AMSC/CMSC 661 Scientific Computing II
Spring 2005
Solution of Sparse Linear Systems
Part 2: Iterative methods

Dianne P. O'Leary

©2005

Solving Sparse Linear Systems: Iterative methods

The plan:

- Iterative methods:
  - Basic (slow) iterations: Jacobi, Gauss-Seidel, SOR.
  - Krylov subspace methods
  - Preconditioning (where direct meets iterative)
- A special purpose method: Multigrid

Basic iterations

The idea: Given an initial guess $x^{(0)}$ for the solution to $Ax^* = b$, construct a sequence of guesses $\{x^{(0)}, x^{(1)}, x^{(2)}, \ldots\}$ converging to x^*.

The amount of work to construct each new guess from the previous one should be a small multiple of the number of nonzeros in A.

Stationary Iterative Methods

These methods grew up in the engineering and mathematical literature. They were **very popular** in the 1960s and are still sometimes used.

Today, they are **almost never** the best algorithms to use (because they take too many iterations), but they are useful **preconditioners** for Krylov subspace methods.

We will define three of them:

- Jacobi (Simultaneous displacement)
- Gauss-Seidel (Successive displacement)
- SOR

Theme: All of these methods split A as $M - N$ for some nonsingular matrix M. Other splittings of this form are also useful.

The Jacobi iteration

Idea: The ith component of the **residual vector** r is defined by

$$r_i = b_i - \mathbf{a_{i1}x_1} - \mathbf{a_{i2}x_2} - \ldots - \mathbf{a_{i,i-1}x_{i-1}} - \mathbf{a_{ii}x_i} - \mathbf{a_{i,i+1}x_{i+1}} - \ldots - \mathbf{a_{in}x_n}.$$

Let's modify x_i to make $r_i = 0$.

Given $x^{(k)}$, construct $x^{(k+1)}$ by

$$\mathbf{x_i^{(k+1)}} = (b_i - \sum_{\mathbf{j=1}}^{\mathbf{i-1}} \mathbf{a_{ij}x_j^{(k)}} - \sum_{\mathbf{j=i+1}}^{\mathbf{n}} \mathbf{a_{ij}x_j^{(k)}})/\mathbf{a_{ii}},\ i = 1, \ldots, n.$$

Observations:

- We must require A to have nonzeros on its main diagonal.
- The algorithm is easy to program! We only need to store two x vectors, $x^{(k)}$ and $x^{(k+1)}$.
- The iteration may or may not converge, depending on the properties of A.
- We should only touch the nonzeros in A – otherwise the work per iteration would be $O(n^2)$ instead of $O(nz)$.
- If we partition A as $L + D + U$, where D contains the diagonal entries, U contains the entries above the diagonal, and L contains the entries below the diagonal, then we can express the iteration as

  $$Dx^{(k+1)} = b - (L + U)x^{(k)}$$

  and this is useful for analyzing convergence. $(M = D,\ N = -(L + U))$

The Gauss-Seidel iteration

Idea: If we really believe that we have improved the ith component of the solution by our Jacobi iteration, then it makes sense to use its **latest value** in the iteration:

Given $x^{(k)}$, construct $x^{(k+1)}$ by

$$x_i^{(k+1)} = (b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)})/a_{ii}, \ i = 1, \ldots, n.$$

Observations:

- We still require A to have nonzeros on its main diagonal.
- The algorithm is easier to program, since we only need to keep one x vector around!
- The iteration may or may not converge, depending on the properties of A.
- We should only touch the nonzeros in A – otherwise the work per iteration would be $O(n^2)$ instead of $O(nz)$.
- If we partition A as $L + D + U$, where D contains the diagonal entries, U contains the entries above the diagonal, and L contains the entries below the diagonal, then we can express the iteration as

  $$(D + L)x^{(k+1)} = b - Ux^{(k)}$$

  and this is useful for analyzing convergence. $(M = D + L, \ N = -U)$

The SOR (Successive Over-Relaxation) iteration

Idea: People who used these iterations on finite difference matrices discovered that Gauss-Seidel (GS) converged faster than Jacobi (J), and they could improve its convergence rate by **going a little further in the GS direction**:

