a, const(2), name = "b")

# Evaluate b at a = 2.
values <- run(b, list(a = 2))

# Retrieve the value of b.
```

```
values$b
```

---

`set` *Change the Value of a Node in the Graph*

---

## Description

Change the value of node `name` in the active graph.

## Usage

```
set(name, value)
```

## Arguments

| | |
|---|---|
| `name` | character scalar, name of the node that is changed. |
| `value` | R object, new value of the node. |

## Value

nothing.

## Note

The cached value of all nodes that directly or indirectly dependend on node `name` is removed. The value of these nodes will be re-computed the next time val is called.

## Author(s)

Ron Triepels

## Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add a parameter
a <- parm(20, name = "a")

# Change value of a
set(a, 40)

# Evaluate a
val(a)
```

`val` *Evaluate a Node in the Graph*

## Description

Evaluate node `name` in the active graph.

## Usage

```
val(name)
```

## Arguments

| | |
|---|---|
| `name` | character scalar, name of the node that is evaluated. |

## Value

R object, the value of the node.

## Note

The values of all nodes are cached for performance reasons. Only those nodes needed to compute node `name` and that have not yet been retrieved by val are computed.

## Author(s)

Ron Triepels

## Examples

```
# Initialize a new computational graph.
x <- cgraph$new()

# Add a parameter
a <- parm(20, name = "a")

# Evaluate a
val(a)
```

# Index