Chapter 5. Conjugate Gradient Methods

* Excellent for large problems
* both linear & non-linear problems
* pre-conditioning for large problems

5.1 Linear Method

Solve $Ax=b$ where:
A is an $n \times n$ symmetric, positive definite matrix.

Rewrite $Ax=b$ as one of having to
minimize: $\phi(x) = \frac{1}{2} x^T A x - b^T x$.

Then: $\boxed{\nabla \phi(x) = Ax - b \stackrel{\text{def}}{=} r(x) \text{ residual}}$

Similar to coordinate descent, but we want to generate the solution by moving in conjugate directions.

Conjugate Direction Methods

A set of vectors $\{p_0, p_1, \ldots, p_l\}$ is conjugate with respect to a symmetric positive definite A if:

$$p_i^T A p_j = 0 \quad \forall i \neq j.$$

The basic idea is that this is a list of orthogonal directions with respet to A.

The solution to $Ax=b$ is:

$$\boxed{\begin{aligned} x_{k+1} &= x_k + \alpha p_k \\ \alpha_k &= \frac{-r_k^T p_k}{p_k^T A p_k} \end{aligned}} \quad \Delta$$

which comes from p. 55:

$$\min \tfrac{1}{2} x^T Q x + b^T x + c \qquad \text{convex}$$

is $\alpha_k = \dfrac{-\nabla f_k^T p_k}{p_k^T Q p_k}$

The basic contribution of CG is that $x_{k+1} = x_k + \alpha p_k$ will not have to be repeated for all n steps, but we may get convergence for a limited number of steps and p_k are easy to compute.

Thm 5.1 For any starting point x_0 generated for the CG algorithm, we have convergence to x^* in at-most n steps.

Proof: Since $p_0, \dots, p_{n-1}$ are linearly independent, we have:

$$x^* - x_0 = \sigma_0 p_0 + \sigma_1 p_1 + \cdots + \sigma_{n-1} p_{n-1}$$

for some $\sigma_0, \dots, \sigma_{n-1}$. Pre-multiply by $p_k^T A$ to get:

$$p_k A (x^* - x_0) = \sum_{i=0}^{n-1} p_k A(\sigma_i p_i)$$

$$= p_k A(\sigma_k p_k)$$

Thus: $$\sigma_k = \frac{p_k^T A (x^* - x_0)}{p_k^T A p_k} \quad \text{—} \circledast$$

We now show that $\circledast$ is generated by the algorithm.

Start with:

$$x_k = x_0 + \alpha_0 p_0 + \alpha_1 p_1 + \cdots + \alpha_{k-1} p_{k-1}.$$

Again, pre-multiply by $p_k^T A$:

$$p_k^T A x_k = p_k^T A x_0 + \sum_{i=0}^{k-1} p_k^T A (\alpha_i p_i)$$

$$= p_k^T A x_0 + 0$$

$$\Rightarrow \boxed{p_k^T A x_k = p_k^T A x_0}$$

$r_k = Ax_k - b$

Start from:

$$p_k^T A (x^* - x_0) = p_k^T A (x^* - x_k)$$

$$= p_k^T (Ax^*) - p_k^T A x_k$$

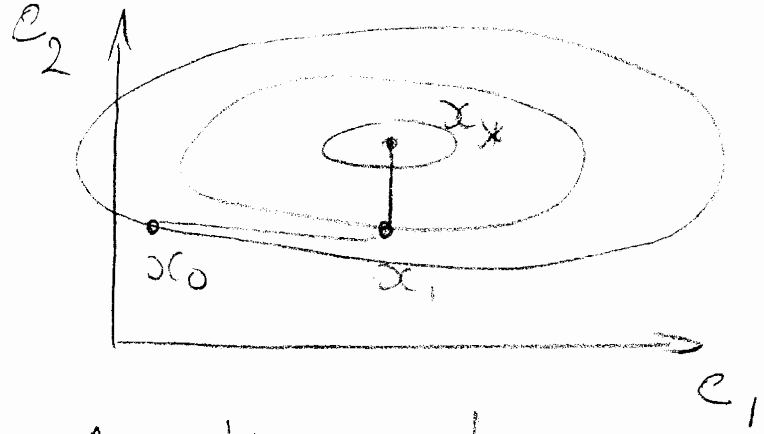
$$= p_k^T b - p_k^T (r_k + b)$$

From $Ax^* = b$

$$= -p_k^T r_k \quad \text{by } r_k \text{ \underline{def}}$$

Substitute $\square$ in $\circledast$ to get $\triangle$. ■

A geometric interpretation:



For A diagonal.

