SI152: Numerical Optimization

Lecture 11: Conjugate Gradient Method[1]

Hao Wang

Email: haw309@gmail.com

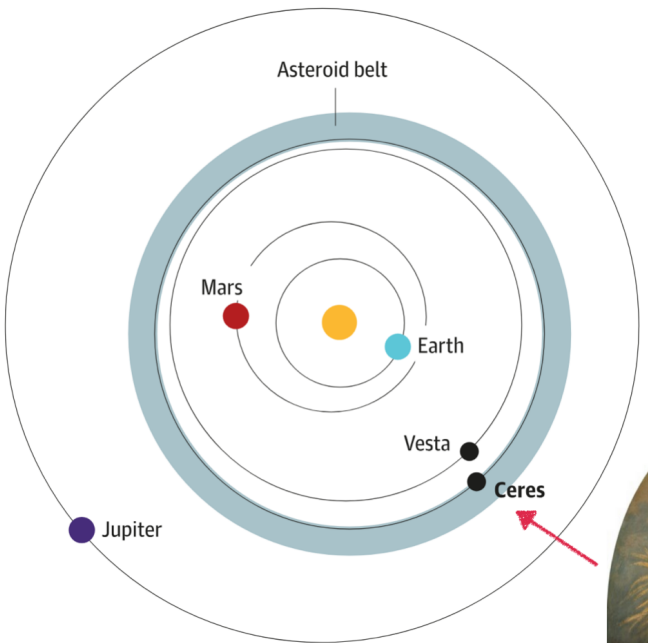
ShanghaiTech University

November 6, 2025

[1] Chapter 5, Numerical Optimization

$$K^T K x = y \iff \min_x \tfrac{1}{2}\|Kx - y\|_2^2$$

$$Ax = b, \quad A \succ 0 \iff \min_x \tfrac{1}{2} x^T A x - b^T x$$



Carl Friedrich Gauss, 1777–1855

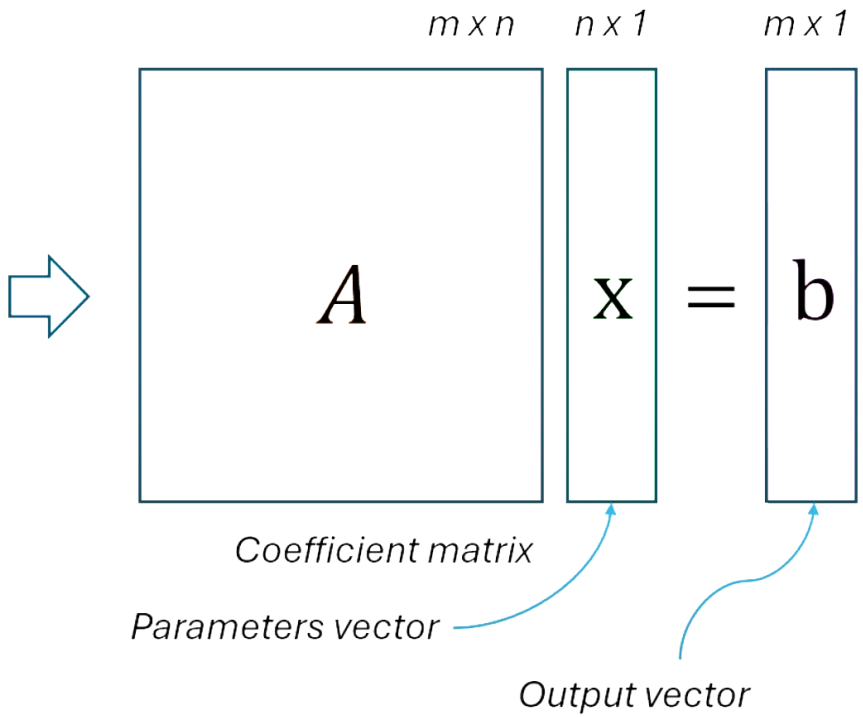
Linear equations are fundamental in many fields:

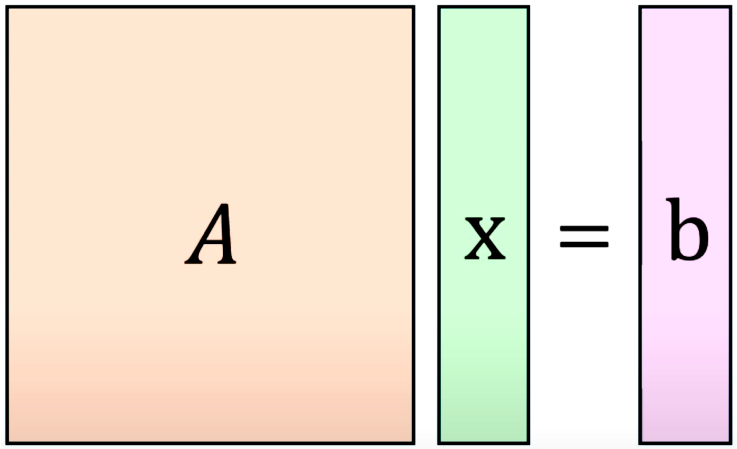
$$Ax = b, \quad A \in \mathbb{R}^{m \times n}$$

Equations: m

Parameters: n

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$
$$\vdots$$
$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m$$





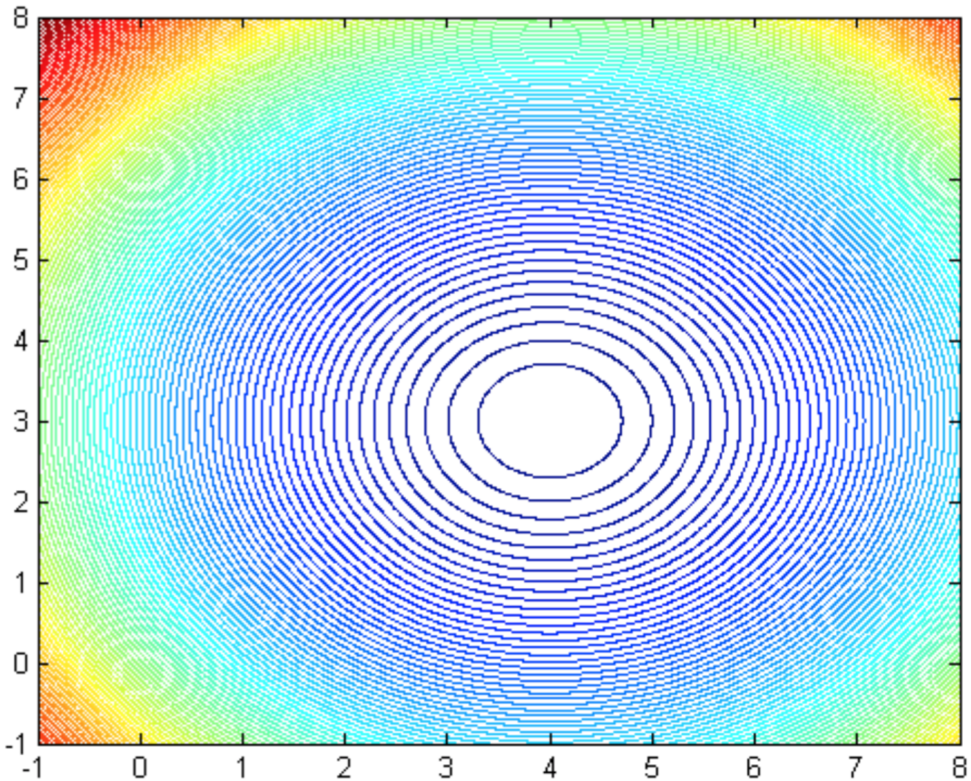
- Direct methods: Gaussian elimination
  Complexity $O(n^3)$: can be inefficient for large systems.
- Iterative methods
- Conjugate gradient method

$$\min_x \tfrac{1}{2} x^T A x - b^T x$$

Example:

- $A = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}, b = \begin{bmatrix} 20 \\ 15 \end{bmatrix}$
- It is easily verified that the solution is

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$



Consider the quadratic optimization problem

$$\min_x f(x) = g^T x + \frac{1}{2} x^T H x,$$

where H is symmetric. Necessary optimality conditions are

$$\nabla f(x) = g + Hx = 0.$$

If $H \succeq 0$, then f is convex, in which case the above conditions are also sufficient.

Thus, we have the following cases based on g and H:

- If $H \succeq 0$, then any solution to $g + Hx = 0$ is optimal
- If $H \succeq 0$ and $g + Hx = 0$ has no solution, or if

  $$d^T H d < 0 \quad \text{for some } d \neq 0,$$

  then the problem is unbounded.

