Krylov Methods

C. T. Kelley
NC State University
tim_kelley@ncsu.edu
Research Supported by NSF, DOE, ARO, USACE

DTU ITMAN, 2011
Outline

Krylov Methods Overview

GMRES
  Analysis of GMRES
  GMRES Implementation

Conjugate Gradient Iteration
  Analysis of CG
  CG Implementation

Other Krylov Methods

Preconditioning

Exercises
References for Krylov Methods I


References for Krylov Methods II


References for Krylov Methods III


Krylov Methods

- Krylov iterative methods obtain $x_n$ from the history of the iteration.
- The ones with theory do this by minimizing an error or residual function over the affine space $x_0 + \mathcal{K}_k$
- $x_0$ is the initial iterate
- $\mathcal{K}_k$ is the $k$th Krylov subspace

$$\mathcal{K}_k = \text{span}(r_0, Ar_0, \ldots, A^{k-1}r_0)$$

for $k \geq 1$. 
Terms and Notation Review

- Equation \( Ax = b \); Solution \( x^* = A^{-1}b \)
- Error \( e = x - x^* \)
- Residual \( b - Ax = Ae \)
These two methods can be expressed in terms of minimization principles.

In GMRES (Generalized Minimum Residual), the $k$th iteration $x_k$ minimizes the residual over $x_0 + \mathcal{K}_k$

$$\| b - Ax_k \| = \min_{x \in x_0 + \mathcal{K}_k} \| b - Ax \|$$

for $\| \cdot \| = \| \cdot \|_2$. For CG, $A$ must be spd and $x_k$ minimizes the $A$-norm of the error.

$$\| x^* - x \|_A = \min_{x \in x_0 + \mathcal{K}_k} \| x^* - x \|_A$$

where

$$\| v \|_A^2 = v^T A v.$$
General Properties of CG/GMRES

- convergence in $N$ iterations (impractical)
- no need for matrix representation of $A$ only matrix-vector products
- sensitive to conditioning and eigenvalue distribution
If \( x \in x_0 + \mathcal{K}_k \) then

\[ r = b - Ax = b - Ax_0 - \sum_{j=0}^{k} \gamma_j A^j r_0 \equiv p(A) r_0 \]

where \( p \in \mathcal{P}_k \), the set of \( k \) degree residual polynomials.

\[ \mathcal{P}_k = \{ p \mid p \text{ is a polynomial of degree } k \text{ and } p(0) = 1. \} \]

This simple observation is the key to analysis of Krylov methods.
Theorem: Let $A$ be nonsingular and let $x_k$ be the $k$th GMRES iteration. Then for all $\bar{p}_k \in P_k$

$$\|r_k\| = \min_{\bar{p} \in P_k} \|p(A)r_0\| \leq \|\bar{p}_k(A)r_0\|.$$
Proof of Theorem

Let $x_k$ the the $k$th GMRES iteration. Then there is $p_k \in \mathcal{P}_k$ such that

$$r_k = b - Ax_k = p_k(A)r_0$$

Since any $x \in x_0 + \mathcal{K}_k$ satisfies

$$r = b - Ax = \bar{p}(A)r_0$$

for some $\bar{p} \in \mathcal{P}_k$, the minimization principle imples that

$$\|r_k\|_2 = \min_{p \in \mathcal{P}_k} \|p(A)r_0\| \leq \|b - Ax\| = \|\bar{p}_k(A)r_0\|.$$
How to Use This Theorem

- Connect properties of the matrix to a polynomial you understand.
- Manufacture a residual polynomial $\bar{p}$ from that.
- Get an upper bound from

$$
\| r_k \| \leq \| \bar{p}(A)r_0 \| \leq \| \bar{p}(A) \| \| r_0 \| 
$$
Corollary: Let $A$ be nonsingular. Then the GMRES algorithm will find the solution within $N$ iterations.

Proof: The characteristic polynomial of $A$ is $p(z) = \text{det}(A - zI)$. $p$ has degree $N$, $p(0) = \text{det}(A) \neq 0$ since $A$ is nonsingular, and so

$$\bar{p}_N(z) = p(z)/p(0) \in \mathcal{P}_N$$

is a residual polynomial. The Cayley-Hamilton theorem says that $\bar{p}_N(A) = 0$, and so

$$\|r_n\| \leq \|\bar{p}_N(A)\| \|r_0\| = 0.$$
Corollary: If \( \| I - A \| \leq \rho < 1 \) then

\[
\| r_k \| \leq \rho^k \| r_0 \|_2.
\]

Proof: Let \( \bar{p}_k = (1 - z)^k \) and use the theorem.
Diagonalizable Matrices

A is **diagonalizable** if there is a nonsingular (possibly complex!) matrix \( V \) such that

\[ A = V \Lambda V^{-1}. \]

If \( A \) is diagonalizable and \( p \) is a polynomial then

\[
p(A) = \sum_{j=0}^{m} a_j \gamma_j A^j = \sum_{j=0}^{m} a_j (V \Lambda V^{-1})^j = V \sum_{j=0}^{m} a_j \Lambda^j V^{-1} = V p(\Lambda) V^{-1}
\]

So

\[
\|p(A)\| \leq \|V\| \|p(A)\| \|V^{-1}\| = \kappa(V) \max_{\lambda \in \sigma(A)} |p(\lambda)|
\]
We just proved . . .

