Numerical Optimization

Lecture Notes #11
Conjugate Gradient Methods — Linear CG, Part #2

Peter Blomgren,
⟨blomgren.peter@gmail.com⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://terminus.sdsu.edu/

Fall 2018

Outline

1. Recap
   - Linear Conjugate Direction Methods
2. The Linear Conjugate Gradient Method
   - $A\bar{\mathbf{p}}$ vs. A^{-1}
   - Adding the Gradient to the Mix...
   - The CG Algorithm... and Krylov Subspaces
3. The CG Algorithm
   - Increased Efficiency
   - The Standard CG Algorithm
   - More Speed... Preconditioning
4. Non-Linear CG for Optimization Problems
   - HW#4, Due 11/2/2018, 12:00pm

Quick Recap: Linear Conjugate **Direction** Methods

We introduced the **conjugate direction method**: Given a starting point $\bar{\mathbf{x}}_0 \in \mathbb{R}^n$ and a set of conjugate directions $\{\bar{\mathbf{p}}_0, \bar{\mathbf{p}}_1, \ldots, \bar{\mathbf{p}}_n\}$ the sequence

$$\bar{\mathbf{x}}_{k+1} = \bar{\mathbf{x}}_k + \alpha_k \bar{\mathbf{p}}_k, \;\; \alpha_k = -\frac{\bar{\mathbf{r}}_k^T \bar{\mathbf{p}}_k}{\bar{\mathbf{p}}_k^T A \bar{\mathbf{p}}_k}, \text{ where } \bar{\mathbf{r}}_k = \bar{\mathbf{r}}(\bar{\mathbf{x}}_k) = A\bar{\mathbf{x}}_k - \bar{\mathbf{b}}$$

We showed that

(1) this method is guaranteed to converge to the solution $\bar{\mathbf{x}}^* = A^{-1}\bar{\mathbf{b}}$ in at most n iterations;

(2) each $\bar{\mathbf{x}}_k$ is the minimizer of $\Phi(\bar{\mathbf{x}}) = \frac{1}{2}\bar{\mathbf{x}}^T A \bar{\mathbf{x}} - \bar{\mathbf{b}}^T \bar{\mathbf{x}}$ over the set $\{\bar{\mathbf{x}} : \bar{\mathbf{x}} = \bar{\mathbf{x}}_0 + \operatorname{span}\{\bar{\mathbf{p}}_0, \bar{\mathbf{p}}_1, \ldots, \bar{\mathbf{p}}_{k-1}\}\}$

We're currently at the **"So what?!?"** stage...

More Questions than Answers

How can we make this useful?

— Given A, how do we get a set of conjugate vectors?

— Efficiently, please!

— Even if we have them, why is this scheme any better than Gaussian elimination? GE requires $\mathcal{O}(n^3)$ operations, and n steps of CG will require n matrix-vector ($A\bar{\mathbf{p}}_k$) products, which require n^2 operations...

— Again, if we have the conjugate vectors, it seems like we will make more progress in certain directions than in others; hence, if we are planning on stopping short of n iterations, the subset of conjugate directions that we select will have an impact on how well we do...

— Where is the gradient?

Comment: $A\bar{\mathbf{p}}$ vs. A^{-1}

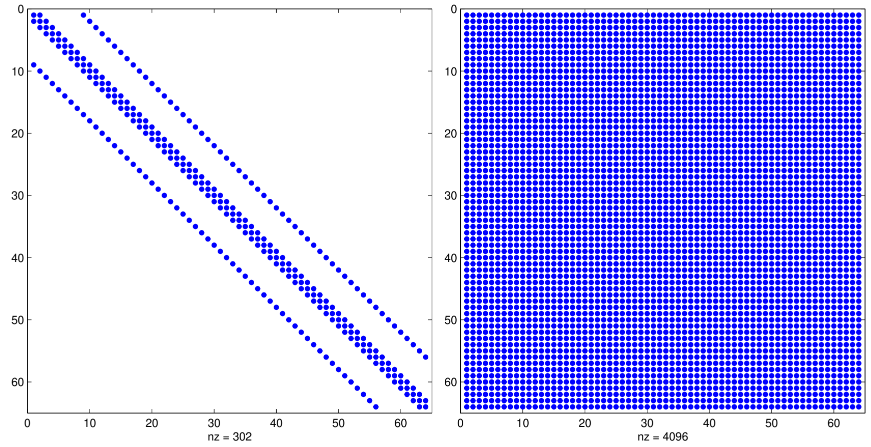

Figure: If the matrix A is sparse (many elements are zero, *e.g.* the matrix illustrated to the left), the computation of the matrix-vector product $A\bar{\mathbf{p}}$ can be economized. However, generally, the inverse of a sparse matrix is dense (the matrix on the right). This is one indication that the conjugate direction method may be useful.