Suppose we did not know the solution and did not have a picture:

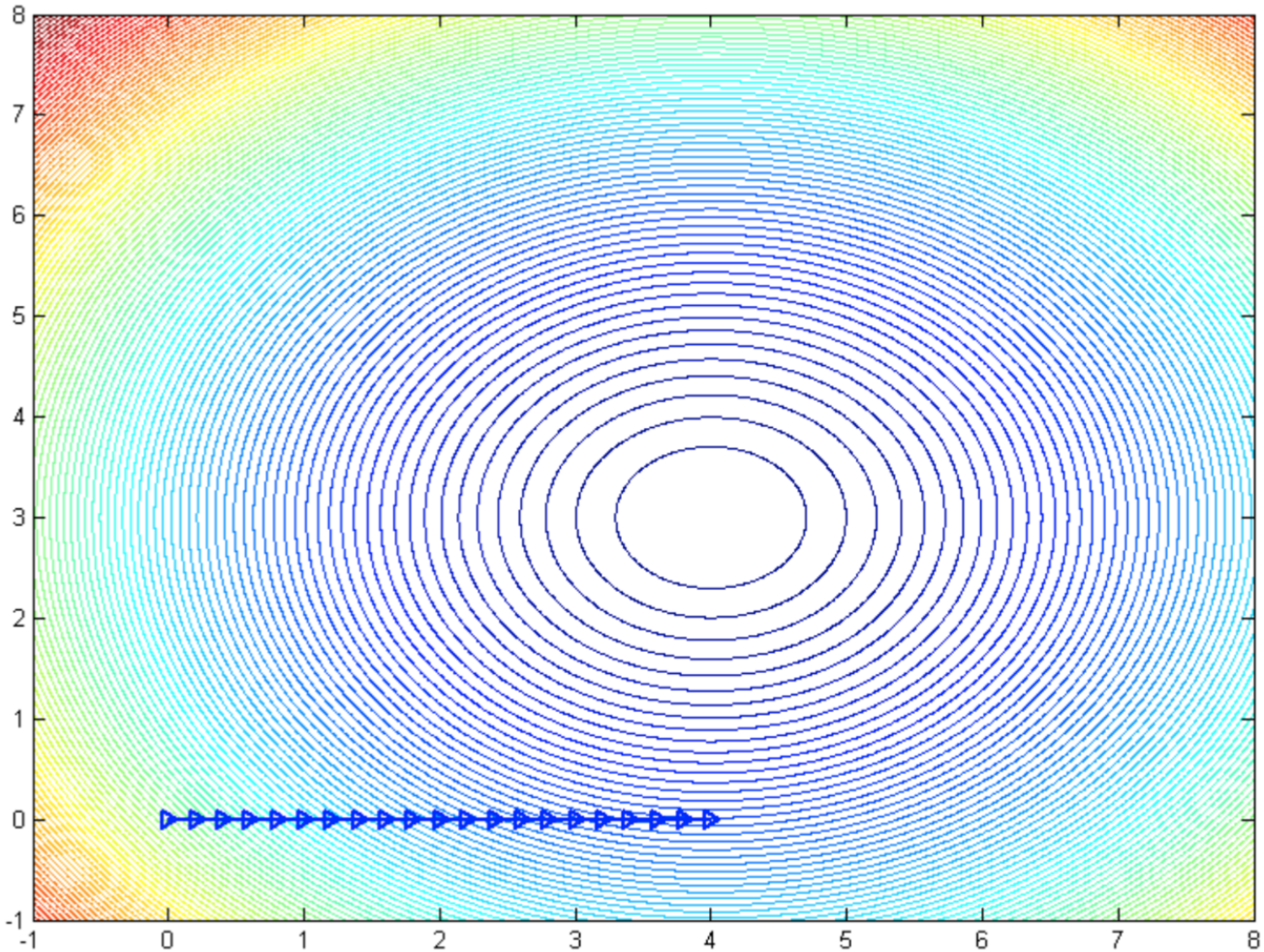
- Can we develop an iterative approach to minimize the quadratic?
- Can we ensure that each operation is cheap?

What about coordinate descent? That is, for an n-dimensional problem, start at the origin, and then minimize successively over

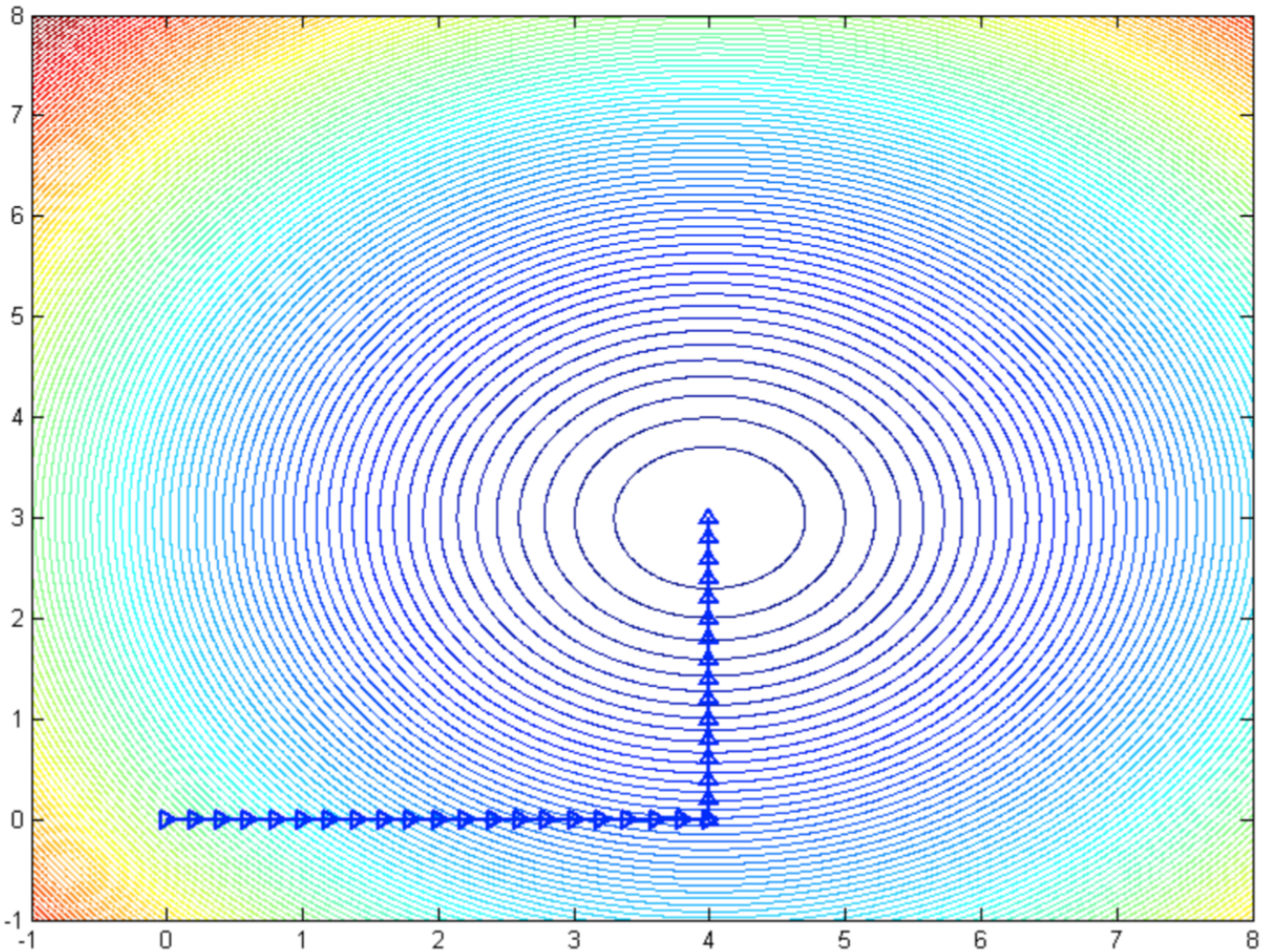
$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix}, \text{ then } \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}, \text{ then } \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}, \text{ then } \ldots$$

Each iteration is then a one-dimensional minimization of a quadratic; easy!

