UW–Madison Math/CS 714

Methods of Computational Mathematics I

Iterative methods II

Instructor: Yue Sun (yue.sun@wisc.edu)

September 25, 2025

The A-conjugate search direction

The previous analysis suggests that the search direction can be very important in convergence. Rather than always choose $\nabla\phi$, we could instead try a different vector p_{k-1}, and update according to

$$u_k = u_{k-1} + \alpha_{k-1} p_{k-1}.$$

As before, we select α_{k-1} to minimize $\phi(u_{k-1} + \alpha p_{k-1})$. Similar to our previous analysis, this gives

$$\alpha_{k-1} = \frac{p_{k-1}^{\mathsf{T}} r_{k-1}}{p_{k-1}^{\mathsf{T}} A p_{k-1}}.$$

The A-conjugate search direction

Ideally, we would like to choose p_{k-1} to point in the direction of the solution, but this is infeasible.

However, in two dimensions, if we take an arbitrary initial guess u_0 and initial step direction p_0, then we can choose p_1 that leads directly to the solution!

The vector p_1 is chosen to be A-conjugate, so that

$$p_1^{\mathsf{T}} A p_0 = 0$$

If $A = I$ this would just imply orthogonality. Hence A-conjugacy generalizes the concept of orthogonality.

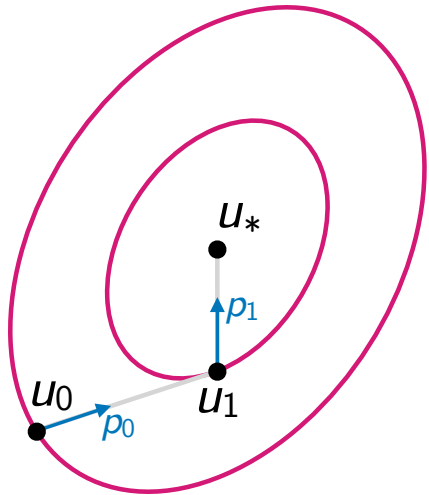
The A-conjugate search direction

To see that p_1 and p_0 are A-conjugate, note that p_0 will be tangent to the level set of ϕ at u_1, so

$$p_0^\mathsf{T} r_1 = p_0^\mathsf{T} A(u_* - u_1) = 0.$$

Since $u_* - u_1 = \alpha p_1$ for some $\alpha \neq 0$, it follows that

$$p_0^\mathsf{T} A p_1 = 0.$$



Generalization

We see that in this case

- $u_1 = \min_\alpha \phi(u_0 + \alpha p_0)$,
- $u_2 = \min_{\alpha,\beta} \phi(u_0 + \alpha p_0 + \beta p_1)$,

and since p_0 and p_1 are linearly independent, u_2 is the global minimizer, u_*.

Generalization

For m dimensions starting from u_0, we could aim to choose search directions $p_0, p_1, p_2, \ldots$ such that

- $u_1 = \min_\alpha \phi(u_0 + \alpha p_0)$,
- $u_2 = \min_{\alpha,\beta} \phi(u_0 + \alpha p_0 + \beta p_1)$,
- $u_3 = \min_{\alpha,\beta,\gamma} \phi(u_0 + \alpha p_0 + \beta p_1 + \gamma p_2)$,
- $\ldots$,

and if the p_j are linearly independent, then we would expect that $u_* = u_m$.

This is the basis of the conjugate gradient method, although it is not obvious how to select the directions p_k.

Conjugate gradient method

The conjugate gradient (CG) method is a famous algorithm in scientific computing. It was originally developed by Hestenes and Stiefel in 1952, although it took time before it was widely used and understood.

Building on the ideas presented, it provides an effective method for solving large SPD matrix systems.

Conjugate gradient method

The CG algorithm is given by

1: Choose initial guess $u_0 = 0$ and tolerance $\epsilon > 0$
2: $r_0 = f - Au_0, p_0 = r_0$
3: **for** $k = 1, 2, 3, \ldots$ **do**
4: $\quad w_{k-1} = Ap_{k-1}$
5: $\quad \alpha_{k-1} = (r_{k-1}^{\mathsf{T}} r_{k-1})/(p_{k-1}^{\mathsf{T}} w_{k-1})$
6: $\quad u_k = u_{k-1} + \alpha_{k-1} p_{k-1}$
7: $\quad r_k = r_{k-1} - \alpha_{k-1} w_{k-1}$
8: $\quad$ If $\|r_k\| < \epsilon$, then stop
9: $\quad \beta_{k-1} = (r_k^{\mathsf{T}} r_k)/(r_{k-1}^{\mathsf{T}} r_{k-1})$
10: $\quad p_k = r_k + \beta_{k-1} p_{k-1}$
11: **end for**

(Note that the initial guess could be arbitrary, but we focus on $u_0 = 0$ to simplify the convergence analysis.)