**Theorem:** Let \( A = \mathbf{V} \Lambda \mathbf{V}^{-1} \) be a nonsingular diagonalizable matrix. Let \( x_k \) be the \( k \)th GMRES iterate. Then for all \( \mathbf{p}_k \in \mathcal{P}_k \)

\[
\frac{\|r_k\|_2}{\|r_0\|_2} \leq \kappa_2(\mathbf{V}) \max_{z \in \sigma(A)} |\mathbf{p}_k(z)|.
\]
If $A$ has $m$ distinct eigenvalues then GMRES will terminate in at most $m$ iterations.

Proof: Use

$$p(z) = \prod_{i=1}^{m} \left( \frac{\lambda_i - \lambda}{\lambda_i} \right)$$

$p(0) = 1$ so $p \in \mathcal{P}_k$. Since $p(\lambda_i) = 0$ for all $i$, $r_N = 0$. This proof is (1) very easy and (2) typical of the way one thinks about Krylov methods.
Let $x_0 = 0$ (so $r_0 = b$) and assume that

- $\sigma(A) \subset (9, 11)$
- $\kappa(V) = 100$

Then if we let $\bar{p}_k(z) = (10 - z)^k/10^k$ we see that

$$\frac{\|r_k\|_2}{\|r_0\|_2} \leq \kappa(V)\|p_k(A)\| \leq (100)10^{-k} = 10^{2-k}.$$ 

So $\|r_k\| \leq \eta\|b\|$ when

$$k > 2 + \log_{10}(\eta).$$

This tells us that an approximate inverse preconditioner could be useful.
Observations

- A normal implies $\kappa(V) = 1$
- If $A$ is not normal and $\kappa(V)$ is large, then $\sigma(A)$ does not tell the whole story.
- The heuristic is that if the eigenvalues are grouped into a few clusters the iteration will perform well.
- If the eigenvalues are clustered near 1, the GMRES is very happy and $A$ is well-conditioned.
Preconditioning means to replace $Ax = b$ with

$$B Ax = Bb \text{ (left)}$$

or

$$ABy = b \text{ (right)}, \text{ and then } x = By$$

and solve the preconditioned equation with GMRES. The preconditioner $B$ should be

- very inexpensive matrix-vector products
- be a good approximate inverse of (part) of $A$

Examples coming later.
Left Preconditioning

Solve

\[ BAx = Bb \]

so

- solution to preconditioned equation is still \( x \)
- preconditioned residual \( Bb - BAx = Br \) should be a better indicator of error
Solve

\[ ABz = b \]

for \( z \). Then set \( x = Bz \).

- The preconditioned residual is the same as the original residual because \( b - A(Bz) = b - Ax \).
- The solution of the preconditioned problem is different.
- The residual may not be a good indicator of the error in \( x \).

More on preconditioning later. But first . . .
And now for the software
The $k$th GMRES iteration is the solution of the linear least squares problem

$$\min \|Ax - b\|$$

where $x = \sum_{j=0}^{k-1} \gamma_j A^j r_0$

The key to a successful implementation is to solve this in an efficient and stable way.
A Questionable GMRES Implementation

How about this?

- As the iteration progresses store $A^j r_0$.
- Let $B_k = (r_0, Ar_0, \ldots, A^{k-1} r_0)$
- Compute the QR factorization of $B_k = Q_k R_k$
- The $x_k = R_k^{-1} Q_k^T b$

What could go wrong?
What could go wrong?

- Accumulating $A^j r_0$ can be unstable
  Example $A = \text{diag}(1, 2, ..., N)$
- The cost of $B_k = Q_k R_k$ is $O(Nk^2)$.
- You have to start over with each $k$ and are not reusing the old columns.
Arnoldi Factorization is Better

Suppose one had an orthogonal projector $V_k$ onto $\mathcal{K}_k$. Then any $z \in \mathcal{K}_k$ can be written as

$$z = \sum_{l=1}^{k} y_l v_i^k$$

where $v_i^k$ is the $i\text{th}$ column of $V_k$. So we can convert the problem for $x_k$ to a problem in $R^k$. Begin by writing any $x \in x_0 + \mathcal{K}_k$ as

$$x = x_0 + V_k y,$$

where $y$ is the vector of coefficients of $x - x_0$ using the columns of $V_k$ as the basis for $\mathcal{K}_k$. 
Arnoldi Part II

So if \( x_k = x_0 + V_k y_k \) then

\[
\| b - Ax_k \| = \| b - A(x_0 + V_k y_k) \|_2 = \| r_0 - AV_k y_k \|_2.
\]

So the least squares problem for \( y \) is

\[
\min \| r_0 - AV_k y \|
\]

If we can build \( V_k \) in a stable way, we have solved the stability problem (but that is not completely simple). Can we do it efficiently?
The Gram-Schmidt process will

- build $V_k$ incrementally, so $V_k = (V_{k-1}, v_k)$,
- enable a fast $QR$ factorization of $AV_k$, and
- be stable (if done correctly).

