July 20, 2013

Chapter 7: Linear Systems: Iterative Methods

Uri M. Ascher and Chen Greif
Department of Computer Science
The University of British Columbia
{ascher,greif}@cs.ubc.ca

Slides for the book
A First Course in Numerical Methods (published by SIAM, 2011)
`http://www.ec-securehost.com/SIAM/CS07.html`

Goals of chapter

- To learn simple and effective iterative methods for linear systems where direct methods are ineffective;
- to analyze these methods, establishing when and where they can be applied and how effective they are;
- to understand modern algorithms, specifically preconditioned conjugate gradients;
- *to get introduced to more advanced and more general Krylov subspace and multigrid techniques which often include the methods of choice for large scale computations.

Outline

- Stationary iteration and relaxation methods
- Application: model Poisson problem
- Convergence of stationary methods
- Conjugate gradient method
- *Krylov subspace methods
- *Multigrid methods

*advanced

Iterative methods for a linear problem

- In this chapter we consider the same problem as in Chapter 5: a linear system

$$A\mathbf{x} = \mathbf{b}$$

  where A is nonsingular $n \times n$.
- Iterative method: starting from initial guess $\mathbf{x}_0$, generate iterates $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k, \ldots$, hopefully converging to solution $\mathbf{x} = \mathbf{x}^*$.
- This approach is typical for nonlinear problems, see Chapter 3 and 9. Here, however, it is applied to a linear problem.
- But why not simply use LU decomposition, or

$$\mathbf{x} = \texttt{A "backslash" b}$$

  in Matlab?
- Generally, the matrix A must be somehow special to consider iterative methods!

Iterative methods for a linear problem

- In this chapter we consider the same problem as in Chapter 5: a linear system

$$A\mathbf{x} = \mathbf{b}$$

  where A is nonsingular $n \times n$.
- Iterative method: starting from initial guess $\mathbf{x}_0$, generate iterates $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k, \ldots$, hopefully converging to solution $\mathbf{x} = \mathbf{x}^*$.
- This approach is typical for nonlinear problems, see Chapter 3 and 9. Here, however, it is applied to a linear problem.
- But why not simply use LU decomposition, or

$$\mathbf{x} = \texttt{A "backslash" b}$$

  in Matlab?
- Generally, the matrix A must be somehow special to consider iterative methods!

Iterative methods for a linear problem

- In this chapter we consider the same problem as in Chapter 5: a linear system

$$A\mathbf{x} = \mathbf{b}$$

  where A is nonsingular $n \times n$.
- Iterative method: starting from initial guess $\mathbf{x}_0$, generate iterates $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k, \ldots$, hopefully converging to solution $\mathbf{x} = \mathbf{x}^*$.
- This approach is typical for nonlinear problems, see Chapter 3 and 9. Here, however, it is applied to a linear problem.
- But why not simply use LU decomposition, or

$$\mathbf{x} = \texttt{A "backslash" } \mathbf{b}$$

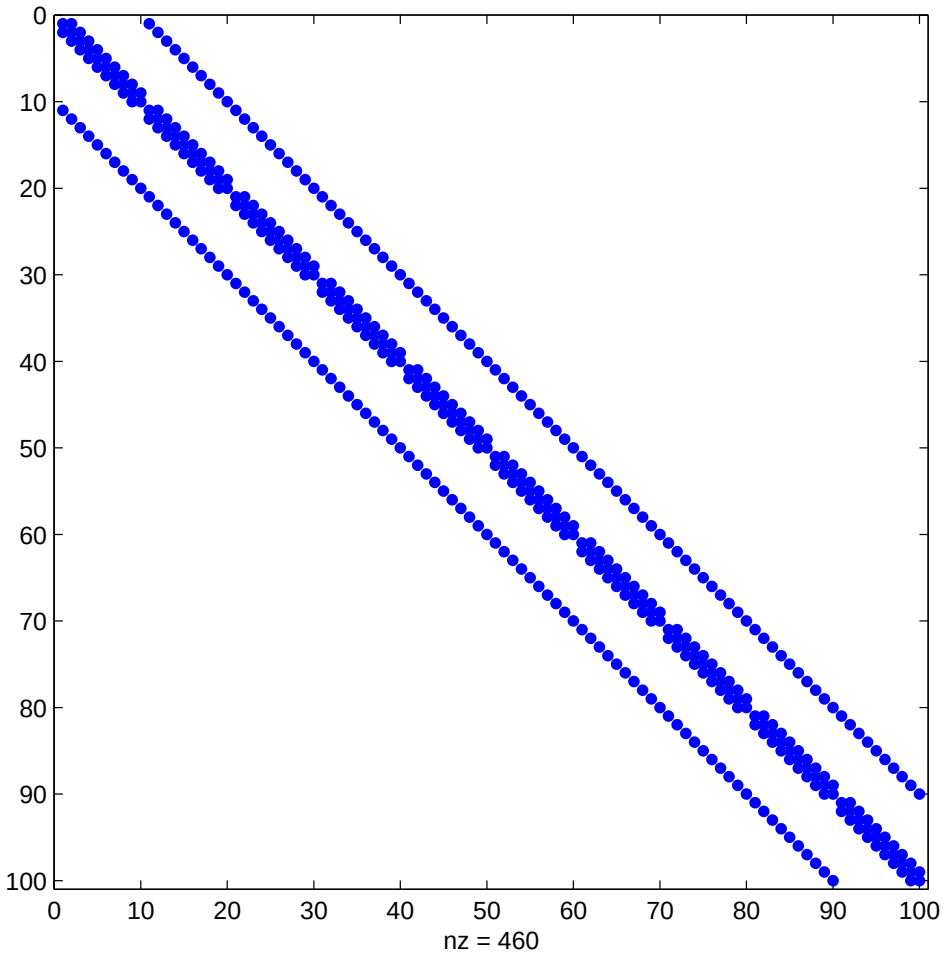
  in Matlab?
- Generally, the matrix A must be somehow special to consider iterative methods!

Why (or when) not to use a direct method

- If A is large and sparse, LU decomposition (Gaussian elimination) may introduce fill-in.
- Want to take advantage when only a *rough approximation* $\hat{\mathbf{x}}$ to $\mathbf{x}$ is required.
- Want to take advantage when a good $\mathbf{x}_0$ approximating $\mathbf{x}$ is known (warm start).
- Sometimes A is not explicitly available, only matrix-vector products $A\mathbf{v}$ for any vector $\mathbf{v}$ can be efficiently carried out.

The famous Poisson matrix

A particularly famous example of a sparse matrix is that of the discretization of the Poisson partial differential equation. Here is an example of the sparsity pattern of a 100×100 Poisson matrix, using the MATLAB command spy.



Outline

- Stationary iteration and relaxation methods
- Application: model Poisson problem
- Convergence of stationary methods
- Conjugate gradient method
- *Krylov subspace methods
- *Multigrid methods

Jacobi and Gauss-Seidel relaxation methods

Given A, denote by D its diagonal part and by E its lower triangular part:

`D = diag(diag(A)) ; E = tril(A)`

e.g.

$$A = \begin{pmatrix} 7 & 3 & 1 \\ -3 & 10 & 2 \\ 1 & 7 & -15 \end{pmatrix}, \quad \Rightarrow D = \begin{pmatrix} 7 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & -15 \end{pmatrix}, \quad E = \begin{pmatrix} 7 & 0 & 0 \\ -3 & 10 & 0 \\ 1 & 7 & -15 \end{pmatrix}.$$

Given iterates $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k, \ldots$ denote residual vector

$$\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k.$$

Jacobi's method

$$\mathbf{x}_{k+1} = \mathbf{x}_k + D^{-1}\mathbf{r}_k$$

Gauss-Seidel method

$$\mathbf{x}_{k+1} = \mathbf{x}_k + E^{-1}\mathbf{r}_k$$

Jacobi and Gauss-Seidel relaxation methods

Given A, denote by D its diagonal part and by E its lower triangular part:

`D = diag(diag(A)) ; E = tril(A)`

e.g.

$$A = \begin{pmatrix} 7 & 3 & 1 \\ -3 & 10 & 2 \\ 1 & 7 & -15 \end{pmatrix}, \quad \Rightarrow D = \begin{pmatrix} 7 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & -15 \end{pmatrix}, \quad E = \begin{pmatrix} 7 & 0 & 0 \\ -3 & 10 & 0 \\ 1 & 7 & -15 \end{pmatrix}.$$

Given iterates $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k, \ldots$ denote residual vector

$$\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k.$$

Jacobi's method

$$\mathbf{x}_{k+1} = \mathbf{x}_k + D^{-1}\mathbf{r}_k$$

Gauss-Seidel method

$$\mathbf{x}_{k+1} = \mathbf{x}_k + E^{-1}\mathbf{r}_k$$

Jacobi and Gauss-Seidel relaxation methods

Given A, denote by D its diagonal part and by E its lower triangular part:

`D = diag(diag(A)) ; E = tril(A)`

e.g.

