Lecture # 20
The Preconditioned Conjugate Gradient Method

We wish to solve

$$A\mathbf{x} = \mathbf{b} \tag{1}$$

where $A \in \mathbf{R}^{n\times n}$ is symmetric and positive definite (SPD). We then of n are being VERY LARGE, say, $n = 10^6$ or $n = 10^7$. Usually, the matrix is also sparse (mostly zeros) and Cholesky factorization is not feasible.

When A is SPD, solving (1) is equivalent to finding $\mathbf{x}^*$ such that

$$\mathbf{x}^* = \arg \min_{\mathbf{x}\in\mathbf{R}^n} \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}.$$

The conjugate gradient algorithm is one way to solve this problem.

Algorithm 1 (The Conjugate Gradient Algorithm)

$\mathbf{x}_0$ *initial guess (usually* 0*).*
$\mathbf{p}_1 = \mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$*;*
$\mathbf{w} = A\mathbf{p}_1$*;*
$\alpha_1 = \mathbf{r}_0^T\mathbf{r}_0/(\mathbf{p}_1^T\mathbf{w})$*;*
$\mathbf{x}_1 = \mathbf{x}_0 + \alpha_1\mathbf{p}_1$ *;*
$\mathbf{r}_1 = \mathbf{r}_0 - \alpha_1\mathbf{w}$*;*
$k = 1$ *;*
while $\|\mathbf{r}_k\|_2 > \epsilon$ *(some tolerence)*
    $\beta_k = \mathbf{r}_k^T\mathbf{r}_k/(\mathbf{r}_{k-1}^T\mathbf{r}_{k-1})$*;*
    $\mathbf{p}_{k+1} = \mathbf{r}_k + \beta_k\mathbf{p}_k$*;*
    $\mathbf{w} = A\mathbf{p}_{k+1}$*;*
    $\alpha_{k+1} = \mathbf{r}_k^T\mathbf{r}_k/(\mathbf{p}_{k+1}^T\mathbf{w})$*;*
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_{k+1}\mathbf{p}_{k+1}$ *;*
    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_{k+1}\mathbf{w}$*;*
    $k = k + 1$*;*
end*;*
$\mathbf{x} = \mathbf{x}_k$*;*

The conjugate gradient algorithm is essentially optimal in the following sense. We note that

$$\mathbf{x}_k = \mathbf{x}_0 + \sum_{j=1}^{k} \alpha_j \mathbf{p}_j = \mathbf{x}_0 + P_k \mathbf{a}$$

where

$$\begin{aligned} P_k &= (\mathbf{p}_1, \ldots, \mathbf{p}_k) \\ \mathbf{a} &= (\alpha_1, \ldots, \alpha_k)^T \end{aligned}$$

One can show that

$$\mathbf{x}_k = \arg \min_{\mathbf{x}=\mathbf{x}_0+P_k\mathbf{c}} \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}.$$

If we let the A-norm be given by

$$\|\mathbf{w}\|_A = \sqrt{\mathbf{w}^T A \mathbf{w}}$$

then one can show that

$$\|\mathbf{x}_k - \mathbf{x}^*\|_A \leq 2\|\mathbf{x}_0 - \mathbf{x}^*\|_A \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k$$

where

$$\kappa = \kappa_2(A) = \|A^{-1}\|_2 \|A\|_2.$$

Thus if A is well conditioned, the conjugate gradient method converges quickly. This bound is actually very pessimistic!

However, if A is badly conditioned, conjugate gradient can converge slowly, so we look to transform (1) into a well conditioned problem. We writing A in the form of a *splitting*

$$A = M - N$$

where M is SPD. Since M is SPD, it has a Cholesky factorization, thus

$$M = R^T R.$$

If we let

$$\hat{A} = R^{-T} A R^{-1}, \quad \hat{\mathbf{b}} = R^{-T}\mathbf{b}$$

then solving (1) is equivalent to solving

$$\hat{A}\mathbf{y} = \hat{\mathbf{b}}$$

where

$$\mathbf{y} = R\mathbf{x}.$$

We hope that $\kappa_2(\hat{A}) \ll \kappa_2(A)$, and we solve (1) by performing conjugate gradient on $\hat{A}$. In that formulation, we would have

$$\hat{\mathbf{y}}_k = R\mathbf{x}_k$$

and

$$\hat{\mathbf{r}}_k = \hat{\mathbf{b}} - \hat{A}\hat{\mathbf{y}}_k = R^{-T}(\mathbf{b} - A\mathbf{x}_k) = R^{-T}\mathbf{r}_k.$$