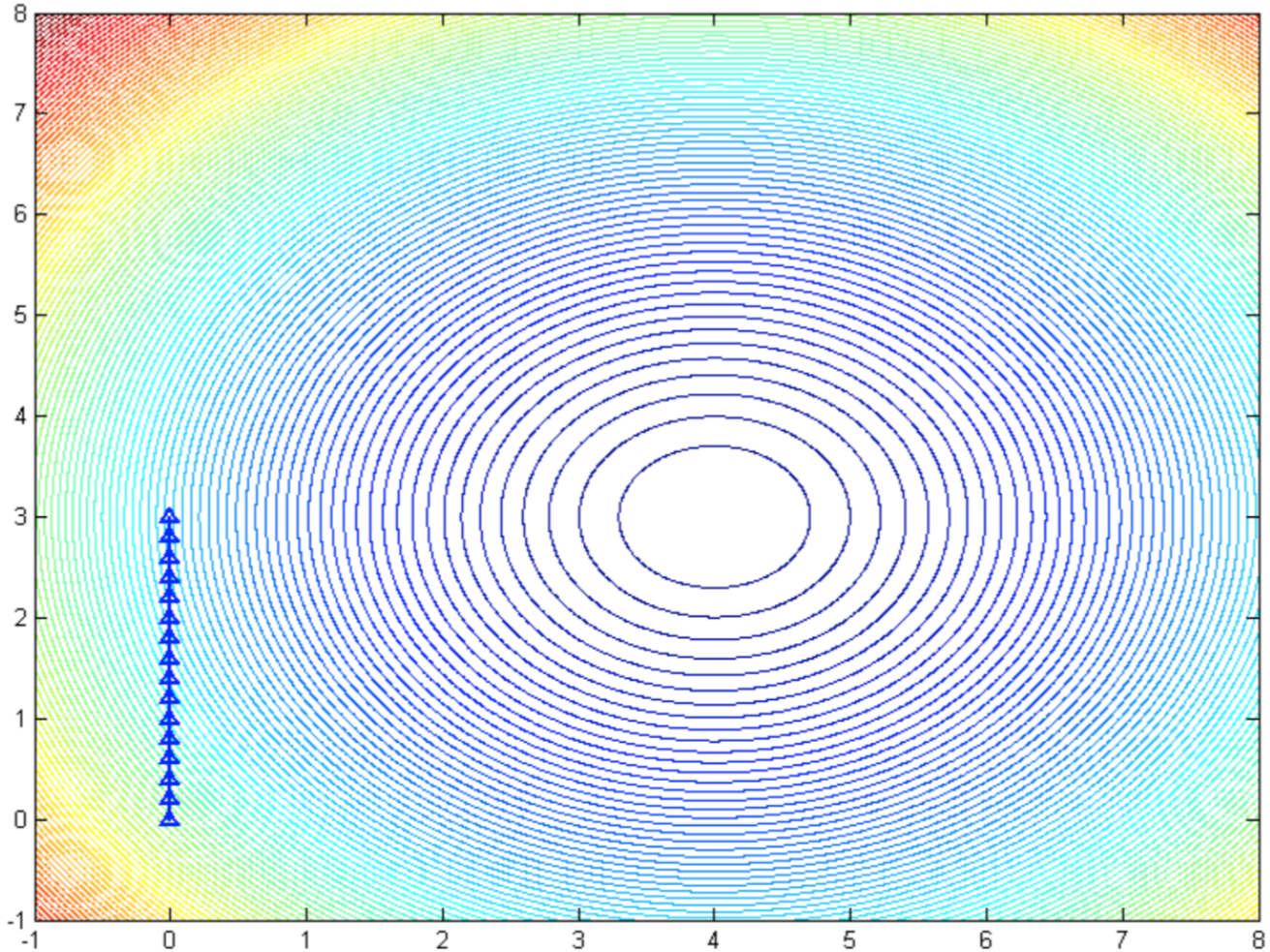
First, minimize over $(1, 0)$:



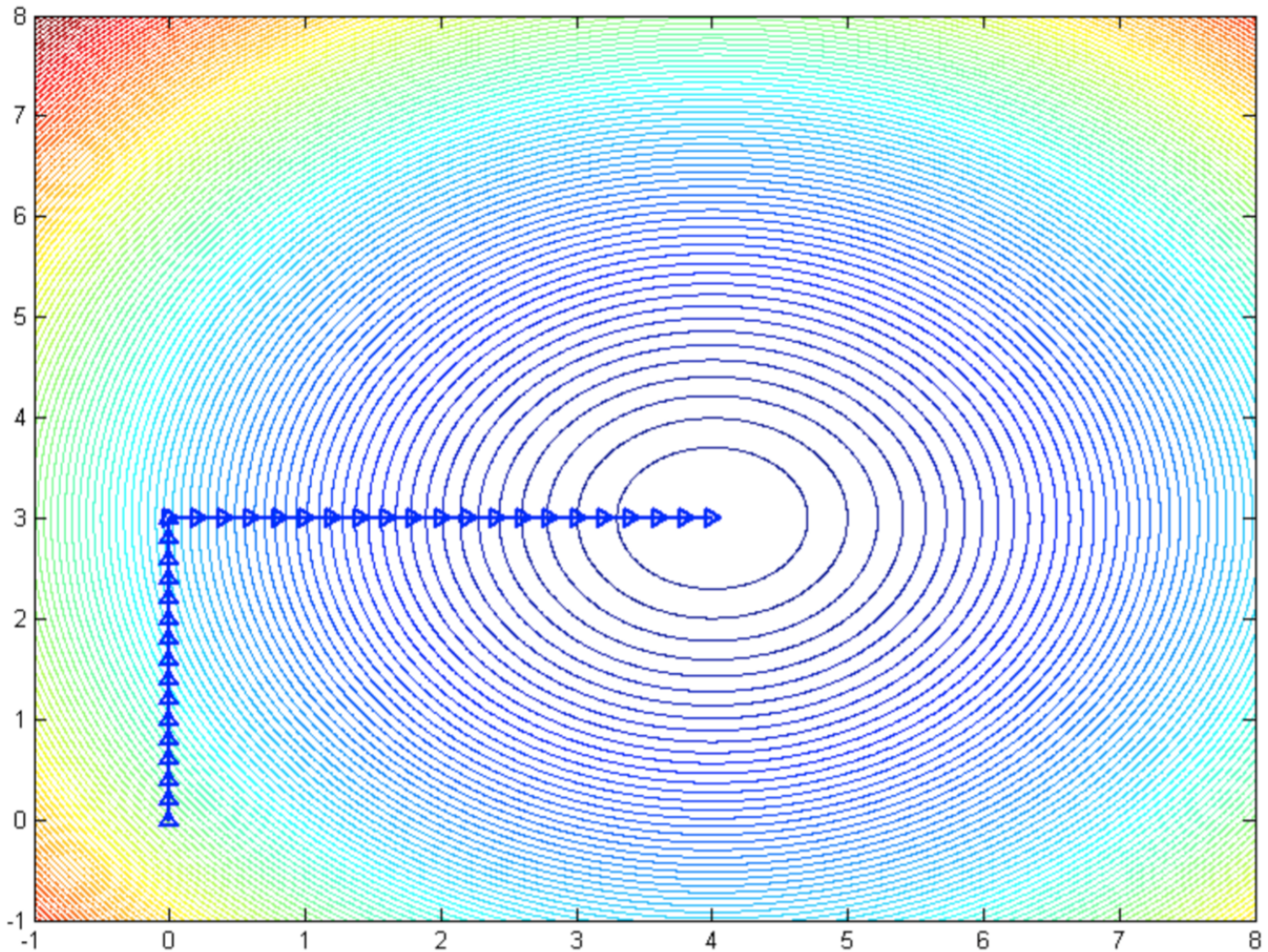
Then, from the new point, minimize over $(0, 1)$: Done!



For this problem, order doesn't matter. We could instead first minimize over $(0, 1)$



Then, from the new point, minimize over $(1, 0)$: Done!



This will happen in general for certain quadratics.

- Successive minimization over the directions

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix}, \text{ then } \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}, \text{ then } \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}, \text{ then } \ldots$$

  solves the problem.
- Each iteration is cheap — one-dimensional minimization of a quadratic.
- Problem is solved in (at most) n iterations.

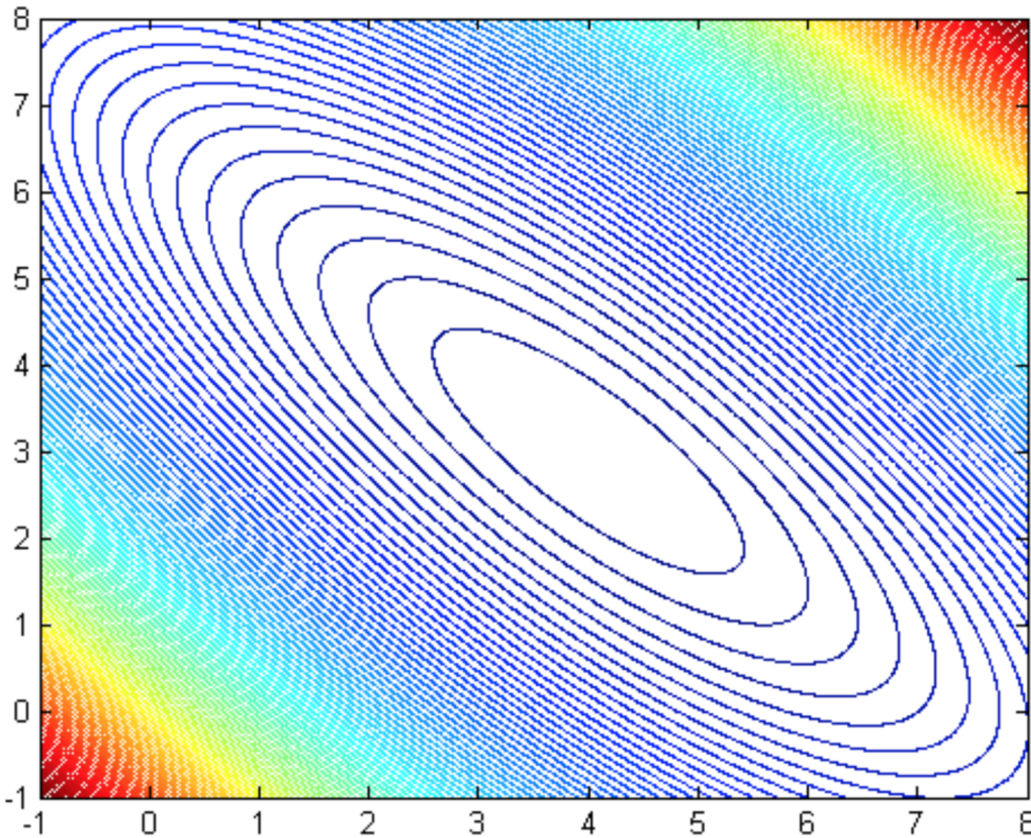
Wouldn't it be great if we could always do this?