Orthogonalization is the central part of the **Arnoldi** method.
Arnoldi Part IV

The algorithm **orthogonalizes** each $Av_i$ against the columns of $V_{k-1}$ to construct $v_k$

$V = \text{arnoldi}(x_0, b, A, k)$

$r_0 = b - Ax_0; \quad v_1 = r_0 / \| r_0 \|$

**for** $i = 1 : k - 1$ **do**

$w = Av_i$

**for** $j = 1 : i$ **do**

$h_{ji} = w^T v_j (= (Av_i)^T v_j); \quad w = w - h_{ji} v_j$

**end for**

$h_{ki} = \| w \|; \quad v_{i+1} = w / h_{ki}$

**end for**

At the end you have $V_k$. Columns orthonormal basis for $K_k$. 
Examine the Arnoldi Loops

What if you divide by zero in

\[ v_1 = \frac{r_0}{\|r_0\|} \text{ or } v_{i+1} = \frac{w}{\|w\|} \]?

- If \( r_0 = 0 \), then \( x_0 \) is the solution and the GMRES iteration would terminate.
- If \( w = 0 \), then you have a happy breakdown of the Arnoldi process. This implies that you found the solution as \( x_{k-1} \).
- A well-designed implementation would stop before division by zero.
The Happy Breakdown Theorem

**Theorem:** Let $A$ be nonsingular, let the vectors $v_j$ be generated by the Arnoldi process, and for which

$$Av_i - \sum_{j=1}^{i} ((Av_i)^T v_j) v_j = 0.$$  

Then $x = A^{-1}b \in x_0 + K_i$. 
Proof: The Happy Breakdown Theorem

- By hypothesis $Av_i \in \mathcal{K}_i$, so $A\mathcal{K}_i \subset \mathcal{K}_i$.
- The columns of $V_i$ are an orthonormal basis for $\mathcal{K}_i$, so $AV_i = V_iH$ where $H$ is an $i \times i$ matrix. $H$ is nonsingular since $A$ is.
- Set $\beta = \|r_0\|_2$ and $e_1 = (1, 0, \ldots, 0)^T \in \mathbb{R}^i$, then $\|r_i\|_2 = \|b - Ax_i\|_2 = \|r_0 - A(x_i - x_0)\|_2$.
- Now, $x_i - x_0 \in \mathcal{K}_i$ so there is $y \in \mathbb{R}^i$ such that $x_i - x_0 = V_iy$.
- Since $r_0 = \beta V_i e_1$ and $V_i$ is an orthogonal matrix
  \[ \|r_i\|_2 = \|V_i(\beta e_1 - Hy)\|_2 = \|\beta e_1 - Hy\|_{R^i}, \]
- Set $y = \beta H^{-1} e_1$ to show $r_i = 0$. 
What about $H$?

- Assuming that there is no breakdown, then
  \[ h_{ij} = (Av_j)^T v_i = 0 \text{ if } i > j + 1, \text{ so } H \text{ is upper Hessenberg.} \]
- So, the Arnoldi process produces $AV_k = V_{k+1}H_k$.
- This means (with $\beta = \|r_0\|$)
  \[
  r_k = b - Ax_k = r_0 - A(x_k - x_0) = V_{k+1}(\beta e_1 - H_ky^k).
  \]
- Hence $x_k = x_0 + V_ky^k$, where $y^k$ minimizes $\|\beta e_1 - H_ky\|_2$.
- This is great. We can test for termination without wasting a matrix-vector product to compute $b - Ax_k$ by testing
  \[
  \|r_k\| = \|\beta e_1 - H_ky_k\|
  \]
A Framework for GMRES Implementation

\[ r = b - Ax, \quad v_1 = r / \| r \|_2, \quad \rho = \| r \|_2, \quad \beta = \rho, \quad k = 0 \]

\[ \textbf{while } \rho > \epsilon \| b \|_2 \textbf{ and } k < k_{\text{max}} \textbf{ do} \]

\[ k = k + 1 \]

Apply Arnoldi to obtain \( H_k \) and \( V_{k+1} \) from \( V_k \) and \( H_{k-1} \)

\[ e_1 = (1, 0, \ldots, 0)^T \in R^{k+1} \]

Solve \( \min \| \beta e_1 - H_k y_k \|_{R^{k+1}} \) for \( y_k \in R^k \).