$$A = \begin{pmatrix} 7 & 3 & 1 \\ -3 & 10 & 2 \\ 1 & 7 & -15 \end{pmatrix}, \quad \Rightarrow D = \begin{pmatrix} 7 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & -15 \end{pmatrix}, \quad E = \begin{pmatrix} 7 & 0 & 0 \\ -3 & 10 & 0 \\ 1 & 7 & -15 \end{pmatrix}.$$

Given iterates $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k, \ldots$ denote residual vector

$$\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k.$$

Jacobi's method

$$\mathbf{x}_{k+1} = \mathbf{x}_k + D^{-1}\mathbf{r}_k$$

Gauss-Seidel method

$$\mathbf{x}_{k+1} = \mathbf{x}_k + E^{-1}\mathbf{r}_k$$

Jacobi and Gauss-Seidel relaxation methods

Given A, denote by D its diagonal part and by E its lower triangular part:

`D = diag(diag(A)) ; E = tril(A)`

e.g.

$$A = \begin{pmatrix} 7 & 3 & 1 \\ -3 & 10 & 2 \\ 1 & 7 & -15 \end{pmatrix}, \quad \Rightarrow D = \begin{pmatrix} 7 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & -15 \end{pmatrix}, \quad E = \begin{pmatrix} 7 & 0 & 0 \\ -3 & 10 & 0 \\ 1 & 7 & -15 \end{pmatrix}.$$

Given iterates $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k, \ldots$ denote residual vector

$$\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k.$$

Jacobi's method

$$\mathbf{x}_{k+1} = \mathbf{x}_k + D^{-1}\mathbf{r}_k$$

Gauss-Seidel method

$$\mathbf{x}_{k+1} = \mathbf{x}_k + E^{-1}\mathbf{r}_k$$

Example: Jacobi (simultaneous relaxation)

- Consider the linear system

$$\begin{aligned} 7x_1 + 3x_2 + x_3 &= 3 \\ -3x_1 + 10x_2 + 2x_3 &= 4 \\ x_1 + 7x_2 - 15x_3 &= 2\ . \end{aligned}$$

- Write as

$$\begin{aligned} 7x_1 &= 3 - 3x_2 - x_3 \\ 10x_2 &= 4 + 3x_1 - 2x_3 \\ -15x_3 &= 2 - x_1 - 7x_2\ . \end{aligned}$$

(Corresponds to a splitting $A = M - N$ with $M = D$.)

- Evaluate right hand side at current iterate k and left hand side as unknown

$$\begin{aligned} x_1^{(k+1)} &= (3 - 3x_2^{(k)} - x_3^{(k)})/7 \\ x_2^{(k+1)} &= (4 + 3x_1^{(k)} - 2x_3^{(k)})/10 \\ x_3^{(k+1)} &= (2 - x_1^{(k)} - 7x_2^{(k)})/(-15)\ , \end{aligned}$$

for $k = 0, 1, 2, \ldots$

Example: Gauss-Seidel

- Write same linear system as

$$\begin{aligned} 7x_1 &= 3 - 3x_2 - x_3 \\ 10x_2 - 3x_1 &= 4 - 2x_3 \\ -15x_3 + x_1 + 7x_2 &= 2 \ . \end{aligned}$$

(Corresponds to a splitting $A = M - N$ with $M = E$.)

- Evaluate right hand side at current iterate k and left hand side as unknown (i.e., forward substitution)

$$\begin{aligned} x_1^{(k+1)} &= (3 - 3x_2^{(k)} - x_3^{(k)})/7 \\ x_2^{(k+1)} &= (4 + 3x_1^{(k+1)} - 2x_3^{(k)})/10 \\ x_3^{(k+1)} &= (2 - x_1^{(k+1)} - 7x_2^{(k+1)})/(-15) \ , \end{aligned}$$

for $k = 0, 1, 2, \ldots$

Properties of Jacobi and Gauss-Seidel relaxations

- Jacobi is more easily parallelized.
- Jacobi matrix M is symmetric.
- GS converges whenever Jacobi converges and often (but not always) twice as fast.
- Both methods are simple but slow. Used as building blocks for faster, more complex methods.
- Both Jacobi and GS are simple examples of a stationary method.
- In general, based on a splitting $A = M - N$, the (fixed point) stationary iterative method is

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}\mathbf{r}_k.$$

- Can be written equivalently as

$$M\mathbf{x}_{k+1} = N\mathbf{x}_k + \mathbf{b}.$$

- It is called *stationary* because M is independent of the iteration counter k.

Properties of Jacobi and Gauss-Seidel relaxations

- Jacobi is more easily parallelized.
- Jacobi matrix M is symmetric.
- GS converges whenever Jacobi converges and often (but not always) twice as fast.
- Both methods are simple but slow. Used as building blocks for faster, more complex methods.
- Both Jacobi and GS are simple examples of a stationary method.
- In general, based on a splitting $A = M - N$, the (fixed point) stationary iterative method is

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}\mathbf{r}_k.$$

- Can be written equivalently as

$$M\mathbf{x}_{k+1} = N\mathbf{x}_k + \mathbf{b}.$$

- It is called *stationary* because M is independent of the iteration counter k.

Over-relaxation and under-relaxation

- There are more sophisticated stationary methods than Jacobi and GS. The methods introduced below are based on a simple modification of the ancient ones.
- Let $\mathbf{x}_{k+1}$ be obtained from $\mathbf{x}_k$ by either Jacobi or GS.
  Modify it further by

$$\mathbf{x}_{k+1} \leftarrow \omega \mathbf{x}_{k+1} + (1-\omega)\mathbf{x}_k$$

  where ω is a parameter.
- Two useful variants:
  - Based on Gauss-Seidel (GS) and $1 < \omega < 2$, obtain faster successive over-relaxation (SOR).

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \omega[(1-\omega)D + \omega E]^{-1}\mathbf{r}_k.$$

  - Based on Jacobi and $\omega \approx 0.8$, obtain slower under-relaxation which is a good smoother in some applications.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \omega D^{-1}\mathbf{r}_k.$$

Over-relaxation and under-relaxation

- There are more sophisticated stationary methods than Jacobi and GS. The methods introduced below are based on a simple modification of the ancient ones.
- Let $\mathbf{x}_{k+1}$ be obtained from $\mathbf{x}_k$ by either Jacobi or GS.
  Modify it further by

$$\mathbf{x}_{k+1} \leftarrow \omega \mathbf{x}_{k+1} + (1-\omega)\mathbf{x}_k$$

  where ω is a parameter.
- Two useful variants:
  - Based on Gauss-Seidel (GS) and $1 < \omega < 2$, obtain faster successive over-relaxation (SOR).

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \omega[(1-\omega)D + \omega E]^{-1}\mathbf{r}_k.$$

  - Based on Jacobi and $\omega \approx 0.8$, obtain slower under-relaxation which is a good smoother in some applications.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \omega D^{-1}\mathbf{r}_k.$$

Over-relaxation and under-relaxation

- There are more sophisticated stationary methods than Jacobi and GS. The methods introduced below are based on a simple modification of the ancient ones.
- Let $\mathbf{x}_{k+1}$ be obtained from $\mathbf{x}_k$ by either Jacobi or GS.
  Modify it further by

$$\mathbf{x}_{k+1} \leftarrow \omega \mathbf{x}_{k+1} + (1-\omega)\mathbf{x}_k$$

  where ω is a parameter.
- Two useful variants:
  - Based on Gauss-Seidel (GS) and $1 < \omega < 2$, obtain faster successive over-relaxation (SOR).

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \omega[(1-\omega)D + \omega E]^{-1}\mathbf{r}_k.$$

  - Based on Jacobi and $\omega \approx 0.8$, obtain slower under-relaxation which is a good smoother in some applications.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \omega D^{-1}\mathbf{r}_k.$$

Outline

- Stationary iteration and relaxation methods
- Application: model Poisson problem
- Convergence of stationary methods
- Conjugate gradient method
- *Krylov subspace methods
- *Multigrid methods

