Package ‘cPCG’

January 11, 2019

Type  Package
Title  Efficient and Customized Preconditioned Conjugate Gradient Method for Solving System of Linear Equations
Version 1.0
Date 2018-12-30
Author Yongwen Zhuang
Maintainer Yongwen Zhuang <zyongwen@umich.edu>
Depends R (>= 3.0.0)
License GPL (>= 2)
Imports Rcpp (>= 0.12.19)
LinkingTo Rcpp, RcppArmadillo
RoxygenNote 6.1.1
Encoding UTF-8
Suggests knitr, rmarkdown
VignetteBuilder knitr
NeedsCompilation yes
Repository CRAN
Date/Publication 2019-01-11 17:00:10 UTC

R topics documented:

- cPCG-package ................................................................. 2
- cgsolve ................................................................. 3
- icc ................................................................. 4
- pcgsolve ................................................................. 5

Index 7
Description


Details

Functions in this package serve the purpose of solving for \( x \) in \( Ax = b \), where \( A \) is a symmetric and positive definite matrix, \( b \) is a column vector.

To improve scalability of conjugate gradient methods for larger matrices, the Armadillo templated C++ linear algebra library is used for the implementation. The package also provides flexibility to have user-specified preconditioner options to cater for different optimization needs.

The DESCRIPTION file:

Package: cPCG
Type: Package
Title: Efficient and Customized Preconditioned Conjugate Gradient Method for Solving System of Linear Equations
Version: 1.0
Date: 2018-12-30
Author: Yongwen Zhuang
Maintainer: Yongwen Zhuang <zyongwen@umich.edu>
Depends: R (>= 3.0.0)
License: GPL (>= 2)
Imports: Rcpp (>= 0.12.19)
LinkingTo: Rcpp, RcppArmadillo
RoxygenNote: 6.1.1
Encoding: UTF-8
Suggests: knitr, rmarkdown
VignetteBuilder: knitr

Index of help topics:

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cPCG-package</td>
<td>Efficient and Customized Preconditioned Conjugate Gradient Method for Solving System of Linear Equations</td>
</tr>
<tr>
<td>cgssolve</td>
<td>Conjugate gradient method</td>
</tr>
<tr>
<td>icc</td>
<td>Incomplete Cholesky Factorization</td>
</tr>
<tr>
<td>pcgsolve</td>
<td>Preconditioned conjugate gradient method</td>
</tr>
</tbody>
</table>
Author(s)

Yongwen Zhuang

References


Examples

```r
# generate test data
test_A <- matrix(c(4,1,1,3), ncol = 2)
test_b <- matrix(1:2, ncol = 1)

cgsolve(test_A, test_b, 1e-6, 1000)

# conjugate gradient method solver
pcgsolve(test_A, test_b, "icc")
```

Description

Conjugate gradient method for solving system of linear equations \( Ax = b \), where \( A \) is symmetric and positive definite, \( b \) is a column vector.

Usage

```
$cgsolve(A, b, tol = 1e-6, maxIter = 1000)$
```

Arguments

- \( A \): matrix, symmetric and positive definite.
- \( b \): vector, with same dimension as number of rows of \( A \).
- \( tol \): numeric, threshold for convergence, default is \( 1e-6 \).
- \( maxIter \): numeric, maximum iteration, default is \( 1000 \).
Details

The idea of conjugate gradient method is to find a set of mutually conjugate directions for the unconstrained problem

$$\arg\min_x f(x)$$

where $$f(x) = 0.5b^TAb - bx + z$$ and $$z$$ is a constant. The problem is equivalent to solving $$Ax = b$$.

This function implements an iterative procedure to reduce the number of matrix-vector multiplications [1]. The conjugate gradient method improves memory efficiency and computational complexity, especially when $$A$$ is relatively sparse.

Value

Returns a vector representing solution $$x$$.

Warning

Users need to check that input matrix $$A$$ is symmetric and positive definite before applying the function.

References


See Also

pcgsolve

Examples

```r
## Not run:
test_A <- matrix(c(4,1,1,3), ncol = 2)
test_b <- matrix(c(1:2, ncol = 1)
cgsolve(test_A, test_b, 1e-6, 1000)
## End(Not run)
```

### icc

**Incomplete Cholesky Factorization**

Description

Incomplete Cholesky factorization method to generate preconditioning matrix for conjugate gradient method.

Usage

icc(A)
**pcgsolve**

**Arguments**
- \( A \)  
  matrix, symmetric and positive definite.

**Details**
Performs incomplete Cholesky factorization on the input matrix \( A \), the output matrix is used for preconditioning in `pcgsolve()` if "ICC" is specified as the preconditioner.

**Value**
Returns a matrix after incomplete Cholesky factorization.

**Warning**
Users need to check that input matrix \( A \) is symmetric and positive definite before applying the function.

**See Also**
- `pcgsolve`

**Examples**

```r
## Not run:
test_A <- matrix(c(4,1,1,3), ncol = 2)
out <- icc(test_A)
## End(Not run)
```

---

**pcgsolve**  
*Preconditioned conjugate gradient method*

**Description**
Preconditioned conjugate gradient method for solving system of linear equations \( Ax = b \), where \( A \) is symmetric and positive definite, \( b \) is a column vector.

**Usage**

```r
pcgsolve(A, b, preconditioner = "Jacobi", tol = 1e-6, maxIter = 1000)
```

**Arguments**
- \( A \)  
  matrix, symmetric and positive definite.
- \( b \)  
  vector, with same dimension as number of rows of \( A \).
- preconditioner  
  string, method for preconditioning: "Jacobi" (default), "SSOR", or "ICC".
- tol  
  numeric, threshold for convergence, default is 1e-6.
- maxIter  
  numeric, maximum iteration, default is 1000.
Details

When the condition number for $A$ is large, the conjugate gradient (CG) method may fail to converge in a reasonable number of iterations. The Preconditioned Conjugate Gradient (PCG) Method applies a precondition matrix $C$ and approaches the problem by solving:

$$C^{-1}Ax = C^{-1}b$$

where the symmetric and positive-definite matrix $C$ approximates $A$ and $C^{-1}A$ improves the condition number of $A$.

Common choices for the preconditioner include: Jacobi preconditioning, symmetric successive over-relaxation (SSOR), and incomplete Cholesky factorization [2].

Value

Returns a vector representing solution $x$.

Preconditioners

Jacobi: The Jacobi preconditioner is the diagonal of the matrix $A$, with an assumption that all diagonal elements are non-zero.

SSOR: The symmetric successive over-relaxation preconditioner, implemented as $M = (D+L)D^{-1}(D+L)^T$. [1]

ICC: The incomplete Cholesky factorization preconditioner. [2]

Warning

Users need to check that input matrix $A$ is symmetric and positive definite before applying the function.

References


See Also

cgsolve

Examples

```r
# Not run:
test_A <- matrix(c(4, 1, 1, 3), ncol = 2)
test_b <- matrix(1:2, ncol = 1)
pcgsolve(test_A, test_b, "icc")
```

```r
# End(Not run)```
Index

*Topic **methods**
  - cg.solve, 3
  - icc, 4
  - pcgsolve, 5

*Topic **optimize**
  - cg.solve, 3
  - pcgsolve, 5

*Topic **package**
  - cPCG-package, 2

  cg.solve, 3, 6
  cPCG (cPCG-package), 2
  cPCG-package, 2

  icc, 4

  pcgsolve, 4, 5, 5
  preconditioner (pcgsolve), 5