Enter: The Gradient

The **Conjugate Gradient** method is a conjugate direction method, which

— Generates the next conjugate vector $\bar{\mathbf{p}}_k$ using only the previous vector $\bar{\mathbf{p}}_{k-1}$ (earlier vectors are not needed.)

— Cheap to compute, and store.

— Each direction $\bar{\mathbf{p}}_k$ is a clever linear combination of $\bar{\mathbf{p}}_{k-1}$ and the negative gradient of the objective $-\nabla\Phi(\bar{\mathbf{x}}_k) = -\bar{\mathbf{r}}(\bar{\mathbf{x}}_k)$ (a.k.a "the (negative) residual," or "the steepest descent direction.")

— Recall that we have a cheap update for the residual

$$\bar{\mathbf{r}}_k = \bar{\mathbf{r}}_{k-1} + \alpha_{k-1}\underbrace{A\bar{\mathbf{p}}_{k-1}}_{\text{"Free"}}.$$

A New Conjugate Direction

We let the new conjugate direction be

$$\bar{\mathbf{p}}_k = -\bar{\mathbf{r}}_k + \beta_k\bar{\mathbf{p}}_{k-1},$$

and we select the scalar β_k so that $\bar{\mathbf{p}}_k$ and $\bar{\mathbf{p}}_{k-1}$ are A-conjugate

$$\bar{\mathbf{p}}_{k-1}^T A\bar{\mathbf{p}}_k = -\bar{\mathbf{p}}_{k-1}^T A\bar{\mathbf{r}}_k + \beta_k\bar{\mathbf{p}}_{k-1}^T A\bar{\mathbf{p}}_{k-1} = 0.$$

Hence,

$$\beta_k = \frac{\bar{\mathbf{p}}_{k-1}^T A\bar{\mathbf{r}}_k}{\bar{\mathbf{p}}_{k-1}^T A\bar{\mathbf{p}}_{k-1}} = \frac{\bar{\mathbf{r}}_k^T A\bar{\mathbf{p}}_{k-1}}{\bar{\mathbf{p}}_{k-1}^T A\bar{\mathbf{p}}_{k-1}},$$

where, again, the quantities $[A\bar{\mathbf{p}}_{k-1}]$ and $\bar{\mathbf{r}}_k^T A\bar{\mathbf{p}}_{k-1}$ are "free" (already computed).

Note!!! The first direction $\bar{\mathbf{p}}_0$ is set to be the steepest descent direction at the initial point $\bar{\mathbf{x}}_0$.

The Conjugate Gradient Algorithm (version 0.99α)

Algorithm: Preliminary Conjugate Gradient

```
Given $A$, $\bar{\mathbf{b}}$ and $\bar{\mathbf{x}}_0$:
$\bar{\mathbf{r}}_0 = A\bar{\mathbf{x}}_0 - \bar{\mathbf{b}}$, $\bar{\mathbf{p}}_0 = -\bar{\mathbf{r}}_0$, $k = 0$
while ( $\|\bar{\mathbf{r}}_k\| > 0$, or other stopping condition )
    $\alpha_k = -\dfrac{\bar{\mathbf{r}}_k^T\bar{\mathbf{p}}_k}{\bar{\mathbf{p}}_k^T A\bar{\mathbf{p}}_k}$,      Store the vector $A\bar{\mathbf{p}}_k$
                                             and the scalar $\bar{\mathbf{p}}_k^T A\bar{\mathbf{p}}_k$
    $\bar{\mathbf{x}}_{k+1} = \bar{\mathbf{x}}_k + \alpha_k\bar{\mathbf{p}}_k$
    $\bar{\mathbf{r}}_{k+1} = \bar{\mathbf{r}}_k + \alpha_k \mathbf{A\bar{p}_k}$
    $\beta_{k+1} = \dfrac{\bar{\mathbf{r}}_{k+1}^T \mathbf{A\bar{p}_k}}{\mathbf{\bar{p}_k^T A\bar{p}_k}}$
    $\bar{\mathbf{p}}_{k+1} = -\bar{\mathbf{r}}_{k+1} + \beta_{k+1}\bar{\mathbf{p}}_k$
    $k = k+1$
end-while
```

Does the CG Algorithm Work?