Poisson problem

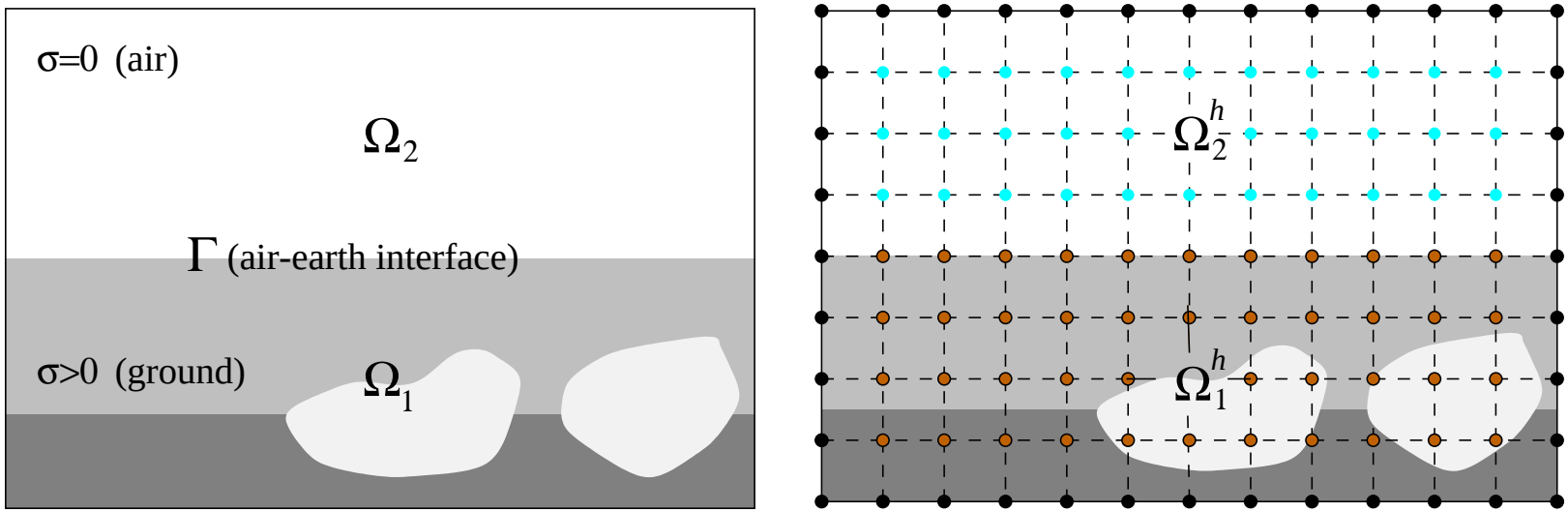
- There are many practical problems involving large, sparse matrices (not our 3×3 example!) where simple stationary methods are relevant. We now develop one such prototype example.
- The Poisson equation is a partial differential equation that in its simplest form is defined on the open *unit square*, $0 < x, y < 1$, and reads

$$-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = g(x, y).$$

  Here $u(x, y)$ is the unknown function sought and $g(x, y)$ is a given *source*.
- Boundary conditions: assume that the sought function $u(x, y)$ satisfies *homogeneous Dirichlet boundary conditions* along the entire boundary of the unit square, written as

$$u(x, 0) = u(x, 1) = u(0, y) = u(1, y) = 0.$$

The need to discretize



(a) An earthly domain

(b) With a discretization grid

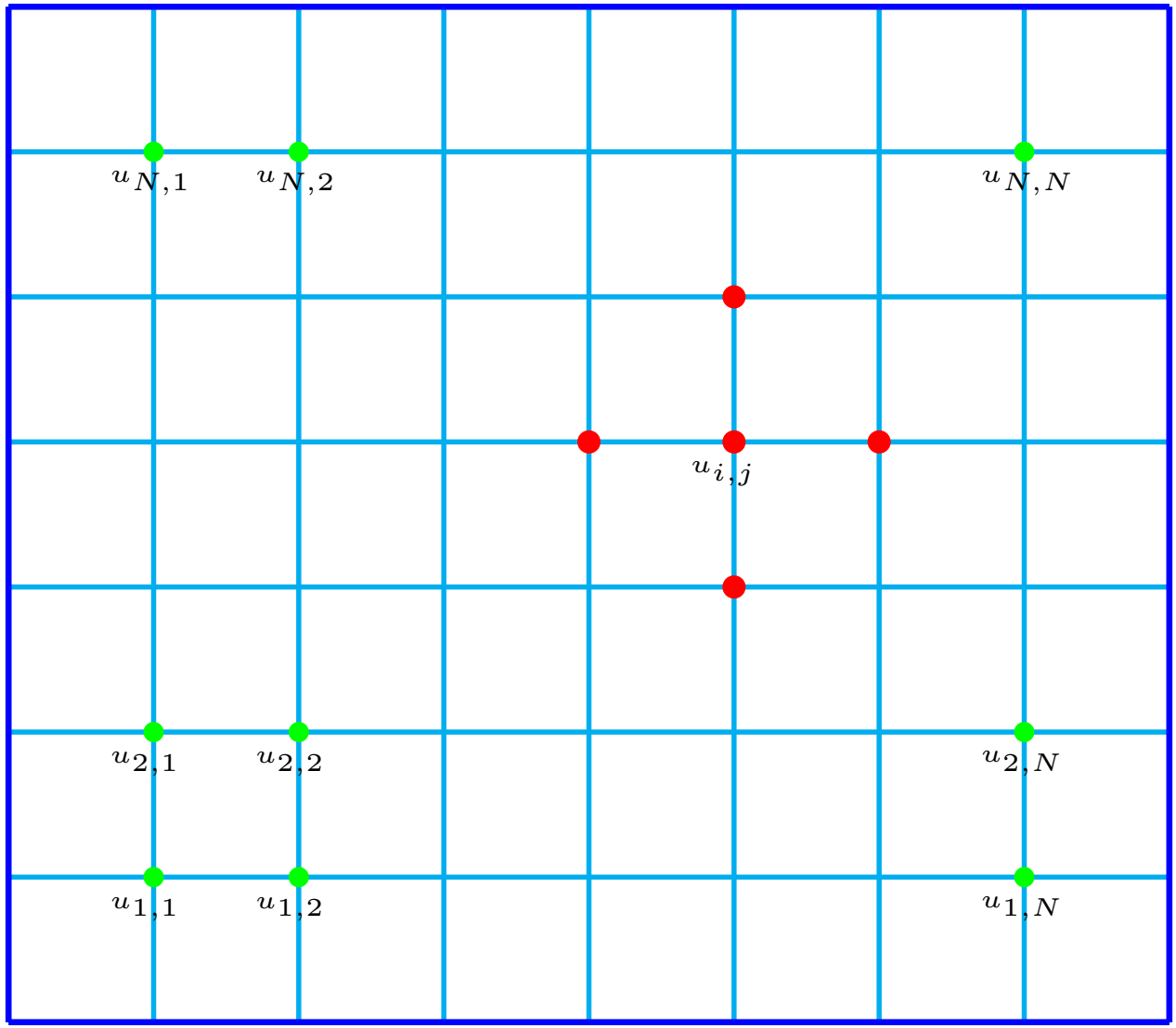
FIGURE: A $2D$ cross-section of a $3D$ domain with a square grid added.

Finite Difference Discretization

- Discretizing using centred differences for the partial derivatives we obtain the equations

$$\begin{aligned} 4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} &= b_{i,j}, \quad 1 \le i,j \le N, \\ u_{i,j} &= 0 \quad \text{otherwise.} \end{aligned}$$

- In this difference scheme, $u_{i,j}$ is the value at the $(i,j)th$ node of a square planar grid, and $b_{i,j} = h^2 g(ih, jh)$ are given values at the same grid locations. Here $h = 1/(N+1)$ is the *grid width*.
- See the grid on the next slide, and note in particular the location of a $u_{i,j}$ and those of its neighbours which appear in the above formula (distinguished by red dots).
- For $u_{i,j}$ to approximate the differential equation solution $u(ih, jh)$ well, need to set h "sufficiently small". Hence, N can easily become large. For example, $h = 0.01$ gives $N = 99$.


$u_{N,1}$
$u_{N,2}$
$u_{N,N}$
$u_{i,j}$
$u_{2,1}$
$u_{2,2}$
$u_{2,N}$
$u_{1,1}$
$u_{1,2}$
$u_{1,N}$

A linear system

- Obviously, these are linear relations, so we can express them as a system of linear equations

$$A\mathbf{u} = \mathbf{b},$$

  where $\mathbf{u}$ consists of the $n = N^2$ unknowns $\{u_{i,j}\}$ somehow organized as a vector, and $\mathbf{b}$ is composed likewise from the values $\{b_{i,j}\}$.
- The two-dimensional problem has features related to sparsity that are not seen in the one-dimensional problem; we will return to this point soon.
- Note that n can easily become very large. Even for a simple 2D case with $N = 100$, the matrix dimension is $n = 10,000$. In 3D and the same h we would get $n = 10^6$, so A cannot even be stored as a full matrix. This, for the simplest problem of its type!

Ordering the unknowns

How should we order the grid unknowns $\{u_{i,j}\}_{i,j=1}^{N}$ into a vector $\mathbf{u}$? We can do this in many ways. A simple (and rational) way is lexicographically, say by columns, which yields

$$\mathbf{u} = \begin{pmatrix} u_{1,1} \\ u_{2,1} \\ \vdots \\ u_{N,1} \\ u_{1,2} \\ u_{2,2} \\ \vdots \\ u_{N,2} \\ u_{1,3} \\ \vdots \\ u_{N,N} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_{1,1} \\ b_{2,1} \\ \vdots \\ b_{N,1} \\ b_{1,2} \\ b_{2,2} \\ \vdots \\ b_{N,2} \\ b_{1,3} \\ \vdots \\ b_{N,N} \end{pmatrix}.$$

Block tridiagonal matrix

The $n \times n$ matrix A (with $n = N^2$) has the form

$$A = \begin{pmatrix} J & -I & & & \\ -I & J & -I & & \\ & \ddots & \ddots & \ddots & \\ & & -I & J & -I \\ & & & -I & J \end{pmatrix},$$

where J is the tridiagonal $N \times N$ matrix

$$J = \begin{pmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 4 & -1 \\ & & & -1 & 4 \end{pmatrix},$$

and I denotes the identity matrix of size N.