Conjugate gradient method

We shall now discuss CG in more detail — it's certainly not obvious why this works!

Let $u_* = A^{-1}f$ denote the exact solution, and let $e_k \equiv u_* - u_k$ denote the error at step k.

Also, let $\|\cdot\|_A$ denote the norm

$$\|u\|_A \equiv \sqrt{u^\mathsf{T} A u}.$$

Krylov subspaces

The only way that we obtain information about the matrix A is via multiplication. This motivates the definition of a Krylov sequence, which for a given vector b is

$$\{b, Ab, A^2b, A^3b, \ldots\}.$$

The corresponding Krylov subspaces are the spaces spanned by successive groups of these vectors:

$$\mathcal{K}_m(A, b) \equiv \text{span}\{b, Ab, A^2b, \ldots, A^{m-1}b\}.$$

Conjugate gradient method

Theorem: The CG iterate u_k is the unique member of $\mathcal{K}_k(A, f)$ that minimizes $\|e_k\|_A$. Also, $u_k = u_*$ for some $k \leq n$.

Proof: This result relies on a set of identities which can be derived (by induction) from the CG algorithm:

(i) $\mathcal{K}_k(A, f) = \text{span}\{u_1, u_2, \ldots, u_k\} = \text{span}\{p_0, p_1 \ldots, p_{k-1}\}$
$= \text{span}\{r_0, r_1, \ldots, r_{k-1}\}$

(ii) $r_k^{\mathsf{T}} r_j = 0$ for $j < k$

(iii) $p_k^{\mathsf{T}} A p_j = 0$ for $j < k$

Conjugate gradient method

From the first identity above, it follows that $u_k \in \mathcal{K}_k(A, f)$.

We will now show that u_k is the unique minimizer in $\mathcal{K}_k(A, f)$.

Let $\tilde{u} \in \mathcal{K}_k(A, f)$ be another "candidate minimizer" and let $\Delta u \equiv u_k - \tilde{u}$, then

$$\begin{aligned} \|u_* - \tilde{u}\|_A^2 &= \|(u_* - u_k) + (u_k - \tilde{u})\|_A^2 \\ &= \|e_k + \Delta u\|_A^2 \\ &= (e_k + \Delta u)^\mathsf{T} A (e_k + \Delta u) \\ &= e_k^\mathsf{T} A e_k + 2 e_k^\mathsf{T} A \Delta u + \Delta u^\mathsf{T} A \Delta u \end{aligned}$$

Conjugate gradient method

Next, let $r(u_k) = f - Au_k$ denote the residual at step k, so that

$$r(u_k) = f - Au_k = f - A(u_{k-1} + \alpha_k p_{k-1}) = r(u_{k-1}) - \alpha_k A p_{k-1}$$

Since $r(u_0) = f = r_0$, by induction we see that for r_k computed in line 7 of CG,

$$r_k = r_{k-1} - \alpha_k A p_{k-1}$$

we have $r_k = r(u_k)$, $k = 1, 2, \ldots$

Conjugate gradient method

Now, recall our expression for $\|u_* - \tilde{u}\|_A^2$:

$$\|u_* - \tilde{u}\|_A^2 = e_k^\mathsf{T} A e_k + 2e_k^\mathsf{T} A \Delta u + \Delta u^\mathsf{T} A \Delta u$$

and note that

$$2e_k^\mathsf{T} A \Delta u = 2\Delta u^\mathsf{T} A(u_* - u_k) = 2\Delta u^\mathsf{T}(f - Au_k) = 2\Delta u^\mathsf{T} r_k$$

Now,

- $\Delta u = u_k - \tilde{u} \in \mathcal{K}_k(A, f)$
- from (i), we have that $\mathcal{K}_k(A, f) = \text{span}\{r_0, r_1, \ldots, r_{k-1}\}$
- from (ii), we have that $r_k \perp \text{span}\{r_0, r_1, \ldots, r_{k-1}\}$

Therefore, we have $2e_k^\mathsf{T} A \Delta u = 2\Delta u^\mathsf{T} r_k = 0$

Conjugate gradient method

This implies that,

$$\|u_* - \tilde{u}\|_A^2 = e_k^\mathsf{T} A e_k + \Delta u^\mathsf{T} A \Delta u \geq \|e_k\|_A^2,$$

with equality only when $\Delta u = 0$, hence $u_k \in \mathcal{K}_k(A, f)$ is the unique minimizer!