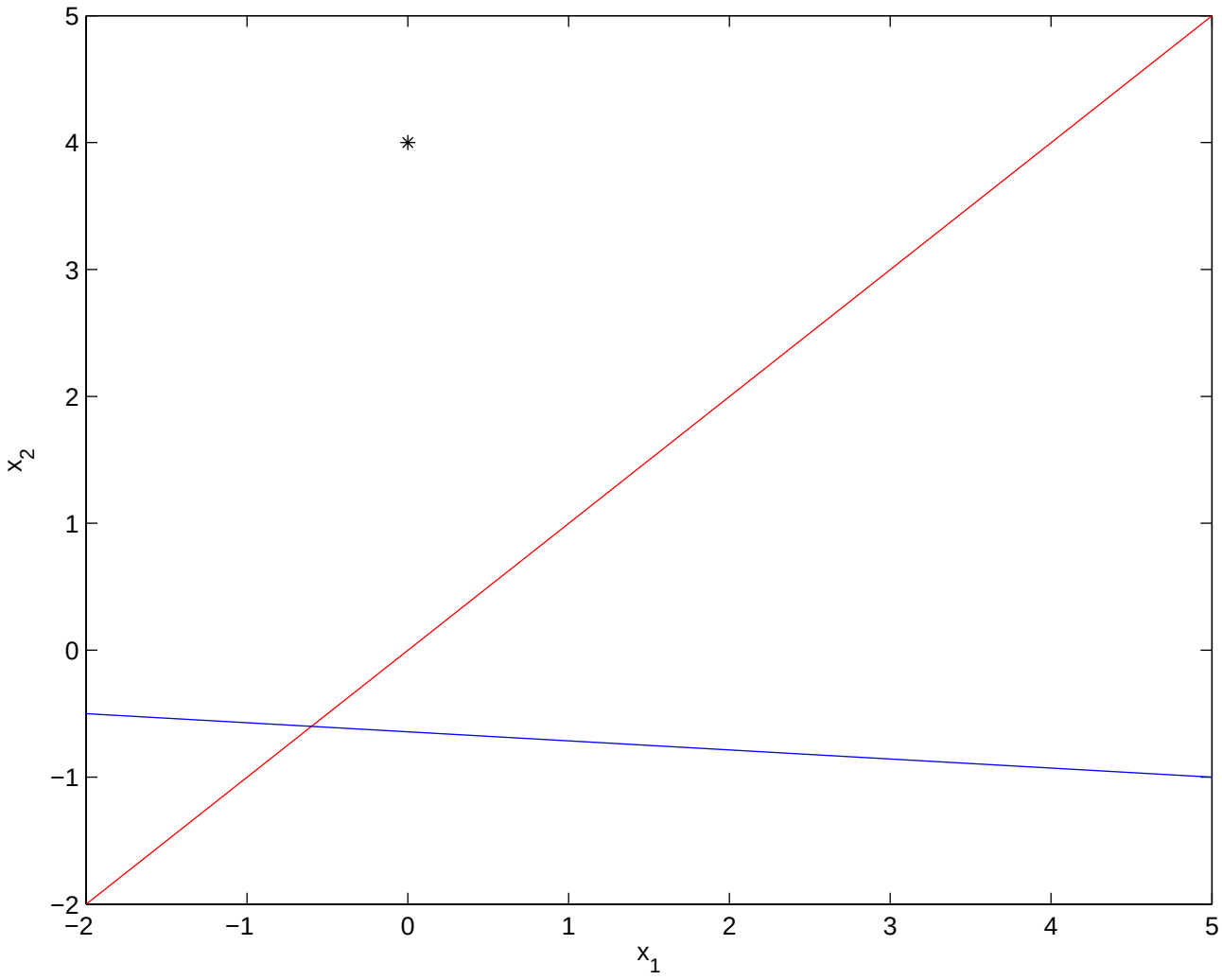
Given $x^{(k)}$, construct $x^{(k+1)}$ by

$$x^{(k+1)} = (1 - \omega)x^{(k)} + \omega x_{GS}^{(k+1)}$$

where ω is a number between 1 and 2.

Unquiz: Suppose $n = 2$ and our linear system can be graphed as in the figure. Draw the first 3 Jacobi iterates and the first 3 Gauss-Seidel iterates using the point marked with a star as $x^{(0)}$. Does either iteration depend on the ordering of the equations or unknowns? []

Convergence of Stationary iterative methods


5
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4
5
x2
x1

- All of these iterations can be expressed as

$$x^{(k+1)} = Gx^{(k)} + c$$

  where $G = M^{-1}N$ is a matrix that depends on A and c is a vector that depends on A and b.

- For all of these iterations, $x^* = Gx^* + c$.
- Subtracting, we see that the error $e^{(k)} = x^{(k)} - x^*$ satisfies

$$e^{(k+1)} = Ge^{(k)},$$

  and it can be shown that the error converges to zero for any initial $x^{(0)}$ if and only if all of the eigenvalues of G lie inside the unit circle.