Example

For instance, if $N = 3$ then

$$A = \left(\begin{array}{ccc|ccc|ccc} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ \hline -1 & 0 & 0 & 4 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ \hline 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 \end{array}\right).$$

Note zero-diagonals within the band! In general there are $N - 2$ such diagonals, because neighbouring unknowns $u_{i,j}$, $u_{i\pm1,j\pm1}$ are no longer consecutive in the vector $\mathbf{u}$. Recall figure of the sparsity structure of the 100×100 (i.e., $N = 10$) "Poisson matrix".

Sparsity issues

- There is a fundamental difference between the 1D and the 2D & 3D problems.
- The 1D problem (i.e., only x, no y variable) produces a tridiagonal matrix, for which Gaussian elimination (GE) requires a *linear*, i.e., $\mathcal{O}(n)$, number of floating point operations (optimal order of complexity). No iterative methods are needed here.
- Higher-dimensional problems give rise to matrices that are sparse *within the band*, and complexity of GE is certainly not linear (in 2D it is $\mathcal{O}(n^2)$). This is in fact where iterative methods can be very effective.
- As a rule of thumb, for this problem in 2D special direct methods are still competitive, but in 3D they no longer are.

Eigenvalues of the Poisson matrix

- For any size N, the matrix A is diagonally dominant and nonsingular.
- It can be verified directly that the $n = N^2$ eigenvalues of A are given by

$$\lambda_{l,m} = 4 - 2\left(\cos(l\pi h) + \cos(m\pi h)\right), \quad 1 \le l, m \le N$$

  (recall $(N+1)h = 1$).
- Thus $\lambda_{l,m} > 0$ for all $1 \le l, m \le N$, so the matrix A is symmetric positive definite.
- It also follows that the condition number satisfies $\kappa(A) = \frac{\lambda_{N,N}}{\lambda_{1,1}} = \mathcal{O}(n)$.
- Knowing the eigenvalues explicitly is helpful in understanding performance, convergence and accuracy issues related to iterative solvers. Such specific knowledge is not typically available for more complex problems of this type, yet the conclusions drawn for the model Poisson problem are often indicative of what happens more generally.

Performance of our relaxation methods for Poisson

- Recall that A is composed of relations of the form

$$4u_{i,j} - u_{1+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} = b_{i,j}, \quad 1 \le i, j \le N.$$

- Can apply the relaxation methods directly, without ever forming A, but bear in mind vectorization issues.
- On a 15×15 grid: $N = 15,\ n = 225$, obtain

relaxation method	ω	Error after 2 itns	Error after 20 itns
Jacobi	1	7.1e-2	5.4e-2
GS	1	6.9e-2	3.8e-2
SOR	1.69	5.6e-2	4.8e-4

- So, Jacobi and GS are both very slow to converge, SOR with $\omega = 1.69$ is significantly faster.
- For larger N these methods require even more iterations, as we shall soon see.

Outline

- Stationary iteration and relaxation methods
- Application: model Poisson problem
- Convergence of stationary methods
- Conjugate gradient method
- *Krylov subspace methods
- *Multigrid methods

General stationary method

Based on splitting $A = M - N$, the (fixed point) iterative method is

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}\mathbf{r}_k.$$

Can be written equivalently as

$$M\mathbf{x}_{k+1} = N\mathbf{x}_k + \mathbf{b}.$$

This is called *stationary* because M is independent of iteration counter k.

Iteration matrix

$$M\mathbf{x}_{k+1} = N\mathbf{x}_k + \mathbf{b}, \quad k = 0, 1, 2, \ldots.$$

- For the exact solution, of course

$$M\mathbf{x} = N\mathbf{x} + \mathbf{b}, \quad N = M - A.$$

- Define error in the kth iteration

$$\mathbf{e}_k = \mathbf{x} - \mathbf{x}_k.$$

Then

$$M\mathbf{e}_{k+1} = (M - A)\mathbf{e}_k, \quad k = 0, 1, 2, \ldots.$$

- Let $T = I - M^{-1}A$, the iteration matrix. Then

$$\mathbf{e}_{k+1} = T\mathbf{e}_k = T(T\mathbf{e}_{k-1}) = \cdots = T^{k+1}\mathbf{e}_0.$$

- Thus, convergence iff $T^k \to 0$.

Convergence of stationary methods

- Method converges if and only if

$$\rho(T) < 1,$$

where to recall (Chapter 4), the spectral radius of a square matrix B with eigenvalues $\mu_1, \ldots, \mu_n$ is the maximum eigenvalue magnitude

$$\rho(B) = \max_i |\mu_i|.$$

- How fast does the iteration converge? Define rate of convergence

$$rate = -\log_{10} \rho(T).$$

- Then number of iterations required to reduce error by factor 10 (i.e., gain a decimal digit) is

$$k \approx \frac{1}{rate}.$$

Convergence of stationary methods

- Method converges if and only if

$$\rho(T) < 1,$$

where to recall (Chapter 4), the spectral radius of a square matrix B with eigenvalues $\mu_1, \ldots, \mu_n$ is the maximum eigenvalue magnitude

$$\rho(B) = \max_i |\mu_i|.$$

- How fast does the iteration converge? Define rate of convergence

$$rate = -\log_{10} \rho(T).$$

- Then number of iterations required to reduce error by factor 10 (i.e., gain a decimal digit) is

$$k \approx \frac{1}{rate}.$$

Convergence of Jacobi for model Poisson problem

- Recall matrix is defined by

$$4u_{i,j} - u_{1+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} = b_{i,j}, \quad 1 \le i,j \le N.$$

  Eigenvalues

$$\lambda_{l,m} = 4 - 2\left(\cos\frac{l\pi}{N+1} + \cos\frac{m\pi}{N+1}\right), \quad 1 \le l,m \le N\,.$$

- Consider Jacobi: $M = D$. Then

$$T = I - D^{-1}A = I - \frac{1}{4}A.$$

- So eigenvalues of iteration matrix are

$$\mu_{l,m} = 1 - \frac{1}{4}\lambda_{l,m} = \frac{1}{2}\left(\cos\frac{l\pi}{N+1} + \cos\frac{m\pi}{N+1}\right), \quad 1 \le l,m \le N\,.$$

Convergence of Jacobi cont.

- Eigenvalues of T are

$$\mu_{l,m} = 1 - \frac{1}{4}\lambda_{l,m} = \frac{1}{2}\left(\cos\frac{l\pi}{N+1} + \cos\frac{m\pi}{N+1}\right), \quad 1 \le l, m \le N\,.$$

- Spectral radius

$$\rho(T) = \rho_J(T) = \mu_{1,1} = \cos\frac{\pi}{N+1} \le 1 - \frac{c}{n}$$

- Rate of convergence

$$rate = -\log\rho(T) \sim 1/n.$$

Thus, $\mathcal{O}(n)$ iterations are required for error reduction by a constant factor. (Asymptotically the same cost as GE.)

Optimal parameter for SOR

1. Denote the optimal parameter (i.e., that which reduces error most rapidly) by ω_{opt}.
2. For a class of matrices including model Poisson, the optimal parameter is given by

