

Lecture 08 - C

§8.4 Gradient Methods

T. Takahashi

Department of Mathematics and Statistics
University of New Mexico



We will answer these questions:

- What are gradient methods?
- What are some drawbacks of gradient descent?
- What is conjugate gradient?
- What makes conjugate gradient such a powerful algorithm?



Review: Iterative Methods



Iterative Methods

- We have used iterations to solve linear systems:

Jacobi and Gauss-Seidel

- We have also used iterations to solve nonlinear equations:

Bisection and Newton's method

Common Idea

Start with an approximation $x^{(0)}$ and update it:

$$x^{(k)} \longrightarrow x^{(k+1)}.$$



Why Is Newton Faster?

Newton's Method

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}.$$

- Bisection utilizes the continuity of the function to converge
- Newton uses the **derivative** to converge even faster



Why Is Newton Faster?

Newton's Method

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}.$$

- Bisection utilizes the continuity of the function to converge
- Newton uses the **derivative** to converge even faster

Question

How can we create a method that converges faster than Jacobi and GS?



Using Derivatives in Several Variables

- In one variable, we use the derivative:

$$f'(x).$$

- In several variables, we use the **gradient**:

$$\nabla\phi(x).$$

Question

How can we use the gradient to help solve $Ax = b$?



Using Derivatives in Several Variables

- In one variable, we use the derivative:

$$f'(x).$$

- In several variables, we use the **gradient**:

$$\nabla\phi(x).$$

Question

How can we use the gradient to help solve $Ax = b$?

Key Idea

Rewrite $Ax = b$ as a minimization problem.

Gradient Methods



The Gradient

Definition: Gradient

The **gradient** of a function $f(x)$ for $x \in \mathbb{R}^n$ is

$$\nabla f(x) = [\partial_1 f \quad \partial_2 f \quad \dots \quad \partial_n f]^T$$

The **gradient operator** ∇ is a differential operator that returns an $n \times 1$ vector.

Two Main Properties

- 1 When $\nabla f(x) \neq 0$, it points in the direction of **steepest ascent**.

$-\nabla f(x)$: direction of steepest descent.

- 2 At a differentiable local minimum,

$$\nabla f(x) = 0.$$

The vector x are critical points.

The $\phi(x)$ function

Definition: Objective function

We define the **objective function** or **phi function** to be

$$\phi(x) = \frac{1}{2}x^T Ax - x^T b,$$

where x and b are $n \times 1$ vectors, and A is a $n \times n$ SPD matrix.

- For $n = 1$: an upward-opening parabola.
- For $n = 2$: a bowl-shaped surface.
- SPD means $x^T Ax > 0$ for every $x \neq 0$.

Main Property

For SPD A , this quadratic is strictly convex and has a unique absolute minimum.



Example in 2D

Objective Function

This is the plot of $\phi(x) = \frac{1}{2}x^T A x - x^T b$, with

$$A = \begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

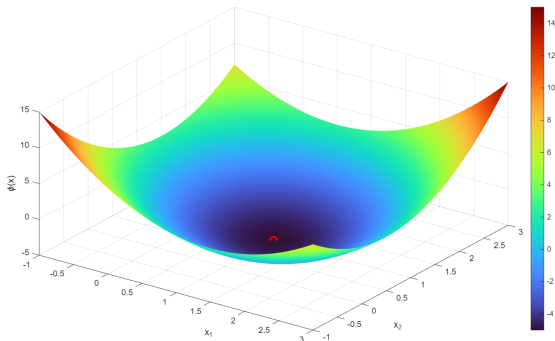


Figure: This is ϕ in 2D. It has a bowl shape. The minimum is circled in red.

Local Minimum = Absolute Minimum

- Consider a simple 1D example of the $\phi(x) = \frac{1}{2}x^T Ax - x^T b$,

$$\phi(x) = 4x^2 - 2x$$

- Since $\phi''(x) = 8 > 0$, its critical point is the unique absolute minimum.
- We can find the local minimum calculus:

$$\begin{aligned}\phi'(x) &= 8x - 2 \equiv 0 \\ \implies x &= 1/4\end{aligned}$$



Local Minimum = Absolute Minimum

- Consider a simple 1D example of the $\phi(x) = \frac{1}{2}x^T Ax - x^T b, :$

$$\phi(x) = 4x^2 - 2x$$

- Since $\phi''(x) = 8 > 0$, its critical point is the unique absolute minimum.
- We can find the local minimum calculus:

$$\begin{aligned}\phi'(x) &= 8x - 2 \equiv 0 \\ \implies x &= 1/4\end{aligned}$$

Question

How can you find the minimum in higher dimensions?