- Many conditions on A have been found that guarantee convergence of these methods; see the SIM notes.

This is enough about slow methods.

From SIM to Krylov subspace methods

So far: Stationary iterative methods.

- $Ax = b$ is replaced by $x = Gx + c$.
- $x^{(k+1)} = Gx^{(k)} + c$
- If $x^{(0)} = 0$, then

$$\begin{aligned} x^{(1)} &= c \\ x^{(2)} &\in \text{span}\{c, Gc\} \\ x^{(3)} &\in \text{span}\{c, Gc, G^2c\} \\ x^{(k)} &\in \text{span}\{c, Gc, G^2c, \ldots, G^{k-1}c\} \\ &\equiv \mathcal{K}_k(G, c) \end{aligned}$$

  and we call $\mathcal{K}_k(G, c)$ a **Krylov subspace**.

- The work per iteration is $O(nz)$ plus a small multiple of n.
- Note that $\mathcal{K}_k(G, c) = \mathcal{K}_k(\hat{G}, c)$ if $\hat{G} = I - G$.

The idea behind Krylov subspace methods: Instead of making the GS choice (for example) from the Krylov subspace, let's try to pick the **best** vector without doing a lot of extra work.

What is "best"?

- **The variational approach:** Choose $x^{(k)} \in \mathcal{K}_k(G, c)$ to minimize

$$\|x - x^*\|_Z$$

where $\|y\|_Z^2 = y^T Z y$ and Z is a symmetric positive definite matrix.

- **The Galerkin approach:** Choose $x^{(k)} \in \mathcal{K}_k(G, c)$ to make the residual $r^{(k)} = b - Ax^{(k)}$ orthogonal to every vector in $\mathcal{K}_k(G, c)$ for some choice of inner product.

Practicalities

- Note that we **expand** the subspace $\mathcal{K}$ at each iteration. This gives us a very important property: **After at most n iterations, Krylov subspace iterations terminate with the true solution.**
- **This finite termination property is less useful than it might seem, since we think of applying these methods when n is so large (thousands, millions, billions, etc.) that we can't afford more than a few hundred iterations.**
- The only way to make the iteration practical is to make a very clever **choice of basis** for $\mathcal{K}$. If we use the obvious choice of $c, Gc, G^2c, \ldots$, then after just a few iterations, our algorithm will lose accuracy.
- We need to choose between (variational) minimization and (Galerkin) projection.

Krylov Ingredient 1: A practical basis

An **orthonormal basis** for $\mathcal{K}$ makes the iteration practical. We say that a vector v is **B-orthogonal** to a vector u **if**

$$u^T B v = 0.$$

where B is a symmetric positive definite matrix. Similarly, we define $\|u\|_B^2 = u^T B u$.

Let's construct our basis for $\mathcal{K}_k(\hat{G}, c)$.

Our first basis vector is

$$p^{(0)} = c/\|c\|_B$$

Now suppose that we have $j+1$ basis vectors $p^{(0)}, \ldots, p^{(j)}$ for $\mathcal{K}_{j+1}(\hat{G}, c)$, and that we have some vector $r \in \mathcal{K}_{j+2}(\hat{G}, c)$ but $r \notin \mathcal{K}_{j+1}(\hat{G}, c)$. Often, we take r to be $\hat{G}p^{(j)}$.

We define the next basis vector by the process of **Gram-Schmidt orthogonalization**:

$$p^{(j+1)} = (r - h_{0,j}p^{(0)} - \ldots - h_{j,j}p^{(j)})/h_{j+1,j}$$

where $h_{i,j} = p^{(i)T}Br$ $(i = 0, \ldots, j)$ and $h_{j+1,j}$ is chosen so that $p^{(j+1)T}Bp^{(j+1)} = 1$.

In matrix form, we can express this relation as

$$\hat{G}p^{(j)} = \begin{bmatrix} p^{(0)} & p^{(1)} & \ldots & p^{(j+1)} \end{bmatrix} \begin{bmatrix} h_{0,j} \\ h_{1,j} \\ \vdots \\ h_{j+1,j} \end{bmatrix},$$

so after k steps we have

$$\hat{G}P_k = P_{k+1}H_k \quad (*)$$

where H_k is a $(k+1) \times k$ matrix with entries h_{ij} (zero if $i > j+1$) and P_k is $n \times k$ and contains the first k basis vectors as its columns.

