# cPCG: Efficient and Customized Preconditioned Conjugate Gradient Method

Yongwen Zhuang

**Abstract**

Computing the product of a large matrix inversion and a vector is of special interest in genetic association analysis with the coefficient matrix being a large covariance matrix for more than 100,000 individuals. Since such a computation is equivalent to solving a system of linear equations, a possible solution to the problem is preconditioned conjugate gradient (PCG) algorithm. Existing implementation of the algorithm in R is limited in scalability and flexibility of direction update rules. The 'cPCG' package applies the Armadillo templated C++ linear algebra library for the implementation, and provides flexibility to have user-specified preconditioner options to cater for different optimization needs.

## 1 Introduction: solving system of linear equations

In biostatistical studies, it is a common task to compute the product of a large matrix inversion and a vector, with the matrix being positive definite. For example, when estimating coefficients in a generalized estimating equations (GEE) model for genetic association analysis, the matrix involved is often a genetic relationship matrix (GRM), which is a sample covariance matrix averaged over genetic variations among all individuals. With the development of sequencing technologies and collaboration among multiple study centers, the scale of such matrices has expanded to orders of more than 100,000, making the direct evaluation of the inverse computationally difficult.

Specifically, let $\boldsymbol{\Sigma}$ be a $n \times n$ symmetric, positive definite matrix with real entries, and $\boldsymbol{y}$ be a $n$-vector. The problem is to solve

$$\boldsymbol{x} = \boldsymbol{\Sigma}^{-1}\boldsymbol{y}$$

where $\boldsymbol{\Sigma}$ and $\boldsymbol{y}$ are known.

### 1.1 Conjugate gradient method

As $\boldsymbol{\Sigma}$ is symmetric and positive definite, the above problem is equivalent to the unconstrained optimization problem

$$\arg\min_{\boldsymbol{x}} f(\boldsymbol{x})$$

**Algorithm 1** Conjugate gradient method (pseudo code)

```
r[0] = b - A x[0]
p[0] = r[0]
i = 0
while (i < maxIter):
  a[i] = r[i].t() r[i] / (p[i].t() A p[i])
  x[i+1] = x[i] + a[i] p[i]
  r[i+1] = r[i] - a[i] A p[i]
      if (r[i+1]- r[i]) converge: break
      else:
            beta[i] = r[i+1].t() r[i+1] / (r[i].t() r[i])
            p[i+1] = r[i+1] + beta[i] p[i]
            i++
```

where $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{y}^\top \boldsymbol{\Sigma} \boldsymbol{y} - \boldsymbol{y}\boldsymbol{x} + z$ and $z$ is a constant.

One approach to solving the alternative problem is by using the conjugate gradient (CG) algorithm [2]. The idea is to find a set of directions $\{\boldsymbol{d}_i\}_{i=1}^n$ such that

$$\boldsymbol{d}_i^\top \boldsymbol{\Sigma} \boldsymbol{d}_j = 0, \text{ for any } i, j \in \{1, 2, ..., n\} \text{ and } i \neq j$$

and

$$\alpha_k = \frac{\boldsymbol{d}_k^\top \boldsymbol{y}}{\boldsymbol{d}_k \boldsymbol{\Sigma} \boldsymbol{d}_k}, \ k = 1, 2, ..., n$$

An iterative procedure can be applied to the PCG method to reduce the number of matrix-vector multiplications [7], see Algorithm 1 for the pseudo code of the algorithm. With a relatively sparse $\boldsymbol{\Sigma}$, which is generally the case for matrices like GRMs, the GC method outperforms other methods in memory efficiency and computational complexity.

## 1.2 Preconditioned conjugate gradient method

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

$$\boldsymbol{C}^{-1}\boldsymbol{\Sigma}\boldsymbol{x} = \boldsymbol{C}^{-1}\boldsymbol{y}$$

where the symmetric and positive-definite matrix $\boldsymbol{C}$ approximates $\boldsymbol{\Sigma}$ and $\boldsymbol{C}^{-1}\boldsymbol{\Sigma}$ improves the condition number of $\boldsymbol{\Sigma}$.
There are several common choices for the preconditioner. The Jacobi preconditioner is the diagonal of the matrix $\boldsymbol{\Sigma}$, with an assumption that all diagonal elements are non-zero; the symmetric successive over-relaxation (SSOR) preconditioner is a combination of diagonal

and lower triangular matrix of $\mathbf{\Sigma}$ [8]; and the incomplete Cholesky factorization preconditioner is given by approximating $\mathbf{\Sigma}$ using a sparse lower triangular matrix [3].