\[ \rho = \| \beta e_1 - H_k y_k \|_{R^{k+1}}. \]

\[ \textbf{end while} \]

\[ x_k = x_0 + V_k y_k. \]
Orthogonalization: Classical Gram-Schmidt

for $j = 1 : k$ do
  $h_{jk} = (A v_k)^T v_j$
end for

$v_{k+1} = A v_k - \sum_{j=1}^{k} h_{jk} v_j$
$h_{k+1,k} = \|v_{k+1}\|_2$
$v_{k+1} = v_{k+1}/\|v_{k+1}\|_2$

Advantage (huge): the for loop is trivially parallel/vectorizable.
Disadvantage: unstable, which means . . .
Instability in Orthogonalization

- Classical Gram-Schmidt can produce $V$’s with non-orthogonal columns.
- In this case, the reduction to upper Hessenberg form is wrong,
- and $\|r_k\| \neq \|\beta e_1 - H_k y_k\|$. 

So we have to fix it.


Classical Gram-Schmidt Twice

\[ \text{for } j = 1 : k \text{ do} \]
\[ h_{jk} = (A v_k)^T v_j \]
\[ \text{end for} \]
\[ v_{k+1} = A v_k - \sum_{j=1}^{k} h_{jk} v_j \]
\[ \text{for } j = 1 : k \text{ do} \]
\[ \tilde{h}_{jk} = v_{k+1}^T v_j \]
\[ h_{jk} = h_{jk} + \tilde{h}_{jk} \]
\[ \text{end for} \]
\[ v_{k+1} = v_{k+1} - \sum_{j=1}^{k} \tilde{h}_{jk} v_j \]
\[ h_{k+1,k} = \| v_{k+1} \| \]
\[ v_{k+1} = v_{k+1} / \| v_{k+1} \| \]

Still parallel, but twice the work.
Orthogonalization: Modified Gram-Schmidt (MGS)

\[ v_{k+1} = Av_k \]

\[ \text{for } j = 1 : k \text{ do} \]

\[ h_{jk} = v_{k+1}^T v_j \]

\[ v_{k+1} = v_{k+1} - h_{jk} v_j \]

\[ \text{end for} \]

\[ \text{if Loss of orthogonality then} \]

\[ \text{Reorthogonalize} \]

\[ \text{end if} \]

\[ h_{k+1,k} = \| v_{k+1} \|_2 \]

\[ v_{k+1} = v_{k+1} / \| v_{k+1} \|_2 \]

More stable than CGS, but parallelism is lost.
Test for loss of orthogonality

If

$$\|Av_k\|_2 + \delta\|v_{k+1}\|_2 = \|Av_k\|_2$$

to working precision, then you should reorthogonalize because there is very little information in $v_{k+1}$. MGS and the test is the default in our MATLAB codes, but ...
Observations

- If you have as few as four cores, CGS-twice is faster.
- Storage is the main problem with GMRES.
- Low-storage methods for non-symmetric matrices have problems (more later).
Solving upper-Hessenberg Least Squares Problems

The last thing to do is to solve

$$\min ||\beta e_1 - H_k y||.$$ 

We do this by forming the QR factorization of $H_k$ with Givens rotations.
A $2 \times 2$ **Givens rotation** is a matrix of the form

$$G = \begin{pmatrix} c & -s \\ s & c \end{pmatrix}$$

(1)

where $c = \cos(\theta)$, $s = \sin(\theta)$ for $\theta \in [-\pi, \pi]$. $G$ rotates a vector in $\mathbb{R}^2$ by $\theta$. In particular

$$G \begin{pmatrix} c \\ -s \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$
Givens Rotations: II

An $N \times N$ Givens rotation replaces a $2 \times 2$ block on the diagonal of the $N \times N$ identity matrix with a $2 \times 2$ Givens rotation.

$$G_j = \begin{pmatrix}
1 & 0 & \cdots & 0 \\
0 & \ddots & \ddots & \ddots \\
\vdots & \ddots & c & -s \\
0 & 1 & \ddots & \ddots \\
0 & \cdots & \cdots & 0 \\
0 & \cdots & \cdots & 0
\end{pmatrix}$$  \hspace{1cm} (2)

Columns $j$ and $j + 1$ are changed.
Givens Rotations: III

To build the $QR$ factorization of $H_k$, we apply Givens rotations. Step 1: Multiply $H_k$ by a Givens rotation that annihilates $h_{21}$ (and, of course, changes $h_{11}$ and the subsequent columns). We define $G_1 = G_1(c_1, s_1)$ by

$$\begin{align*}
c_1 &= h_{11}/\sqrt{h_{11}^2 + h_{21}^2} \quad \text{and} \quad s_1 = -h_{21}/\sqrt{h_{11}^2 + h_{21}^2}.
\end{align*}$$

Then $R_k \leftarrow G_1 H_k$ has zero in the 22 entry.