A conjugate gradient step would be

$$\begin{aligned} \hat{\mathbf{p}}_{k+1} &= \hat{\mathbf{r}}_k + \beta_k \hat{\mathbf{p}}_k \\ \mathbf{y}_{k+1} &= \mathbf{y}_k + \alpha_{k+1}\hat{\mathbf{p}}_{k+1} \end{aligned}$$

We prefer to update $\mathbf{x}_k = R^{-1}\mathbf{y}_k$. We have that

$$\hat{\mathbf{p}}_{k+1} = R^{-T}\mathbf{r}_k + \beta_k \hat{\mathbf{p}}_k,$$

so

$$\mathbf{x}_{k+1} = R^{-1}\mathbf{y}_{k+1} = R^{-1}\mathbf{y}_k + \alpha_{k+1}R^{-1}\hat{\mathbf{p}}_{k+1}$$

Thus

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_{k+1}\mathbf{p}_{k+1}$$

where

$$\mathbf{p}_{k+1} = R^{-1}R^{-T}\mathbf{r}_k + \beta_k \mathbf{p}_k.$$

Since

$$M^{-1} = R^{-1}R^{-T}$$

if we let

$$\mathbf{z}_k = M^{-1}\mathbf{r}_k$$

then

$$\mathbf{p}_{k+1} = \mathbf{z}_k + \beta_k \mathbf{p}_k$$

is the update to the descent direction. The formulas for β_k and α_{k+1} update to

$$\begin{aligned} \beta_k &= (\mathbf{r}_k^T \mathbf{z}_k)/(\mathbf{r}_{k-1}^T \mathbf{z}_{k-1}) \\ \alpha_{k+1} &= (\mathbf{r}_k^T \mathbf{z}_k)/(\mathbf{p}_{k+1}^T A\mathbf{p}_{k+1}). \end{aligned}$$

These modifications lead to the preconditioned conjugate gradient method.

Algorithm 2 (The Preconditioned Conjugate Gradient Algorithm)

$\mathbf{x}_0$ *initial guess (usually* 0*).*
$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$*;*
Solve $M\mathbf{z}_0 = \mathbf{r}_0$*;*
$\mathbf{p}_1 = \mathbf{z}_0$*;*
$\mathbf{w} = A\mathbf{p}_1$*;*
$\alpha_1 = \mathbf{r}_0^T\mathbf{z}_0/(\mathbf{p}_1^T\mathbf{w})$*;*
$\mathbf{x}_1 = \mathbf{x}_0 + \alpha_1\mathbf{p}_1$ *;*
$\mathbf{r}_1 = \mathbf{r}_0 - \alpha_1\mathbf{w}$*;*
$k = 1$ *;*
while $\|\mathbf{r}_k\|_2 > \epsilon$ *(some tolerence)*
    Solve $M\mathbf{z}_k = \mathbf{r}_k$*;*
    $\beta_k = \mathbf{r}_k^T\mathbf{z}_k/(\mathbf{r}_{k-1}^T\mathbf{z}_{k-1})$*;*
    $\mathbf{p}_{k+1} = \mathbf{z}_k + \beta_k\mathbf{p}_k$*;*
    $\mathbf{w} = A\mathbf{p}_{k+1}$*;*
    $\alpha_{k+1} = \mathbf{r}_k^T\mathbf{z}_k/(\mathbf{p}_{k+1}^T\mathbf{w})$*;*
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_{k+1}\mathbf{p}_{k+1}$ *;*
    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_{k+1}\mathbf{w}$*;*
    $k = k + 1$*;*
end*;*
$\mathbf{x} = \mathbf{x}_k$*;*

So far, I have ignored the question of how to choose M. It is a very important question. Although this version of the conjugate gradient method dates from Concus, Golub, and O'Leary (1976), people still write papers on how to choose preconditioners (including your instructor!).

Here are some straightforward choices.

