Statistics and Numerical Methods, Tsinghua University

Numerical Linear Algebra

Xuening Bai (白雪宁), Zhuo Chen (陈卓)

Institute for Advanced Study (IASTU) & Department of Astronomy (DoA)

清華大學

Tsinghua University

Oct. 8, 2024

Spectral radius

For a matrix $A \in \mathbb{R}^{n\times n}$, its spectral radius is defined as

$$\rho(A) \equiv \max\{|\lambda| : \lambda \in \lambda(A)\}$$ (largest absolute eigenvalue)

Theorem:

For any matrix norm $\|\cdot\|$, one has $\rho(A) \le \|A\|$, but for any ϵ >0, there exists a matrix norm so that $\|A\| \le \rho(A) + \epsilon$.

Gershgorin theorem: for a complex matrix A, the eigenvalues of each row locate in circles that centered at a_{ii} , with radii $\rho_i = \sum_{j\neq i} |a_{ij}|$.

Theorem: $\lim_{k\to\infty} A^k = 0 \iff \rho(A) < 1 \iff \sum_{k=0}^{\infty} A^k$ converges .

In this case, $\sum_{k=0}^{\infty} A^k = (I - A)^{-1}$.

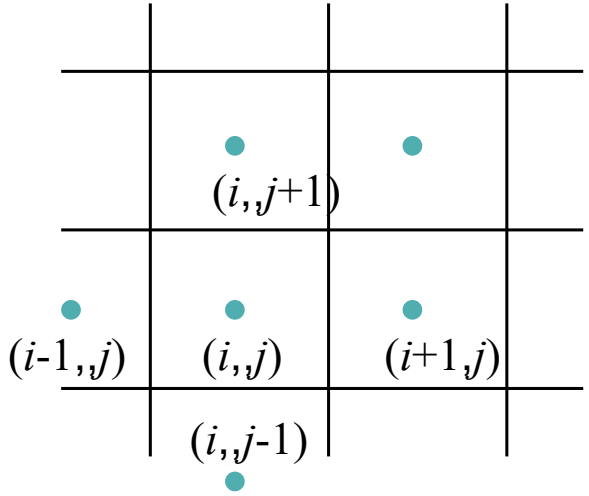
Outline

- Solving linear systems: direct methods
- Sensitivity analysis
- QR factorization
- Solving linear systems: classical iterative methods
- Solving linear systems: conjugate gradient methods

Sparse matrix: example

Let us consider solving the Poisson equation on an *N* by *N* grid:

$$\nabla^2 \Phi = 4\pi G\rho \ .$$



We can approximate the Laplace operator by finite difference, to obtain:

$$\frac{\Phi_{i+1,j} + \Phi_{i-1,j} + \Phi_{i,j+1} + \Phi_{i,j-1} - 4\Phi_{i,j}}{\Delta x^2} = 4\pi G\rho_{i,j}$$

Ax ***b***

There will be N^2 such equations for N^2 variables (including boundary conditions).

However, if written in the form of $Ax=b$, the matrix A would be $N^2 \times N^2$, with every row/column containing only 5 non-zero elements: this is extremely sparse!

Sparse matrix solvers

- Storage:

  In general, never write the full matrix, but only store non-zero elements and their indices, which depends on the exact form of the matrix.

- Direct solvers

  General Gaussian elimination / LU can not retain sparse matrix structures.

  Special strategies needed for certain form of sparse patterns (we do not cover this).

- Iterative solvers

  Give up to find exact solution, but develop iterative procedures that quickly converge towards the solution.

  There are a wide variety of methods far beyond the scope of this course. We will cover some most basic methods.

Basic iterative solvers

An iterative method is defined by $\boldsymbol{x}^{(k+1)} = \Psi(\boldsymbol{x}^{(k)})$.