In order to guarantee convergence in n steps, the directions $\{\bar{\mathbf{p}}_i\}$ must be A-conjugate; maybe we should show this?! But, first a definition:

Definition (Krylov Subspace)

A **Krylov** subspace of degree k for $\bar{\mathbf{r}}_0$ is the space

$$\mathcal{K}(\bar{\mathbf{r}}_0, k) \overset{\text{def}}{=} \text{span}\{\bar{\mathbf{r}}_0, A\bar{\mathbf{r}}_0, A^2\bar{\mathbf{r}}_0, \dots, A^{k-1}\bar{\mathbf{r}}_0\}.$$

We state a theorem which shows that the directions are indeed conjugate; further it shows that the residuals are mutually orthogonal, and that the search directions and residuals are contained in a Krylov subspace. These facts will allow us to optimize the CG algorithm for speed (computational effort).

"The CG Theorem"

Theorem

Suppose that the k^{th} iterate generated by the CG method is not the solution (i.e. $\bar{\mathbf{x}}_k \neq \bar{\mathbf{x}}^*$). The following properties hold

$$\begin{array}{lrcl}
(1) & \bar{\mathbf{r}}_k^T\bar{\mathbf{r}}_i & = & 0, \quad i = 0, 1, \dots, k-1 \\
(2) & \text{span}\{\bar{\mathbf{r}}_0, \bar{\mathbf{r}}_1, \dots, \bar{\mathbf{r}}_k\} & = & \text{span}\{\bar{\mathbf{r}}_0, A\bar{\mathbf{r}}_0, A^2\bar{\mathbf{r}}_0, \dots, A^k\bar{\mathbf{r}}_0\} \\
(3) & \text{span}\{\bar{\mathbf{p}}_0, \bar{\mathbf{p}}_1, \dots, \bar{\mathbf{p}}_k\} & = & \text{span}\{\bar{\mathbf{r}}_0, A\bar{\mathbf{r}}_0, A^2\bar{\mathbf{r}}_0, \dots, A^k\bar{\mathbf{r}}_0\} \\
(4) & \bar{\mathbf{p}}_k^T A\bar{\mathbf{p}}_i & = & 0, \quad i = 0, 1, \dots, k-1
\end{array}$$

Therefore, the sequence $\{\bar{\mathbf{x}}_k\}$ converges to $\bar{\mathbf{x}}^*$ in at most n steps.

Note: The theorem is true **if and only if** the first search direction is the steepest descent direction. We notice that the search direction (and not the gradients/residuals) are conjugate in the "conjugate gradient method."

A More Efficient Implementation α_k (Numerator)

We now combine our results in order to tighten up the algorithm.

First, we use the relation (update for $\bar{\mathbf{p}}_k$ in the algorithm)

$$\bar{\mathbf{p}}_k = -\bar{\mathbf{r}}_k + \beta_k\bar{\mathbf{p}}_{k-1},$$

and the result (from lecture #10, or slide #10)

$$\bar{\mathbf{r}}_k^T\bar{\mathbf{p}}_i = 0, \; i = 0, 1, \dots, k-1,$$

thus the numerator in the expression for α_k can be rewritten:

$$\bar{\mathbf{r}}_k^T\bar{\mathbf{p}}_k = \bar{\mathbf{r}}_k^T(-\bar{\mathbf{r}}_k + \beta_k\bar{\mathbf{p}}_{k-1}) = -\bar{\mathbf{r}}_k^T\bar{\mathbf{r}}_k + \beta_k \underbrace{\bar{\mathbf{r}}_k^T\bar{\mathbf{p}}_{k-1}}_{0} = -\bar{\mathbf{r}}_k^T\bar{\mathbf{r}}_k.$$

A More Efficient Implementation β_k

Second, we use the update formula for the residual

$$\bar{\mathbf{r}}_k = \bar{\mathbf{r}}_{k-1} + \alpha_{k-1}A\bar{\mathbf{p}}_{k-1} \quad \Leftrightarrow \quad \alpha_{k-1}A\bar{\mathbf{p}}_{k-1} = \bar{\mathbf{r}}_k - \bar{\mathbf{r}}_{k-1},$$

and again (from lecture #10, or slide #10)

$$\bar{\mathbf{r}}_k^T\bar{\mathbf{p}}_i = 0, \; i = 0, 1, \dots, k-1,$$

as well as the update for $\bar{\mathbf{p}}_k$ in the algorithm

$$\bar{\mathbf{p}}_k = -\bar{\mathbf{r}}_k + \beta_k\bar{\mathbf{p}}_{k-1}$$

We get

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

The CG Algorithm (version 1.0, "Standard")