The Jacobi preconditioner. Choose

$$M = D = \mathrm{diag}(a_{11}, a_{22}, \ldots, a_{nn}).$$

The resulting $\hat{A}$ is

$$\hat{A} = D^{-1/2}AD^{-1/2}$$

which is the same as diagonally scaling to produce $\hat{A}$ with ones on the diagonal.

Block versions of the Jacobi preconditioner are also possible.That is, if we write A in the form

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & \cdots & A_{1s} \\ A_{21} & A_{22} & \cdots & \cdots & A_{2s} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ A_{s1} & A_{s2} & \cdots & \cdots & A_{ss} \end{pmatrix}$$

where each A_{ij} is $q \times q$ matrix. A block Jacobi preconditioner is the matrix

$$M = D = \begin{pmatrix} A_{11} & 0 & \cdots & \cdots & 0 \\ 0 & A_{22} & \cdots & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & A_{ss} \end{pmatrix}.$$

This matrix M is symmetric and positive definite. The solution of

$$M\mathbf{z} = \mathbf{r}$$

solves

$$A_{kk}\mathbf{z}_k = \mathbf{r}_k$$

where

$$\mathbf{z} = \begin{pmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_s \end{pmatrix}, \quad \mathbf{r} = \begin{pmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_s \end{pmatrix}.$$

The SSOR preconditioner For $\omega \in (0, 2)$ choose

$$M = [\omega(2-\omega)]^{-1}(D + \omega L)D^{-1}(D + \omega L^T)$$

The value of ω matters little so we usually choose $\omega = 1$, which yields

$$M = (D + L)D^{-1}(D + L^T)$$

In this case,

$$N = LD^{-1}L^T.$$

The work here is just a back and forward solve.

Incomplete Cholesky preconditioner Do Cholesky, but ignore fill elements. If A is large and sparse in the Cholesky factorization

$$A = R^T R \tag{2}$$

the matrix R will often have many more nonzeros than A. This is one of the reasons that conjugate gradient is cheaper than Cholesky in some instances.

First, let us write a componentwise version of the Cholesky algorithm to compute (2).

for $k = 1{:}\, n-1$
    $r_{kk} = \sqrt{a_{kk}}$;
    for $j = k+1 : n$
        $r_{kj} = a_{kj}/r_{kk}$;
    end ;
    for $i = k+1 : n$
        for $j = i{:}\, n$ % Upper triangle only!
            $a_{ij} = a_{ij} - r_{ki} r_{kj}$;
        end;
    end;
end;
$r_{nn} = \sqrt{a_{nn}}$;

Incomplete Cholesky ignores the fill elements. We get

$$M = \hat{R}^T \hat{R}$$

where $R_M = (\hat{r}_{kj})$ from the following algorithm. Here $r_{kj} = 0$ if $a_{kj} = 0$.

Algorithm 3 (Incomplete Cholesky Factorization)

for $k = 1{:}\, n-1$
    $\hat{r}_{kk} = \sqrt{a_{kk}}$;
    for $j = k+1 : n$
        $\hat{r}_{kj} = a_{kj}/\hat{r}_{kk}$;
    end ;
    for $i = k+1 : n$

```
        for j = i:n % Upper triangle only!
            if a_ij ≠ 0
               a_ij = a_ij − r̂_ki r̂_kj;
            end;
        end;
    end;
end;
r_nn = √a_nn;
```

One downside to this algorithm, is that even if A is SPD, it is possible that a_{kk} could be negative or zero when it is time for r_{kk} to be evaluated at the beginning of the main loop. Thus, unlike the Jacobi and SSOR preconditioners, the incomplete Cholesky preconditioner is not defined for all SPD matrices!

However, if, in addition, A is *irreducibly diagonally dominant*, that is,

$$|a_{jj}| \geq \sum_{i \neq j} |a_{ij}|, \quad j = 1, \ldots, n$$

with at least one inequality strict and no decoupling, that is, no permutation such that

$$PAP^T = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix},$$

then incomplete Cholesky is guaranteed to exist.

Incomplete Cholesky is a popular preconditioner in many environments. Much has been written about it and continues to be written about it. The many variants of it consist of algorithms for specifying which fill elements to accept and which to reject. They can become quite sophisticated.