The exact solution satisfies $\boldsymbol{x}^* = \Psi(\boldsymbol{x}^*)$, and we define the error as $\boldsymbol{e}^{(k)} = \boldsymbol{x}^{(k)} - \boldsymbol{x}^*$.

For linear system, there must be a matrix C such that $\boldsymbol{e}^{(k+1)} = C\boldsymbol{e}^{(k)}$.

Convergence requires $C^k \to 0 \ (\text{as } k \to \infty)$ <=> spectral radius $\rho(C) < 1$

Classical iterative solvers work by splitting matrix A as: $A = M - N$,
with the following format:

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

$$\Rightarrow \quad C = M^{-1}N = I - M^{-1}A$$

The question is given matrix type, find the right split to achieve fast convergence.

Classical iterative methods

Consider again $A\boldsymbol{x} = \boldsymbol{b}$ for square, non-singular matrix A.

Let $A \equiv D + L + U$

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}, \quad L = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}, \quad U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}.$$

$$\Rightarrow D\boldsymbol{x} = -(L+U)\boldsymbol{x} + \boldsymbol{b} \qquad \Rightarrow \boldsymbol{x} = -(D^{-1}L)\boldsymbol{x} - (D^{-1}U)\boldsymbol{x} + D^{-1}\boldsymbol{b}$$

Three classical methods that are built upon this splitting:

- Jacobi method
- Gauss-Seidel method
- Successive over-relaxation

(applicable mostly to diagonal-dominant matrices)

Jacobi method

Starting from $\boldsymbol{x} = -(D^{-1}L)\boldsymbol{x} - (D^{-1}U)\boldsymbol{x} + D^{-1}\boldsymbol{b}$

Iteration procedure: $\boldsymbol{x}^{(k+1)} = [-D^{-1}(L+U)]\boldsymbol{x}^{(k)} + D^{-1}\boldsymbol{b}$

Jacobi matrix

$$D^{-1}(L+U) = \begin{bmatrix} 0 & a_{12}/a_{11} & a_{13}/a_{11} & \dots & a_{1n}/a_{11} \\ a_{21}/a_{22} & 0 & a_{23}/a_{22} & \dots & a_{2n}/a_{22} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1}/a_{nn} & a_{n2}/a_{nn} & a_{n3}/a_{nn} & \dots & 0 \end{bmatrix}$$

$$\Rightarrow \quad x_i^{(k+1)} = \frac{1}{a_{ii}}\left[-\sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} + b_i \right]$$

L U

Gauss-Seidel (GS) method

Starting from $\boldsymbol{x} = -(D^{-1}L)\boldsymbol{x} - (D^{-1}U)\boldsymbol{x} + D^{-1}\boldsymbol{b}$

Iteration procedure: $\boldsymbol{x}^{(k+1)} = -(D^{-1}L)\boldsymbol{x}^{(k+1)} - (D^{-1}U)\boldsymbol{x}^{(k)} + D^{-1}\boldsymbol{b}$

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left[-\sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} + b_i \right]$$

L U

Taking the advantage that $x_j^{(k+1)}$ ($j<i$) is already available when computing $x_i^{(k+1)}$.

More formally, the iteration formula is

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

Successive over-relaxation (SOR)

Recall from GS: $\boldsymbol{x}^{(k+1)} = -(D^{-1}L)\boldsymbol{x}^{(k+1)} - (D^{-1}U)\boldsymbol{x}^{(k)} + D^{-1}\boldsymbol{b}$

Introducing a relaxation factor ω, and modify GS iteration into:

$$\boldsymbol{x}^{(k+1)} = (1-\omega)\boldsymbol{x}^{(k)} + \omega[-(D^{-1}L)\boldsymbol{x}^{(k+1)} - (D^{-1}U)\boldsymbol{x}^{(k)} + D^{-1}\boldsymbol{b}]$$

The corresponding procedure is:

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left[-\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)} + b_i\right]$$

More formally, the iteration formula is

$$\boldsymbol{x}^{(k+1)} = (D+\omega L)^{-1}[(1-\omega)D - \omega U]\boldsymbol{x}^{(k)} + \omega(D+\omega L)^{-1}\boldsymbol{b}$$

Why doing this: tuning ω to achieve faster convergence.

