Scientific Computing with Case Studies

SIAM Press, 2009
http://www.cs.umd.edu/users/oleary/SCCSwebpage

Lecture Notes for Unit VII

Sparse Matrix Computations

Part 2: Iterative Methods

Dianne P. O'Leary

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

Reference: Chapter 28.

Basic iterations

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

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

Stationary Iterative Methods (SIMs)

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 $\mathbf{A}$ as $\mathbf{M} - \mathbf{N}$ for some nonsingular matrix $\mathbf{M}$. Other splittings of this form are also useful.

The Jacobi iteration

Idea: The ith component of the residual vector $\mathbf{r}$ is defined by

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

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

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

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

Observations:

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

  $$\mathbf{D}\mathbf{x}^{(k+1)} = \mathbf{b} - (\mathbf{L} + \mathbf{U})\mathbf{x}^{(k)}$$

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

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 $\mathbf{x}^{(k)}$, construct $\mathbf{x}^{(k+1)}$ 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 $\mathbf{L} + \mathbf{D} + \mathbf{U}$, 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

  $$(\mathbf{D} + \mathbf{L})\mathbf{x}^{(k+1)} = \mathbf{b} - \mathbf{U}\mathbf{x}^{(k)}$$

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

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

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

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

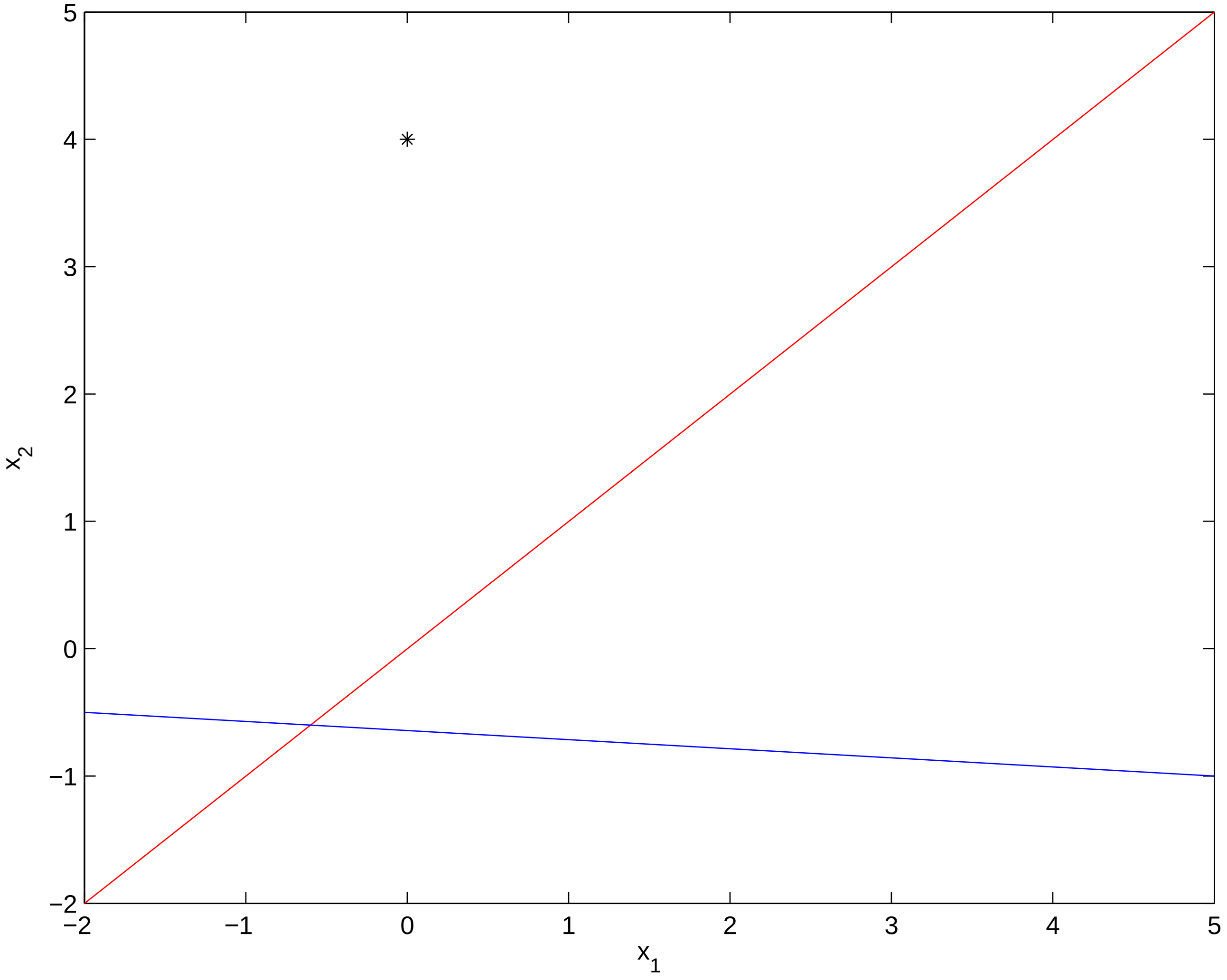
- All of these iterations can be expressed as

