Package ‘sanic’

September 22, 2020

Type Package

Title Solving Ax = b Nimbly in C++

Version 0.0.1

Date 2020-09-04

Author Nikolas Kuschnig [aut, cre] (<https://orcid.org/0000-0002-6642-2543>)

Maintainer Nikolas Kuschnig <nikolas.kuschnig@wu.ac.at>

Description Routines for solving large systems of linear equations in R.
Direct and iterative solvers from the Eigen C++ library are made available.
Solvers include Cholesky, LU, QR, and Krylov subspace methods (Conjugate
Gradient, BiCGSTAB). Both dense and sparse problems are supported.

URL https://github.com/nk027/sanic

BugReports https://github.com/nk027/sanic/issues

Depends R (>= 3.3.0)

Imports Rcpp (>= 1.0.5), Matrix, methods

License GPL-3

Encoding UTF-8

LinkingTo Rcpp, RcppEigen

RoxygenNote 7.1.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-09-22 08:40:03 UTC

R topics documented:

sanic *Solving Ax = b Nimbly in C++*

Description

Routines for solving large systems of linear equations in R. Direct and iterative solvers from the Eigen C++ library are made available. Solvers include Cholesky, LU, QR, and Krylov subspace methods (Conjugate Gradient, BiCGSTAB). Both dense and sparse problems are supported.

solve_cg *Solve a System of Equations using Iterative Methods*

Description

Function to use Conjugate Gradient (CG) methods to solver systems of equations.

Usage

```
solve_cg(
  a,
  b,
  x0,
  type = c("BiCGSTAB", "LSCG", "CG"),
  tol,
  iter,
  verbose = FALSE
)
```

Arguments

`a`	Square numeric matrix with the coefficients of the linear system. Both dense and sparse matrices are supported (see sparsify).
`b`	Numeric vector or matrix at the right-hand side of the linear system. If missing, 'b' is set to an identity matrix and 'a' is inverted.
`x0`	Numeric vector or matrix with an initial guess. Must be of the same dimension as 'b'.
`type`	Character scalar. Whether to use the BiCGSTAB, least squares CG or classic CG method.
`tol`	Numeric scalar with the desired tolerance. Defaults to the machine precision.
`iter`	Integer scalar with the maximum number of iterations. Defaults to the theoretical maximum, i.e. the number of columns in 'a'.
`verbose`	Logical scalar. Whether to print iterations and tolerance.

Value

Solves for x and returns a numeric matrix with the results.

Examples

```
# Solve via least squares or bi-conjugate gradient methods
A <- matrix(rnorm(9), nrow = 3, ncol = 3)
# The matrix A should be of class 'dgCMatrix' (otherwise it is converted)
A <- sparsify(A)
x <- rnorm(3)
b  <- A %*% x

x_bi <- solve_cg(A, b)
x_ls <- solve_cg(A, b, type = "LS")

# Solve via conjugate gradient for symmetric matrices
AA <- A %*% A
b <- AA %*% x
x_cg <- solve_cg(AA, b, type = "CG")
```

solve_chol *Solve a System of Equations Using Direct Methods*

Description

Functions to access specific direct solvers for systems of equations.

Usage

```
solve_chol(a, b)

solve_lu(a, b)

solve_qr(a, b)
```

Arguments

a	Square numeric matrix with the coefficients of the linear system. Both dense and sparse matrices are supported (see sparsify).
b	Numeric vector or matrix at the right-hand side of the linear system. If missing, 'b' is set to an identity matrix and 'a' is inverted.

Value

Solves for x and returns a numeric matrix with the results.

Examples

```
# Solve via LU and QR for general matrices
A <- matrix(rnorm(9), nrow = 3, ncol = 3)
x <- rnorm(3)
b  <- A %*% x

x_lu <- solve_lu(A, b)
x_qr <- solve_qr(A, b)

# Solve via Cholesky for symmetric matrices
AA <- crossprod(A)
b <- AA %*% x

x_chol <- solve_chol(AA, b)

# Sparse methods are available for the 'dgCMatrix' class from Matrix
x_slu <- solve_lu(sparsify(A), b)
```

sparsify *Transform a Matrix to Be Sparse.*

Description

Concise function to transform dense to sparse matrices of class `dgCMatrix` (see sparseMatrix).

Usage

```
sparsify(x)
```

Arguments

`x`	Numeric matrix to transform to a sparse 'dgCMatrix'.

Value

Returns 'x' as `dgCMatrix`.

Examples

```
sparsify(matrix(rnorm(9L), 3L))
```

Index