Incomplete Cholesky Decomposition

Dona Hasini Gammune & Mananage Sanjaya Kumara

Handout

Introduction

- An **incomplete Cholesky factorization** of a **`Symmetric Positive definite matrix`** is a sparse approximation of the Cholesky factorization.
- It is a fundamental tool in the solution of large systems of linear equations.
- Let **`A`** be a **`Symmetric Positive definite matrix`**. An **incomplete Cholesky factorization** of A is such that

$$A = LL^T + R \text{ , } l_{ij} = 0 \text{ if } (i,j) \notin S \text{ and } r_{ij} = 0 \text{ if } (i,j) \in S$$

where,
- L is a **`Lower triangular matrix`**.
- S is a **`symmetric sparsity pattern`**.
- R is an **`error matrix`** which does not have to be formed.

Implementing the Incomplete Cholesky algorithm

- One popular way to find such a matrix L is to use the algorithm for finding the exact `Cholesky decomposition`, **except that any entry is set to zero if the corresponding entry in A is also zero**.

Algorithm

For $i = 1 : N$ and $j = 1 : N$

if$(a_{ij} = 0)$,then $L_{ij} = 0$

else do the following:

For i from 1 to N:

$$L_{ii} = \left(a_{ii} - \sum_{k=1}^{i-1} L_{ik}^2 \right)^{\frac{1}{2}}$$

For j from $i + 1$ to N:

$$L_{ji} = \frac{1}{L_{ii}} \left(a_{ji} - \sum_{k=1}^{i-1} L_{ik} L_{jk} \right)$$

Implementation of the algorithm : In R

```
> ichol<-function(A){
+   n <-nrow(A)
+     for(k in 1:n){
+     A[k,k]<-sqrt(A[k,k])
+     i<-k+1
+     while(i<=n){
+       if(A[i,k]!=0){
+       A[i,k]<-A[i,k]/A[k,k]
+     }
+      i<-i+1
+    }
+     j<-k+1
+     while(j<=n){
+     for (i in j:n){
+        if(A[i,j]!=0){
+        A[i,j]<-(A[i,j]-A[i,k]*A[j,k])
+       }
+     }
+     j<-j+1
+     }
+   }
+   return(A*lower.tri(A,TRUE))
+ }
```

Implementation of the algorithm: In Rcpp

```
> library(Rcpp)
> library(RcppEigen)
> sourceCpp(code = '
+ #include <Rcpp.h>
+ #include <RcppEigen.h>
+ // [[Rcpp::depends(RcppEigen)]]
+ using namespace std;
+ using namespace Rcpp;
+ using namespace Eigen;
+ // [[Rcpp::export]]
+   MatrixXd rcpp_ichol(MatrixXd A) {
+   A=A.triangularView<Lower>();
+   int n = A.rows();
+    for(int k=0;k<n;k++){
+    A(k,k)=sqrt(A(k,k));
+     for(int i=k+1;i<n;i++){
+       if(A(i,k)!=0){
+       A(i,k)=A(i,k)/A(k,k);
+       }
+     }
+     for(int j=k+1;j<n;j++){
+       for (int i=j;i<n;i++){
+        if(A(i,j)!=0){
+        A(i,j)=(A(i,j)-A(i,k)*A(j,k));
+       }
+     }
+     }
+    }
+   return A;
+ }
+ ')
```

In-built function for Incomplete cholesky factorization cPCG::icc()

```
> library(cPCG)
> icc(A)
>
```

Arguments:

- **A** - matrix, symmetric and positive definite.
- Returns a lower triabgular matrix after incomplete Cholesky factorization.
- Need to check that input matrix A is **symmetric and positive definite** before applying the function.
- 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.

In-built function to solve for x using Incomplete Cholesky factorization: cPCG::pcgsolve()

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

```
> library(cPCG)
> pcgsolve(A, b, preconditioner = ..., tol = ..., maxIter = ...)
```

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`.

Value:

- Returns a vector representing solution **x**.

Examples

Example 1

Let

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

```
The L matrix is
           [,1]       [,2]     [,3]
[1,]  1.4142136  0.0000000 0.000000
[2,] -0.7071068  1.2247449 0.000000
[3,]  0.0000000 -0.8164966 1.154701
```

This is a lower triangular matrix.