## 2 The R package cPCG

Implementation of the CG or PCG method in R is limited in either scalability to larger matrices or flexibility of algorithm to address the specific problem of solving system of linear equations. The cPCG package integrates the Armadillo templated C++ library into R, which enables efficient linear algebra operations. In addition, the package provides flexibility to have preconditioner options to cater for different user-specified optimization needs.

### 2.1 Usage of functions

There are two major functions in the package, cgsolve() and pcgsolve(), corresponding to the conjugate gradient method with or without preconditioning. While users have the option to choose between the two methods, it is recommended that preconditioning be used for large scale matrix computation. Figure 1 shows an example for the usage of the two functions.

The required arguments for both cgsolve() and pcgsolve() are:

| | |
|---|---|
| A | A symmetric and positive definite matrix. |
| b | A numeric vector with the same dimension as number of rows of A. |

Optional arguments for both cgsolve() and pcgsolve() are:

| | |
|---|---|
| tol | A numeric threshold for convergence, default is $10^{-6}$. |
| maxIter | An integer indicating maximum number of iterations, default is 1000. |

Function pcgsolve() has one additional optional argument:

| | |
|---|---|
| preconditioner | A string indicating method for preconditioning:<br>Jacobi [default]: the Jacobi (diagonal) preconditioner;<br>SSOR: the symmetric successive over-relaxation preconditioner;<br>ICC: the incomplete Cholesky factorization preconditioner. |

### 2.2 Performance

To assess the performance of the cPCG package's CG and PCG implementations, the cgsolve() and pcgsolve() function with three preconditioner methods were tested based on simulated data along with an existing implementation of the preconditioned conjugate gradient method (hereafter referred to as benchmark) [4]. A set of symmetric and positive definite matrices were generated with increasing dimensions $n \times n$, $(n \in \{50, 100, 500, 1000\})$, to-

**Figure 1** Simple example using functions in package

```
> library("cPCG")
> test_A <- matrix(c(4,1,1,3), ncol = 2)
> test_b <- matrix(1:2, ncol = 1)
> cgsolve(test_A, test_b) # CG
> pcgsolve(test_A, test_b, "ICC") # PCG with ICC preconditioner
                [,1]
[1,] 0.09074421
[2,] 0.63682348
```