This also tells us that if $u_* \in \mathcal{K}_k(A, f)$, then $u_k = u_*$

Therefore[1] CG will converge to u_* in at most n iterations since $\mathcal{K}_k(A, f)$ is a subspace of $\mathbb{R}^n$ of dimension k $\square$

[1] Assuming exact arithmetic!

Conjugate gradient method

Note that the theoretical guarantee that CG will converge in n steps is of no practical use.

In floating point arithmetic we will not get exact convergence to u_*.

More importantly, we assume n is huge, so we want to terminate CG well before n iterations anyway.

Nevertheless, the guarantee of convergence in at most n steps is of historical interest.

Hestenes and Stiefel originally viewed CG as a direct method that will converge after a finite number of steps.

Simple conjugate gradient example

The program *simple_cg.py* uses CG to solve the one-dimensional Poisson equation for $u(x)$,

$$\frac{\partial^2 u}{\partial x^2} = f$$

on the interval $[0, 1]$, with Dirichlet conditions $u(0) = u(1) = 0$.

Discretize as $u_j = u(jh), f_j = f(jh)$ where $h = \frac{1}{n+1}$. Hence $u_0 = u_{n+1} = 0$ and

$$\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} = f_j$$

for $j = 1, \ldots, n$.

Conjugate gradient method

We now consider the convergence of the CG method. A famous result for CG is that if A has 2-norm condition number κ, then

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k$$

Hence smaller condition number implies faster convergence.

Conjugate gradient method

Suppose we want to terminate CG when

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq \epsilon$$

for some $\epsilon > 0$, how many CG iterations will this require?

We have the identities

$$\begin{aligned} 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k &= 2\left(\frac{\sqrt{\kappa}+(1-1)-1}{\sqrt{\kappa}+1}\right)^k \\ &= 2\left(1-\frac{2}{\sqrt{\kappa}+1}\right)^k \\ &= 2\left(1-\frac{2/\sqrt{\kappa}}{1+1/\sqrt{\kappa}}\right)^k. \end{aligned}$$

Conjugate gradient method

For large κ it follows that

$$\frac{\|e_k\|_A}{\|e_0\|_A} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \approx 2\left(1-\frac{2}{\sqrt{\kappa}}\right)^k.$$

Hence we terminate CG when

$$\left(1-\frac{2}{\sqrt{\kappa}}\right)^k \approx \frac{\epsilon}{2}.$$

Conjugate gradient method

Taking logs gives

$$k \approx \log(\epsilon/2)/\log\left(1 - \frac{2}{\sqrt{\kappa}}\right) \approx \frac{1}{2}|\log(\epsilon/2)|\sqrt{\kappa}.$$

where the last expression follows from the Taylor expansion:

$$\log\left(1 - \frac{2}{\sqrt{\kappa}}\right) \approx \log(1) - \frac{2}{\sqrt{\kappa}} = -\frac{2}{\sqrt{\kappa}}.$$

This analysis shows that the number of CG iterations for a given tolerance ϵ grows approximately as $\sqrt{\kappa}$

Conjugate gradient method

For the discrete Laplacian, we have $\kappa = O(h^{-2})$, hence number of CG iterations should grow as $O(\sqrt{\kappa}) = O(h^{-1})$

For $\epsilon = 10^{-4}$, with the program *poisson_cg.py*, we obtain the following convergence results

h	κ	CG iterations
4×10^{-2}	3.67×10^{2}	32
2×10^{-2}	1.47×10^{3}	65
1×10^{-2}	5.89×10^{3}	133
5×10^{-3}	2.36×10^{4}	272

Conjugate gradient method

These results indicate that CG gets more expensive for Poisson equation as h is reduced for two reasons:

- The matrix and vectors get larger, hence each CG iteration is more expensive.
- We require more iterations since the condition number gets larger.

Convergence

Convergence of the conjugate gradient method is better when the matrix A has a small condition number

A way to improve convergence is to use preconditioning. We find a matrix M that is an approximation to A, and solve $M^{-1}Ax = M^{-1}b$. We want

- M is symmetric and positive definite
- $M^{-1}A$ is well conditioned and has few extreme eigenvalues
- $Mx = b$ is easy to solve