Gradient of $\phi(x)$

- Since $x^T b = x_1 b_1 + \cdots + x_n b_n$,

$$\nabla(x^T b) = [b_1 \quad \cdots \quad b_n]^T = b.$$

- For $x^T A x$, applying the product rule:

$$\nabla(x^T A x) = (A + A^T)x = 2Ax \quad \text{since } A = A^T.$$

- Therefore the gradient is

$$\nabla\phi(x) = \nabla\left(\frac{1}{2}x^T A x - x^T b\right) = Ax - b$$

Derivative of ϕ

In higher dimensions, the derivative of $\phi(x)$ is

$$\nabla\phi(x) = Ax - b.$$

Absolute Minimum of $\phi(x)$

In 1D

- Calculate $\phi'(x)$
- Set $\phi'(x) = 0$
- Solve for x

In $\mathbb{R}^n$

- Calculate the gradient: $\nabla\phi(x) = Ax - b$
- Set it equal to zero: $Ax - b = 0$
- Then solve for x : $x = A^{-1}b$

Question

What do you notice about the minimizer of $\phi(x)$?

Equivalent Problems

Problem A

Solve the system $Ax = b$ where A is SPD.

$\updownarrow$ reformulation

Problem B

Find the absolute minimum of $\phi(x) = \frac{1}{2}x^T Ax - x^T b$ where A is SPD.

- Problem B reformulates Problem A as an **optimization problem**.
- **Gradient methods** can solve problem B.

Key Idea

For SPD A , solving $Ax = b$ is equivalent to minimizing $\phi(x)$.



Gradient Methods

Definition: Gradient Methods

Use the gradient to construct successive approximations to a minimizer of $\phi(x)$.

Update

$$x^{(k+1)} = x^{(k)} + \alpha_k s^{(k)}.$$

- α_k = step size
- $s^{(k)}$ = search direction
- Choose a direction and step size to decrease ϕ .
- For our quadratic, $\nabla\phi(x^{(k)}) = -r^{(k)}$.

Stopping Condition

$$\|r^{(k)}\| < \delta.$$

Gradient Descent

- Follow the direction of **steepest descent**:

$$s^{(k)} = -\nabla\phi(x^{(k)}) = r^{(k)}.$$

Gradient Descent Update

$$x^{(k+1)} = x^{(k)} + \alpha r^{(k)}, \quad r^{(k)} = b - Ax^{(k)}.$$

- Choose a fixed step size $\alpha > 0$.
- The step size must be small enough for convergence.

Method: Gradient Descent

- Given:** SPD matrix A , RHS b , step α , tolerance
- Output:** Approximate solution to $Ax = b$
- Idea:** Move in the direction of steepest descent.

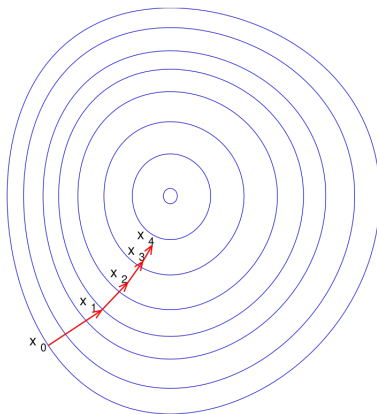
Gradient Descent



Gradient descent is like a blind person trying to reach the bottom of a pit. At each step they feel the slope and move in the steepest downhill direction.



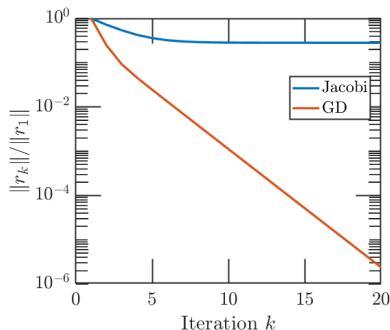
Gradient Descent Visualized



- This is a simple example for 2×2 matrix
- blue curves = contour map for $\phi(x)$
- The minimum is at the center of the innermost circle
- red arrows = vectors indicating $-\nabla\phi(x_k)$



Performance



Pros:

- Very easy to code
- One matrix-vector product per iteration.

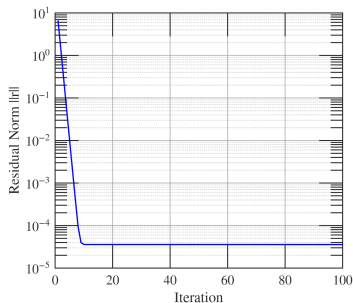
```
1 alpha = 0.8/max(eigs(A));  
2 rnorms = zeros(N,1);  
3 for i = 1:N  
4     r = b - A*x;           % residue  
5     rnorms(i) = norm(r);   % compute norm  
6     x = x + alpha*r;       % update  
7 end
```

Not the Best Method

- Consider the matrix

$$A = \begin{bmatrix} 4 & 2 \\ 2 & 1.001 \end{bmatrix}$$

- This matrix is SPD, but $\text{cond}(A) = 6.2501\text{e}+03$



- The error cannot achieve machine precision



Issues with Gradient Descent

- **Step size:** too small is slow; too large can diverge.
- **Ill-conditioning:** elongated contours slow progress.
- **Zigzagging:** downhill steps can repeatedly cross the valley.