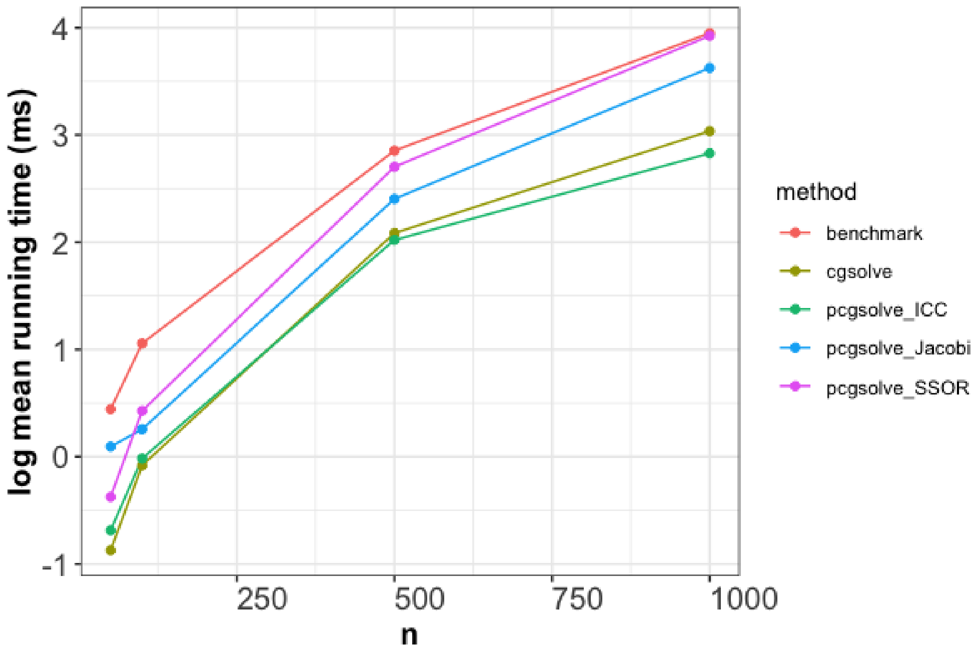

Figure 2: Mean running time of different methods with $n \times n$ matrix

gether with a set of column vectors of dimension $n \times 1$. Running time in milliseconds was averaged over 10 evaluations for each case.

As shown in Figure 2, functions in the cPCG package have better performance over the benchmark method consistently over all evaluation settings. For solving system of linear equations $\boldsymbol{\Sigma x} = \boldsymbol{y}$ where matrix $\boldsymbol{\Sigma}$ has a relatively small dimension, the conjugate gradient method has a faster running time compared to other methods. As the dimension increases, the incomplete Cholesky factorization preconditioned conjugate gradient method outperforms other algorithms being evaluated, which confirms the notion to use preconditioners for larger scale computation.

## 2.3 Code availability

Package development and assessment were performed using R 3.5.1 [6], the RcppArmadillo (v0.9.200.4.0) [1] and the microbenchmark (v1.4-6) [5] packages. The cPCG package was developed under the GNU General Public License v3.0 (GPL-3). Full code, documentation, and latest updates for the package can be found at `<https://github.com/styvon/cPCG>`.

## 3 Summary and discussion

This article introduces an R package cPCG that uses conjugate gradient methods for solving system of linear equations $\boldsymbol{\Sigma x} = \boldsymbol{y}$, where $\boldsymbol{\Sigma}$ is a symmetric and positive definite matrix. Using an iterative algorithm with integration of the C++ Armadillo library, the functions in the package show better performance over existing implementation in R, and provides more options of preconditioners for user-customized usage.

For illustration purpose, the comparison of computation time among the methods was conducted for matrices up to dimension $1000 \times 1000$. While the performance evaluation in this article indicate a better performance using incomplete Cholesky factorization over other methods for larger dimensions, it is necessary for future studies to confirm the result with a more robust assessment for different types of data and an even larger scale.

With improved performance in efficiency and flexibility, the cPCG package enables scaling up computations to higher dimension scenario, such as statistical analyses with genetic relationship matrix of large populations.

## References

[1] Dirk Eddelbuettel and Conrad Sanderson. "RcppArmadillo: Accelerating R with high-performance C++ linear algebra". In: *Computational Statistics and Data Analysis* 71 (2014), pp. 1054–1063. URL: `http://dx.doi.org/10.1016/j.csda.2013.02.005`.

[2] Reeves Fletcher and Colin M Reeves. "Function minimization by conjugate gradients". In: *The computer journal* 7.2 (1964), pp. 149–154.

[3] David S Kershaw. "The incomplete Cholesky—conjugate gradient method for the iterative solution of systems of linear equations". In: *Journal of computational physics* 26.1 (1978), pp. 43–65.

[4] B N Mandal and Jun Ma. *pcg: Preconditioned Conjugate Gradient Algorithm for solving Ax=b*. R package version 1.1. 2014. URL: `https://CRAN.R-project.org/package=pcg`.

[5] Olaf Mersmann. *microbenchmark: Accurate Timing Functions*. R package version 1.4-6. 2018. URL: `https://CRAN.R-project.org/package=microbenchmark`.

[6] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2018. URL: `https://www.R-project.org/`.

[7] Yousef Saad. *Iterative methods for sparse linear systems*. Vol. 82. siam, 2003.

[8] David Young. "Iterative methods for solving partial difference equations of elliptic type". In: *Transactions of the American Mathematical Society* 76.1 (1954), pp. 92–111.