Error matrix

- The error matrix $R = A - LL'$ is

```
> A-ichol(A)%*%t(ichol(A))
              [,1]         [,2] [,3]
[1,] -4.440892e-16 0.000000e+00    0
[2,]  0.000000e+00 4.440892e-16    0
[3,]  0.000000e+00 0.000000e+00    0
```

- Here the error matrix **R** is almost equal to zero.

Testing the algorithms

```
> identical(icc(A),rcpp_ichol(A),as.matrix(ichol(A)))
[1] TRUE
```

- The factorization (L matrix) obtained by the implemented algorithms in R and Rcpp are identical to that obtained by `icc()`.

Example 2: Aplplication in statistics

- Consider the `mpg` data from `ggplot2` package.

```
> data(mpg, package = 'ggplot2')
> y<-mpg$hwy
> x<-model.matrix(~cty+class,data = mpg)
> C<-crossprod(x)
> b<-crossprod(x,y)
> head(C)
            (Intercept)   cty classcompact classmidsize classminivan
(Intercept)         234  3945           47           41           11
cty                3945 70729          946          769          174
classcompact         47   946           47            0            0
classmidsize         41   769            0           41            0
classminivan         11   174            0            0           11
classpickup          33   429            0            0            0
            classpickup classsubcompact classsuv
(Intercept)          33              35       62
cty                 429             713      837
classcompact          0               0        0
classmidsize          0               0        0
classminivan          0               0        0
classpickup          33               0        0
```

- The matrix C is a Symmetric positive definite matrix that contains many zeros as it's entries.
- Therefore we can use Incomplete Cholesky factorization.

Incomplete Cholesky factorization of $C = X'X$

```
> ichol(C)
                (Intercept)        cty classcompact classmidsize classminivan
(Intercept)      15.2970585  0.0000000      0.00000      0.00000     0.000000
cty             257.8927177 64.9641913      0.00000      0.00000     0.000000
classcompact      3.0724861  2.3648136      5.65398      0.00000     0.000000
classmidsize      2.6802538  1.1973065      0.00000      5.69058     0.000000
classminivan      0.7190925 -0.1762312      0.00000      0.00000     3.232932
classpickup       2.1572775 -1.9602515      0.00000      0.00000     0.000000
classsubcompact   2.2880216  1.8923640      0.00000      0.00000     0.000000
classsuv          4.0530668 -3.2057108      0.00000      0.00000     0.000000
                classpickup classsubcompact classsuv
(Intercept)        0.000000        0.000000 0.000000
cty                0.000000        0.000000 0.000000
classcompact       0.000000        0.000000 0.000000
classmidsize       0.000000        0.000000 0.000000
classminivan       0.000000        0.000000 0.000000
classpickup        4.950108        0.000000 0.000000
classsubcompact    0.000000        5.117022 0.000000
classsuv           0.000000        0.000000 5.941049
```

```
> identical(icc(C),rcpp_ichol(C),as.matrix(ichol(C)))
[1] TRUE
```

- The factorization (L matrix) obtained by the implemented algorithms in R and Rcpp are identical to that obtained by `icc()`.

Example 3 : Aplplication in statistics

- Consider the `mpg` data from `ggplot2` package.

```
> x1 <- model.matrix(~ cty + class+ displ + drv, data = mpg)
> y<-mpg$hwy
> D<-crossprod(x1)
> b1<-crossprod(x1,y)
```