Convergence in these methods

Mathematically, all the above can be written in a form

$$x^{(k+1)} = Cx^{(k)} + M^{-1}b \qquad C = I - M^{-1}A$$

It is straightforward to show that the necessary and sufficient condition for convergence is

$$C^k \to 0 \quad (k \to \infty) \Rightarrow \rho(C) < 1$$ (spectral radius, i.e. largest eigenvalue)

Also note that spectral radius is a lower bound of matrix norms.

For instance, consider the Jacobi matrix, and compute its ∞-norm:

$$\|I - D^{-1}A\|_\infty = \max_i \sum_{j \neq i} |a_{ij}/a_{ii}|$$

We see that this is satisfied if A is strictly diagonally dominant: $|a_{ii}| > \sum_{j \neq i} |a_{i,j}|$.

Sufficient conditions for convergence

(but not necessary)

Jacobi method:

A is strictly diagonally dominant: $|a_{ii}| > \sum_{j \neq i} |a_{i,j}| \ .$

Gauss-Seidel method:

A is strictly diagonally dominant, or symmetric and positive definite.

SOR method:

More complex.

For A being symmetric and positive definite, convergence is guaranteed for $0 < \omega < 2$, with some optimal value in between that can be much faster than Jacobi and GS.

Outline

- Solving linear systems: direct methods
- Sensitivity analysis
- QR factorization
- Solving linear systems: classical iterative methods
- Solving linear systems: conjugate gradient methods

Conjugate gradient method

The classical iterative methods usually requires the spectral radius of the iteration matrix to be < 1. This is often too demanding.

The conjugate gradient (CG) method was developed since the 1950s, which is perhaps the most popular and important sparse matrix solver.

It only references A through matrix-vector multiplication.

The original CG method applies to symmetric, positive-definite matrices.

It can be considered as a prototype for the more general Krylov subspace methods, which can apply to any matrices, developed since 1980-1990s.

General philosophy

Consider $A\boldsymbol{x} = \boldsymbol{b}$ with A being real symmetric and positive definite.

Solving this equation is equivalent to minimizing:

$$\phi(\boldsymbol{x}) \equiv \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}$$

We can easily see $\nabla\phi$ gives the residual: $\nabla\phi(\boldsymbol{x}) = A\boldsymbol{x} - \boldsymbol{b} \equiv -\boldsymbol{r}$

If $\boldsymbol{x}^*$ is the solution, we have for any $\boldsymbol{y}$:

$$\phi(\boldsymbol{x}^* + \boldsymbol{y}) = \frac{1}{2}(\boldsymbol{x}^* + \boldsymbol{y})^T A(\boldsymbol{x}^* + \boldsymbol{y}) - \boldsymbol{b}^T(\boldsymbol{x}^* + \boldsymbol{y}) = \text{const} + \frac{1}{2}\boldsymbol{y}^T A\boldsymbol{y}$$

Basic idea:

Start from any point $\boldsymbol{x}_0$ and perform gradient descent towards minimum.
Given any line what is the point on the line that minimize phi?