Algorithm: Conjugate Gradient

```
Given $A$, $\bar{\mathbf{b}}$ and $\bar{\mathbf{x}}_0$:
$\bar{\mathbf{r}}_0 = A\bar{\mathbf{x}}_0 - \bar{\mathbf{b}}$, $\bar{\mathbf{p}}_0 = -\bar{\mathbf{r}}_0$, $k = 0$
while ( $\|r_k\| > 0$, or other stopping condition )
    $\alpha_k = \frac{\bar{\mathbf{r}}_k^T\bar{\mathbf{r}}_k}{\bar{\mathbf{p}}_k^T A\bar{\mathbf{p}}_k}$,        Store the vector $A\bar{\mathbf{p}}_k$
                                       and the scalar $\bar{\mathbf{r}}_k^T\bar{\mathbf{r}}_k$
    $\bar{\mathbf{x}}_{k+1} = \bar{\mathbf{x}}_k + \alpha_k\bar{\mathbf{p}}_k$
    $\bar{\mathbf{r}}_{k+1} = \bar{\mathbf{r}}_k + \alpha_k \mathbf{A\bar{p}_k}$
    $\beta_{k+1} = \frac{\bar{\mathbf{r}}_{k+1}^T\bar{\mathbf{r}}_{k+1}}{\bar{\mathbf{r}}_k^T\bar{\mathbf{r}}_k}$,     Keep numerator for next step!
    $\bar{\mathbf{p}}_{k+1} = -\bar{\mathbf{r}}_{k+1} + \beta_{k+1}\bar{\mathbf{p}}_k$
    $k = k + 1$
end-while
```

The CG Algorithm (version 1.0, "Standard") Work

The work per iteration for this version of the CG algorithm consists of

— One matrix-vector product $A\bar{\mathbf{p}}_k$
$\sim n^2$ operations (if A is dense)
$\sim \mathcal{O}(n)$ in many cases, when A is sparse.

— Two inner products: $\bar{\mathbf{p}}_k^T(A\bar{\mathbf{p}}_k)$ and $\bar{\mathbf{r}}_{k+1}^T\bar{\mathbf{r}}_{k+1}$
$\sim n$ additions, and $\sim n$ multiplications

— Three vector sums
$\sim 3n$ additions

The CG Algorithm — Convergence

In **exact arithmetic** CG converges in at most n iterations.

In many cases, the algorithm will find the solution in many fewer iterations. We leave the detailed convergence analysis for some other day, but state some key results:

Theorem

If A has only r distinct eigenvalues, then the CG iteration will terminate at the solution $\bar{\mathbf{x}}^*$ in at most r iterations.

Theorem

If A has eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, we have that

$$\|\bar{\mathbf{x}}_{k+1} - \bar{\mathbf{x}}^*\|_A \leq \left[\frac{\lambda_{n-k} - \lambda_1}{\lambda_{n-k} + \lambda_1}\right] \|\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}^*\|_A.$$

The CG Algorithm — Convergence: Comments

The second theorem tells us that the CG algorithm selects **exactly** the optimal sequence of conjugate search directions $\{\bar{\mathbf{p}}_i\}$.

If there is a cluster of eigenvalues of A around λ_1, *i.e.* $\lambda_1 = 1$, $\lambda_{3900} = 1.0002$, $\lambda_{4000} = 1.03$, and $\lambda_n = \lambda_{4032}$, then after 32 iterations we would have

$$\|\bar{\mathbf{x}}_{32} - \bar{\mathbf{x}}^*\|_A \leq \left[\frac{0.03}{2.03}\right] \|\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}^*\|_A$$

and after another 100 iterations

$$\|\bar{\mathbf{x}}_{132} - \bar{\mathbf{x}}^*\|_A \leq \left[\frac{0.0002}{2.0002}\right] \|\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}^*\|_A$$

With tight clustering (which is quite common) we often achieve very good convergence after a $k \ll n$ iterations.