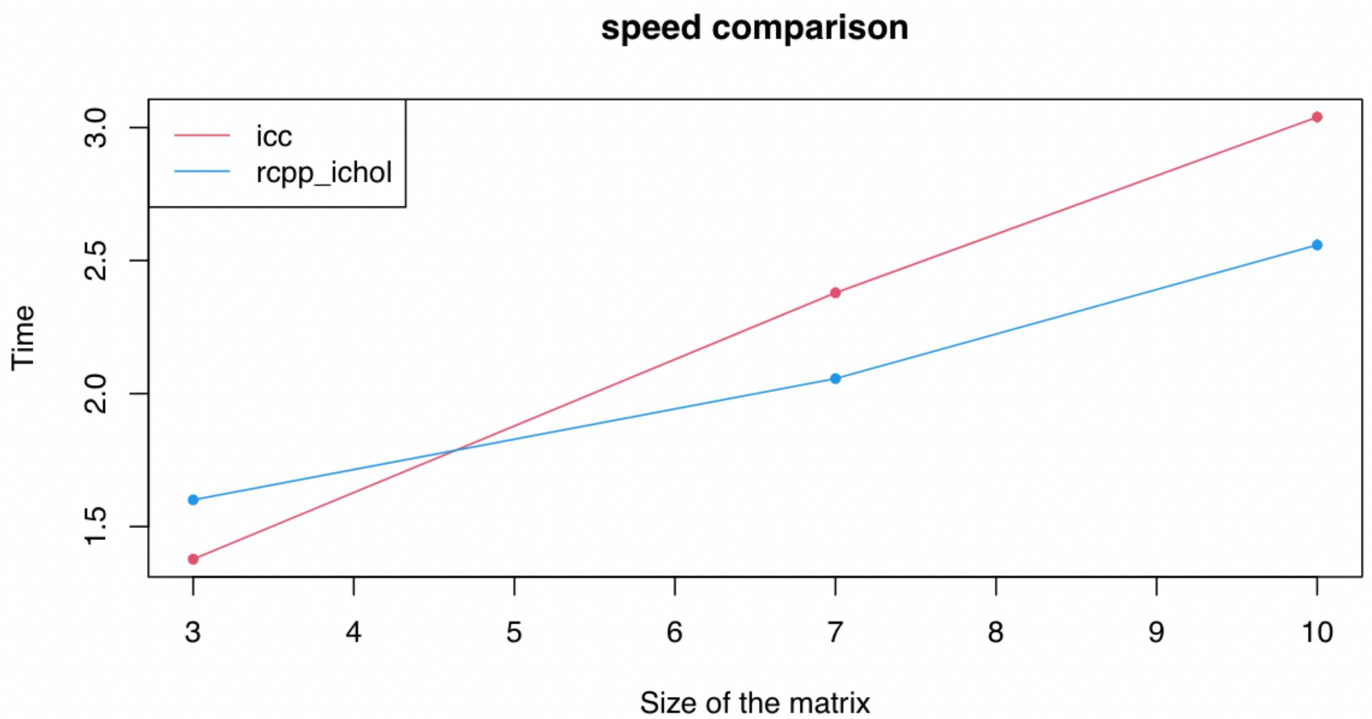
- Here we have added one more explanatory variable to the model.
- The matrix D is still a Symmetric positive definite matrix that contains many zeros as it's entries.
- Therefore we can use Incomplete Cholesky factorization.

```
> ichol(D)
               (Intercept)         cty classcompact classmidsize classminivan
(Intercept)     15.2970585   0.0000000    0.0000000     0.000000     0.000000
cty            257.8927177  64.9641913    0.0000000     0.000000     0.000000
classcompact     3.0724861   2.3648136    5.6539796     0.000000     0.000000
classmidsize     2.6802538   1.1973065    0.0000000     5.690580     0.000000
classminivan     0.7190925  -0.1762312    0.0000000     0.000000     3.232932
classpickup      2.1572775  -1.9602515    0.0000000     0.000000     0.000000
classsubcompact  2.2880216   1.8923640    0.0000000     0.000000     0.000000
classsuv         4.0530668  -3.2057108    0.0000000     0.000000     0.000000
displ           53.1082494 -15.7476103   -2.9420331    -0.648242    -1.133635
drvf             6.9294368   5.0789321    0.3004386     2.345334     2.138047
drvr             1.6343011  -1.0694254    0.0000000     0.000000     0.000000
               classpickup classsubcompact  classsuv     displ     drvf
(Intercept)     0.00000000       0.0000000 0.0000000  0.000000 0.000000
cty             0.00000000       0.0000000 0.0000000  0.000000 0.000000
classcompact    0.00000000       0.0000000 0.0000000  0.000000 0.000000
classmidsize    0.00000000       0.0000000 0.0000000  0.000000 0.000000
classminivan    0.00000000       0.0000000 0.0000000  0.000000 0.000000
classpickup     4.95010787       0.0000000 0.0000000  0.000000 0.000000
classsubcompact 0.00000000       5.1170222 0.0000000  0.000000 0.000000
classsuv        0.00000000       0.0000000 5.9410494  0.000000 0.000000
displ           0.07302718       0.2711324 1.7785466 11.283778 0.000000
drvf            0.00000000      -0.6773254 0.0000000 -1.047305 4.524338
drvr            0.00000000       1.4235674 0.1595341  2.223947 0.000000
                    drvr
(Intercept)     0.000000
cty             0.000000
classcompact    0.000000
classmidsize    0.000000
classminivan    0.000000
classpickup     0.000000
classsubcompact 0.000000
classsuv        0.000000
displ           0.000000
drvf            0.000000
drvr            3.766624
> identical(icc(D),rcpp_ichol(D),as.matrix(ichol(D)))
[1] TRUE
```