Step 2: Multiply $R$ by $G_2(c_2, s_2)$ where

$$\begin{align*}
c_2 &= h_{22}/\sqrt{h_{22}^2 + h_{32}^2} \quad \text{and} \quad s_1 = -h_{32}/\sqrt{h_{22}^2 + h_{32}^2}.
\end{align*}$$

Continue . . .
Continuing we obtain, at the end,

\[ R_k = G_k \ldots G_1 H_k \]

is upper triangular. Set

\[ Q_k = (G_k \ldots G_1)^T \]

and \( H_k = Q_k R_k \). Cost = \( O(N) \).
The implementation stores $Q_k$ by
- storing the sequences $\{c_j\}$ and $\{s_j\}$
- computing the action of $Q_k$ on a vector $x \in \mathbb{R}^{k+1}$ by applying $G_j(c_j, s_j)$
- and obtain $Q_k x = G_1(c_k, s_k)^T \ldots G_k(c_1, s_1)^T x$.
- We overwrite $H_k$ with the triangular part of the QR factorization of $H_k$, so
- we do not store $H_k$, rather $R_k$. 
Givens Rotations: $V$

At iteration $k$ you have $H_{k-1}$ overwritten with $R_{k-1}$

- $g = \rho(1, 0, \ldots, 0)^T \in \mathbb{R}^k$
- Compute $h_{jk}$ for $1 \leq j \leq k + 1$
- $Q_k = I$

1. If $k > 1$ apply $Q_{k-1}$ to the $k$th column of $H$.
2. $\nu = \sqrt{h_{k,k}^2 + h_{k+1,k}^2}$.
3. $c_k = h_{k,k}/\nu$, $s_k = -h_{k+1,k}/\nu$
   
   $h_{k,k} = c_k h_{k,k} - s_k h_{k+1,k}$, $h_{k+1,k} = 0$

4. $g = G_k(c_k, s_k)g$.
5. $Q_k^T = G_k Q_{k-1}^T$.
6. $\rho = \|(g)_{k+1}\|$.
CG’s Minimization Principle

Solve $Ax = b$ where $A$ is spd.
For CG, $x_k$ minimizes the $A$-norm of the error

$$\|x^* - x\|_A = \min_{x \in x_0 + \mathcal{K}_k} \|x^* - x\|_A$$

over $x_0 + \mathcal{K}_k$, where

$$\|v\|_A^2 = v^T A v.$$
As with GMRES, any \( x \in x_0 + \mathcal{K}_k \) can be written

\[
x = x_0 + \sum_{j=0}^{k-1} \gamma_j A^j r_0
\]

Let \( x^* = A^{-1} b \) and \( e = x^* - x \). Since \( r = b - Ax = Ae \),

\[
x^* - x = e = x^* - x_0 - \sum_{j=0}^{k-1} \gamma_j A^j r_0
= e_0 - \sum_{j=1}^{k} \gamma_j A^j e_0 = p(A)e_0
\]

for some \( p \in \mathcal{P}_k \).
Minimization Principle

So, if $x_k$ is the $k$th CG iteration

$$\|e_k\|_A \leq \|p(A)e_0\|_A$$

for all $p \in \mathcal{P}_k$.

So what does this mean?
What is the $A$-norm of $p(A)$

Since $A$ is spd, $A$ has a unique spd square root,

$$A = U \Lambda U^T \quad \text{and} \quad \sqrt{A} = U \sqrt{\Lambda} U^T$$

so

$$\|x\|_A^2 = x^T Ax = (\sqrt{A}x)^T (\sqrt{A}x) = \|\sqrt{A}x\|^2$$

which means

$$\|p(A)x\|^2 = \|\sqrt{A}p(A)x\|^2 = \|p(A)(\sqrt{A}x)\|^2$$

Hence

$$\|p(A)\|_A = \max_{\lambda \in \sigma(A)} |p(\lambda)|$$
As with GMRES

\[ \| e_k \|_A \leq \max_{\lambda \in \sigma(A)} |p(\lambda)| \| e_0 \|_A \]

So, for example, if \( \sigma(A) \subset (.9, .1) \) then

\[ \| e_k \|_A \leq \| e_0 \| 10^{-k} \]

which we get by using \( p(z) = (1 - z)^k \).
Convergence within $N$ Iterations

**Theorem:** Let $A$ be spd. Then the CG algorithm will find the solution within $N$ iterations.

**Proof:** Use

$$p(z) = \prod \left( \frac{\lambda_i - z}{\lambda_i} \right)$$
Theorem: Let $0 < \lambda_1 \leq \lambda_2 \leq \lambda_N$ be the eigenvalues of $A$ (so $\kappa(A) = \lambda_N/\lambda_1$). Let $x_k$ be the $k$th CG iteration. Then

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq \left[ \frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right]^k.$$ 

This can be pessimistic if the eigenvalues are clustered.
Termination

It's standard to terminate the iteration when the residual is small

\[ \|r_k\| \leq \|b - Ax_k\| \leq \eta \|r_0\|. \]

How is this connected to the \(A\)-norm of \(e\)?