If A is not diagonal, then use:

$$\hat{x} = S^{-1} x$$

where:

$$S = [\, p_0 \;\; p_1 \;\; \cdots \;\; p_{n-1} \,]$$

conjugate directions are column vectors.

<u>to transform it to a diagonal problem.</u>

Thm 5.2

Let $x_0 \in \mathbb{R}^n$ be a starting point.
Then $\{x_k\}$ by CG satisfies:

* $r_k^T p_i = 0, \quad i = 0, \dots, k-1$
* x_k minimizes $\phi(x) = \frac{1}{2} x^T A x - b^T x$
  over: $\{ x \mid x = x_0 + \mathrm{span}\{p_0, \dots, p_{k-1}\} \}$

Proof: Very nice, see book, p. 106.

NB
* Note that setting the eigenvectors to $\{p_0, \dots, p_{n-1}\}$ will also work
* any set of orthogonal directions will work.

CG method: Computing p_k (1st attempt)

$$\boxed{\begin{aligned} p_k &= -r_k + \beta_k p_{k-1}, \quad p_0 = -\nabla f(x_0) \\ \beta_k &= \frac{r_k^T A p_{k-1}}{p_{k-1}^T A p_{k-1}} \\ -r_k &= -\nabla \phi(x_k) \sim * \end{aligned}}$$

The proof that this method converges is given in theorem 5.3.

Standard form of CG (w/out pre-conditioner)

Given x_0:

$r_0 = Ax_0 - b$ ← residual

$p_0 = -r_0$ ← move in steepest-descent

$k = 0$

while $(r_k \neq 0)$ & $(k <= n-1)$

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$$ ← step-length

$x_{k+1} = x_k + \alpha_k p_k$ ← new location

$$\left.\begin{aligned} r_{k+1} &= r_k + \alpha_k A p_k \\ \beta_{k+1} &= r_{k+1}^T r_{k+1} / r_k^T r_k \\ p_{k+1} &= -r_{k+1} + \beta_{k+1} p_k \end{aligned}\right\}$$ ← new direction

$k = k+1$

end

Comments on the usage of CG:

Storage requirements: we only need two "copies" of each variable: $x_k, r_k, \alpha_k, \beta_k, p_k$.

* for large problems, use CG, else use Gaussian elimination, or singular value decomposition, or other methods.
* CG can be sensitive to rounding errors, while Gaussian elimination & SVD are not (as much)
* The reason to use CG is because:
  - $p_k A p_k$ can be computed very quickly if A is sparse, and
  - it may converge in a very small number of iterations.

Preconditioning

Set $\hat{x} = Cx$, C non-singular:

Then: $\phi(x) = \frac{1}{2} x^T A x - b^T x$

becomes:

$$\begin{cases} x = C^{-1}\hat{x} \\ x^T = \hat{x}^T C^{-T} \end{cases}$$

and: $\hat{\phi}(\hat{x}) = \frac{1}{2} \hat{x}^T (C^{-T} A C) \hat{x} - (C^{-T} b)^T \hat{x}$

which, as before, will solve:

$$(C^{-T} A C^{-1}) \hat{x} = C^{-T} b \quad — \; (*)$$

Convergence of $(*)$ depends on the eigenvalues of $(C^{-T} A C^{-1})$.

See discussion on pre-conditioning by your textbook. Also, set $p_0 = -y_0$ in Algorithm 5.3, and also discussion on convergence right before.

5.2 Nonlinear CG

Replace the following from the standard algorithm:

* replace $\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$ by line search along p_k (strong Wolfe cond.)
* replace r_k by ∇f_k.

This gives the Fletcher-Reeves CG method (see textbook for the steps).

For details on how to implement PR+ line-searches for global convergence, see reference [103] of your textbook.

The change must be slight.

PR+ algorithm (modified 5.4)

Given x_0;

Evaluate $f_0 = f(x_0)$, $\nabla f_0 = \nabla f(x_0)$

$p_0 = -\nabla f_0$

$k = 0$

<u>while</u> $(\nabla f_k \neq 0)$

Compute α_k using line-search (but see [103])

$$x_{k+1} = x_k + \alpha_k p_k$$

Evaluate ∇f_{k+1}

$$\beta_{k+1}^{PR} = \frac{\nabla f_{k+1}^T (\nabla f_{k+1} - \nabla f_k)}{\|\nabla f_k\|^2}$$

$$\beta_{k+1}^{PR+} = \max(\beta_{k+1}^{PR}, 0)$$

$$p_{k+1} = -\nabla f_{k+1} + \beta_{k+1}^{FR} p_k$$

$k = k+1$

<u>end</u>