Steepest descent

Assume a line in the direction of p, thus

$$f(x+\alpha p) = \frac{1}{2}(x+\alpha p)^T A(x+\alpha p) - (x+\alpha p)^T b$$
$$= \frac{1}{2}\alpha^2 p^T Ap + \alpha p^T(Ax-b) + c$$

The quadratic form has a minimum at $\alpha = \dfrac{p^T r}{p^T Ap}$

It is natural to take $\boldsymbol{p}$ in the direction of $\boldsymbol{r}$, because $r = Ax - b = \nabla\phi$

This yields: $\alpha = \dfrac{\boldsymbol{r}^T\boldsymbol{r}}{\boldsymbol{r}^T A\boldsymbol{r}}$

Steepest descent

So the overall procedure is:

1. Start with some initial value $\boldsymbol{x}_0$, and compute the residual $\boldsymbol{r}_0$.
2. Repeat the following until the residual $\boldsymbol{r}_k$ is sufficiently small:

$$\alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{r}_k^T A \boldsymbol{r}_k} \ , \quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha \boldsymbol{r}_k \ , \quad \boldsymbol{r}_{k+1} = \boldsymbol{b} - A\boldsymbol{x}_{k+1}$$

How fast does it converge?

Let us define $\|x\|_A \equiv \sqrt{\boldsymbol{x}^T A \boldsymbol{x}}$,
and it can be shown that:

$$\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_A \leq \left(\frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} \right)^k \|\boldsymbol{x}_0 - \boldsymbol{x}_*\|_A$$

max/min eigenvalues

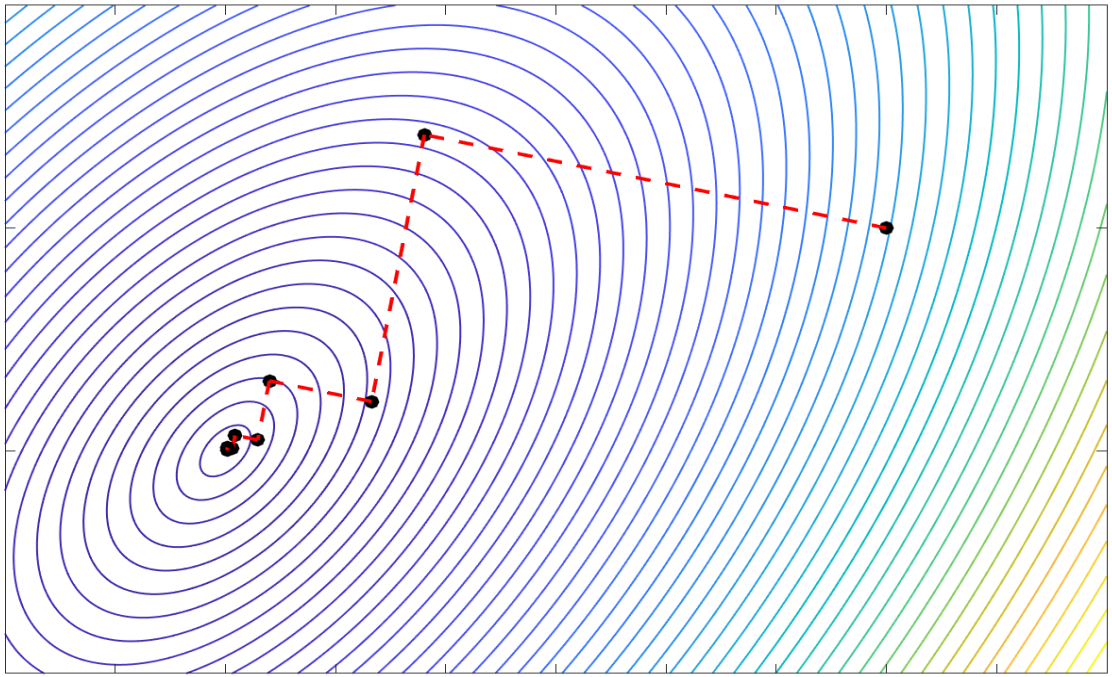
Convergence can be very slow for ill-conditioned matrix..

Why is steepest descent slow

The direction of descent is orthogonal to the previous direction:

$$\boldsymbol{r}_{k+1}^T \boldsymbol{r}_k = (\boldsymbol{r}_k - \alpha A \boldsymbol{r}_k)^T \boldsymbol{r}_k = 0$$

Therefore, iteration follows a zig-zag trajectory.