  $$\mathbf{x}^{(k+1)} = \mathbf{G}\mathbf{x}^{(k)} + \mathbf{c}$$

  where $\mathbf{G} = \mathbf{M}^{-1}\mathbf{N}$ is a matrix that depends on **A** and **c** is a vector that depends on **A** and **b**.
- For all of these iterations, $\mathbf{x}^* = \mathbf{G}\mathbf{x}^* + \mathbf{c}$.
- Subtracting, we see that the error $\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}^*$ satisfies

  $$\mathbf{e}^{(k+1)} = \mathbf{G}\mathbf{e}^{(k)},$$

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


5
4
3
2
1
0
−1
−2
−2
−1
0
1
2
3
4
5
x_1
x_2

- Many conditions on $\mathbf{A}$ 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.

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

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

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

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

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 $\mathbf{x}^{(k)} \in \mathcal{K}_k(\mathbf{G}, \mathbf{c})$ to minimize

$$\|\mathbf{x} - \mathbf{x}^*\|_{\mathbf{Z}}$$

  where $\|\mathbf{y}\|_{\mathbf{Z}}^2 = \mathbf{y}^T\mathbf{Zy}$ and $\mathbf{Z}$ is a symmetric positive definite matrix.

- The Galerkin approach: Choose $\mathbf{x}^{(k)} \in \mathcal{K}_k(\mathbf{G}, \mathbf{c})$ to make the residual $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{Ax}^{(k)}$ orthogonal to every vector in $\mathcal{K}_k(\mathbf{G}, \mathbf{c})$ 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 $\mathbf{c}, \mathbf{Gc}, \mathbf{G}^2\mathbf{c}, \ldots$, 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 $\mathbf{v}$ is **B**-orthogonal to a vector $\mathbf{u}$ if

$$\mathbf{u}^T\mathbf{B}\mathbf{v} = 0.$$

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

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

Our first basis vector is

$$\mathbf{v}_1 = \mathbf{c}/\|\mathbf{c}\|_{\mathbf{B}}$$

Now suppose that we have j basis vectors $\mathbf{v}_1, \ldots, \mathbf{v}_j$ for $\mathcal{K}_j(\widehat{\mathbf{G}}, \mathbf{c})$, and that we have some vector $\mathbf{z} \in \mathcal{K}_{j+1}(\widehat{\mathbf{G}}, \mathbf{c})$ but $\mathbf{z} \notin \mathcal{K}_j(\widehat{\mathbf{G}}, \mathbf{c})$. Often, we take $\mathbf{z}$ to be $\widehat{\mathbf{G}}\mathbf{v}_j$.