Since

\[ \sqrt{\lambda_1} \|x\| \leq \|x\|_A \leq \sqrt{\lambda_N} \|x\| \]

we have

\[ \frac{\|r_k\|}{\|r_0\|} = \frac{\|Ae_k\|}{\|Ae_0\|} \leq \sqrt{\kappa(A)} \frac{\sqrt{\lambda} \|e_k\|}{\sqrt{\lambda} \|e_0\|} = \sqrt{\kappa(A)} \frac{\|e_k\|_A}{\|e_0\|_A} \]
Example

Let $x_0 = 0$ and assume $\sigma(A) \subset (9, 11)$. Using $p(z) = (10 - z)^k / 10^k$ we see

$$\|e_k\|_A / \|e_0\|_A \leq 10^{-k}.$$  

So the $A$-norm of the error will be reduced by a factor of $10^{-3}$ after 3 iterations.

What about the residual? All we know is that $\kappa(A) \leq 11/9$, so

$$\frac{\|r_k\|}{\|r_0\|} \leq 10^{-k} \sqrt{11/9}$$

and we need 4 iterations to guarantee a residual reduction of $10^{-3}$. 
Alternative Minimization Principle

Theorem: The \( k \)th iterate \( x_k \) of CG minimizes

\[
\phi(x) = \frac{1}{2} x^T A x - x^T b
\]

over \( x_0 + K_k \)

Remark: Note that if \( \tilde{x} \) is any a stationary point,

\[
\nabla \phi(\tilde{x}) = A \tilde{x} - b = 0
\]

then \( \tilde{x} = x^* \).
Proof

Note that

\[ \|x - x^*\|_A^2 = (x - x^*)^T A (x - x^*) = x^T Ax - x^T A x^* - (x^*)^T A x + (x^*)^T A x^*. \]

Since \( A \) is symmetric and \( Ax^* = b \)

\[ -x^T A x^* - (x^*)^T A x = -2x^T A x^* = -2x^T b. \]

Therefore

\[ \|x - x^*\|_A^2 = 2\phi(x) + (x^*)^T A x^*. \]

So \( x \) minimizes \( \phi \) over any set if and only if \( x \) minimizes \( \|x - x^*\|_A^2 \).
CG Implementation

cg(x, b, A, \epsilon, kmax)

\[ r = b - Ax, \quad \rho_0 = \|r\|_2^2, \quad k = 1. \]

while \( \sqrt{\rho_{k-1}} > \epsilon \|b\| \) and \( k < k_{\text{max}} \) do

if \( k = 1 \) then

\[ p = r \]

else

\[ \beta = \rho_{k-1}/\rho_{k-2} \] and \( p = r + \beta p \)

end if

\[ w = Ap \]
\[ \alpha = \rho_{k-1}/p^Tw \]
\[ x = x + \alpha p \]
\[ r = r - \alpha w \]
\[ \rho_k = \|r\|^2 \]
\[ k = k + 1 \]

end while
CG Implementation: Cost I, two scalar products

\[ \text{cg}(x, b, A, \epsilon, k_{\text{max}}) \]

\[ r = b - Ax, \quad \rho_0 = \|r\|^2, \quad k = 1. \]

\[ \text{while } \sqrt{\rho_{k-1}} > \epsilon \|b\| \text{ and } k < k_{\text{max}} \text{ do} \]

\[ \text{if } k = 1 \text{ then} \]
\[ p = r \]
\[ \text{else} \]
\[ \beta = \rho_{k-1}/\rho_{k-2} \text{ and } p = r + \beta p \]
\[ \text{end if} \]
\[ w = Ap \]
\[ \alpha = \rho_{k-1}/p^T w \]
\[ x = x + \alpha p \]
\[ r = r - \alpha w \]
\[ \rho_k = \|r\|^2 \]
\[ k = k + 1 \]
\[ \text{end while} \]
CG Implementation: Cost II, three daxpys

cg(x, b, A, ϵ, kmax)

\[ r = b - Ax, \quad \rho_0 = \|r\|_2^2, \quad k = 1. \]

while \( \sqrt{\rho_{k-1}} > \epsilon \|b\| \) and \( k < k_{max} \) do

if \( k = 1 \) then

\[ p = r \]

else

\[ \beta = \frac{\rho_{k-1}}{\rho_{k-2}} \] and \( p = r + \beta p \)

end if

\[ w = Ap \]

\[ \alpha = \frac{\rho_{k-1}}{p^T w} \]

\[ x = x + \alpha p \]

\[ r = r - \alpha w \]

\[ \rho_k = \|r\|^2 \]

\[ k = k + 1 \]

end while
Each iteration requires

- one matrix-vector product,
- two scalar products,
- three daxpys,

and the storage of $x, b, r, p, w$ five vectors!

Compare to GMRES ($k$ vectors and $O(k)$ scalar products).
Preconditioned CG (PCG)

Right (or left) preconditioning is a problem because

\[ BA \] or \[ AB \]

need not be spd.