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

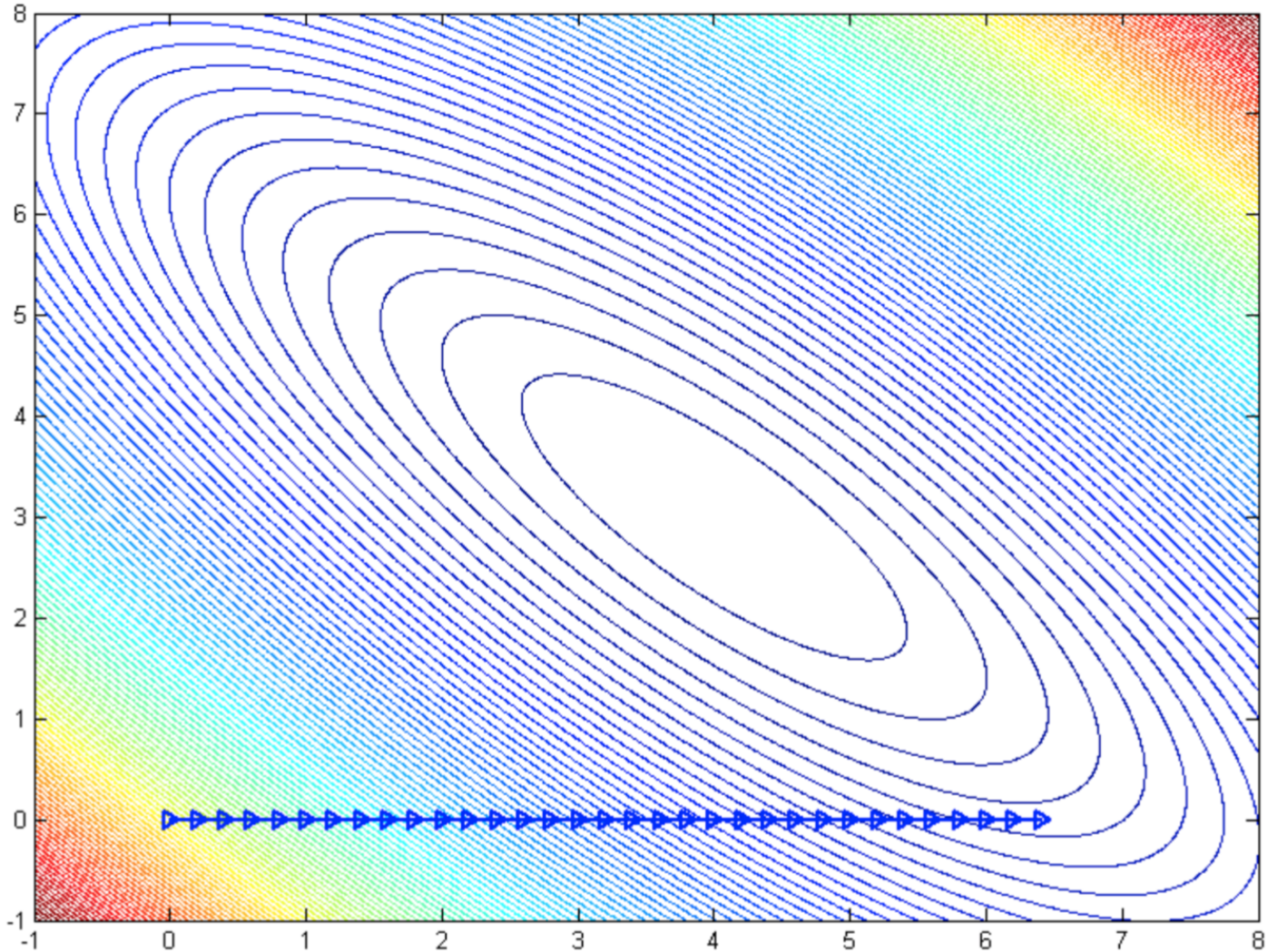
Example:

- $A = \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix}, b = \begin{bmatrix} 32 \\ 31 \end{bmatrix}$
- It is easily verified that the solution is

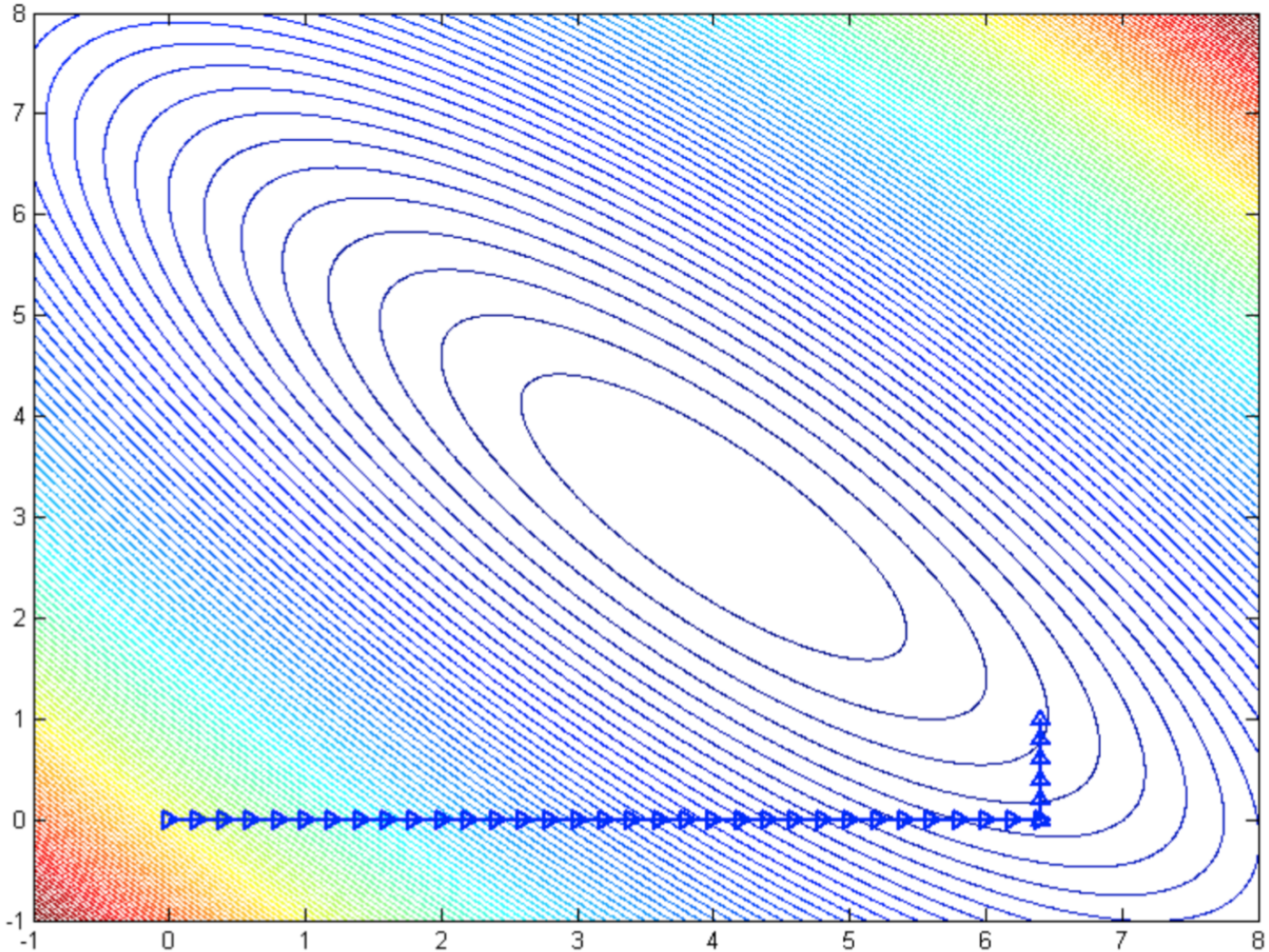
$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$