We define the next basis vector by the process of Gram-Schmidt orthogonalization (see Section 5.3.2):

$$\mathbf{v}_{j+1} = (\mathbf{z} - h_{1,j}\mathbf{v}_1 - \ldots - h_{j,j}\mathbf{v}_j)/h_{j+1,j}$$

where $h_{i,j} = \mathbf{v}_i^T\mathbf{B}\mathbf{z}$ $(i = 1, \ldots, j)$ and $h_{j+1,j}$ is chosen so that $\mathbf{v}_{j+1}^T\mathbf{B}\mathbf{v}_{j+1} = 1$.

In matrix form, we can express this relation as

$$\widehat{\mathbf{G}}\mathbf{v}_j = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \ldots & \mathbf{v}_{j+1} \end{bmatrix} \begin{bmatrix} h_{1,j} \\ h_{2,j} \\ \vdots \\ h_{j+1,j} \end{bmatrix},$$

so after k steps we have

$$\widehat{\mathbf{G}}\mathbf{V}_k = \mathbf{V}_{k+1}\mathbf{H}_k \quad (*)$$

where $\mathbf{H}_k$ is a $(k+1) \times k$ matrix with entries h_{ij} (zero if $i > j+1$) and $\mathbf{V}_k$ 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

$$\widehat{\mathbf{G}} = \mathbf{V}_n \mathbf{H}_n \mathbf{V}_n{}^{-1}$$

(and note that $\mathbf{V}_n{}^{-1} = \mathbf{V}_n^T$ if $\mathbf{B} = \mathbf{I}$)

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

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

We have just constructed the Arnoldi algorithm.

The Arnoldi algorithm

```
[V, H] = Arnoldi($m, \widehat{\mathbf{G}}, \mathbf{B}, \mathbf{v}_1$)
Given a positive integer $m$, a symmetric positive definite matrix $\mathbf{B}$, a matrix $\widehat{\mathbf{G}}$, and a vector $\mathbf{v}_1$ with $\|\mathbf{v}_1\|_{\mathbf{B}} = 1$.
for $j = 1, \ldots, m$,
    $\mathbf{v}_{j+1} = \widehat{\mathbf{G}}\mathbf{v}_j$.
    for $i = 1, \ldots, j$,
        $h_{ij} = \mathbf{v}_i^T \mathbf{B} \mathbf{v}_{j+1}$
        $\mathbf{v}_{j+1} = \mathbf{v}_{j+1} - h_{ij}\mathbf{v}_i$
    end (for $i$)
    $h_{j+1,j} = (\mathbf{v}_{j+1}^T \mathbf{B} \mathbf{v}_{j+1})^{1/2}$
    $\mathbf{v}_{j+1} = \mathbf{v}_{j+1}/h_{j+1,j}$.
end (for $j$)
```

Notes:

- In practice, $\mathbf{B}$ is either the identity matrix or a matrix closely related to $\widehat{\mathbf{G}}$.
- Note that we need only 1 matrix-vector product by $\widehat{\mathbf{G}}$ per iteration.
- After m iterations, we have done $O(m^2)$ inner products of length n each, and this work becomes significant as m increases.
- If $\mathbf{B}\widehat{\mathbf{G}}$ is symmetric, then by (*), so is $\mathbf{H}_m$, 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 $O(m)$. Then the Arnoldi algorithm is called Lanczos tridiagonalization.

- In writing this algorithm, we took advantage of the fact that $\mathbf{v}_i^T\mathbf{B}\mathbf{v}_{j+1}$ is mathematically the same, whether we use the original vector $\mathbf{v}_{j+1}$ 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 $\mathbf{x}^{(k)} \in \mathcal{K}_k$ so that $\mathbf{x}^{(k)}$ minimizes

$$\|\mathbf{x} - \mathbf{x}^*\|_{\mathbf{Z}}$$

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

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

$$\|\mathbf{x}^{(k)} - \mathbf{x}^*\|_{\mathbf{Z}}^2 = (\mathbf{V}_k\mathbf{y}^{(k)} - \mathbf{x}^*)^T\mathbf{Z}(\mathbf{V}_k\mathbf{y}^{(k)} - \mathbf{x}^*).$$