Preconditioned conjugate gradient method

The preconditioned CG algorithm is given by

1: Choose tolerance $\epsilon > 0$
2: $u_0 = 0$, $r_0 = f$, $p_1 = M^{-1}f$, $y_0 = M^{-1}r_0$
3: **for** $k = 1, 2, 3, \ldots$ **do**
4: $\quad z = Ap_k$
5: $\quad \nu_k = (y_{k-1}^{\mathsf{T}} r_{k-1})/(p_k^{\mathsf{T}} z)$
6: $\quad u_k = u_{k-1} + \nu_k p_k$
7: $\quad r_k = r_{k-1} - \nu_k z$
8: $\quad$ If $\|r_k\| < \epsilon$, then stop
9: $\quad y_k = M^{-1} r_k$
10: $\quad \mu_k = (y_k^{\mathsf{T}} r_k)/(y_{k-1}^{\mathsf{T}} r_{k-1})$
11: $\quad p_{k+1} = y_k + \mu_k p_k$
12: **end for**

Examples of preconditioning

- Diagonal (Jacobi) preconditioning: define $M = \mathrm{diag}(a_{11}, a_{22}, \ldots, a_{nn})$. Straightforward to invert.
- Block Jacobi preconditioning: Write the matrix in block form as

$$A = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1k} \\ \vdots & \vdots & \ddots & \vdots \\ A_{k1} & A_{k2} & \cdots & A_{kk} \end{pmatrix}$$

Define

$$M = \begin{pmatrix} A_{11} & & & \\ & \ddots & & \\ & & & A_{kk} \end{pmatrix}$$

Performing M^{-1} requires inverting each block—much faster than solving the original matrix

Examples of preconditioning

- Incomplete LU/Cholesky factorization: a full LU or Cholesky factorization of a sparse matrix results in fill-in of the zero entries. Adjust algorithm to obtain approximate result with minimum fill-in.
- Multigrid: the multigrid algorithm is an iterative procedure for solving matrix problems, by applying successive V-cycles. Let M^{-1} be the matrix applying one V-cycle—good approximation to the inverse of A.

Krylov subspace methods

We saw that the properties of CG are closely connected to the Krylov subspace $\mathcal{K}_k(A, f)$, and is therefore called a Krylov subspace method.

There are many other examples of Krylov subspace methods, which work via evaluating matrix–vector multiplications only:

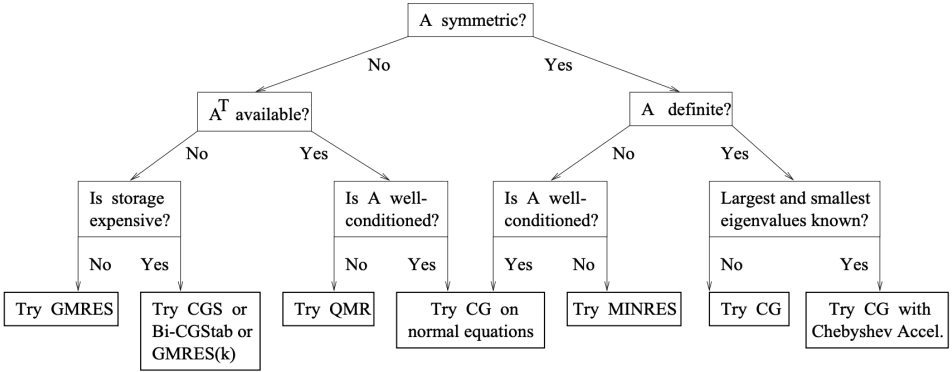
- Arnoldi iteration
- Lanczos iteration
- Generalized minimum residual (GMRES) method
- Conjugate residual method
- Biconjugate gradient stabilized (BiCGSTAB) method

Some of these methods are no longer restricted to SPD matrices; we can use Krylov methods to solve arbitrary linear systems $Au = f$.[2]

[2] See *Applied Numerical Linear Algebra* by James Demmel for more information.

Krylov subspace methods

There are many related algorithms for solving different types of linear systems. The following flow chart from the textbook[3] illustrates some of the different possibilities.



[3] J. A. Demmel, *Applied Numerical Linear Algebra*, SIAM 1997.

Other methods for solving elliptic PDEs

There are many other methods for solving elliptic PDEs, including

- **Fast direct solvers**[4]**:** Exploit hierarchical low-rank structure to accelerate sparse PDE matrices or dense boundary integral systems
  - Recursive skeletonization, HIF-DE, HSS/HODLR methods
  - Often accelerated by tools like the fast multipole method (FMM) or randomized linear algebra
- **Domain decomposition methods:** Decompose the domain into subdomains, solve on each subdomain, and iterate or couple with coarse solves
- **Boundary integral methods**[5]**:** Recast the PDE as an integral equation on the boundary; leads to dense systems often accelerated by FMM or hierarchical solvers

[4] https://fastalgorithms.github.io/

[5] https://users.flatironinstitute.org/~ahb/BIE/