Equation (*) is very important: Since the algorithm terminates after n vectors have been formed, we have actually **factored** our matrix

$$\hat{G} = P_n H_n P_n^{-1}$$

(and note that $P_n^{-1} = P_n^T$ if $B = I$)

Therefore, the matrix H_n is closely related to $\hat{G}$ – it has the same **eigenvalues**. In fact, the leading $k \times k$ piece of H_n (available after k steps) is in some sense a **good approximation** to $\hat{G}$. This is the basis of algorithms for

- solving linear systems of equations involving $\hat{G}$.
- finding approximations to eigenvalues and eigenvectors of $\hat{G}$.

We have just constructed the **Arnoldi algorithm**.

The Arnoldi algorithm

$[P, H]$ = **Arnoldi**$(k, \hat{G}, B, p^{(0)})$
Given a positive integer k, a symmetric positive definite matrix B, a matrix $\hat{G}$, and a vector $p^{(0)}$ with $\|p^{(0)}\|_B = 1$.
for $j = 0, 1, \ldots, k-1$,

$p^{(j+1)} = \hat{\mathbf{G}}\mathbf{p}^{(\mathbf{j})}$.

for $i = 0, \ldots, j$,

$h_{ij} = p^{(i)T} B p^{(j+1)}$
$p^{(j+1)} = p^{(j+1)} - h_{ij} p^{(i)}$

end (for i)

$h_{j+1,j} = (p^{(j+1)T} B p^{(j+1)})^{1/2}$

$p^{(j+1)} = p^{(j+1)} / h_{j+1,j}$.

end (for j)

Notes:

- In practice, B is either the identity matrix or a matrix closely related to $\hat{G}$.
- Note that we need only **1 matrix-vector product** by $\hat{G}$ per iteration.
- After k iterations, we have done $O(k^2)$ inner products of length n each, and this work becomes significant as k increases.
- **If $B\hat{G}$ is symmetric, then by (*), so is H_k, so all but 2 of the inner products at step j are zero. In this case, we can let the loop index $i = j-1 : j$ and the number of inner products drops to $O(k)$.** Then the **Arnoldi** algorithm is called **Lanczos tridiagonalization**.
- In writing this algorithm, we took advantage of the fact that $p^{(i)T} B p^{(j+1)}$ is **mathematically** the same, whether we use the original vector $p^{(j+1)}$ or the updated one. **Numerically**, using the updated one works a bit better, but both eventually lose orthogonality, and the algorithm sometimes needs to be restarted to overcome this.

Krylov Ingredient 2: A definition of "best"

Two good choices:

- (variational) minimization
- (Galerkin) projection.

Using Krylov minimization

Problem: Find $x^{(k)} \in \mathcal{K}_k$ so that $x^{(k)}$ minimizes

$$\|x - x^*\|_Z$$

over all choices of $x \in \mathcal{K}_k$.

Solution: Let $x^{(k)} = P_k y^{(k)}$, where $y^{(k)}$ is a vector with k components. Then

$$\|x^{(k)} - x^*\|_Z^2 = (P_k y^{(k)} - x^*)^T Z (P_k y^{(k)} - x^*).$$

Differentiating with respect to the components of $y^{(k)}$, and setting the derivative to zero yields

$$P_k^T Z P_k \mathbf{y^{(k)}} = P_k^T Z \mathbf{x^*}.$$

Since $y^{(k)}$ and x^* are both unknown, we **usually** can't solve this. But we can if we are clever about our choice of Z.

1st special choice of Z

Recall:

- We need to solve $P_k^T Z P_k \mathbf{y^{(k)}} = P_k^T Z \mathbf{x^*}$, and $\hat{G} x^* = c$.
- $\hat{G} P_k = P_{k+1} H_k \ \ (*)$
- $P_{k+1}^T B P_{k+1} = I_{k+1}$

Let $Z = \hat{G}^T B \hat{G}$. (This is symmetric, and positive definite if $\hat{G}$ is nonsingular.) Then

$$P_k^T Z \mathbf{x^*} = P_k^T \hat{G}^T B \hat{G} x^* = P_k^T \hat{G}^T B c = \mathbf{H_k^T P_{k+1}^T B c}$$

is computable! The left-hand side also simplifies:

$$P_k^T Z P_k = P_k^T \hat{G}^T B \hat{G} P_k = H_k^T P_{k+1}^T B P_{k+1} H_k = H_k^T H_k.$$

So we need to solve

$$H_k^T H_k y^{(k)} = \mathbf{H_k^T P_{k+1}^T B c}.$$

This algorithm is called **GMRES** (generalized minimum residual), due to Saad and Schultz in 1986, and is probably the most often used Krylov method.