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

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

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

1st special choice of Z

Recall:

- We need to solve $\mathbf{V}_k^T\mathbf{Z}\mathbf{V}_k\mathbf{y}^{(k)} = \mathbf{V}_k^T\mathbf{Z}\mathbf{x}^*$, and $\widehat{\mathbf{G}}\mathbf{x}^* = \mathbf{c}$.
- $\widehat{\mathbf{G}}\mathbf{V}_k = \mathbf{V}_{k+1}\mathbf{H}_k \quad (*)$
- $\mathbf{V}_{k+1}^T\mathbf{B}\mathbf{V}_{k+1} = \mathbf{I}_{k+1}$

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

$$\mathbf{V}_k^T\mathbf{Z}\mathbf{x}^* = \mathbf{V}_k^T\widehat{\mathbf{G}}^T\mathbf{B}\widehat{\mathbf{G}}\mathbf{x}^* = \mathbf{V}_k^T\widehat{\mathbf{G}}^T\mathbf{B}\mathbf{c} = \mathbf{H}_k^T\mathbf{V}_{k+1}^T\mathbf{B}\mathbf{c}$$

is computable! The left-hand side also simplifies:

$$\mathbf{V}_k^T\mathbf{Z}\mathbf{V}_k = \mathbf{V}_k^T\widehat{\mathbf{G}}^T\mathbf{B}\widehat{\mathbf{G}}\mathbf{V}_k = \mathbf{H}_k^T\mathbf{V}_{k+1}^T\mathbf{B}\mathbf{V}_{k+1}\mathbf{H}_k = \mathbf{H}_k^T\mathbf{H}_k.$$

So we need to solve

$$\mathbf{H}_k^T \mathbf{H}_k \mathbf{y}^{(k)} = \mathbf{H}_k^T \mathbf{V}_{k+1}^T \mathbf{B}\mathbf{c}.$$

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 $\mathbf{V}_k^T \mathbf{Z} \mathbf{V}_k \mathbf{y}^{(k)} = \mathbf{V}_k^T \mathbf{Z} \mathbf{x}^*$ and $\widehat{\mathbf{G}}\mathbf{x}^* = \mathbf{c}$.
- $\widehat{\mathbf{G}}\mathbf{V}_k = \mathbf{V}_{k+1}\mathbf{H}_k \ \ (*)$
- $\mathbf{V}_{k+1}^T \mathbf{B} \mathbf{V}_{k+1} = \mathbf{I}_{k+1}$

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

$$\mathbf{V}_k^T \mathbf{Z} \mathbf{x}^* = \mathbf{V}_k^T \mathbf{B}\widehat{\mathbf{G}}\mathbf{x}^* = \mathbf{V}_k^T \mathbf{B}\mathbf{c}$$

is computable. The left-hand side also simplifies:

$$\mathbf{V}_k^T \mathbf{Z} \mathbf{V}_k = \mathbf{V}_k^T \mathbf{B}\widehat{\mathbf{G}}\mathbf{V}_k = \mathbf{V}_k^T \mathbf{B}\mathbf{V}_{k+1}\mathbf{H}_k = \bar{\mathbf{H}}_k$$

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

$$\bar{\mathbf{H}}_k \mathbf{y}^{(k)} = \mathbf{V}_k^T \mathbf{B}\mathbf{c}.$$

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

- $\widehat{\mathbf{G}}\mathbf{x}^* = \mathbf{c}$.
- $\widehat{\mathbf{G}}\mathbf{V}_k = \mathbf{V}_{k+1}\mathbf{H}_k \ \ (*)$
- $\mathbf{V}_{k+1}^T \mathbf{B} \mathbf{V}_{k+1} = \mathbf{I}_{k+1}$