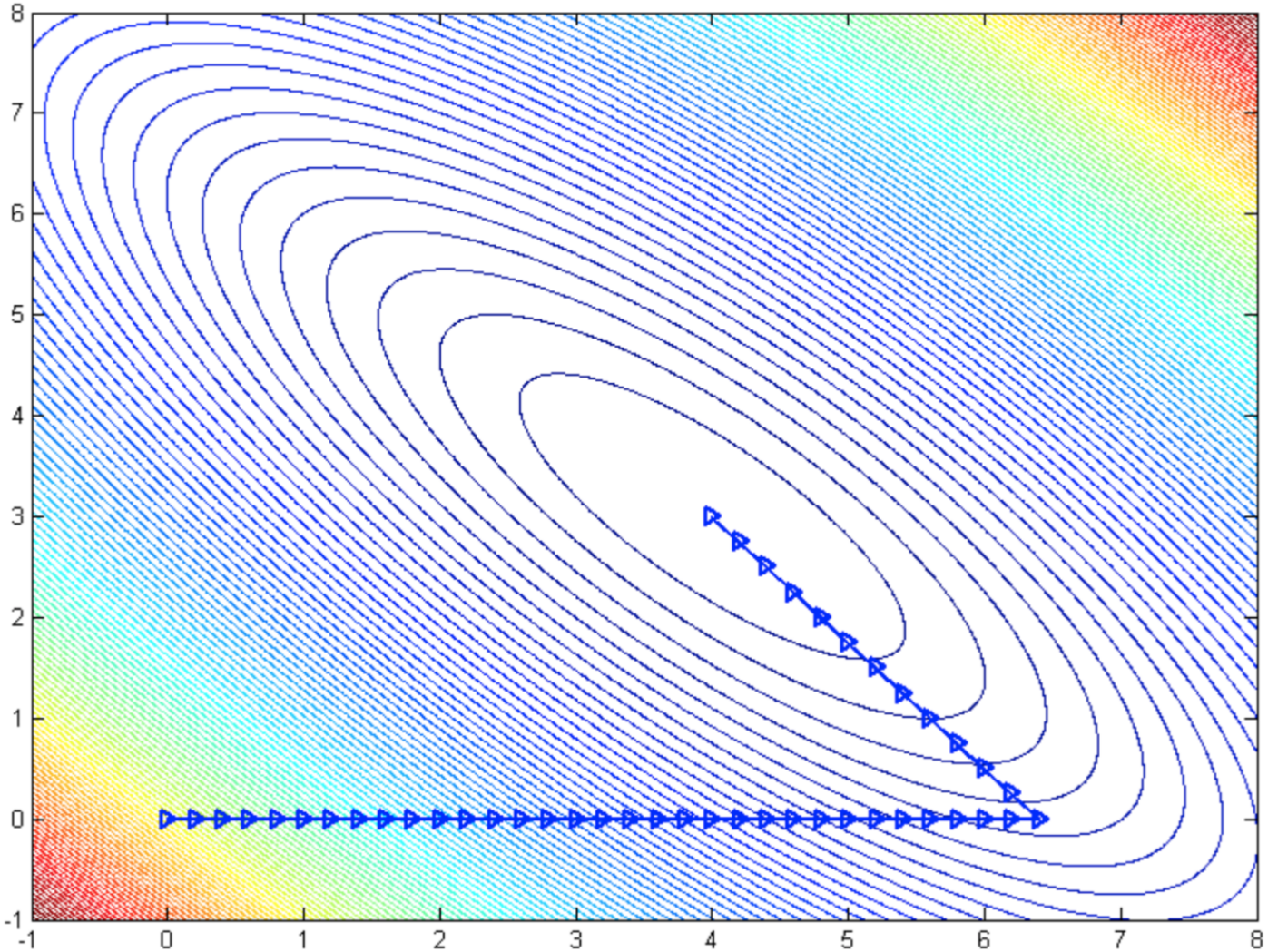
First, minimize over $(1, 0)$:



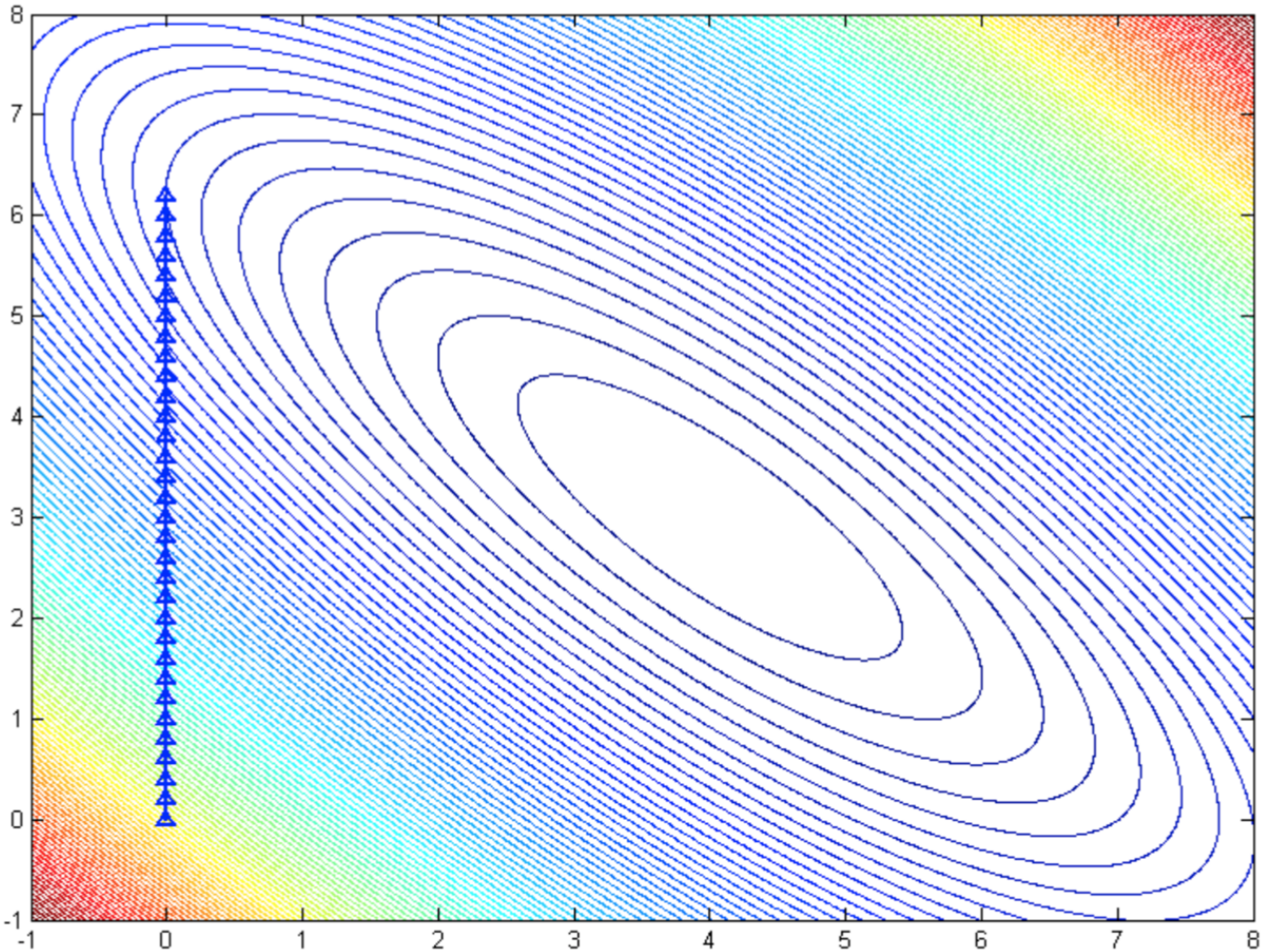
Then, from the new point, minimize over $(0, 1)$: Er, not so good.