While every step is locally optimized, the strategy is not necessarily optimized from a global perspective.

Conjugate gradient method

Instead of choosing the locally steepest direction as the direction of gradient descent, consider the plane spanned by this steepest descent direction and the direction used in all previous step.

Choose the most optimal direction and length in this plane to minimize $\boldsymbol{\phi}$.

$$x_{k+1} = x_0 + \sum_k \alpha_k p_k$$

To minimize $\boldsymbol{\phi}$ is equivalent to

$$\phi = \frac{1}{2}(x_0 + \sum_{i=0}^{k} \alpha_i p_i)^T A(x_0 + \sum_{i=0}^{k} \alpha_i p_i) - (x_0 + \sum_{i=0}^{k} \alpha_i p_i)^T b$$

Impose A-orthogonal (conjugate orthogonal) $\quad p_i^T A p_j = 0$

$$\phi = \frac{1}{2} x_0^T A x_0 + \sum_k (\frac{1}{2}\alpha_i^2 p_i^T A p_i + \alpha_i p_i^T (A x_0 - b)) \qquad \text{How to find the } \alpha \text{ and p?}$$

Conjugate gradient method

Let p_{k-1} be the direction of gradient descent in the previous step, hence:

$$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \alpha_{k-1}\boldsymbol{p}_{k-1} \quad \text{and} \quad \boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k$$

Next, we need to determine what is

$$\boldsymbol{p}_k = \boldsymbol{r}_k + \beta_{k-1}\boldsymbol{p}_{k-1} \qquad \text{so that} \qquad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$$

Search dir. grad. descent dir. Previous search dir.

"Tridiagonalization"
See *Matrix computation* by *Golub & Van loan*.

Essentially, we need to determine β_{k-1} and α_k. to minimize $\phi(\boldsymbol{x}_{k+1})$.

The way to proceed is to consider ϕ as a function of α_k and β_{k-1} it is a matter of algebra to find the solution, which gives

$$\beta_{k-1} = -\frac{\boldsymbol{r}_k^T A \boldsymbol{p}_{k-1}}{\boldsymbol{p}_{k-1}^T A \boldsymbol{p}_{k-1}} \quad , \quad \alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{p}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$$

Conjugate gradient method

The algorithm is as follows:

Start with an arbitrary $\boldsymbol{x}_0$, with $\boldsymbol{r}_0=\boldsymbol{b}-A\boldsymbol{x}_0$.

Initially, choose $\boldsymbol{p}_0=\boldsymbol{r}_0$ as the initial direction of gradient descent.

Then loop over the following:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k \qquad \text{with} \quad \alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$$

$$\boldsymbol{r}_{k+1} = \boldsymbol{b} - A\boldsymbol{x}_{k+1} = \boldsymbol{r}_k - \alpha_k A \boldsymbol{p}_k$$

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k \qquad \text{with} \quad \beta_k = -\frac{\boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^T \boldsymbol{r}_k}$$

Note: the expressions are substantially better optimized so that matrix-vector product is evaluated only once.

Conjugate gradient method

For CG, one can verify show they satisfy the following

(1) $\boldsymbol{p}_i^T \boldsymbol{r}_j = 0 \ , \ 0 \le i < j \le k \ ;$

(2) $\boldsymbol{r}_i^T \boldsymbol{r}_j = 0 \ , \ 0 \le i, j \le k \ , \ i \ne j \ ;$

orthogonal bases

(3) $\boldsymbol{p}_i^T A \boldsymbol{p}_j = 0 \ , \ 0 \le i, j \le k \ , \ i \ne j \ ;$

conjugate orthogonal basis
(which is why it is named “conjugate gradient”)