2nd special choice of Z

Recall:

- We need to solve $P_k^T Z P_k \mathbf{y^{(k)}} = P_k^T Z \mathbf{x^*}$ and $\hat{G} x^* = c$.
- $\hat{G} P_k = P_{k+1} H_k \ \ (*)$
- $P_{k+1}^T B P_{k+1} = I_{k+1}$

Added assumption: Assume $Z = B\hat{G}$ is symmetric and positive definite. **(Note that we need that Z be symmetric and positive definite in order to be minimizing a norm of the error. Our assumption here is that $B\hat{G}$ is symmetric and positive definite.)**

$$P_k^T Z \mathbf{x}^* = P_k^T B\hat{G}x^* = P_k^T Bc$$

is computable. The left-hand side also simplifies:

$$P_k^T Z P_k = P_k^T B\hat{G}P_k = P_k^T BP_{k+1}H_k = \bar{H}_k$$

where $\bar{H}_k$ contains the first k rows of H_k. So we need to solve

$$\bar{H}_k y^{(k)} = P_k^T Bc.$$

This algorithm is called **conjugate gradients** (CG), due to Hestenes and Stiefel in 1952. It is the most often used Krylov method for symmetric problems.

Using Krylov projection

- $\hat{G}x^* = c.$
- $\hat{G}P_k = P_{k+1}H_k \ \ (*)$
- $P_{k+1}^T BP_{k+1} = I_{k+1}$

Problem: Find $x^{(k)} \in \mathcal{K}_k$ so that $r^{(k)} = c - \hat{G}x^{(k)}$ is B-orthogonal to the columns of P_k.

Solution:

$$0 = P_k^T B(c - \hat{G}x^{(k)}) = P_k^T B(c - \hat{G}P_k y^{(k)})$$

so we need to solve

$$P_k^T B\hat{G}P_k y^{(k)} = P_k^T Bc.$$

or

$$\bar{H}_k y^{(k)} = P_k^T Bc.$$

This algorithm is called the **Arnoldi iteration**. (**Note:** Same equation as cg, but no assumption of symmetry or positive definiteness.)

An important alternative

The algorithms we have discussed (GMRES, CG, and Arnoldi) all use the **Arnoldi basis**.

There is another convenient basis, derived using the **nonsymmetric Lanczos algorithm**. This basis gives rise to several useful algorithms:

- CG (alternate derivation)
- bi-conjugate gradients (Bi-CG) (Fletcher 1976)
- Bi-CGStab (van der Vorst 1992)
- quasi-minimum residual (QMR) (Freund and Nachtigal 1991)
- transpose-free QMR (Freund and Nachtigal 1993)
- (the most useful) CGStab (CG-squared stabilized) (van der Vorst 1989)

Advantages: Only a fixed number of vectors are saved, not k.
Disadvantages: The basic algorithms can break down – terminate without obtaining the solution to the linear system. Fixing this up is messy.

We won't take the time to discuss these methods in detail, but they can be useful.

The conjugate gradient method

- In general, we need to **save all of the old vectors** in order to accomplish the projection of the residual.
- For some special classes of matrices, we only need a few old vectors.
- The most important of these classes is **symmetric positive definite matrices**, just as we need for **self-adjoint** elliptic PDEs. The resulting algorithm is called **conjugate gradients** (CG).
- It is both a **minimization** algorithm (in the energy norm) and a **projection** algorithm ($B = I$).
- There is a very compact and practical form for the algorithm.

The conjugate gradient algorithm

$[x, r] = \mathbf{cg}(A, M, b, tol)$
Given symmetric positive definite matrices A and M, a vector b, and a tolerance tol, compute an approximate solution to $Ax = b$.
Let $r = b$, $x = 0$, solve $Mz = r$ for z, and let $\gamma = r^T z$, $p = z$.
for $k = 0, 1, \ldots,$ until $\|r\| < tol$,

$\alpha = \gamma/(p^T Ap)$

$x = x + \alpha p$

$r = r - \alpha Ap$

Solve $Mz = r$ for z.

$\hat{\gamma} = r^T z$

$\beta = \hat{\gamma}/\gamma$, $\gamma = \hat{\gamma}$

$p = z + \beta p$

end (for k)

The matrix M is called the **preconditioner**, and our next task is to understand what it does and why we might need it.