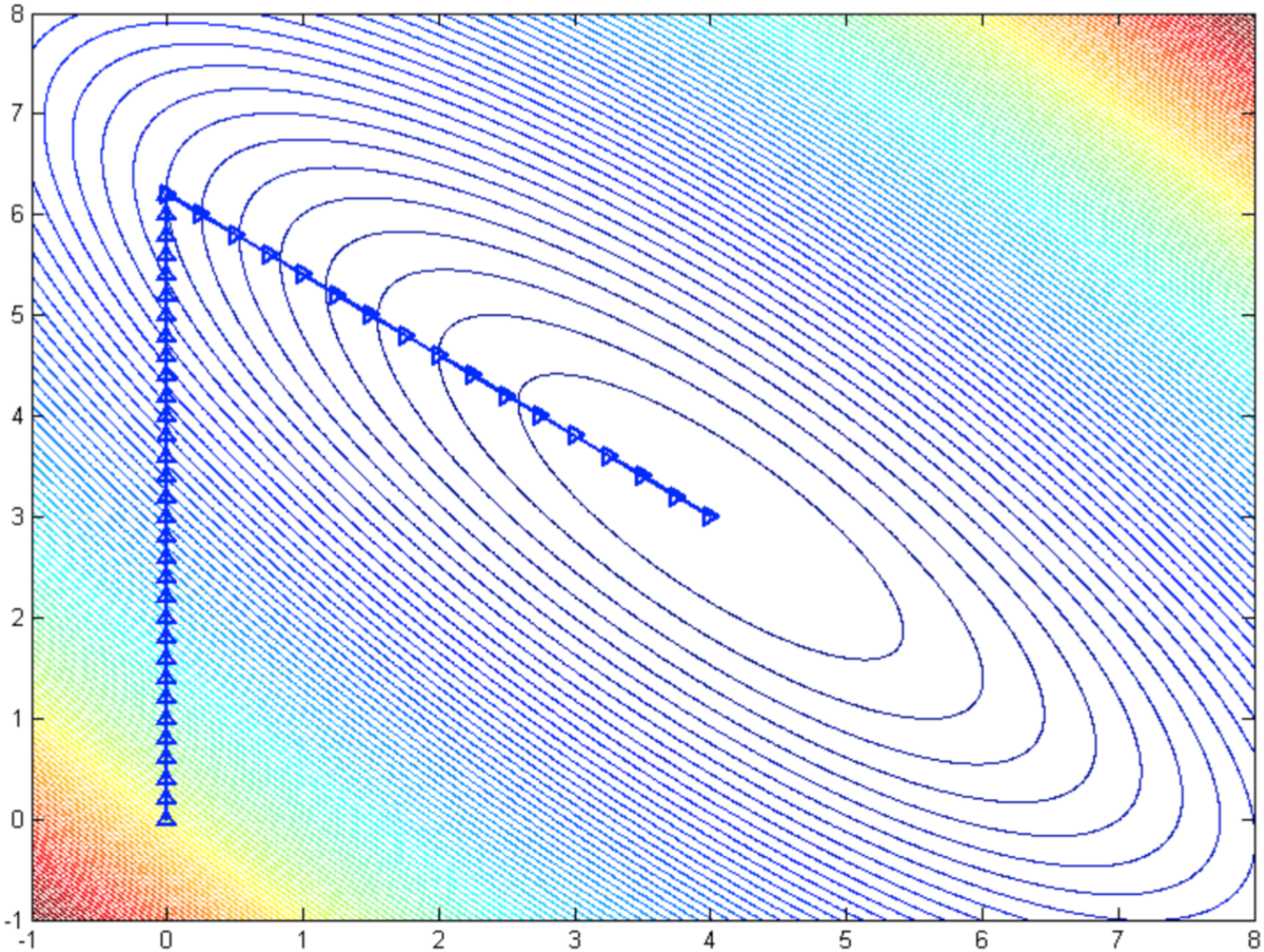
After step 1, it would be great to know this direction:



Suppose we minimized over $(0, 1)$ first: similar to before.



Now, after step 1, it would be great to know this direction:



The unique property of example 1 was that the Hessian was diagonal

- If the minimization problem

$$\min_x \tfrac{1}{2} x^T A x - b^T x$$

  has a diagonal A, then coordinate descent works nicely.
- Otherwise, coordinate descent is not the way to go.

What to do if the Hessian is not diagonal? One idea:

- Find a nonsingular matrix P such that $P^T AP$ is a diagonal matrix.
- Then, we can recast our problem as one to solve

$$\min_{\tilde{x}} \tfrac{1}{2}\tilde{x}^T P^T AP\tilde{x} - b^T P\tilde{x},$$

  since it is easily seen that $x = P\tilde{x}$ then solves our original problem.
- Preconditioning scales the contours to be aligned with coordinate directions

What are the properties of such a P and how could we possibly find such a matrix?

Let A be any positive definite $n \times n$ matrix.

- For any matrix (without any zero columns)

$$P = [p_0, \ldots, p_m],$$

the product

$$P^T AP$$

will be diagonal if and only if

$$p_i^T A p_j = 0, \text{ for all } i \neq j.$$

- We call the columns of such a matrix A-conjugate.

(We already know $p_j^T A p_j > 0$ for all j. How?)

Note:

- Any set of vectors that are A-conjugate are linearly independent (prove this).
- Thus, the matrix P on the previous page has at most n columns.

Suppose we have a set of n vectors, $p_0, ..., p_{n-1}$, that are A-conjugate. Then, consider the following algorithms starting at any x_0:

- Compute the current residual

$$r_k = Ax_k - b.$$

- Compute a steplength to minimize $\phi(x)$ along $x_k + \alpha p_k$:

$$\alpha_k = -\frac{r_k^T p_k}{p_k^T A p_k}. \quad \text{(why?)}$$

- Update $x_{k+1} \leftarrow x_k + \alpha_k p_k$ and go to step 1.

Theorem 1

For any x_0, the sequence $\{x_k\}$ generated by the conjugate direction method converges to the minimizer x_* of $\phi(x)$ in at most n steps.

Proof.

Since span$\{p_0, ..., p_{n-1}\} = \mathbb{R}^n$, there exists $\{\sigma_0, ..., \sigma_{n-1}\} \subset \mathbb{R}$ such that

$$x_* - x_0 = \sigma_0 p_0 + ... + \sigma_{n-1} p_{n-1}.$$

Premultiplying the above expression by $p_k^T A$, this means $\sigma_k = p_k^T A(x_* - x_0)/p_k^T A p_k$. However, we know from the algorithm that

$$p_k^T A x_k = p_k^T A(x_0 + \alpha_0 p_0 + ... + \alpha_{k-1} p_{k-1}) \implies p_k^T A x_k = p_k^T A x_0,$$

so we have

$$\sigma_k = \frac{p_k^T A(x_* - x_0)}{p_k^T A p_k} = \frac{p_k^T A(x_* - x_k)}{p_k^T A p_k} = \frac{p_k^T (b - A x_k)}{p_k^T A p_k} = -\frac{p_k^T r_k}{p_k^T A p_k} = \alpha_k.$$

□

Theorem 2

The sequence $\{x_k\}$ *generated by the conjugate direction method yields*

$$r_k^T p_j = 0, \qquad \text{for all } j = 0, \ldots, k-1,$$

and each iterate x_k *is the minimizer of* $\phi(x)$ *over the set*

$$x_0 + \text{span}\{p_0, \ldots, p_{k-1}\}.$$

How can we find an $n \times n$ matrix P with A-conjugate columns?

- If computing such a matrix is expensive (say, as expensive as computing a matrix of eigenvectors), then conjugate directions is impractical.

Fortunately (and amazingly), computing conjugate directions can be cheap!

Start by minimizing over the steepest descent direction for $\phi(d)$ at initial point x_0:

- The initial residual is $r_0 = Ax_0 - b = \nabla\phi(x_0)$, so we start by minimizing from x_0 over
$$p_0 = -r_0.$$

- The minimizer is easily found to be $x_1 = x_0 + \alpha_0 p_0$, where
$$\alpha_0 = -\frac{r_0^T p_0}{p_0^T A p_0},$$
yielding the new residual $r_1 = Ax_1 - b$.

Now, suppose we define the new direction as

$$p_1 = -r_1 + \beta_1 p_0.$$

What value for β_1?

$$0 = p_0^T A p_1 = p_0^T A(-r_1 + \beta_1 p_0) \implies \beta_1 = \frac{r_1^T A p_0}{p_0^T A p_0}.$$

This way, p_0 and p_1 are conjugate, but if we continue to set

$$p_k = -r_k + \beta_k p_{k-1},$$

with

$$\beta_k = \frac{r_k^T A p_{k-1}}{p_{k-1}^T A p_{k-1}},$$

will p_k be conjugate to all of $p_0, \ldots, p_{k-1}$? Yes, and more. First, the algorithm...

Given x_0, set $r_0 = Ax_0 - b$, $p_0 = -r_0$, and for $k = 0, 1, 2, \ldots$:

$$\begin{aligned}
\alpha_k &\leftarrow - \frac{r_k^T p_k}{p_k^T A p_k} \\
x_{k+1} &\leftarrow x_k + \alpha_k p_k \\
r_{k+1} &\leftarrow A x_{k+1} - b \\
\beta_{k+1} &\leftarrow \frac{r_{k+1}^T A p_k}{p_k^T A p_k} \\
p_{k+1} &\leftarrow - r_{k+1} + \beta_{k+1} p_k
\end{aligned}$$

Each iteration requires:

- 2 matrix-vector products: Ap_k, Ax_{k+1}.
- 3 vector-vector products: $r_k^T p_k$, $(Ap_k)^T p_k$, $(Ap_k)^T r_{k+1}$.
- 3 vector additions and 2 vector scalings.