(4) $\text{span}\{\boldsymbol{r}_0, \ \boldsymbol{r}_1, \ldots \boldsymbol{r}_k\} = \text{span}\{\boldsymbol{p}_0, \ \boldsymbol{p}_1, \ldots \boldsymbol{p}_k\} = \mathcal{K}_{k+1}(A, \boldsymbol{r}_0)$

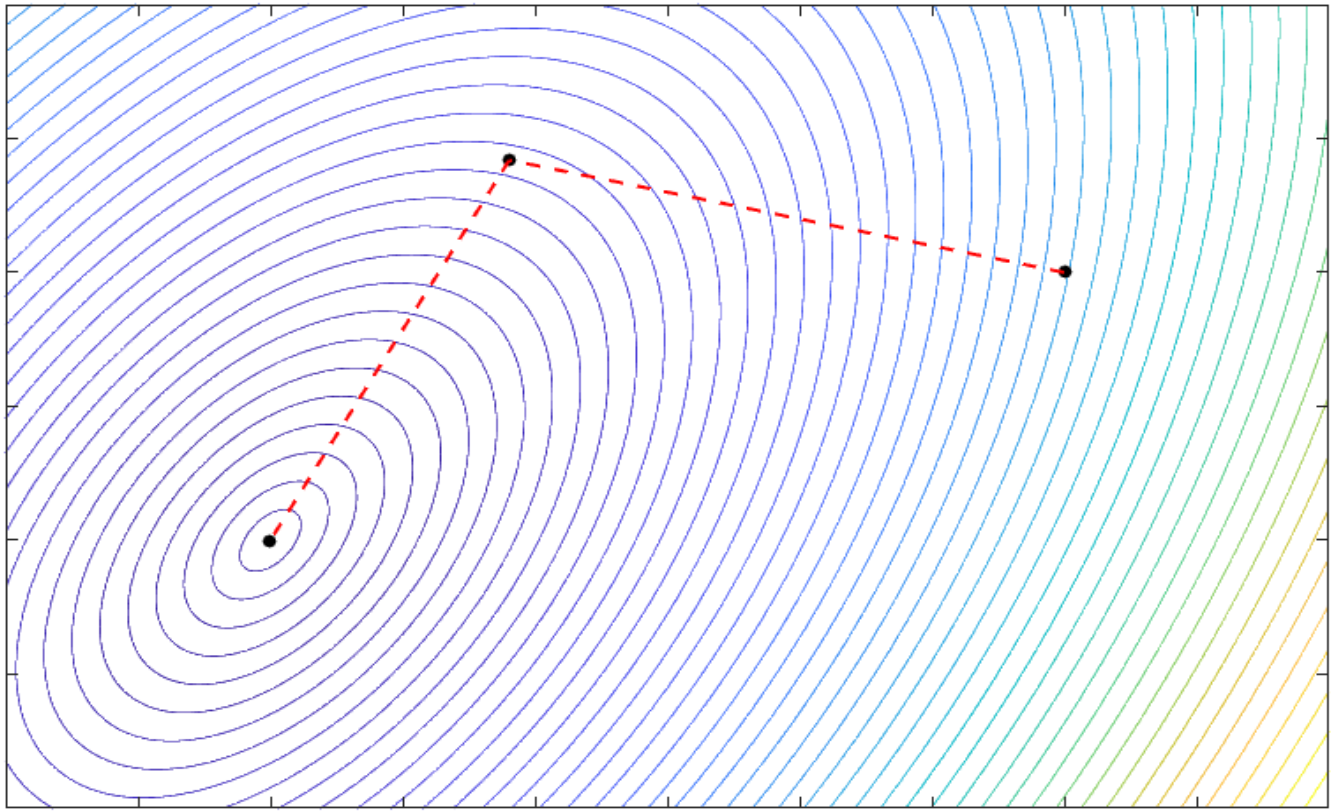
where $\mathcal{K}_m(A, \boldsymbol{r}_0) \equiv \text{span}\{\boldsymbol{r}_0, \ A\boldsymbol{r}_0, \ldots A^{m-1}\boldsymbol{r}_0\}$

This is called “Krylov subspace” (we will come back to this).