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho_J^2}} > 1,$$

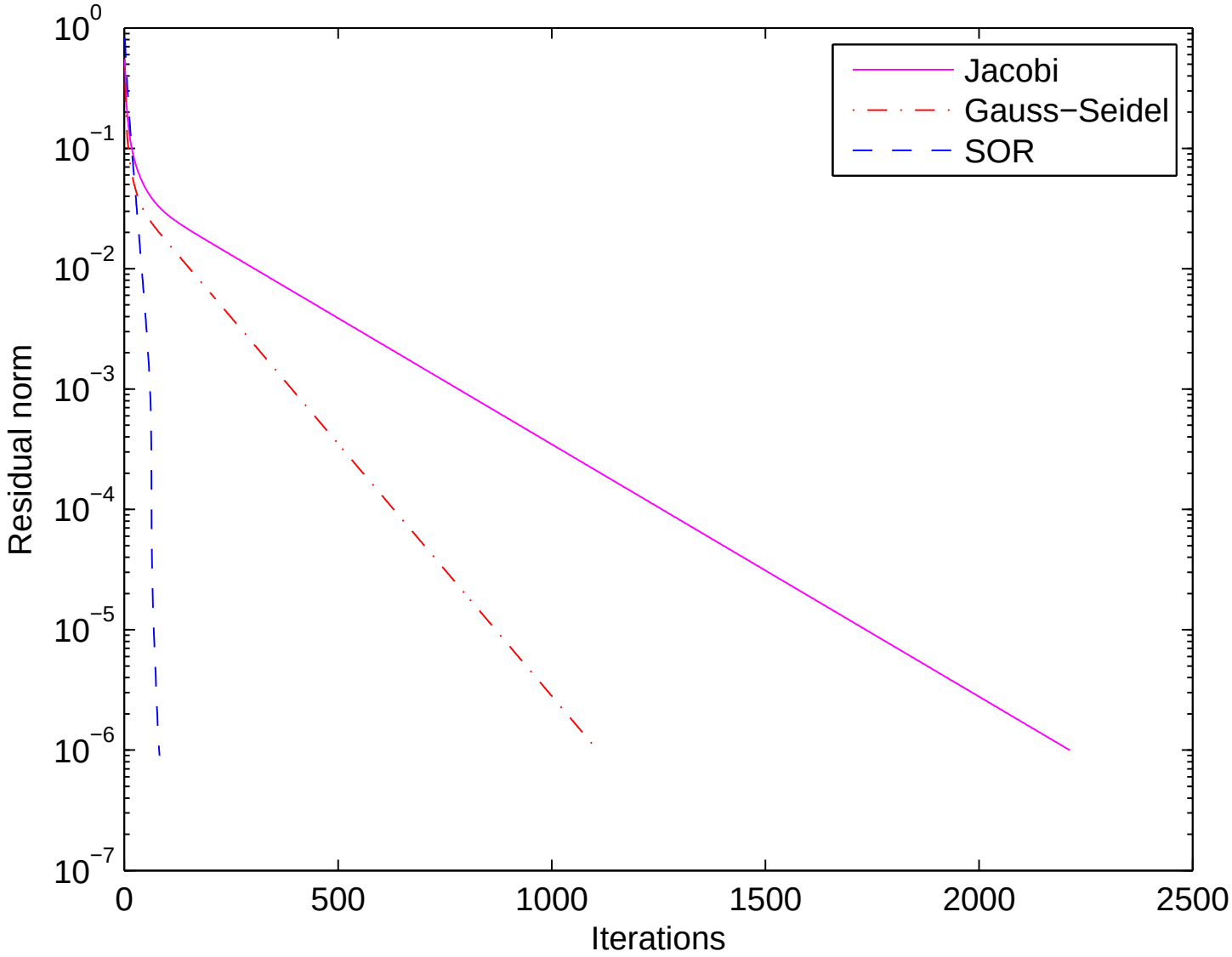
   where ρ_J is the spectral radius of the Jacobi iteration matrix.
3. For the model Poisson problem we therefore obtain

$$\omega_{opt} = \frac{2}{1 + \sin\left(\frac{\pi}{N+1}\right)}.$$

4. For this class of matrices, the spectral radius of the SOR matrix for ω_{opt} is $\omega_{opt} - 1 = 1 - \frac{c}{N}$.

Convergence of stationary methods for Poisson

For $N = 15,\ n = 225$:



Relaxation methods for the model Poisson problem

- Jacobi requires $\mathcal{O}(n)$ iterations hence $\mathcal{O}(n^2)$ flops
- Gauss-Seidel is twice as fast, so same $\mathcal{O}$
- SOR with optimal ω requires $\mathcal{O}(N)$ iterations hence $\mathcal{O}(n^{3/2})$ flops!
- Lower bound: the best possible method would require at least $\mathcal{O}(n)$ flops.
- So there seems to be room for improvement: are there better methods?

Relaxation methods for the model Poisson problem

- Jacobi requires $\mathcal{O}(n)$ iterations hence $\mathcal{O}(n^2)$ flops
- Gauss-Seidel is twice as fast, so same $\mathcal{O}$
- SOR with optimal ω requires $\mathcal{O}(N)$ iterations hence $\mathcal{O}(n^{3/2})$ flops!
- Lower bound: the best possible method would require at least $\mathcal{O}(n)$ flops.
- So there seems to be room for improvement: are there better methods?

Outline

- Stationary iteration and relaxation methods
- Application: model Poisson problem
- Convergence of stationary methods
- Conjugate gradient method
- *Krylov subspace methods
- *Multigrid methods

Nonstationary methods

- None of the stationary methods we have seen is very fast even for the model Poisson problem when n is really large.
- Another, general disadvantage of all these relaxation methods is that they require an explicitly given matrix A: what if A is only given implicitly, through matrix-vector products?
- Try to take advantage of accumulating knowledge as the iteration proceeds.
- Consider $M = M_k$, and more generally

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k.$$

The scalar $\alpha_k > 0$ is a step size; $\mathbf{p}_k$ is a search direction.

- Restrict consideration to cases where A is symmetric positive definite.
- Then solving $A\mathbf{x} = \mathbf{b}$ is equivalent to

$$\min_{\mathbf{x}} \phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}.$$

Nonstationary methods

- None of the stationary methods we have seen is very fast even for the model Poisson problem when n is really large.
- Another, general disadvantage of all these relaxation methods is that they require an explicitly given matrix A: what if A is only given implicitly, through matrix-vector products?
- Try to take advantage of accumulating knowledge as the iteration proceeds.
- Consider $M = M_k$, and more generally

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k.$$

The scalar $\alpha_k > 0$ is a step size; $\mathbf{p}_k$ is a search direction.

- Restrict consideration to cases where A is symmetric positive definite.
- Then solving $A\mathbf{x} = \mathbf{b}$ is equivalent to

$$\min_{\mathbf{x}} \phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}.$$

Nonstationary methods

- None of the stationary methods we have seen is very fast even for the model Poisson problem when n is really large.
- Another, general disadvantage of all these relaxation methods is that they require an explicitly given matrix A: what if A is only given implicitly, through matrix-vector products?
- Try to take advantage of accumulating knowledge as the iteration proceeds.
- Consider $M = M_k$, and more generally

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k.$$

The scalar $\alpha_k > 0$ is a step size; $\mathbf{p}_k$ is a search direction.

- Restrict consideration to cases where A is symmetric positive definite.
- Then solving $A\mathbf{x} = \mathbf{b}$ is equivalent to

$$\min_{\mathbf{x}} \phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}.$$

Gradient descent and steepest descent

- Gradient descent: take $\mathbf{p}_k = \mathbf{r}_k$, i.e., $M_k = \alpha_k^{-1} I$.
- How to choose the step size?
- Simple-minded, greedy approach: exact line search.

$$\min_{\alpha} \phi(\mathbf{x}_k + \alpha_k \mathbf{r}_k) = \frac{1}{2}(\mathbf{x}_k + \alpha \mathbf{r}_k)^T A(\mathbf{x}_k + \alpha \mathbf{r}_k) - \mathbf{b}^T(\mathbf{x}_k + \alpha \mathbf{r}_k).$$

- Critical point: differentiate wrto α and set to 0. Obtain steepest descent (poor, but popular, choice of name) with

$$\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{r}_k^T A \mathbf{r}_k} = \frac{\langle \mathbf{r}_k, \mathbf{r}_k \rangle}{\langle \mathbf{r}_k, A\mathbf{r}_k \rangle}.$$

Gradient descent and steepest descent

- Gradient descent: take $\mathbf{p}_k = \mathbf{r}_k$, i.e., $M_k = \alpha_k^{-1} I$.
- How to choose the step size?
- Simple-minded, greedy approach: exact line search.

$$\min_{\alpha} \phi(\mathbf{x}_k + \alpha_k \mathbf{r}_k) = \frac{1}{2}(\mathbf{x}_k + \alpha \mathbf{r}_k)^T A(\mathbf{x}_k + \alpha \mathbf{r}_k) - \mathbf{b}^T(\mathbf{x}_k + \alpha \mathbf{r}_k).$$

- Critical point: differentiate wrto α and set to 0. Obtain steepest descent (poor, but popular, choice of name) with

$$\alpha_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{r}_k^T A \mathbf{r}_k} = \frac{\langle \mathbf{r}_k, \mathbf{r}_k \rangle}{\langle \mathbf{r}_k, A\mathbf{r}_k \rangle}.$$

Steepest descent algorithm

Given an initial guess $\mathbf{x}_0$ and a tolerance tol, set at first $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\delta_0 = \langle \mathbf{r}_0, \mathbf{r}_0 \rangle$, $b_\delta = \langle \mathbf{b}, \mathbf{b} \rangle$ and $k = 0$. Then:
While $\delta_k > tol^2 \; b_\delta$,

$$\begin{aligned} \mathbf{s}_k &= A\mathbf{r}_k \\ \alpha_k &= \frac{\delta_k}{\langle \mathbf{r}_k, \mathbf{s}_k \rangle} \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{r}_k \\ \mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k \mathbf{s}_k \\ \delta_{k+1} &= \langle \mathbf{r}_{k+1}, \mathbf{r}_{k+1} \rangle \\ k &= k + 1. \end{aligned}$$

Note organization so that only one matrix-vector multiplication per iteration is required.
Properties of this algorithm will be summarized after we introduce the better, conjugate gradient algorithm.