Theorem (Conjugate gradient properties)[2]

Define the Krylov subspace of degree k for r_0 as

$$\mathcal{K}(r_0; k) = \text{span}(r_0, Ar_0, A^2 r_0, ..., A^k r_0\}.$$

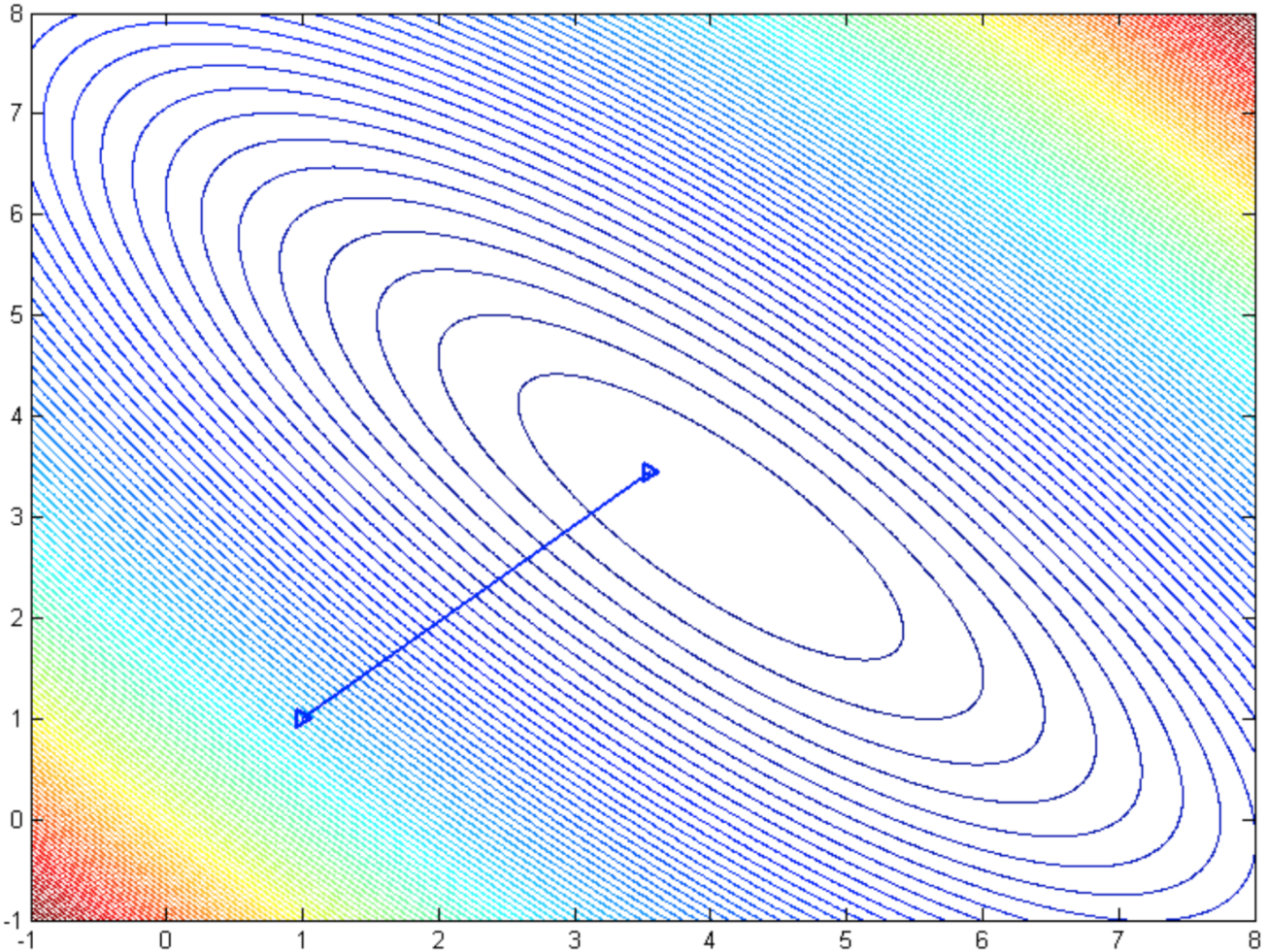
During the conjugate gradient algorithm, the following hold:

$$\begin{aligned} r_k^T r_j &= 0, \quad \text{for } j = 0, 1, ..., k-1; \\ \text{span}\{r_0, r_1, ..., r_k\} &= \mathcal{K}(r_0; k) \\ \text{span}\{p_0, p_1, ..., p_k\} &= \mathcal{K}(r_k; k) \\ p_k^T A p_j &= 0, \quad \text{for } j = 0, 1, ..., k-1. \end{aligned}$$

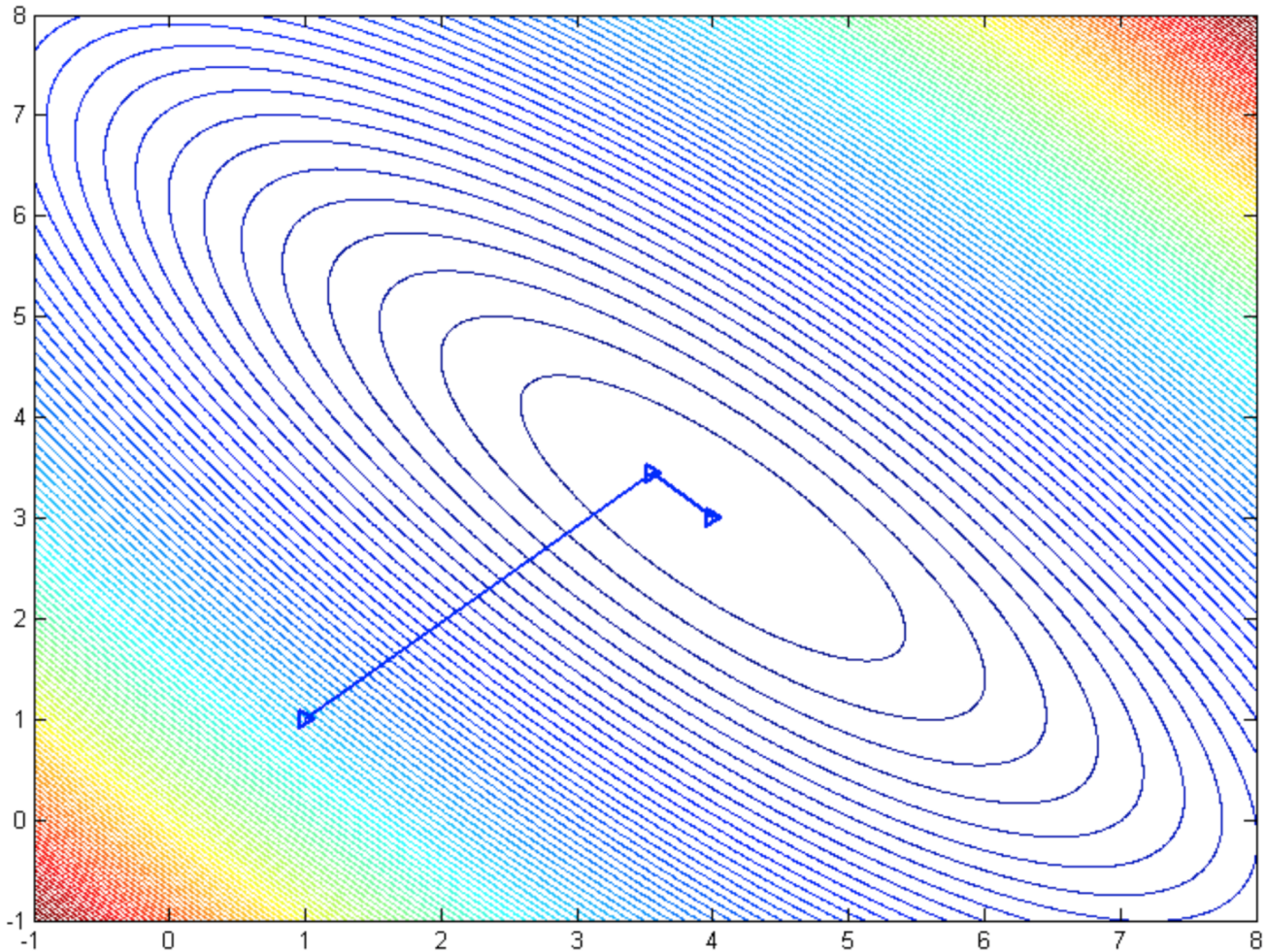
Thus, the sequence $\{x_k\}$ converges to x_* in at most n steps!

[2] Theorem 5.3., Numerical Optimization

Minimizing along steepest descent direction $p_0 = -r_0 = -\nabla\phi(x_0)$



Minimizing along p_1: Done!