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

Solution:

$$0 = \mathbf{V}_k^T \mathbf{B}(\mathbf{c} - \widehat{\mathbf{G}}\mathbf{x}^{(k)}) = \mathbf{V}_k^T \mathbf{B}(\mathbf{c} - \widehat{\mathbf{G}}\mathbf{V}_k\mathbf{y}^{(k)})$$

so we need to solve

$$\mathbf{V}_k^T \mathbf{B}\widehat{\mathbf{G}}\mathbf{V}_k \mathbf{y}^{(k)} = \mathbf{V}_k^T \mathbf{B}\mathbf{c}.$$

or

$$\bar{\mathbf{H}}_k \mathbf{y}^{(k)} = \mathbf{V}_k^T \mathbf{B}\mathbf{c}.$$

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 ($\mathbf{B} = \mathbf{I}$).
- There is a very compact and practical form for the algorithm.

The conjugate gradient algorithm

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

The matrix $\mathbf{M}$ is called the preconditioner, and we will need to understand what it does and why we might need it.

The practical form of GMRES

We'll write the form of the algorithm used to solve $\widehat{\mathbf{G}}\mathbf{x}^* = \mathbf{c}$, given a positive integer m (the restart parameter) and with $\mathbf{B} = \mathbf{I}$.

```
Initially, $\mathbf{x}^{(m)} = \mathbf{0}$ and $k = m$.
Until termination,
    • Set $k = k + 1$. If $k = m + 1$, then set $k = 1$ and $\mathbf{x}^{(0)} = \mathbf{x}^{(m)}$.
    • Increase the dimension of the Krylov subspace to dimension $k$ using the starting
      vector $\mathbf{c} - \widehat{\mathbf{G}}\mathbf{x}^{(0)}$ and the matrix $\widehat{\mathbf{G}}$, giving a matrix $\mathbf{V}_k$ of directions and $\mathbf{H}_k$ of
      coefficients.
    • Solve $\mathbf{H}_k^T\mathbf{H}_k\mathbf{y}^{(k)} = \mathbf{H}_k^T\mathbf{V}_{k+1}^T\mathbf{c}$, and set $\mathbf{x}^{(k)} = \mathbf{x}^{(0)} + \mathbf{V}_k\mathbf{y}^{(k)}$.
    • If $\|\widehat{\mathbf{G}}\mathbf{x}^{(k)} - \mathbf{c}\|$ is small enough, set $\mathbf{x}_{final} = \mathbf{x}^{(k)}$ and terminate.
```

Note: Ideally, $m = n$, but since the storage is $O(mn)$, and the time to solve the systems involving $\mathbf{H}_1, \ldots \mathbf{H}_m$ is $O(m^3)$ (using matrix updating techniques and Cholesky decomposition), in practice we keep m to 20 or 100 at most.

The plan:

So far:

- Iterative methods:
    - Basic (slow) iterations: Jacobi, Gauss-Seidel, SOR.
    - Krylov subspace methods: the algorithms

Next:

- Krylov subspace methods: convergence theory
- Preconditioning (where direct meets iterative)
- A special purpose method: Multigrid

Reference: Chapter 28

Convergence results for GMRES(m)

- Convergence on positive definite matrices. (Saad p.205, Thm 6.30) If $(\widehat{\mathbf{G}} + \widehat{\mathbf{G}}^T)/2$ is positive definite, then GMRES(m) converges for any $m \geq 1$.

