Package: sparsediff (via r-universe)

June 4, 2026

Type Package

Title R Interface to the 'SparseDiffEngine' Sparse Differentiation Backend

Version 0.4.0

Description Bindings for the 'SparseDiffEngine' C library, the sparse Jacobian and Hessian differentiation backend used by 'CVXPY' for its Disciplined Nonlinear Programming (DNLP) extension. Provides low-level routines for building nonlinear expression graphs and evaluating sparse derivatives, intended as a backend for higher-level modeling layers such as 'CVXR'. This is the R analog of the 'sparsediffpy' Python package and wraps the same C library.

License Apache License (== 2.0)

Copyright file inst/COPYRIGHTS

URL https://bnaras.github.io/sparsediff/, https://github.com/bnaras/sparsediff

BugReports https://github.com/bnaras/sparsediff/issues

Encoding UTF-8

SystemRequirements GNU make

LinkingTo cpp11

Suggests cpp11, knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation yes

RoxygenNote 7.3.3

Config/pak/sysreqs make

Repository https://bnaras.r-universe.dev

Date/Publication 2026-06-03 04:40:49 UTC

RemoteUrl https://github.com/bnaras/sparsediff

RemoteRef HEAD

RemoteSha a2b5cf95d239abd4f311d868307f1d5ea2b681cc

Contents

`engine_version` *Bundled SparseDiffEngine version*

Description

Returns the version string of the SparseDiffEngine C library bundled with this package.

Usage

```
engine_version()
```

Value

A character scalar, e.g. `"0.3.0"`.

Examples

```
engine_version()
```

`sparsediff-affine` *Affine and shape atoms*

Description

Affine combinations and shape manipulations of expressions. These have constant (zero) second derivatives but participate in the Jacobian.

Arguments

`left`, `right`, `child`, `c`	expression handles.
`d1`, `d2`	target row and column dimensions (for `sd_promote`, `sd_reshape`, `sd_broadcast`, `sd_index`).
`indices`	0-based column-major flat indices selected by `sd_index`.
`args`	a list of expression handles to stack (`sd_hstack`, `sd_vstack`).
`n_vars`	total number of variables in the problem.
`axis`	reduction axis for `sd_sum`: `-1` (all entries), `0` (down rows) or `1` (across columns).

Details

`sd_add` elementwise sum of two expressions.

`sd_sum` sum reduction along `axis`.

`sd_trace` matrix trace.

`sd_transpose` matrix transpose.

`sd_diag_vec` diagonal matrix from a vector.

`sd_diag_mat` diagonal vector from a matrix.

`sd_upper_tri` the strict upper-triangular entries.

`sd_promote` promote a scalar to shape `d1 x d2`.

`sd_reshape` reshape to `d1 x d2` (column-major).

`sd_broadcast` broadcast to `d1 x d2`.

`sd_index` select entries by 0-based flat `indices`.

`sd_hstack`, `sd_vstack` horizontal / vertical stacking.

Value

An expression handle.

See Also

`sparsediff-elementwise`, `sparsediff-matrix`

sparsediff-bivariate *Bivariate atoms*

Description

Functions of two expression arguments.

Arguments

`l`, `r`, `x`, `y`	expression handles.

Details

`sd_elementwise_mult` elementwise (Hadamard) product.

`sd_matmul` matrix product xy.

`sd_quad_over_lin` the quadratic-over-linear $\|x\|^2/y$.

`sd_rel_entr` elementwise relative entropy $x \log(x/y)$.

`sd_rel_entr_first_scalar`, `sd_rel_entr_second_scalar` relative entropy with a scalar first or second argument broadcast against the other.

Value

An expression handle.

See Also

`sparsediff-elementwise`, `sparsediff-reduction`

sparsediff-elementwise *Elementwise atoms*

Description

Smooth elementwise functions of a single expression. Each returns an expression of the same shape as its argument.

Arguments

`child`, `c`	an expression handle (the argument).
`p`	exponent for `sd_power`.

Details

`sd_exp`, `sd_log` exponential and natural logarithm.

`sd_sin`, `sd_cos`, `sd_tan` trigonometric functions.

`sd_sinh`, `sd_tanh`, `sd_asinh`, `sd_atanh` hyperbolic and inverse-hyperbolic functions.

`sd_logistic` the logistic $\log(1 + e^x)$.

`sd_xexp` xe^x.

`sd_normal_cdf` the standard normal CDF.

`sd_entr` the elementwise entropy $-x \log x$.

`sd_power` the power x^p.

`sd_neg` negation $-x$.

Value

An expression handle.

See Also

`sparsediff-affine`, `sparsediff-bivariate`

`sparsediff-leaves` *Leaf expressions: variables and parameters*

Description

Create the leaves of an expression graph. A variable is a slice of the differentiation vector; a parameter is fixed data that can be updated between evaluations (see `sd_register_params`).

Arguments

`d1`, `d2`	row and column dimensions of the leaf.
`var_id`	0-based flat (column-major) offset of this variable's first entry within the primal vector `u`.
`param_id`	0-based parameter offset, analogous to `var_id`.
`n_vars`	total number of variables in the problem.
`values`	numeric data for the parameter (length `d1 * d2`, column-major).