This suggests that CG method is guaranteed to converge to exact solution in finite (n) steps: it is actually a direct method!

Example

When solving lower-dimension (2) problem, CG quickly finds the exact solution.



Conjugate gradient method

In practice, it does not take long for the vector basis to lose orthogonality due to round-off errors.

Also, when N is large (e.g., 10^6), one cannot afford to wait for N iterations.

Therefore, CG still serves as an iterative method in practice.

One can show that the rate of convergence is

$$\|\boldsymbol{x}_k - \boldsymbol{x}_*\|_A \leq 2\left(\frac{\sqrt{\kappa_2}-1}{\sqrt{\kappa_2}+1}\right)^k \|\boldsymbol{x}_0 - \boldsymbol{x}_*\|_A \quad \text{where} \quad \|x\|_A \equiv \sqrt{\boldsymbol{x}^T A \boldsymbol{x}}$$

$$\kappa_2 \equiv \mathrm{cond}_2 A$$

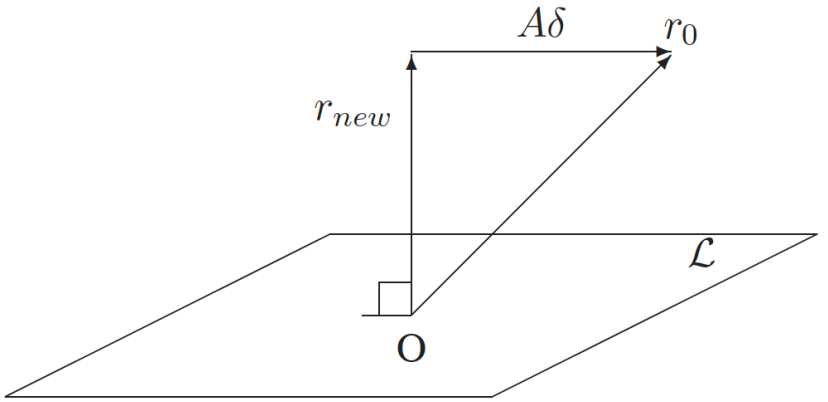
In practice, convergence is usually much faster.

Optional: projection methods

Let A be an nxn matrix and $\mathcal{K}, \mathcal{L}$ are m-dimensional subspaces of R^n.

In solving $A\boldsymbol{x}=\boldsymbol{b}$ starting from an initial guess $\boldsymbol{x}_0$, a projection technique is to

$$\text{Find } \tilde{\boldsymbol{x}} \in \boldsymbol{x}_0 + \mathcal{K}, \text{ such that } \boldsymbol{b} - A\tilde{\boldsymbol{x}} \perp \mathcal{L}$$



Two popular choices:

$\mathcal{L} = A\mathcal{K}$: residual projection methods

$$\|\tilde{\boldsymbol{r}}\|_2 \leq \|\boldsymbol{r}_0\|_2$$

$\mathcal{L} = \mathcal{K}$: error projection methods

$$\|\tilde{\boldsymbol{x}} - \boldsymbol{x}^*\|_A \leq \|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_A$$ for A being symmetric positive definite.

See book "*Iterative Methods for Sparse Linear Systems*" by Y. Saad

Previous iteration methods: a new look

Gauss-Seidel method:

$$\mathcal{K} = \mathcal{L} = \text{span}(\boldsymbol{e}_i)$$ (every iteration cycles through i=1,..,n)

Steepest descent:

$$\mathcal{K} = \mathcal{L} = \text{span}(\boldsymbol{r})$$ (one dimensional, m=1)

There is also the minimum residual method: (one dimensional, m=1)

$$\mathcal{K} = \text{span}(\boldsymbol{r}) \ , \ \mathcal{L} = \text{span}(A\boldsymbol{r})$$

It converges for positive definite matrix (no need to be symmetric).