- Convergence on diagonalizable matrices. (Saad p.206, Prop 6.32) If $\widehat{\mathbf{G}} = \mathbf{X}\mathbf{\Lambda}\mathbf{X}^{-1}$ where $\mathbf{\Lambda}$ is the matrix of eigenvalues, then for $k = 1, \ldots, m$,

$$\|\mathbf{r}^{(k)}\|_2 \leq \kappa(\mathbf{X})\epsilon_k\|\mathbf{r}^{(0)}\|_2,$$

  where
  $\mathbf{r}^{(i)} = \mathbf{c} - \widehat{\mathbf{G}}\mathbf{x}^{(i)}$ is the residual,
  $\kappa(\mathbf{X})$ is the square-root of the ratio of the largest eigenvalue of $\mathbf{X}^T\mathbf{X}$ to the smallest,
  and

$$\epsilon_k = \min_{\rho \in \mathcal{P}_k} \max_{j=1,\ldots,n} |\rho(\lambda_j)|.$$

  where $\mathcal{P}_k$ is the set of all polynomials of degree at most k satisfying $\rho(0) = 1$.

- For normal matrices: If $\widehat{\mathbf{G}}\widehat{\mathbf{G}}^T = \widehat{\mathbf{G}}^T\widehat{\mathbf{G}}$, then $\widehat{\mathbf{G}}$ is normal and $\kappa(\mathbf{X}) = 1$. Otherwise the bound is not very useful.

- (Saad p.206, Cor 6.33) If all of the eigenvalues of $\widehat{\mathbf{G}}$ are in an ellipse that doesn't contain the origin, and the ellipse is centered at $(c, 0)$ in the complex plane, with focal distance d (pure real or pure imaginary) and semimajor axis a, then

$$\epsilon_k \leq \frac{\left(a/d + \sqrt{(a/d)^2 - 1}\right)^k + \left(a/d + \sqrt{(a/d)^2 - 1}\right)^{-k}}{\left(c/d + \sqrt{(c/d)^2 - 1}\right)^k + \left(c/d + \sqrt{(c/d)^2 - 1}\right)^{-k}}$$

Convergence result for CG

(Saad, p.205, eqn (6.128)) In the energy norm,

$$\|\mathbf{x}^{(k)} - \mathbf{x}^*\|_{\widehat{\mathbf{G}}} \leq 2 \left(\frac{\sqrt{\kappa(\widehat{\mathbf{G}})} - 1}{\sqrt{\kappa(\widehat{\mathbf{G}})} + 1} \right)^k \|\mathbf{x}^{(0)} - \mathbf{x}^*\|_{\widehat{\mathbf{G}}}$$

where $\kappa(\widehat{\mathbf{G}})$ is the ratio of the largest and smallest eigenvalues of $\widehat{\mathbf{G}}$.

Preconditioning GMRES and CG

- For fast iterations, we need to be able to solve linear systems involving **M** very quickly, since this must be done once per iteration.
- To make the number of iterations small, we want **M** to be a good approximation to **A** so that the eigenvalues are in a small ellipse (GMRES) or a small interval (CG).
- For CG, we need to require that **M** be symmetric and positive definite.
- Note that the linear system
$$\mathbf{Mz} = \mathbf{r}$$
is typically solved using a direct method, so the better **M** is, the closer we are to solving $\mathbf{Ax} = \mathbf{b}$ using a direct method.

Some common choices of preconditioning matrices M

- **M** = the diagonal of **A**.
- **M** = a banded piece of **A**.
- **M** = an incomplete factorization of **A**, leaving out inconvenient elements (the ILU preconditioner).
- $\mathbf{M}^{-1}$ = a sparse approximation to $\mathbf{A}^{-1}$. (the sparse approximate inverse preconditioner (SAIP))
- **M** = a related matrix; e.g., if **A** is a discretization of a differential operator, **M** might be a discretization of a related operator that is easier to solve. Or **M** might be the block diagonal piece of the matrix after ordering for nested dissection.
- **M** might be the matrix from any stationary iterative method (SIM) or from multigrid (to be discussed).

- In some situations, it is a good idea to let **M** change at each iteration. The resulting algorithm is called flexible-GMRES. It is sometimes useful, but we won't discuss it here.