Steepest descent algorithm

Given an initial guess $\mathbf{x}_0$ and a tolerance tol, set at first $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\delta_0 = \langle \mathbf{r}_0, \mathbf{r}_0 \rangle$, $b_\delta = \langle \mathbf{b}, \mathbf{b} \rangle$ and $k = 0$. Then:
While $\delta_k > tol^2 \; b_\delta$,

$$\begin{aligned} \mathbf{s}_k &= A\mathbf{r}_k \\ \alpha_k &= \frac{\delta_k}{\langle \mathbf{r}_k, \mathbf{s}_k \rangle} \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{r}_k \\ \mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k \mathbf{s}_k \\ \delta_{k+1} &= \langle \mathbf{r}_{k+1}, \mathbf{r}_{k+1} \rangle \\ k &= k + 1. \end{aligned}$$

Note organization so that only one matrix-vector multiplication per iteration is required.
Properties of this algorithm will be summarized after we introduce the better, conjugate gradient algorithm.

Conjugate gradient (CG) algorithm

Here the search direction is no longer the residual, but rather, a judicious linear combination of the residual with the previous search direction.

Given an initial guess $\mathbf{x}_0$ and a tolerance tol, set at first $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\delta_0 = \langle \mathbf{r}_0, \mathbf{r}_0 \rangle$, $b_\delta = \langle \mathbf{b}, \mathbf{b} \rangle$, $k = 0$ and $\mathbf{p}_0 = \mathbf{r}_0$. Then:
While $\delta_k > tol^2 \; b_\delta$,

$$
\begin{aligned}
\mathbf{s}_k &= A\mathbf{p}_k \\
\alpha_k &= \frac{\delta_k}{\langle \mathbf{p}_k, \mathbf{s}_k \rangle} \\
\mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k \\
\mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k \mathbf{s}_k \\
\delta_{k+1} &= \langle \mathbf{r}_{k+1}, \mathbf{r}_{k+1} \rangle \\
\mathbf{p}_{k+1} &= \mathbf{r}_{k+1} + \frac{\delta_{k+1}}{\delta_k} \mathbf{p}_k \\
k &= k + 1.
\end{aligned}
$$

Conjugate gradient (CG) algorithm

Here the search direction is no longer the residual, but rather, a judicious linear combination of the residual with the previous search direction.

Given an initial guess $\mathbf{x}_0$ and a tolerance tol, set at first $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\delta_0 = \langle \mathbf{r}_0, \mathbf{r}_0 \rangle$, $b_\delta = \langle \mathbf{b}, \mathbf{b} \rangle$, $k = 0$ and $\mathbf{p}_0 = \mathbf{r}_0$. Then:
While $\delta_k > tol^2 \; b_\delta$,

$$
\begin{aligned}
\mathbf{s}_k &= A\mathbf{p}_k \\
\alpha_k &= \frac{\delta_k}{\langle \mathbf{p}_k, \mathbf{s}_k \rangle} \\
\mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k \\
\mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k \mathbf{s}_k \\
\delta_{k+1} &= \langle \mathbf{r}_{k+1}, \mathbf{r}_{k+1} \rangle \\
\mathbf{p}_{k+1} &= \mathbf{r}_{k+1} + \frac{\delta_{k+1}}{\delta_k} \mathbf{p}_k \\
k &= k + 1.
\end{aligned}
$$

Krylov subspace

- Easy to see, for both methods, that

$$\mathbf{r}_k = p_k(A)\mathbf{r}_0,$$

  where p_k is a polynomial of degree k satisfying $p_k(0) = 1$.

- Also

$$\mathbf{e}_k = p_k(A)\mathbf{e}_0, \quad \mathbf{x}_k - \mathbf{x}_0 = q_{k-1}(A)\mathbf{r}_0.$$

- Define Krylov Subspace of nonsingular C with respect to $\mathbf{y}$ by

$$\mathcal{K}_k(C;\mathbf{y}) = \text{span}\{\mathbf{y}, C\mathbf{y}, C^2\mathbf{y}, \ldots, C^{k-1}\mathbf{y}\}.$$

- Thus,

$$\begin{aligned} \mathbf{r}_k &\in \mathcal{K}_{k+1}(A;\mathbf{r}_0), \quad \text{and} \\ \mathbf{x}_k - \mathbf{x}_0 &\in \mathcal{K}_k(A;\mathbf{r}_0). \end{aligned}$$

Energy norm

- For B a symmetric positive definite matrix,

$$\|\mathbf{z}\|_B = \sqrt{\mathbf{z}^T B\mathbf{z}} \equiv \sqrt{\langle \mathbf{z}, B\mathbf{z}\rangle}.$$

- Then

$$\|\mathbf{r}_k\|_{A^{-1}} = \|\mathbf{e}_k\|_A.$$

- Can interpret CG as minimizing in kth iteration the energy norm $\|\mathbf{e}_k\|_A$ over space $\mathbf{x}_0 + \mathcal{K}_k(A; \mathbf{r}_0)$.
- Hence, in exact arithmetic, exact solution is obtained after n iterations.

Energy norm

- For B a symmetric positive definite matrix,

$$\|\mathbf{z}\|_B = \sqrt{\mathbf{z}^T B\mathbf{z}} \equiv \sqrt{\langle \mathbf{z}, B\mathbf{z}\rangle}.$$

- Then

$$\|\mathbf{r}_k\|_{A^{-1}} = \|\mathbf{e}_k\|_A.$$

- Can interpret CG as minimizing in kth iteration the energy norm $\|\mathbf{e}_k\|_A$ over space $\mathbf{x}_0 + \mathcal{K}_k(A; \mathbf{r}_0)$.
- Hence, in exact arithmetic, exact solution is obtained after n iterations.

CG properties

- Subspace minimization (and uninteresting exact termination)
- Search directions are A-conjugate:

$$\langle \mathbf{p}_l, A\mathbf{p}_j \rangle = 0, \ l \neq j$$

- Residual directions are orthogonal:

$$\langle \mathbf{r}_l, \mathbf{r}_j \rangle = 0, \ l \neq j$$

- Key convergence rate estimate:

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

hence for large κ, number of iterations k for reducing initial error by factor c

$$k \leq .5\sqrt{\kappa(A)} \ln(2/c) + 1.$$

CG properties

- Subspace minimization (and uninteresting exact termination)
- Search directions are A-conjugate:

$$\langle \mathbf{p}_l, A\mathbf{p}_j \rangle = 0, \ l \neq j$$

- Residual directions are orthogonal:

$$\langle \mathbf{r}_l, \mathbf{r}_j \rangle = 0, \ l \neq j$$

- Key convergence rate estimate:

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

hence for large κ, number of iterations k for reducing initial error by factor c

$$k \leq .5\sqrt{\kappa(A)} \ln(2/c) + 1.$$

CG properties

- Subspace minimization (and uninteresting exact termination)
- Search directions are A-conjugate:

$$\langle \mathbf{p}_l, A\mathbf{p}_j \rangle = 0, \ l \neq j$$

- Residual directions are orthogonal:

$$\langle \mathbf{r}_l, \mathbf{r}_j \rangle = 0, \ l \neq j$$

- Key convergence rate estimate:

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

hence for large κ, number of iterations k for reducing initial error by factor c

$$k \leq .5\sqrt{\kappa(A)} \ln(2/c) + 1.$$

CG properties

- Subspace minimization (and uninteresting exact termination)
- Search directions are A-conjugate:

$$\langle \mathbf{p}_l, A\mathbf{p}_j \rangle = 0, \ l \neq j$$

- Residual directions are orthogonal:

$$\langle \mathbf{r}_l, \mathbf{r}_j \rangle = 0, \ l \neq j$$

- Key convergence rate estimate:

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

hence for large κ, number of iterations k for reducing initial error by factor c

$$k \leq .5\sqrt{\kappa(A)} \ln(2/c) + 1.$$

Steepest descent (SD) vs Conjugate gradients (CG)

- Gradient descent uses residual $\mathbf{r}_k$ as search direction; CG uses $\mathbf{p}_k = \mathbf{r}_k + \frac{\|\mathbf{r}_k\|^2}{\|\mathbf{r}_{k-1}\|^2}\mathbf{p}_{k-1}$.
- SD is simpler, greedy, one-step, easier to show convergence. CG is two-step (requires initialization), can be more fragile.
- SD requires $\mathcal{O}(\kappa(A))$ iterations whereas CG requires $\mathcal{O}(\sqrt{\kappa(A)})$ iterations!
- For both methods

$$\mathbf{r}_k \in \mathcal{K}_{k+1}(A;\mathbf{r}_0), \quad \mathbf{x}_k - \mathbf{x}_0 \in \mathcal{K}_k(A;\mathbf{r}_0).$$

- For CG also

$$\mathbf{x}_k = \arg\min_{\mathbf{y}\in S} \phi(\mathbf{y}) = \arg\min_{\mathbf{z}\in S} \|\mathbf{x}-\mathbf{z}\|_A$$

where $S = \mathbf{x}_0 + \mathcal{K}_k(A;\mathbf{r}_0)$. Thus, convergence (in exact arithmetic) in at most n iterations.