Question

Can we choose a better step size and search direction?



Conjugate Gradient



Step Size

- One issue with the Gradient Descent is that α is fixed

$$x^{(k+1)} = x^{(k)} + \underbrace{\alpha}_{\text{fixed}} s^{(k)}$$

- It's suboptimal
- Annoying to tune
- It would be better to make α dynamic:

$$\alpha \rightarrow \alpha_k$$



Improving α

Choosing the Best Step

Goal

Given the search direction $s^{(k)}$, choose the stepsize α_k optimally in

$$x^{(k+1)} = x^{(k)} + \alpha_k s^{(k)}.$$

1. Along the direction $s^{(k)}$, define

$$g(\alpha) = \phi\left(x^{(k)} + \alpha s^{(k)}\right).$$

2. Choose α to minimize this one-variable function:

$$\alpha_k = \arg \min_{\alpha} g(\alpha).$$

3. Therefore, at the optimal stepsize,

$$g'(\alpha_k) = 0.$$



Improving α

The Optimality Condition

4. Using the chain rule,

$$\begin{aligned} 0 &= g'(\alpha) \\ &= \nabla \phi \left(x^{(k)} + \alpha s^{(k)} \right)^T \frac{d}{d\alpha} \left(x^{(k)} + \alpha s^{(k)} \right) \\ &= \nabla \phi \left(x^{(k+1)} \right)^T s^{(k)}. \end{aligned}$$

5. Recall that the gradient and residual are related

$$\nabla \phi(x) = Ax - b = -r.$$

6. Therefore,

$$0 = -(r^{(k+1)})^T s^{(k)},$$

or equivalently,

$$\boxed{(r^{(k+1)})^T s^{(k)} = 0.}$$

Key Idea

At the optimal stepsize, the new residual is orthogonal to the search direction.

Improving α

Solving for the Stepsize

7. Rewrite $r^{(k+1)}$ in terms of $r^{(k)}$

$$\begin{aligned}r^{(k+1)} &= b - Ax^{(k+1)} \\&= b - A \left(x^{(k)} + \alpha s^{(k)} \right) \\&= r^{(k)} - \alpha As^{(k)}.\end{aligned}$$

8. Substitute this into

$$(r^{(k+1)})^T s^{(k)} = 0 :$$

and then solve for α .

$$\begin{aligned}0 &= \left(r^{(k)} - \alpha As^{(k)} \right)^T s^{(k)} \\&= (r^{(k)})^T s^{(k)} - \alpha (s^{(k)})^T As^{(k)}.\end{aligned}$$

Thus,

$$\boxed{\alpha_k = \frac{(r^{(k)})^T s^{(k)}}{(s^{(k)})^T As^{(k)}}}.$$



Search Direction

- Gradient Descent with variable step size

$$x^{(k+1)} = x^{(k)} + \alpha_k s^{(k)}$$

is still not good enough

- It cannot prevent zig-zagging

Goal

Derive a formula for search direction $s^{(k)}$ in the gradient method

$$x^{(k+1)} = x^{(k)} + \alpha_k s^{(k)}$$

which eliminates the zigzagging behavior in GD.



How to improve $s^{(k)}$: Orthogonal Directions

- **Key Issue:**

- Redundant search directions
- New directions are nearly parallel to old directions

- **Fix:** Force new directions to be *perpendicular* to previous directions

Definition: Orthogonality

- Two vectors are **orthogonal** if $u^T v = 0$.
- Two vectors are **A-orthogonal (A-conjugate)** if $u^T A v = 0$ where A is $n \times n$.



Improving the Search Direction

- Start with the steepest descent direction:

$$s^{(0)} = r^{(0)}.$$

- At the next iteration, include the previous direction.