The CG Algorithm — Example 1 of 2

Let

$$A = \mathrm{diag}(k^2\, I_k,\ k = 1\ldots5), \quad \bar{\mathbf{b}} = \bar{\mathbf{1}}, \quad \bar{\mathbf{x}}_0 = \bar{\mathbf{0}}.$$

We get $\bar{\mathbf{r}}_0$, $\bar{\mathbf{r}}_1$, $\bar{\mathbf{r}}_2$, $\bar{\mathbf{r}}_3$, $\bar{\mathbf{r}}_4$, $\bar{\mathbf{r}}_5$:

$\|\bar{\mathbf{r}}_0\| = \sqrt{15}$, $\|\bar{\mathbf{r}}_1\| = 2.16025$, $\|\bar{\mathbf{r}}_2\| = 1.54919$, $\|\bar{\mathbf{r}}_3\| = 1.13389$, $\|\bar{\mathbf{r}}_4\| = 0.745356$, $\|\bar{\mathbf{r}}_5\| = 2.20786 \times 10^{-14}$

$$\begin{bmatrix} 1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1 \end{bmatrix}, \begin{bmatrix} 0.933\\0.733\\0.733\\0.4\\0.4\\0.4\\-0.0667\\-0.0667\\-0.0667\\-0.0667\\-0.667\\-0.667\\-0.667\\-0.667\\-0.667 \end{bmatrix}, \begin{bmatrix} 0.8\\0.286\\0.286\\-0.286\\-0.286\\-0.286\\-0.486\\-0.486\\-0.486\\-0.486\\0.286\\0.286\\0.286\\0.286\\0.286 \end{bmatrix}, \begin{bmatrix} 0.6\\-0.171\\-0.171\\-0.386\\-0.386\\-0.386\\0.314\\0.314\\0.314\\0.314\\-0.0714\\-0.0714\\-0.0714\\-0.0714\\-0.0714 \end{bmatrix}, \begin{bmatrix} 0.333\\-0.381\\-0.381\\0.214\\0.214\\0.214\\-0.0635\\-0.0635\\-0.0635\\-0.0635\\0.00794\\0.00794\\0.00794\\0.00794\\0.00794 \end{bmatrix}, \begin{bmatrix} 1.11 \times 10^{-16}\\-1.11 \times 10^{-16}\\-1.11 \times 10^{-16}\\-1.39 \times 10^{-16}\\-1.39 \times 10^{-16}\\-1.39 \times 10^{-16}\\8.05 \times 10^{-16}\\8.05 \times 10^{-16}\\8.05 \times 10^{-16}\\8.05 \times 10^{-16}\\-6.98 \times 10^{-15}\\-6.98 \times 10^{-15}\\-6.98 \times 10^{-15}\\-6.98 \times 10^{-15}\\-6.98 \times 10^{-15} \end{bmatrix}$$

San Diego State University

The CG Algorithm — Example 2 of 2

Let

$$A = \mathrm{diag}(k^2\, I_k,\ k = 1\ldots5), \quad \bar{\mathbf{b}} = \bar{\mathbf{1}}, \quad \bar{\mathbf{x}}_0 = \bar{\mathbf{0}}.$$

We get $\bar{\mathbf{x}}_1$, $\bar{\mathbf{x}}_2$, $\bar{\mathbf{x}}_3$, $\bar{\mathbf{x}}_4$, $\bar{\mathbf{x}}_5 = \bar{\mathbf{x}}^*$:

$$\begin{bmatrix} 0.0667\\0.0667\\0.0667\\0.0667\\0.0667\\0.0667\\0.0667\\0.0667\\0.0667\\0.0667\\0.0667\\0.0667\\0.0667\\0.0667\\0.0667 \end{bmatrix}, \begin{bmatrix} 0.2\\0.179\\0.179\\0.143\\0.143\\0.143\\0.0929\\0.0929\\0.0929\\0.0929\\0.0286\\0.0286\\0.0286\\0.0286\\0.0286 \end{bmatrix}, \begin{bmatrix} 0.4\\0.293\\0.293\\0.154\\0.154\\0.154\\0.0429\\0.0429\\0.0429\\0.0429\\0.0429\\0.0429\\0.0429\\0.0429\\0.0429 \end{bmatrix}, \begin{bmatrix} 0.667\\0.345\\0.345\\0.0873\\0.0873\\0.0873\\0.0665\\0.0665\\0.0665\\0.0665\\0.0397\\0.0397\\0.0397\\0.0397\\0.0397 \end{bmatrix}, \begin{bmatrix} 1\\0.25\\0.25\\0.111\\0.111\\0.111\\0.0625\\0.0625\\0.0625\\0.0625\\0.04\\0.04\\0.04\\0.04\\0.04 \end{bmatrix}$$