- The factorization (L matrix) obtained by the implemented algorithms in R and Rcpp are identical to that obtained by `icc()`.

Speed comparison plot:: The computation of `L` matrix

- The `ichol()` function is slow because multiple `for loops` are involved.Therefore here we have compared the speed of `icc()` and `rcpp_ichol()` for different matrix sizes.



- It can be clearly seen that the time taken for the Incomplete Cholesky factorization using the `rcpp_ichol()` function decreases as the size of th matrix; i.e, the number of explanatory variables increases.
- Therefore this method is more applicable in factorizing the large dense systems.
- Also the time taken in factorizing a matrix using Incomplete Cholesky factorization is less when compared with that factorized used Cholesky factorization.

```
> microbenchmark::microbenchmark(rcpp_ichol(C),chol(C))
Unit: microseconds
          expr   min    lq     mean median     uq      max neval cld
 rcpp_ichol(C) 1.375 1.667  3.11985  1.875 2.1045   89.251   100   a
       chol(C) 3.751 4.063 31.51443  5.188 5.8335 2615.709   100   a
```

Speed comparison :: solving for β

- `backsolve()` and `forwardsolve()` are more efficient as L' and L are upper triangular and lower triangular, respectively.
- Example 2

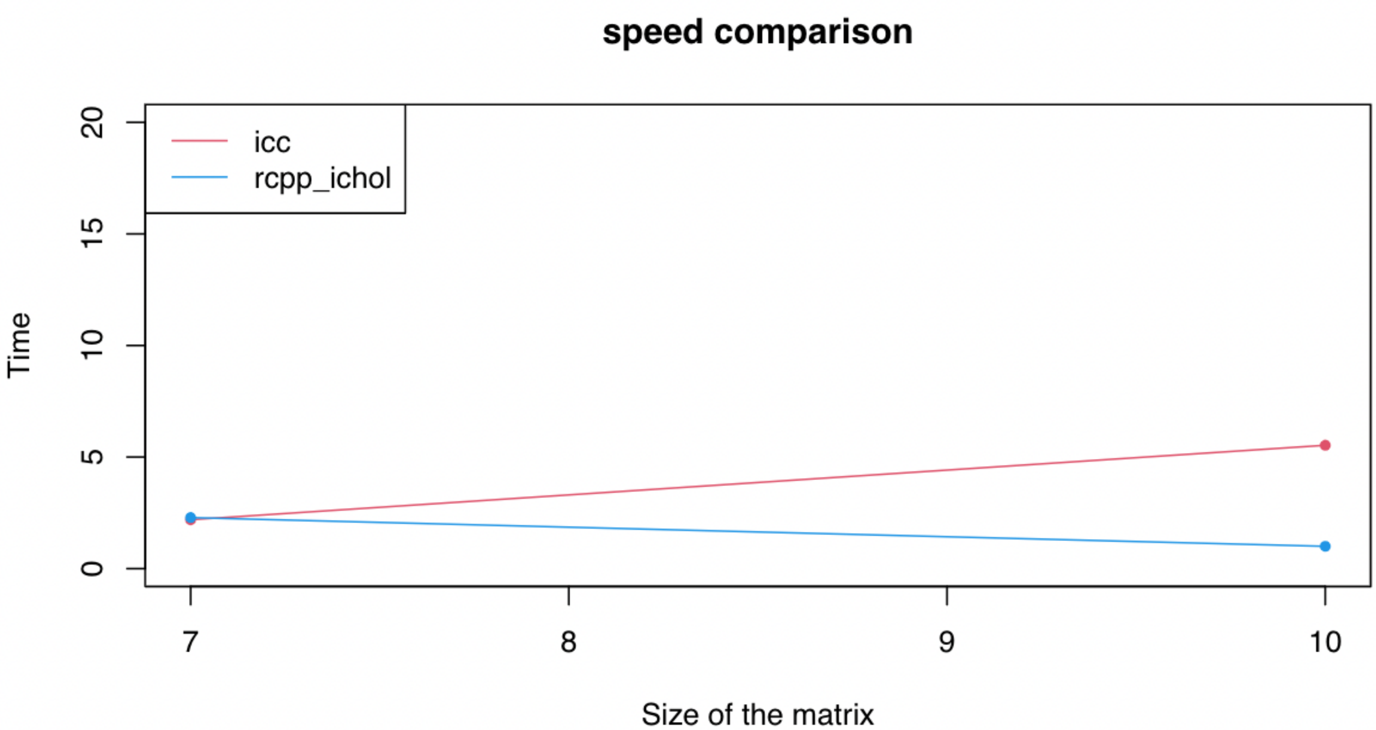
```
> set.seed(12345)
> ICr<-t(ichol(C))
> ICrcpp<-t(rcpp_ichol(C))
> ICicc<-t(icc(C))
>
> Csol_r<-backsolve(ICr,forwardsolve(t(ICr),b))
> Csol_rcpp<-backsolve(ICrcpp,forwardsolve(t(ICrcpp),b))
> Csol_icc<-backsolve(ICicc,forwardsolve(t(ICicc),b))
> identical(Csol_r,Csol_rcpp,Csol_icc)
[1] TRUE
```