CG method:

$$\mathcal{K} = \mathcal{L} = \mathcal{K}_m(A, \boldsymbol{r}_0) \equiv \text{span}\{\boldsymbol{r}_0, \ A\boldsymbol{r}_0, \ldots A^{m-1}\boldsymbol{r}_0\}$$

Tuned for symmetric positive definite matrix.

Krylov subspace methods

The CG method is the prototype of a much richer class of methods, known as Krylov subspace methods, which can be applied to general matrices.

Projection methods applied to Krylov subspaces.

$$\mathcal{K}_m(A, \boldsymbol{r}_0) \equiv \text{span}\{\boldsymbol{r}_0,\ A\boldsymbol{r}_0, \ldots A^{m-1}\boldsymbol{r}_0\}$$

General rationale:

Replace $A^{-1}\boldsymbol{b}$ by $p(A)\boldsymbol{b}$, where $p(A)$ is some "good" polynomial.

Many variations with deep mathematics, but roughly speaking, two broad classes:

$$\mathcal{K}_m = \mathcal{K}_m(A, \boldsymbol{r}_0)\ ,\ \mathcal{L}_m = \mathcal{K}_m \ \text{ or } \ \mathcal{L}_m = A\mathcal{K}_m$$

$$\mathcal{K}_m = \mathcal{K}_m(A, \boldsymbol{r}_0)\ ,\ \mathcal{L}_m = \mathcal{K}_m(A^T, \boldsymbol{r}_0)$$

GMRES and BICG

Two most commonly used methods for non-symmetric sparse matrices.

Generalized minimum residual method (GMRES):

$$\mathcal{K}_m = \mathcal{K}_m(A, \boldsymbol{r}_0) \ , \ \mathcal{L}_m = A\mathcal{K}_m$$

Use Arnoldi (Gram-Schmidt like) method (can also use Householder) to compute orthogonal basis of $\mathcal{K}_m$, and minimize the residual by solving a least square problem.

Bi-conjugate gradient method (BICG):

$$\mathcal{K}_m = \mathcal{K}_m(A, \boldsymbol{r}_0) \ , \ \mathcal{L}_m = \mathcal{K}_m(A^T, \boldsymbol{r}_0)$$

Use Lanczo bi-orthogonalization to compute the basis of $\mathcal{K}_m$ and $\mathcal{L}_m$ which are mutually orthogonal.

Reduces to CG method for symmetric, positive-definite matrix.

The procedure is not numerically stable => improvement by BICGSTAB

Preconditioning

Suppose you have a matrix P such that $PA \approx I$

Then solving the system: $PA\boldsymbol{x} = P\boldsymbol{b}$

would be much easier than solving $A\boldsymbol{x}=\boldsymbol{b}$ especially in ill-conditioned cases.

This procedure is called preconditioning, and P is called a preconditioner.

Preconditioning can also be done on the right side, or on both sides.

How to find P?

sharing the same sparsity pattern of A

Incomplete LU/Cholesky decomposition: $A = \tilde{L}\tilde{U} - R$

The preconditioner is set by: $P^{-1} = \tilde{L}\tilde{U}$

Jacobi/Gauss-Seidel/SOR can also be considered as preconditioners.

Generally the fastest solvers combine a preconditioner with an iterative solver.

Summary: iterative methods

Classical iterative methods:

- Based on splitting: Jacobi/Gauss-Seidel/SOR
- Convergence can be complex, but favors diagonal-dominant matrices.

Symmetric and positive-definite matrix:

- Steepest descent method: local strategy, slow convergence
- Conjugate gradient method (CG): "direct" method, faster convergence

More general framework: projection and the Krylov subspace methods

- Generalized minimum residual method (GMRES)
- Bi-conjugate gradient method stabilized (BICGSTAB)

Usually combine with preconditioning to improve convergence.