San Diego State University

Contrasting Example: "Maximal Subspace Collapse" 1 of 2

What happens is we select a sequence of search directions which collapse the maximal number of dimensions of the residual (which coincidentally, in this example, is the subspace corresponding to the largest eigenvalue)?

$$\bar{\mathbf{p}}_0 = \begin{bmatrix} 0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\1\\1\\1\\1 \end{bmatrix}, \quad \bar{\mathbf{p}}_1 = \begin{bmatrix} 0\\0\\0\\0\\0\\0\\1\\1\\1\\1\\0\\0\\0\\0\\0 \end{bmatrix}, \quad \bar{\mathbf{p}}_2 = \begin{bmatrix} 0\\0\\0\\1\\1\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}, \quad \bar{\mathbf{p}}_3 = \begin{bmatrix} 0\\1\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}, \quad \bar{\mathbf{p}}_4 = \begin{bmatrix} 1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}$$

San Diego State University

Contrasting Example: "Maximal Subspace Collapse" 2 of 2

We get the following sequence of residuals

$$\bar{\mathbf{r}}_1^{\mathrm{MSC}} \approx \begin{bmatrix} 1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\0\\0\\0\\0\\0 \end{bmatrix}, \quad \bar{\mathbf{r}}_2^{\mathrm{MSC}} \approx \begin{bmatrix} 1\\1\\1\\1\\1\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}, \quad \bar{\mathbf{r}}_3^{\mathrm{MSC}} \approx \begin{bmatrix} 1\\1\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}, \quad \bar{\mathbf{r}}_4^{\mathrm{MSC}} \approx \begin{bmatrix} 1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}, \quad \bar{\mathbf{r}}_5^{\mathrm{MSC}} \approx \begin{bmatrix} 0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}$$

	$\|\bar{\mathbf{r}}_0\|$	$\|\bar{\mathbf{r}}_1\|$	$\|\bar{\mathbf{r}}_2\|$	$\|\bar{\mathbf{r}}_3\|$	$\|\bar{\mathbf{r}}_4\|$	$\|\bar{\mathbf{r}}_5\|$
CG	$\sqrt{15}$	2.1603	1.5492	1.1339	0.7454	≈ 0
MSC	$\sqrt{15}$	3.1623	2.4495	1.7321	1.0000	≈ 0

Table: CG gives the optimal sequence of residual lengths at each iteration!

San Diego State University

Contrasting Example #2: Steepest Descent

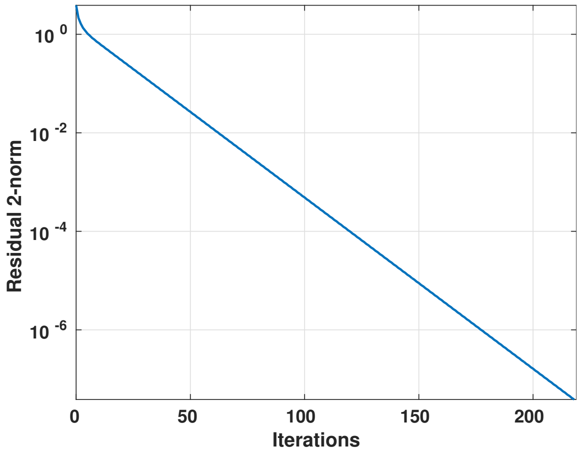

Figure: Since the condition number is only $5^2 = 25$, one may think that maybe steepest descent will do a decent job? But, alas, it takes over 200 iterations to get a reduction of the residual norm by 10^{-8}.

The CG Algorithm — Convergence: More Comments

It is worth noting that the theorem gives an **upper bound** of the error, in practice it is *almost* true that if the eigenvalues of A occur in r distinct clusters, then (compare with the first theorem) the CG algorithm will approximately solve the problem after r steps.

Further it can be shown that for the CG algorithm

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

whereas (forgotten from lecture #5)

$$\|\bar{\mathbf{x}}_{k+1} - \bar{\mathbf{x}}^*\|_A \leq \left[\frac{\kappa(A)-1}{\kappa(A)+1}\right]^k \|\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}^*\|_A,$$

for the steepest descent algorithm. Here $\kappa(A) = \lambda_n/\lambda_1$ is the condition number of the matrix A.