- The solutions for the β coefficients obtained using the Incomplete Cholesky factorization method is not exactly the solutions obtained using the Cholesky factorization method as $LL' \approx C$. But it is an approximation.

```
              [,1]
[1,]  4.186648085
[2,]  1.153605157
[3,]  0.714509151
[4,]  1.665177908
[5,]  0.332024684
[6,] -1.885676949
[7,]  0.006596086
[8,] -1.438032540
```

- Similarly the twe can solve for β in example 3 in example 3 also. But it is an approximation.

Speed comparison plot:: Solving for β



- The time consumed to solve the system for β in both the cases is calculated.
- It can be seen that the time taken to solve for β using the `rcpp_ichol()` function decreases as the size of the matrix increases.

Practice question 1

- Consider the least squares estimator in linear regression models, β, which is the solution to the equation, $X'X\beta = X'y$. Apply the Incomplete cholesky factorization on $X'X$, and use `backsolve()` and `forwardsolve()` to recover the approximate least squares estimators of the model,

```
> data(mpg, package = 'ggplot2')
> fit_mpg <- lm(hwy ~ cty + class+ displ + drv+factor(cyl)+fl, data = mpg)
```

- Also compare the computation time in factorizing the matrix $X'X$ using Cholesky factorization and Incomplete Cholesky factorization.

Practice question 2

- Repeat the pracice question 1 with the model ,

```
> data(mtcars)
> fit_cars <- lm(mpg~disp+hp+drat+qsec+factor(cyl)+factor(vs)+factor(am)+factor(gear), data = mtcars)
```

Advantages & disadvantages of Incomplete Cholesky factorization

Advantages

- Incomplete Cholesky factorization is very efficient in increasing the convergence rates of basic iterative methods.
- Reduces the complicated addressing and high demands for auxiliary storage.
- This factorization is extremely cheap to compute as it reduces the computational time.

Disdvantages

- The product LL' is typically very different from A. But the product LL' will match A on its pattern up to round-off.

Summary

- Incomplete Cholesky factorization is a factorization which contains nonzeros only in the same position as A contains nonzeros.
- This is an approximation of the Cholesky factorization.
- `A` should be a Symmetric and Positive Definite matrix.
- Very usefull in solving large dense sytems.
- In-built function to solve Incomplete Cholesky factorization is `cPCG::icc`. A function is written in Rcpp which beats `cPCG::icc` in speed.

References and contribution report

References

- https://cran.r-project.org/web/packages/cPCG/cPCG.pdf
- https://doi.org/10.1137/S1064827597327334
- https://rdrr.io/cran/cPCG/man/cPCG-package.html
- https://www.mathworks.com/help/matlab/ref/ichol.html

Contribution report

Hasini

- Implemented the function in R.
- Prepared practice problems.

Sanjaya

- Implemented the function in Rcpp.
- Prepared practice problems.

- Both of us prepared the presentation and handout.