The correct way to precondition CG is symmetrically

\[ SASy = Sb \]

and then \( x = Sy \). This means that \( S^2 = B \) is the preconditioner.

So do you have to compute \( S = \sqrt{B} \)?
PCG

pcg(x, b, A, B, \(\epsilon\), \(kmax\))

\[ r = b - Ax, \quad \rho_0 = \|r\|^2, \quad k = 1 \]

while \(\sqrt{\rho_k - 1} > \epsilon\|b\|\) and \(k < kmax\) do

\[ z = Br \]

\[ \tau_{k-1} = z^T r \]

if if \(k = 1\) then

\[ \beta = 0 \text{ and } p = z \]

else

\[ \beta = \frac{\tau_{k-1}}{\tau_{k-2}}, \quad p = z + \beta p \]

end if

\[ w = Ap \]

\[ \alpha = \frac{\tau_{k-1}}{p^T w} \]

\[ x = x + \alpha p; \quad r = r - \alpha w; \quad \rho_k = r^T r \]

\[ k = k + 1 \]

end while
Cost of PCG

Each iteration requires

- one matrix-vector product,
- one preconditioner-vector product,
- three scalar products,
- four daxpys,

and the storage of $x, b, r, z, p, w$ six vectors.
CGNR and CGNE

Conjugate gradient on the normal equations. Two low-storage + provably convergent methods for nonsymmetric problems.

CGNR: Apply CG to

\[ A^T A = A^T b \]

CGNE: Apply CG to

\[ AA^T y = b \text{ and set } x = A^T y. \]
Analysis of CGNR

Apply the minimization property. You minimize

\[ \| x^* - x \|^2_{A^T A} = (x^* - x)^T A^T A (x^* - x) = (Ax^* - Ax)^T (Ax^* - Ax) \]

\[ = (b - Ax)^T (b - Ax)^T = \| r \|^2 \]

over \( x_0 + K_k(A^T A) \). Hence the name Conjugate Gradient on the Normal equations to minimize the Residual.
Analysis of CGNE

Same story,

$$\|y^* - y\|_{AA^T}^2 = (y^* - y)^T (AA^T) (y^* - y)$$

$$= (A^T y^* - A^T y)^T (A^T y^* - A^T y) = \|x^* - x\|^2$$

is minimized over $y_0 + K_k(AA^T)$ at each iterate. **Conjugate Gradient on the Normal equations to minimize the Error.**
Observations

- CGNR and CGNE need two matrix-vector products
  - one is a transpose-vector product
    hard to do in a matrix-free way
  - Condition number is squared, so more iterations are needed
  - Classic time-for-storage trade-off.
We discuss Bi-CGSTAB and TFQMR. Their common properties are

- Constant storage
- Two $A$-vector products per iteration
- No transpose-vector products needed
- Breakdown possible; no complete convergence theory
Bi-CGSTAB

\[ \text{bicgstab}(x, b, A, \epsilon, kmax) \]

\[ r = b - Ax, \; \hat{r}_0 = \hat{r} = r, \; \rho_0 = \alpha = \omega = 1, \; v = p = 0, \; k = 0, \; \rho_1 = \hat{r}_0^T r \]

\textbf{while} \; \|r\| > \epsilon \|b\| \textbf{ and } k < kmax \; \textbf{do} \]

\[ k = k + 1 \]

\[ \beta = (\rho_k / \rho_{k-1})(\alpha / \omega) \quad \text{(breakdown possible; zero-divide)} \]

\[ p = r + \beta(p - \omega v) \quad \text{(two daxpys)} \]

\[ v = Ap \quad \text{(matvec)} \]

\[ \alpha = \rho_k / (\hat{r}_0^T v) \quad \text{(scalar product + breakdown possible; zero-divide)} \]

\[ s = r - \alpha v, \; t = As \quad \text{(daxpy + matvec)} \]

\[ \omega = t^T s / \|t\|^2; \; \rho_{k+1} = -\omega \hat{r}_0^T t \quad \text{(three scalar products)} \]

\[ x = x + \alpha p + \omega s \quad \text{(two daxpys)} \]

\[ r = s - \omega t \quad \text{(daxpy)} \]

\textbf{end while}
Cost of BiCGSTAB

Each iteration requires
- two matrix-vector product,
- four scalar products,
- seven daxpys,

and the storage of \( x, b, r, \hat{r}, p, v, s, t \) eight vectors.