CG vs. Steepest Descent 1 of 2

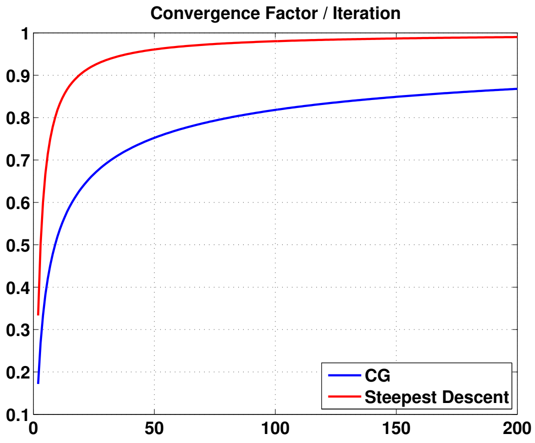

Figure: Comparing the convergence factors $\left[\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right]$ (for Conjugate Gradient) and $\left[\frac{\kappa(A)-1}{\kappa(A)+1}\right]$ (for Steepest Descent) for condition numbers, $\kappa(A) \in [2, 200]$.

CG vs. Steepest Descent 2 of 2

$\kappa(A)$	**CG**	**SD**	**SD/CG**
10	22	69	3.136
100	72	691	9.597
1,000	229	6,908	30.166
10,000	725	69,078	95.280

Table: A comparison of how many iterations Conjugate gradient (CG) and Steepest descent (SD) are required in order to reduce the initial error $\|\bar{\mathbf{x}}_0 - \bar{\mathbf{x}}^*\|_A$ by a factor of 10^{-6}. We notice a "slight" improvement. The speedup is $\sim \sqrt{\kappa(A)}$.

Preconditioning: Speeding Things Up Even More

The CG method can be accelerated further by **preconditioning** the linear system; we [FORMALLY] make a non-singular change of variables

$$\widehat{\mathbf{x}} = C\bar{\mathbf{x}},$$

and solve the linear system (and/or its equivalent minimization problem, $\min \widehat{\Phi}(\bar{\mathbf{x}})$)

$$\left[C^{-T}AC^{-1}\right]\widehat{\mathbf{x}} - C^{-T}\bar{\mathbf{b}}, \quad \widehat{\Phi}(\bar{\mathbf{x}}) = \frac{1}{2}\widehat{\mathbf{x}}^T\left[C^{-T}AC^{-1}\right]\widehat{\mathbf{x}} - \left[C^{-T}\bar{\mathbf{b}}\right]^T\widehat{\mathbf{x}}.$$

Now, the convergence rate will depend on the eigenvalues of $\mathcal{A} = \left[C^{-T}AC^{-1}\right]$. Therefore, we would like to choose C such that the eigenvalues of $\mathcal{A}$ are favorably clustered, and/or the condition number of $\mathcal{A}$ is less than that of A.

As in the case of the transformation guaranteeing n-step convergence, this change of variables does not have to be done explicitly.

The Preconditioned CG Algorithm (a.k.a. "PCG")

Algorithm: PCG

```
Given $A$, $\mathbf{M} = \mathbf{C}^T\mathbf{C}$, $\bar{\mathbf{b}}$ and $\bar{\mathbf{x}}_0$:  compute $\bar{\mathbf{r}}_0 = A\bar{\mathbf{x}}_0 - \bar{\mathbf{b}}$,
$\bar{\mathbf{y}}_0 = \mathbf{M}^{-1}\bar{\mathbf{r}}_0$, $\bar{\mathbf{p}}_0 = -\bar{\mathbf{y}}_0$, $k = 0$
while ( $\|r_k\| > 0$, or other stopping condition )
    $\alpha_k = \dfrac{\bar{\mathbf{r}}_k^T\bar{\mathbf{y}}_k}{\bar{\mathbf{p}}_k^T A\bar{\mathbf{p}}_k}$,        Store the vector $A\bar{\mathbf{p}}_k$
                                  and the scalar $\bar{\mathbf{r}}_k^T\bar{\mathbf{y}}_k$
    $\bar{\mathbf{x}}_{k+1} = \bar{\mathbf{x}}_k + \alpha_k\bar{\mathbf{p}}_k$
    $\bar{\mathbf{r}}_{k+1} = \bar{\mathbf{r}}_k + \alpha_k A\bar{\mathbf{p}}_k$
    $\bar{\mathbf{y}}_{k+1} = \mathbf{M}^{-1}\bar{\mathbf{r}}_{k+1}$
    $\beta_{k+1} = \dfrac{\bar{\mathbf{r}}_{k+1}^T\bar{\mathbf{y}}_{k+1}}{\bar{\mathbf{r}}_k^T\bar{\mathbf{y}}_k}$,      Save the numerator for next step!
    $\bar{\mathbf{p}}_{k+1} = -\bar{\mathbf{y}}_{k+1} + \beta_{k+1}\bar{\mathbf{p}}_k$
    $k = k + 1$
end-while
```

PCG: Comments

If we set $M = I$, then we recover standard CG.