Steepest descent (SD) vs Conjugate gradients (CG)

- Gradient descent uses residual $\mathbf{r}_k$ as search direction; CG uses $\mathbf{p}_k = \mathbf{r}_k + \frac{\|\mathbf{r}_k\|^2}{\|\mathbf{r}_{k-1}\|^2}\mathbf{p}_{k-1}$.
- SD is simpler, greedy, one-step, easier to show convergence. CG is two-step (requires initialization), can be more fragile.
- SD requires $\mathcal{O}(\kappa(A))$ iterations whereas CG requires $\mathcal{O}(\sqrt{\kappa(A)})$ iterations!
- For both methods

$$\mathbf{r}_k \in \mathcal{K}_{k+1}(A;\mathbf{r}_0), \quad \mathbf{x}_k - \mathbf{x}_0 \in \mathcal{K}_k(A;\mathbf{r}_0).$$

- For CG also

$$\mathbf{x}_k = \arg\min_{\mathbf{y}\in S} \phi(\mathbf{y}) = \arg\min_{\mathbf{z}\in S} \|\mathbf{x}-\mathbf{z}\|_A$$

where $S = \mathbf{x}_0 + \mathcal{K}_k(A;\mathbf{r}_0)$. Thus, convergence (in exact arithmetic) in at most n iterations.

Steepest descent (SD) vs Conjugate gradients (CG)

- Gradient descent uses residual $\mathbf{r}_k$ as search direction; CG uses $\mathbf{p}_k = \mathbf{r}_k + \frac{\|\mathbf{r}_k\|^2}{\|\mathbf{r}_{k-1}\|^2}\mathbf{p}_{k-1}$.
- SD is simpler, greedy, one-step, easier to show convergence. CG is two-step (requires initialization), can be more fragile.
- SD requires $\mathcal{O}(\kappa(A))$ iterations whereas CG requires $\mathcal{O}(\sqrt{\kappa(A)})$ iterations!
- For both methods

$$\mathbf{r}_k \in \mathcal{K}_{k+1}(A;\mathbf{r}_0), \quad \mathbf{x}_k - \mathbf{x}_0 \in \mathcal{K}_k(A;\mathbf{r}_0).$$

- For CG also

$$\mathbf{x}_k = \arg\min_{\mathbf{y}\in S} \phi(\mathbf{y}) = \arg\min_{\mathbf{z}\in S} \|\mathbf{x}-\mathbf{z}\|_A$$

where $S = \mathbf{x}_0 + \mathcal{K}_k(A;\mathbf{r}_0)$. Thus, convergence (in exact arithmetic) in at most n iterations.

Steepest descent (SD) vs Conjugate gradients (CG)

- Gradient descent uses residual $\mathbf{r}_k$ as search direction; CG uses $\mathbf{p}_k = \mathbf{r}_k + \frac{\|\mathbf{r}_k\|^2}{\|\mathbf{r}_{k-1}\|^2}\mathbf{p}_{k-1}$.
- SD is simpler, greedy, one-step, easier to show convergence. CG is two-step (requires initialization), can be more fragile.
- SD requires $\mathcal{O}(\kappa(A))$ iterations whereas CG requires $\mathcal{O}(\sqrt{\kappa(A)})$ iterations!
- For both methods

$$\mathbf{r}_k \in \mathcal{K}_{k+1}(A; \mathbf{r}_0), \quad \mathbf{x}_k - \mathbf{x}_0 \in \mathcal{K}_k(A; \mathbf{r}_0).$$

- For CG also

$$\mathbf{x}_k = \arg\min_{\mathbf{y}\in S} \phi(\mathbf{y}) = \arg\min_{\mathbf{z}\in S} \|\mathbf{x} - \mathbf{z}\|_A$$

where $S = \mathbf{x}_0 + \mathcal{K}_k(A; \mathbf{r}_0)$. Thus, convergence (in exact arithmetic) in at most n iterations.

Steepest descent (SD) vs Conjugate gradients (CG)

- Gradient descent uses residual $\mathbf{r}_k$ as search direction; CG uses $\mathbf{p}_k = \mathbf{r}_k + \frac{\|\mathbf{r}_k\|^2}{\|\mathbf{r}_{k-1}\|^2}\mathbf{p}_{k-1}$.
- SD is simpler, greedy, one-step, easier to show convergence. CG is two-step (requires initialization), can be more fragile.
- SD requires $\mathcal{O}(\kappa(A))$ iterations whereas CG requires $\mathcal{O}(\sqrt{\kappa(A)})$ iterations!
- For both methods

$$\mathbf{r}_k \in \mathcal{K}_{k+1}(A; \mathbf{r}_0), \quad \mathbf{x}_k - \mathbf{x}_0 \in \mathcal{K}_k(A; \mathbf{r}_0).$$

- For CG also

$$\mathbf{x}_k = \arg\min_{\mathbf{y}\in S} \phi(\mathbf{y}) = \arg\min_{\mathbf{z}\in S} \|\mathbf{x} - \mathbf{z}\|_A$$

where $S = \mathbf{x}_0 + \mathcal{K}_k(A; \mathbf{r}_0)$. Thus, convergence (in exact arithmetic) in at most n iterations.

CG vs SOR

- When both methods may be applied optimally (i.e., A is s.p.d. for CG, and we know ω_{opt} for SOR) these methods require asymptotically a similar number of iterations.
- But CG uses only matrix-vector multiplications: more applicable.
- Moreover, there are ways to speed up CG by preconditioning, which are not available to SOR.
- Conclusion: continue to improve (and approve) CG.

Preconditioning

- Often, even $\mathcal{O}(\sqrt{\kappa(A)})$ iterations are too many!
  (In the Poisson example, $\sqrt{\kappa(A)} \sim N$.)
- Consider solving instead

$$(P^{-1/2}AP^{-1/2})(P^{1/2}\mathbf{x}) = P^{-1/2}\mathbf{b}$$

  with preconditioner P such that $P^{-1/2}AP^{-1/2}$ is better conditioned.
- Fortunately, can simply consider

$$P^{-1}A\mathbf{x} = P^{-1}\mathbf{b},$$

  and we want $B = P^{-1}A$ to be better conditioned than A.
- *So, the search is on for a preconditioner P that is both close enough to A to get a good condition number for B and far enough from A to be easily invertible.*

Preconditioning

- Often, even $\mathcal{O}(\sqrt{\kappa(A)})$ iterations are too many!
  (In the Poisson example, $\sqrt{\kappa(A)} \sim N$.)
- Consider solving instead

$$(P^{-1/2}AP^{-1/2})(P^{1/2}\mathbf{x}) = P^{-1/2}\mathbf{b}$$

  with preconditioner P such that $P^{-1/2}AP^{-1/2}$ is better conditioned.
- Fortunately, can simply consider

$$P^{-1}A\mathbf{x} = P^{-1}\mathbf{b},$$

  and we want $B = P^{-1}A$ to be better conditioned than A.
- *So, the search is on for a preconditioner P that is both close enough to A to get a good condition number for B and far enough from A to be easily invertible.*

Preconditioned conjugate gradients (PCG)

Given an initial guess $\mathbf{x}_0$ and a tolerance tol, set at first $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\mathbf{h}_0 = P^{-1}\mathbf{r}_0$, $\delta_0 = \langle \mathbf{r}_0, \mathbf{h}_0 \rangle$, $b_\delta = \langle \mathbf{b}, P^{-1}\mathbf{b} \rangle$, $k = 0$ and $\mathbf{p}_0 = \mathbf{h}_0$. Then:
While $\delta_k > tol^2 \; b_\delta$,

$$
\begin{aligned}
\mathbf{s}_k &= A\mathbf{p}_k \\
\alpha_k &= \frac{\delta_k}{\langle \mathbf{p}_k, \mathbf{s}_k \rangle} \\
\mathbf{x}_{k+1} &= \mathbf{x}_k + \alpha_k \mathbf{p}_k \\
\mathbf{r}_{k+1} &= \mathbf{r}_k - \alpha_k \mathbf{s}_k \\
\mathbf{h}_{k+1} &= P^{-1}\mathbf{r}_{k+1} \\
\delta_{k+1} &= \langle \mathbf{r}_{k+1}, \mathbf{h}_{k+1} \rangle \\
\mathbf{p}_{k+1} &= \mathbf{h}_{k+1} + \frac{\delta_{k+1}}{\delta_k} \mathbf{p}_k \\
k &= k + 1.
\end{aligned}
$$

Choices for preconditioner

- Symmetric SOR (SSOR)
- Incomplete Cholesky (IC) $P = FF^T$; more generally incomplete LU (ILU)
  - IC('0') – avoid fill-in altogether
  - IC(tol) – carry out elimination step only if result is above drop tolerance tol
- Often a good preconditioner is unknown in practice!