New Search Direction

$$s^{(k+1)} = r^{(k+1)} + \beta_k s^{(k)}.$$

- Choose β_k so the directions are A -conjugate:

$$\underbrace{(s^{(k+1)})^T}_{\text{new}} A \underbrace{s^{(k)}}_{\text{old}} = 0.$$



Choosing β_k

- Substitute the new direction into the conjugacy condition:

$$(r^{(k+1)} + \beta_k s^{(k)})^T A s^{(k)} = 0.$$

- Solving for β_k gives

$$\beta_k = -\frac{(r^{(k+1)})^T A s^{(k)}}{(s^{(k)})^T A s^{(k)}}.$$

Simplified Formula for CG

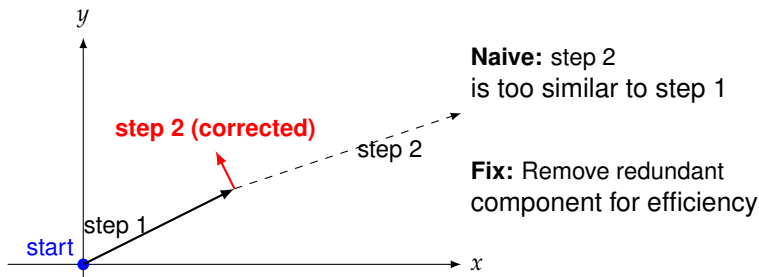
Using the exact line search and CG orthogonality:

$$\beta_k = \frac{(r^{(k+1)})^T r^{(k+1)}}{(r^{(k)})^T r^{(k)}}.$$

Key Idea

Only a scalar correction is needed to update the search direction.

Orthogonal Steps



Key Idea

Instead of searching in a near-identical direction, we remove the redundant component to make progress in an independent **A-orthogonal** direction.



Other Small Improvements

- **Reuse** $As^{(k)}$. The product $p^{(k)} = As^{(k)}$ is needed $(s^{(k)})^\top p^{(k)}$ for α_k and again in the residual update. Compute it once, store in a temporary, and the loop needs only $\text{nnz}(A)$ flops per iteration.
- **Short residual recurrence.**

$$r^{(k+1)} = r^{(k)} - \alpha_k p^{(k)}$$

avoids the extra SpMV that $b - Ax^{(k+1)}$ would require.

- **Dot-product recycling.** The numerator $r^{(k+1)\top} r^{(k+1)}$ is already computed for β_{k+1} . Reuse it for the stopping test $\|r^{(k+1)}\|_2 < \varepsilon$.
- **Minimal storage.** With those tricks, CG needs only four n -vectors in memory: $x^{(k)}, r^{(k)}, s^{(k)}, p^{(k)}$.



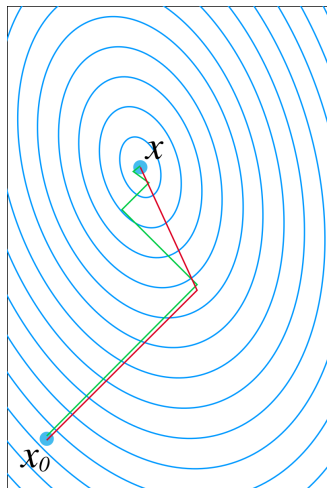
Conjugate Gradient

```
1  $x_0 = \text{initial guess}$ 
2  $r_0 = b - Ax_0$ 
3  $s_0 = r_0$ 
4 for  $k = 0, 1, 2, \dots$ 
5      $\alpha_k = \frac{r_k^T r_k}{s_k^T A s_k}$ 
6      $x_{k+1} = x_k + \alpha_k s_k$ 
7      $r_{k+1} = r_k - \alpha_k A s_k$ 
8      $\beta_{k+1} = r_{k+1}^T r_{k+1} / r_k^T r_k$ 
9      $s_{k+1} = r_{k+1} + \beta_{k+1} s_k$ 
```

Method: Conjugate Gradient

- **Given:** Given A , b , tolerance
- **Output:** Approximation to $Ax = b$
- **Idea:** Very efficient iterative method
 - ▶ Utilize sparsity of A
 - ▶ α is optimized
 - ▶ s_k are chosen to be A -orthogonal to each other

Conjugate Gradient Visualized



- Example A , 2×2 matrix
- blue curves = contour map
- green = gradient descent
- red = conjugate gradient

For a simple 2×2 , CG will converge in just 2 iterations.

Convergence of Conjugate Gradient

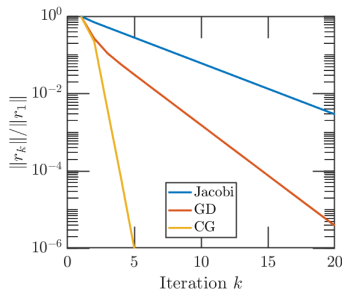
Exact Arithmetic

For an $n \times n$ SPD matrix, CG reaches the exact solution in **at most n iterations**.

- The search directions are mutually A -conjugate.
- In practice, we stop when $\|r^{(k)}\| < \delta$.
- Rounding errors can prevent termination within n iterations.
- Often, far fewer than n iterations are needed.



Performance: CG vs. Jacobi



- The error for Jacobi and CG are shown on a log-linear plot
- CG converges much faster than GD

Key Idea

- CG is a go-to algorithm to solve $Ax = b$ but A must be SPD
- CG is better than GD because it uses optimal stepsize and A-orthogonal search directions