Given x_0, set $r_0 = Ax_0 - b$, $p_0 = -r_0$, and for $k = 0, 1, 2, \ldots$:

$$\begin{aligned}
\alpha_k &\leftarrow \frac{r_k^T r_k}{p_k^T A p_k} \\
x_{k+1} &\leftarrow x_k + \alpha_k p_k \\
r_{k+1} &\leftarrow r_k + \alpha_k A p_k \\
\beta_{k+1} &\leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \\
p_{k+1} &\leftarrow -r_{k+1} + \beta_{k+1} p_k
\end{aligned}$$

Each iteration requires:

- 1 matrix-vector products: Ap_k.
- 2 vector-vector products: $(Ap_k)^T p_k$, $r_{k+1}^T r_{k+1}$.
- 3 vector additions and 3 vector scalings.

We have seen that the CG method converges in at most n steps

- Often, approximate solutions are found much faster.
- We will see that steps to convergence depend on the eigenvalues of A.
- The goals of preconditioning then become clear.

Theorem 3

If A has r distinct eigenvalues, then the CG method will converge in r steps.

Theorem 4

If A has eigenvalues $\lambda_1 \leq \ldots \leq \lambda_n$, then

$$\|x_{k+1} - x_*\|_A^2 \leq \left(\frac{\lambda_{n-k} - \lambda_1}{\lambda_{n-k} + \lambda_1} \right)^2 \|x_0 - x_*\|_A^2.$$

Theorem 5

If A has condition number $\kappa(A)$, then

$$\|x_{k+1} - x_*\|_A^2 \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^{k+1} \|x_0 - x_*\|_A^2.$$

Consider an example where A has 10 distinct (non-clustered) eigenvalues:

```
>> eig(A)'

ans =

    0.0625    0.6405    2.2592    3.2548    5.4752    8.0424   14.3216   16.4549   27.1209   33.3637
```

The `cg.m` code (w/ random x_0) terminates with a small residual in 10 iterations:

```
>> x = cg(A,b,x,e);
==================================+=========================================
   k        ||x||          ||r||     |    ||p||          alpha          beta
==================================+=========================================
     0  2.1305e+000   1.7696e+001 |  1.7696e+001   +5.7003e-002   +2.1162e-001
     1  1.6381e+000   8.1407e+000 |  8.9608e+000   +5.9714e-002   +4.1390e-001
     2  1.3698e+000   5.2374e+000 |  6.4176e+000   +1.2569e-001   +6.4068e-001
     3  1.0608e+000   4.1921e+000 |  5.8719e+000   +5.8855e-002   +2.9281e-001
     4  1.1168e+000   2.2684e+000 |  2.8464e+000   +2.6837e-001   +1.5718e+000
     5  1.6591e+000   2.8440e+000 |  5.3015e+000   +1.8400e-001   +6.6288e-001
     6  2.5724e+000   2.3155e+000 |  4.2085e+000   +4.8497e-001   +6.2108e-001
     7  4.4883e+000   1.8248e+000 |  3.1878e+000   +2.1983e-001   +4.5410e-001
     8  5.0517e+000   1.2297e+000 |  1.8994e+000   +3.7721e+000   +3.0864e+000
     9  1.0846e+001   2.1603e+000 |  6.2476e+000   +7.2375e-001   +6.4061e-015
    10  1.4937e+001   8.8445e-009 |  -----------   ------------   ------------
==================================+=========================================
```

Consider an example where A has 4 distinct (non-clustered) eigenvalues:

```
>> eig(A)'

ans =

    0.0625    3.2548   16.4549   27.1209   27.1209    0.0625    0.0625    3.2548    3.2548    3.2548
```

The `cg.m` code (w/ random x_0) terminates with a small residual in 4 iterations:

```
>> x = cg(A,b,x,e);
==================================+=======================================
   k       ||x||         ||r||    |    ||p||          alpha          beta
==================================+=======================================
   0  2.1305e+000   1.5469e+001 | 1.5469e+001  +5.8751e-002  +1.9457e-001
   1  1.7107e+000   6.8232e+000 | 7.4575e+000  +9.5686e-002  +8.3252e-001
   2  1.3785e+000   6.2257e+000 | 8.7924e+000  +1.5316e-001  +1.2278e-001
   3  1.5024e+000   2.1815e+000 | 2.4340e+000  +1.2793e+001  -1.0603e-014
   4  3.2358e+001   4.1579e-012 | -----------  ------------  ------------
==================================+=======================================
```

Consider an example where A has 8 distinct (≈ 0.0625) eigenvalues:

```
>> eig(A)'

ans =

   27.1209   16.4549    0.0625    0.0625    0.0625    0.0625    0.0625    0.0625    0.0625    0.0625
```

The `cg.m` code (w/ random x_0) terminates with a small residual in 4 iterations:

```
>> x = cg(A,b,x,e);
==================================+=======================================
   k        ||x||          ||r||     |     ||p||          alpha          beta
==================================+=======================================
   0   2.1305e+000   6.9963e+000 |  6.9963e+000   +5.8016e-002   +3.4897e-001
   1   2.1937e+000   4.1330e+000 |  4.8003e+000   +2.2574e-001   +3.5196e+000
   2   2.8302e+000   7.7537e+000 |  1.8589e+001   +2.7376e+000   +2.5892e-012
   3   5.2922e+001   1.2502e-005 |  1.2502e-005   +1.6000e+001   +1.3756e-009
   4   5.2922e+001   3.8277e-011 |  -----------   ------------   ------------
==================================+=======================================
```

(Note that the residual is already relatively small within 3 iterations.)

- CG is a very successful method for solving quadratic subproblems.
- That is, it is successful for computing a single search direction.
- Is it possible to extend these ideas to minimizing general nonlinear functions?

What are the essential components of a CG iteration?

- Minimize the function along a given direction.
- Update iterate and evaluate new "residual" (i.e., steepest descent direction).
- Determine new direction as combination of old one and new steepest descent.

Given x_0, set $r_0 = \nabla f(x_0)$, $p_0 = -r_0$ and for $k = 0, 1, 2, ...$

$$\begin{aligned}
\alpha_k &\leftarrow \text{value to minimize } f \text{ along } x_k + \alpha p_k; \\
x_{k+1} &\leftarrow x_k + \alpha_k p_k \\
r_{k+1} &\leftarrow \nabla f(x_{k+1}) \\
\beta_k^{FR} &\leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \\
p_{k+1} &\leftarrow -r_{k+1} + \beta_k^{FR} p_k
\end{aligned}$$

More succinctly, the steps are

$$\begin{aligned}
\alpha_k &\leftarrow \text{value to minimize } f \text{ along } x_{k+1} = x_k + \alpha p_k; \\
\beta_k^{FR} &\leftarrow \frac{\nabla f(x_{k+1})^T \nabla f(x_{k+1})}{\nabla f(x_k)^T \nabla f(x_k)} \\
p_{k+1} &\leftarrow -\nabla f(x_{k+1}) + \beta_k^{FR} p_k
\end{aligned}$$

- Linear CG is well-understood and clearly works.
- Nonlinear CG works in bizarre ways.

How do we know that the Fletcher-Reeves method is sensible? Note:

$$\nabla f(x_k)^T p_k = -\|\nabla f(x_k)\|^2 + \beta_k^{FR} \nabla f(x_k)^T p_{k-1}.$$

Thus, we have descent directions as long as the latter term does not dominate.

- An exact line search long p_{k-1} yields $\nabla f(x_k)^T p_{k-1} = 0$.
- Inexact line search yields descent if the strong Wolfe conditions are enforced.

Thus, one can show that if the level set

$$\mathcal{L} = \{x \mid f(x) \leq f(x_0)\}$$

is bounded, and if f is continuously differentiable in a neighborhood of $\mathcal{L}$, then

$$\liminf_{k \to \infty} \|\nabla f(x_k)\| = 0.$$

Alternative nonlinear CG methods differ in their choice of β, e.g.,

$$\text{Fletcher-Reeves: } \beta_k^{FR} = \frac{\nabla f(x_{k+1})^T \nabla f(x_{k+1})}{\nabla f(x_k)^T \nabla f(x_k)} = \frac{\|g_{k+1}\|^2}{\|g_k\|^2}$$

$$\text{Polak-Ribére: } \beta_k^{PR} = \frac{\nabla f(x_{k+1})^T (\nabla f(x_{k+1}) - \nabla f(x_k))}{\|\nabla f(x_k)\|^2} = \frac{g_{k+1}^T y_k}{\|g_k\|^2}$$

$$\text{Hestenes-Stiefel: } \beta_k^{HS} = \frac{\nabla f(x_{k+1})^T (\nabla f(x_{k+1}) - \nabla f(x_k))}{(\nabla f(x_{k+1}) - \nabla f(x_k))^T p_k} = \frac{g_{k+1}^T y_k}{y_k^T p_k}$$

$$\text{Dai-Yuan: } \beta_k^{DY} = \frac{\nabla f(x_{k+1})^T \nabla f(x_{k+1})}{(\nabla f(x_{k+1}) - \nabla f(x_k))^T p_k} = \frac{\|g_{k+1}\|^2}{y_k^T p_k}$$

Each perform differently in practice and have different theory (see book).