- Another limitation of CG is the requirement that A be s.p.d.
- Given a more general (nonsingular) matrix, naively we could solve the normal equations problem $A^T A\mathbf{x} = A^T\mathbf{b}$ where now $A^T A$ is s.p.d. However, the condition number, and hence the number of iterations, is squared this way!
- Opt instead for other Krylov subspace methods, considered next.

Choices for preconditioner

- Symmetric SOR (SSOR)
- Incomplete Cholesky (IC) $P = FF^T$; more generally incomplete LU (ILU)
  - IC('0') – avoid fill-in altogether
  - IC(tol) – carry out elimination step only if result is above drop tolerance tol
- Often a good preconditioner is unknown in practice!

- Another limitation of CG is the requirement that A be s.p.d.
- Given a more general (nonsingular) matrix, naively we could solve the normal equations problem $A^T A\mathbf{x} = A^T\mathbf{b}$ where now $A^T A$ is s.p.d. However, the condition number, and hence the number of iterations, is squared this way!
- Opt instead for other Krylov subspace methods, considered next.

Outline

- Stationary iteration and relaxation methods
- Application: model Poisson problem
- Convergence of stationary methods
- Conjugate gradient method
- *Krylov subspace methods
- *Multigrid methods

General nonsingular matrix

- Assume A nonsingular, but not necessarily symmetric positive definite.
- Extend CG by searching in similar Krylov subspaces: $\mathbf{x}_k - \mathbf{x}_0 \in \mathcal{K}_k(A; \mathbf{r}_0)$

$$\mathcal{K}_k(A; \mathbf{r}_0) = \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0\}.$$

- Building blocks
  1. Construct an orthogonal basis for the Krylov subspace.
  2. Define an optimality property.
  3. Use an effective preconditioner.

General nonsingular matrix

- Assume A nonsingular, but not necessarily symmetric positive definite.
- Extend CG by searching in similar Krylov subspaces: $\mathbf{x}_k - \mathbf{x}_0 \in \mathcal{K}_k(A; \mathbf{r}_0)$

$$\mathcal{K}_k(A; \mathbf{r}_0) = \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0\}.$$

- Building blocks
  1. Construct an orthogonal basis for the Krylov subspace.
  2. Define an optimality property.
  3. Use an effective preconditioner.

Orthogonal basis for the subspace: Arnoldi

- The obvious basis (powers $A^j \mathbf{r}_0$) is poorly conditioned.
- So orthonormalize these vectors: Arnoldi algorithm.

$$
\begin{aligned}
&\mathbf{q}_1 = \mathbf{r}_0/\|\mathbf{r}_0\| \\
&\text{for } j = 1 \text{ to } k \\
&\quad \mathbf{z} = A\mathbf{q}_j \\
&\quad \text{for } i = 1 \text{ to } j \\
&\quad\quad h_{i,j} = \langle \mathbf{q}_i, \mathbf{z} \rangle \\
&\quad\quad \mathbf{z} = \mathbf{z} - h_{i,j}\mathbf{q}_i \\
&\quad \text{end} \\
&\quad h_{j+1,j} = \|\mathbf{z}\| \\
&\quad \text{if } h_{j+1,j} = 0, \text{quit} \\
&\quad \mathbf{q}_{j+1} = \mathbf{z}/h_{j+1,j} \\
&\text{end}
\end{aligned}
$$

Orthogonal basis for the subspace: Arnoldi

- The obvious basis (powers $A^j \mathbf{r}_0$) is poorly conditioned.
- So orthonormalize these vectors: Arnoldi algorithm.

$$
\begin{aligned}
&\mathbf{q}_1 = \mathbf{r}_0 / \|\mathbf{r}_0\| \\
&\text{for } j = 1 \text{ to } k \\
&\quad \mathbf{z} = A\mathbf{q}_j \\
&\quad \text{for } i = 1 \text{ to } j \\
&\quad\quad h_{i,j} = \langle \mathbf{q}_i, \mathbf{z} \rangle \\
&\quad\quad \mathbf{z} = \mathbf{z} - h_{i,j}\mathbf{q}_i \\
&\quad \text{end} \\
&\quad h_{j+1,j} = \|\mathbf{z}\| \\
&\quad \text{if } h_{j+1,j} = 0, \text{quit} \\
&\quad \mathbf{q}_{j+1} = \mathbf{z} / h_{j+1,j} \\
&\text{end}
\end{aligned}
$$

Orthogonal basis in symmetric case: Lanczos

- If A is symmetric, the upper Hessenberg H is tridiagonal.
- Obtain three-term recurrence: Lanczos algorithm.

$$\begin{aligned}
\mathbf{z} &= A\mathbf{q}_j \\
\gamma_j &= \langle \mathbf{q}_j, \mathbf{z} \rangle \\
\mathbf{z} &= \mathbf{z} - \beta_{j-1}\mathbf{q}_{j-1} - \gamma_j \mathbf{q}_j \\
\beta_j &= \|\mathbf{z}\| \\
\mathbf{q}_{j+1} &= \mathbf{z}/\beta_j .
\end{aligned}$$

Start with $\mathbf{q}_1 = \mathbf{r}_0/\|\mathbf{r}_0\|,\ \beta_0 = 0$.

GMRES and MINRES

Minimize residual norm.

- Main components of GMRES iteration:
  1. perform a step of the Arnoldi process;
  2. update the QR factorization of the updated upper Hessenberg matrix;
  3. solve the resulting least squares problem.
- GMRES(m): limited memory GMRES – restart after m iterations
- MINRES: for the symmetric case: no memory problem.

GMRES and MINRES

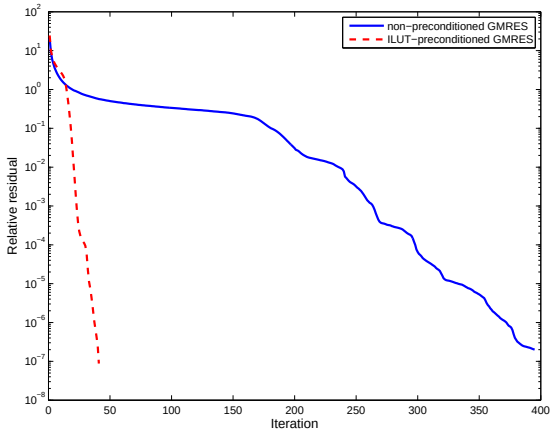
Minimize residual norm.

- Main components of GMRES iteration:
  1. perform a step of the Arnoldi process;
  2. update the QR factorization of the updated upper Hessenberg matrix;
  3. solve the resulting least squares problem.
- GMRES(m): limited memory GMRES – restart after m iterations
- MINRES: for the symmetric case: no memory problem.

Preconditioning

- For a general matrix A, a preconditioner is often more crucial than P in PCG for s.p.d.
- Incomplete LU (ILU)
- Incomplete LU with drop tolerance (ILUT)
- Example: convection-diffusion equation

$$-\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)+\sigma\frac{\partial u}{\partial x}+\tau\frac{\partial u}{\partial y}=g(x,y).$$



Outline

- Stationary iteration and relaxation methods
- Application: model Poisson problem
- Convergence of stationary methods
- Conjugate gradient method
- *Krylov subspace methods
- *Multigrid methods

Multigrid method

- Consider iteration for model problem: Poisson equation on the unit square.
- Highest frequencies of residual or error correspond to largest eigenvalues, most oscillatory eigenvectors. These are "not seen" on a coarser grid.
- Simple relaxations such as damped Jacobi ($\omega = .8$) or Gauss-Seidel are slow methods but fast smoothers (reduce high frequency components of residual/error).
- Smoothed residual can be transferred to a coarser grid, where procedure can be repeated, more economically.

Multigrid method cycle

function $\mathbf{x} = \texttt{multigrid}(A, \mathbf{b}, \mathbf{x}, level, \nu_1, \nu_2, \gamma)$

if level $=$ coarsest, solve exactly $A\mathbf{x} = \mathbf{b}$, *return*

for $j = 1 : \nu_1$, $\mathbf{x} = \texttt{relax}(A, \mathbf{b}, \mathbf{x})$, end

$\mathbf{r} = \mathbf{b} - A\mathbf{x}$

$[A_c, \mathbf{r}_c] = \texttt{restrict}(A, \mathbf{r})$

$\mathbf{v}_c = 0$

for $l = 1 : \gamma$

$\quad \mathbf{v}_c = \texttt{multigrid}(A_c, \mathbf{r}_c, \mathbf{v}_c, level - 1, \nu_1, \nu_2, \gamma)$

end

$\mathbf{x} = \mathbf{x} + \texttt{prolongate}(\mathbf{v}_c)$

for $j = 1 : \nu_2$, $\mathbf{x} = \texttt{relax}(A, \mathbf{b}, \mathbf{x})$, end

Multigrid performance

Can be used as a standalone or as a preconditioner for a Krylov subspace method.
The latter is more robust, and as such is generally preferred.
The figure below is for the model Poisson problem with $n = 255^2$