Breakdown? Pick new \( x_0 \) and try again.
TFQMR

tfqmr(x, b, A, ε, kmax)

\[ k = 0; \quad w_1 = y_1 = r_0 = b - Ax; \quad u_1 = v = Ay_1, \quad d = 0; \quad ρ_0 = r_0^T r_0; \quad τ = \|r\|; \quad θ = 0; \quad η = 0 \]

while \( k < k_{\text{max}} \) do

\[ k = k + 1; \quad \sigma_{k-1} = r_0^T v; \text{ (scalar product)} \]

\[ α = ρ_{k-1}/σ_{k-1}; \text{ (breakdown possible; zero-divide)} \]

\[ y_2 = y_1 - αv; \quad u_2 = Ay_2 \text{ (daxpy + matvec)} \]

for \( j = 1, 2 (m = 2k - 2 + j) \) (all costs doubled in this loop) do

\[ w = w - αu_j; \quad d = y_j + (θ^2 η/α)d \text{ (two daxpys)} \]

\[ θ = \|w\|/τ; \quad c = 1/\sqrt{1 + θ^2} \text{ (scalar product)} \]

\[ τ = τθc; \quad η = c^2 α; \]

\[ x = x + ηd \text{ (daxpy)} \]

If \( τ\sqrt{m+1} ≤ ε\|b\| \) terminate successfully

end for

\[ ρ_k = r_0^T w, \quad β = ρ_k/ρ_{k-1} \text{ (scalar product + breakdown possible; zero-divide)} \]

\[ y_1 = w + βy_2, \quad u_1 = Ay_1 \text{ (daxpy + matvec)} \]

\[ v = u_1 + β(u_2 + βv) \text{ (two daxpys)} \]

end while
Recall that

- convert $Ax = b$ to $x = Mx + c$ with a matrix splitting,
- $M_S$ is the iteration matrix for the method
- Harvest a preconditioner with $BA = I - M$ and then

$$x = Mx + c$$

is the same as $BAx = Bb$. 
Example: Jacobi

- Splitting: $A = D + L + U$
- $M = -D^{-1}(L + U) = I - D^{-1}A$
- so $B = D^{-1}$.

Sometimes Jacobi preconditioning works well.
If you can store $A$ as a sparse matrix then

» you can start a sparse factorization,

» and discard small elements in the factors,

» or enforce sparsity.

The MATLAB commands `ilu` and `ichol` create incomplete LU and Cholesky factorizations.
Many integral equations are well conditioned and CG or GMRES do well.

The transport equation is one example.

The performance of Kyrlov methods is independent of the discretization.

WARNING! Sometime preconditioning can still make a difference.
Suppose you seek to solve an elliptic boundary value problem.

\[ Lu = f \]

with Dirichlet/Neumann/mixed boundary conditions. If you discretize the PDE to obtain

\[ L_h u_h = f_h \]

the resulting discrete problem is very poorly conditioned and Krylov methods will be slow.
Split $L = L_1 + L_0$, where $L_1$ contains the high-order derivatives. If you can find a fast solver for $L_1$ with the same type of boundary conditions, then $L_1^{-1}$ is a mesh-independent preconditioner. Why? $L_1^{-1}L$ is an integral operator. (Manteufel/Parter 1990)
Example of PDE preconditioning

- $-\nabla^2 u + c_1 u_x + c_2 u_y + c_0 u = f(x, y)$ for $0 < x, y < 1$
- $u(x, 0) = u(0, y) = u(x, 1) = u(1, y) = 0$
- $L_1 u = -\nabla^2 u$
- Apply fast Poisson solver $N \log(N)$ work.
Scalability

The scenario:

- Continuous problem: $Lu = f$; Discrete problem: $L_h u_h = f_h$.
- $h = 1/N$ spatial mesh width; $N^2$ number of mesh points.
- Second order accuracy: $u_h - u^* = O(h^2)$
- Preconditioner $B_h$ is “perfect”, i.e. Krylovs needed to reduce error by factor of 10 is $N_k$ for all $h$.
- Cost of $B_h L_h$ matvec is $O(N)$

Then, given $h$ you can find $u_h$ up to truncation error in $O(N)$ work!
Pick $h_0 = 2^p h$ so that $L h_0 u_{h_0} = f_{h_0}$ is easy to solve.

Solve $L h_0 u_0 = f_{h_0}$

for $l=1:p$ do

$h_l = h_{l-1}/2$; $u_l = u_{l-1}$

Apply GMRES to $L h_l u_l = f_{h_l}$ with $u_l$ as the start.

Terminate when residual is reduced by factor of 10.

Accept $u_l$

end for
A matvec for $h_l = 2^l h$ costs $O(2^{-l})N^2$ operations.

We do at most $N_k$ matvecs at each level.

So . . .

\[
\text{Cost} \leq \sum_{l=0}^{p} N_k (2^{-l}N)^2 \leq \sum_{l=0}^{\infty} N_k (2^{-l}N)^2
\]

\[
= N_k N \sum_{l=0}^{\infty} 4^{-l} = 4N_k N^2 / 3.
\]
Exercises

- Modify the pde demo codes klpde2ddemo.m to use BiCGStab and TFQMR. Any problems?
- Write a CGNR code and solve the problem in klpde2ddemo.m with CGNR.
- Solve the source iteration equation with GMRES. What problem would you have if you wanted to solve it with CGNR or CGNE?