We note that in each iteration, we have to solve the linear system $M\bar{\mathbf{y}}_{k+1} = \bar{\mathbf{r}}_{k+1} \Leftrightarrow (\bar{\mathbf{y}}_{k+1} = M^{-1}\bar{\mathbf{r}}_{k+1})$. We must select M so that we can do this quickly, otherwise we lose the overall-work advantage over Steepest descent or Gaussian elimination.

There are several (usually competing) properties we would like M to have:

— M should effectively impact the structure of the eigenvalues.

— M should be cheap to compute and store.

— The linear system $M\bar{\mathbf{y}} = \bar{\mathbf{r}}$ should be "easy."

Finding Good Preconditions 1 of 2

There is no "best" way of finding M. The optimal M for a particular A may even depend on how much memory, etc your computer has.

In general M is a simplified version of A, *e.g.* we may take the tridiagonal part of A:

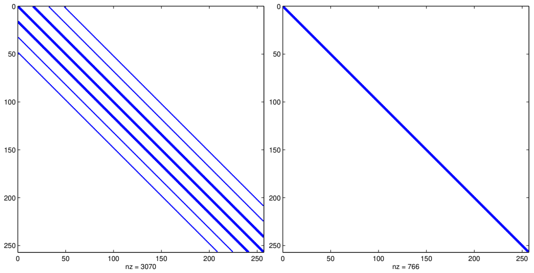

Figure: When A has a banded structure (left) with a significant bandwidth, then a tri-diagonal preconditioner M (right) may be a good choice. Recall that in this case we can solve $M\bar{\mathbf{y}} = \bar{\mathbf{r}}$ in $\mathcal{O}(n)$ operations.

Finding Good Preconditioners 2 of 2

Preconditioning is itself a science (or an art?) which will be revisited in more detail in **(???)**.

One of the more efficient strategies is **incomplete Cholesky factorization**. — The exact Cholesky factorization of an SPD matrix A has the form

$$LL^T = A, \quad \text{where } L \text{ is lower triangular.}$$

Usually, even though A may be sparse, L will be dense (due to **fill-ins**).

In incomplete Cholesky factorization, the same algorithm is followed, but whenever a fill-in occurs, that value is dropped — this way we end up with

$$M = \tilde{L}\tilde{L}^T \approx A,$$

where $\tilde{L}$ and A have the same sparsity patterns.

Homework #4, Problem #1 of 2 Due 11/2/2018, 12:00pm

Problem #1

Implement the standard CG algorithm, and use it to solve linear systems describing the "Helical Coordinate Preconditioner for the Laplacian" in 1D, 2D, and 3D:

Matlab-centric problem matrices; push n until you run out of memory (or patience!)

d = ones(n,1); A = spdiags([d -2*d d], [-1 0 1], n, n);
d = ones(n^2,1); A = spdiags([d d -4*d d d], [-n -1 0 1 n], n^2, n^2);
d = ones(n^3,1); A = spdiags([d d d -6*d d d d], [-n^2 -n -1 0 1 n n^2], n^3, n^3);

Ponder matrix size; number of iterations to drive the initial residual (given the initial guess of all zeros $\vec{x}_0 =$ `zeros(size(d))`, with a right-hand-side of all ones $\vec{b} =$ `ones(size(d))`) to a residual of size *tol* × *initial residual*; execution time; condition numbers; non-zero matrix elements; total # of matrix elements, etc...

Homework #4, Problem #2 of 2

Problem #2

A modified version of NW$^{\text{1st}}$-5.1: Implement the standard CG algorithm, and use it to solve linear systems in which A is the Hilbert matrix, whose elements are $a_{ij} = 1/(i + j - 1)$. Set the right-hand-side to be all ones $\vec{b} =$ `ones(n,1)`, and the initial point to be the origin $\vec{x}_0 =$ `zeros(n,1)`. For dimensions n = 5, 8, 12, 20, plot the norm of the residual as a function of the iteration; stop when the norm is less than 10^{-6}

Note: The Hilbert matrix shows up in the normal equations in least squares approximations, and is an example of a matrix with a nasty condition number.

Compute the condition number for your matrices, and plot the spread of the eigenvalues. From the formulas, estimate how many steepest descent iterations you would need to solve the problem to the same precision. (Can you get a meaningful estimate?)

Index