How to form $M^{-1}r$ for an SIM preconditioner

Consider your favorite stationary iterative method (Jacobi, Gauss-Seidel, SOR, etc.),

$$\mathbf{M}\mathbf{x}^{(k+1)} = \mathbf{N}\mathbf{x}^{(k)} + \mathbf{b}$$

or

$$\mathbf{x}^{(k+1)} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x}^{(k)} + \mathbf{M}^{-1}\mathbf{b}.$$

Manipulating this a bit, we get

$$\begin{aligned}
\mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + (\mathbf{M}^{-1}\mathbf{N} - \mathbf{I})\mathbf{x}^{(k)} + \mathbf{M}^{-1}\mathbf{b} \\
&= \mathbf{x}^{(k)} + \mathbf{M}^{-1}(\mathbf{N} - \mathbf{M})\mathbf{x}^{(k)} + \mathbf{M}^{-1}\mathbf{b} \\
&= \mathbf{x}^{(k)} + \mathbf{M}^{-1}(\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}) \\
&= \mathbf{x}^{(k)} + \mathbf{M}^{-1}\mathbf{r}^{(k)} .
\end{aligned}$$

Therefore, we compute the "preconditioned residual" by taking one step of the SIM starting from the latest CG or GMRES iterate, and returning $\Delta\mathbf{x}$ for $\mathbf{M}^{-1}\mathbf{r}^{(k)}$. It is this matrix $\mathbf{M}^{-1}$, that represents the multiple of the residual that we add on to $\mathbf{x}$, that preconditions CG or GMRES.

Multigrid methods

Reference: Chapter 32

The idea behind multigrid methods

(Idea: Fedorenko 1964, Brandt 1977, Nicolaides, Hackbusch, ...)
Consider our simplest problem

$$-u'' = f(x)$$

on the interval $x \in (0,1)$, with $u(0) = u(1) = 0$.

There are three ingredients to the idea.

- If we use a very coarse grid for finite elements, with $h = .25$, for example, then
  - The linear system of equations is very small ($n = 3$) so we can solve it fast using either a direct or an iterative method.
  - We expect our computed solution u_h to have the same overall shape as the true solution u but to lose a lot of local detail.

- If we use a very fine grid, then
  - The linear system of equations is much more expensive to solve.
  - We expect our computed solution u_h to be very close to u.

- If an iterative method is started with an initial guess that is close to the true solution, we hope to need a very small number of iterations.
  - In particular, if we consider Jacobi, Gauss-Seidel, or SOR, we adjust the solution based on very local information, so these methods are good at filling in the fine details once the overall shape of the solution is known.

Making use of multiple grids

An idea: Nested iteration

```
Set $k = 0$, $h = 1$, and $u_h = 0$.
While the approximation is not good enough,
    Set $k = k + 1$, $n = 2^k - 1$, and $h = 1/(n + 1)$.
    Form the matrix $A_h$ and the right-hand side $b_h$, and solve the matrix problem for the finite element approximation $u_h$ using GS , with the initial guess formed from $u_{2h}$ evaluated at the mesh points.
```

The termination tolerance for the residual on grid h should be proportional to h^2, since that is the size of the local error. This algorithm runs from coarse grid to finest and is useful (although rather silly for 1D PDEs).

The V-Cycle

We can do better if we run from finest grid to coarsest grid and then back to finest.

This algorithm has 3 ingredients:

- An iterative method that converges quickly if most of the error is high frequency – oscillating rapidly – which happens when the overall shape of the solution is already identified.
- A way to transfer values from a fine grid to a coarse one – restriction.
  We let $\mathcal{R}_h$ be the operator that goes from grid h to grid $2h$.
- A way to transfer values from a coarse grid to a fine one – interpolation or prolongation.
  We let $\mathcal{I}_h$ be the operator that goes from grid $2h$ to grid h.

Gauss-Seidel gives us the first ingredient, while our finite element formula for the solution as a sum of basis function components gives us the last two. (For technical reasons, though, restriction should be the adjoint of interpolation, rather than using the finite element choices for both.)

For finite differences, interpolation and restriction can also be defined.

We'll define the V-Cycle idea recursively.