Value

An expression handle.

See Also

`sparsediff-elementwise`, `sd_problem`

sparsediff-matrix *Parameter- and constant-matrix atoms*

Description

Operations that combine an expression with fixed data — a registered parameter node or a constant matrix — so the data can flow through the differentiated graph (and, for parameters, be updated between evaluations).

Arguments

`param`	a parameter expression handle (see `sd_parameter`).
`child`	an expression handle (the variable argument).
`Qp`, `Qi`, `Qx`	the column-pointer, row-index and value arrays of a compressed-sparse-column matrix Q (as in a `Matrix::dgCMatrix`: `@p`, `@i`, `@x`) for `sd_quad_form`'s $x^\top Q x$.
`Ap`, `Ai`, `Ax`	the compressed-sparse-column arrays of a constant matrix A for the sparse matrix products.
`ncol`	number of columns of the sparse constant matrix A.
`m`, `n`	row and column dimensions of the dense constant matrix.
`data`	the dense constant-matrix entries (length `m * n`, column-major).

Details

`sd_scalar_mult`, `sd_vector_mult` multiply a child by a scalar / vector parameter.

`sd_convolve` convolution of a parameter kernel with a child.

`sd_quad_form` the quadratic form $x^\top Q x$ with sparse constant Q.

`sd_left_matmul`, `sd_right_matmul` left / right product with a sparse constant matrix A.

`sd_left_matmul_dense`, `sd_right_matmul_dense` left / right product with a dense constant matrix.

Value

An expression handle.

See Also

`sd_parameter`, `sd_register_params`

sparsediff-oracle *Sparse derivative oracle*

Description

Initialise and evaluate the value, gradient, sparse constraint Jacobian and sparse lower-triangular Lagrangian Hessian of a problem built with sd_problem.

Arguments

prob	a problem handle from sd_problem.
u	a numeric vector: the primal point at which to evaluate, laid out in the engine's column-major flat ordering (the same ordering used by the var_id offsets passed to sd_variable).
obj_w	a scalar weight σ multiplying the objective Hessian.
w	a numeric vector of constraint multipliers (length equal to the total constraint size) weighting the constraint Hessians.

Details

Evaluation is ordered: a forward pass first (sd_objective_forward / sd_constraint_forward) populates the node values at u, after which sd_gradient, sd_jacobian_values and sd_hessian_values read them. Sparsity patterns are structural — fixed once the corresponding sd_init_* routine has run — so they are queried once and reused, while the values are recomputed at each new point. Row and column indices are 0-based (the engine convention; a higher-level modelling layer such as **CVXR** translates them to 1-based as needed).

Value

sd_init_derivatives, sd_init_jacobian, sd_init_jacobian_coo, sd_init_hessian_coo called for their side effect; return NULL invisibly.

sd_objective_forward the scalar objective value at u.

sd_constraint_forward the constraint vector at u.

sd_gradient the objective gradient, length n_vars.

sd_jacobian_sparsity, sd_hessian_sparsity a list with integer rows/cols (0-based COO indices) and nrow/ncol.

sd_jacobian_values the constraint-Jacobian nonzeros, matching the Jacobian sparsity order.

sd_hessian_values the nonzeros of $\sigma\nabla^2 f + \sum_i w_i \nabla^2 g_i$, lower triangle, matching the Hessian sparsity order.

See Also

sd_problem

`sparsediff-problem` *Assemble a differentiable problem*

Description

Combine an objective expression and a list of constraint expressions (built with the `sd_*` atom constructors) into a single problem object whose value and sparse derivatives can be evaluated repeatedly.

Arguments

`objective`	an expression handle for the scalar objective.
`constraints`	a list of expression handles, stacked vertically to form the constraint vector $g(x)$ (may be empty).
`verbose`	logical; if `TRUE`, print diagnostic information while the problem is assembled.
`prob`	a problem handle returned by `sd_problem()`.
`params`	a list of parameter expression handles (see `sd_parameter`) to register for fast re-evaluation under changing data.
`theta`	a numeric vector of new parameter values, concatenated in the order the parameters were registered.

Details

The typical lifecycle is: build expressions with the `sd_*` constructors, assemble them with `sd_problem()`, initialise the derivative structures once (`sd_init_derivatives` and friends), then evaluate the objective/constraints and their sparse derivatives at as many primal points as needed. When the problem has parameters, register them once with `sd_register_params()` and push new values with `sd_update_params()` to re-evaluate without rebuilding the graph (the DPP-style fast path).

Value

`sd_problem()` returns an external-pointer problem handle. The handle owns the underlying expression graph and frees it when garbage collected. `sd_register_params()` and `sd_update_params()` are called for their side effect and return `NULL` invisibly.

See Also

`sd_init_derivatives` for the evaluation oracle, `sd_variable` and the atom constructors for building expressions.

`sparsediff-reduction` *Product-reduction atoms*

Description

Multiplicative reductions of an expression.

Arguments

`c`	an expression handle.

Details

`sd_prod` product of all entries.

`sd_prod_axis_zero` column-wise products (reduce down rows).

`sd_prod_axis_one` row-wise products (reduce across columns).

Value

An expression handle.

See Also

`sparsediff-affine`

Index