$\mathbf{v}_h$ = V-Cycle($\mathbf{v}_h$,$\mathbf{b}_h$, η_1,η_2)

1. Perform η_1 GS iterations on $\mathbf{A}_h\mathbf{u}_h = \mathbf{b}_h$ using $\mathbf{v}_h$ as the initial guess, obtaining an approximate solution that we still call $\mathbf{v}_h$.

2. If h is not the coarsest grid parameter,

   Let $\mathbf{v}_{2h}$ = V-Cycle(0,$\mathcal{R}_h(\mathbf{b}_h - \mathbf{A}_h\mathbf{v}_h)$, η_1,η_2).
   Set $\mathbf{v}_h = \mathbf{v}_h + \mathcal{I}_h\mathbf{v}_{2h}$.

3. Perform η_2 GS iterations on $\mathbf{A}_h\mathbf{u}_h = \mathbf{b}_h$ using $\mathbf{v}_h$ as the initial guess, obtaining an approximate solution that we still call $\mathbf{v}_h$.

The standard multigrid algorithm is to solve $\mathbf{A}_h\mathbf{u}_h = \mathbf{b}_h$ by repeating the V-Cycle until convergence.

Cost per V-Cycle

A GS iteration on a grid of size h costs about $nz(h)$ multiplications, where $nz(h)$ is the number of nonzeros in $\mathbf{A}_h$. Note that $nz(h) \approx 2nz(2h)$ since $\mathbf{A}_{2h}$ has about half as many rows as $\mathbf{A}_h$.

So performing 1 GS iteration on each grid $h, h/2, \dots, 1$ costs less than $nz(h)(1 + 1/2 + 1/4 + \dots) \approx 2nz(h)$ multiplications $\equiv$ 2 work-units.

So the cost of a V-Cycle is at most 2 times the cost of $\eta_1 + \eta_2$ GS iterations on the finest mesh.

Unquiz: Convince yourself that the storage necessary for all of the matrices and vectors is also a modest multiple of the storage necessary for the finest grid. []

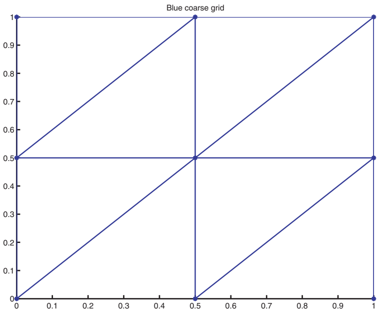
Convergence rate for multigrid

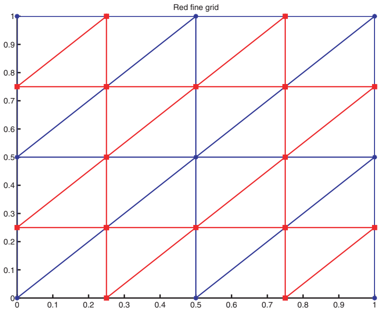
We know that standard iterative methods like GS are very slow (take many iterations), but on our simple problem, we need only a few iterations on each grid, and the total amount of work to solve the full problem to a residual of size $O(h^2)$ is a small number of work-units.

Multigrid for 2-d problems

Develop a sequence of nested grids.

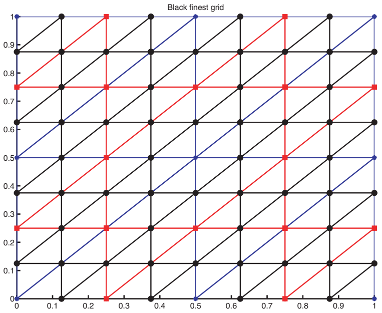
If the PDE is elliptic, it is not to hard to achieve convergence in a small number of work-units.





Multigridders would say that if you don't achieve it, then you have chosen either your iteration or your interpolation/restriction pair "incorrectly".

For the domain $(0,1) \times (0,1)$, we might use the three grids shown here.


Black finest grid
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1