Lecture 09: Iterative Methods - Conjugate Gradient Method

June 18, 2025

Outline

Solution by Steepest Descent

- In Lecture 08 we looked at stationary iterative methods.
- We will now begin to look at other iterative methods for solving $Ax = b$.
- The **steepest descent** method and the **conjugate gradient** method are discussed in this lecture.

Solution by Steepest Descent

- There is an equivalent minimization interpretation of solving $Ax = b$.
- We assume $A \in \mathbb{R}^{n \times n}$ is symmetric positive definite (SPD) and consider the quadratic function

$$F(x) = \frac{1}{2} x^T A x - b^T x, \quad x \in \mathbb{R}^n.$$

- We will show that the solution of $Ax = b$ is equivalent to the solution of the minimization problem

$$\min_x F(x).$$

Solution by Steepest Descent

- Consider visualizing the case with $n = 2$ as shown in Figure 1.
- Here x is a length-2 vector $x = [x_1, x_2]^T$.
- The function $F(x)$ is a scalar that gives height in the plot.
- Plotting F gives a paraboloid with minimum value at $x = A^{-1}b$.
- Note that A being SPD implies F is convex.

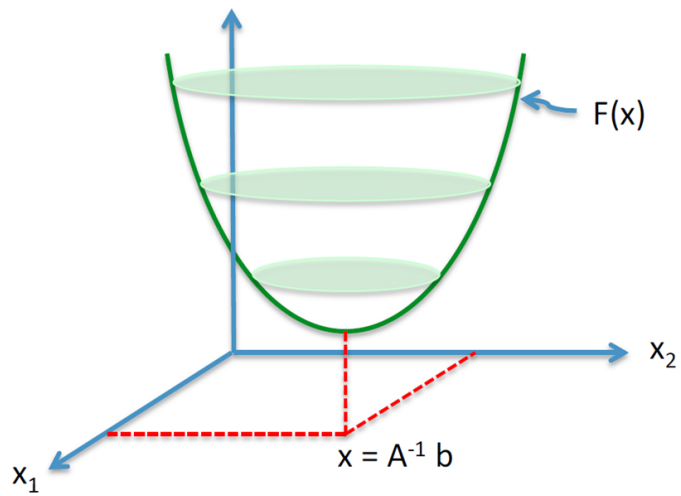


Figure: Visualization of $F(x)$ when $n = 2$.

Solution by Steepest Descent

Theorem 1

The solution of the linear system and minimization form are the same.

Proof.

The solution of the minimization satisfies $\nabla F(x) = 0$, i.e.,

$$\frac{\partial F}{\partial x_i} = 0, \forall i.$$

Since $\nabla F(x) = Ax - b$ (See Lecture Notes), therefore the solution of $Ax = b$ corresponds to a stationary point. However, since $F(x)$ is (strictly) convex any local minimum is the global minimum. □

Solution by Steepest Descent

- By Theorem 1, if we can solve the minimization problem, we have solved the linear system.
- So we will try to find the minimum by "walking downhill".
- The main idea is, at each step, first choose a **search direction** vector $p \neq 0$.
- Then find the point along that direction with the lowest value.
- Therefore, we iterate as $x^{k+1} = x^k + \alpha p$, where $\alpha \in \mathbb{R}$.
- We want to find α that gives the minimum value of $F(x^{k+1}) = F(x^k + \alpha p)$.

Solution by Steepest Descent

Let

$$
\begin{aligned}
f(\alpha) &= F(x^k + \alpha p), \\
&= \frac{1}{2}(x^k + \alpha p)^T A(x^k + \alpha p) - (x^k + \alpha p)^T b, \\
&= \frac{1}{2}(x^k)^T A x^k + \left(\frac{\alpha}{2} p^T A x^k + \frac{\alpha}{2}(x^k)^T A p\right) \\
&\quad + \frac{\alpha^2}{2} p^T A p - (x^k)^T b - \alpha p^T b, \\
&= \underbrace{\left[\frac{1}{2}(x^k)^T A x^k - (x^k)^T b\right]}_{F(x^k)} \\
&\quad + \alpha\left[p^T A x^k - p^T b\right] + \frac{\alpha^2}{2}\left[p^T A p\right].
\end{aligned}
$$

Solution by Steepest Descent

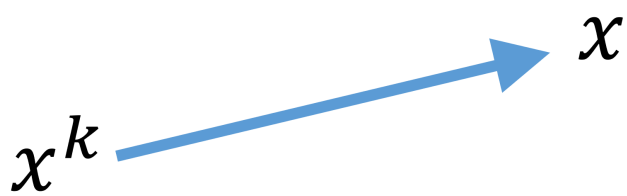
- To optimize we differentiate f with respect to α, set it to 0, and solve for α.

$$f'(\alpha) = -p^T(b - Ax^k) + \alpha p^T Ap = 0,$$
$$\Rightarrow \alpha = \frac{p^T(b - Ax^k)}{p^T Ap} = \frac{p^T r^k}{p^T Ap},$$

  where $r^k = b - Ax^k$ is the residual of the kth iterate.
- This is the optimal α given the update rule $x^{k+1} = x^k + \alpha p$ and search vector $p \neq 0$.
- Note that since A is SPD, $p^T Ap > 0$. So α gives the minimum point along a given search direction.

Solution by Steepest Descent

- Now consider choosing the search directions p.
- What is the *optimal* search direction?
- A vector pointing straight from x^k towards the solution x, so $p = x - x^k$.



- This p would give the solution in **one step**.
- Unfortunately we do not know x!
- Therefore, we pick the search directions in a *locally optimal* manner.
- That is, we determine what direction reduces F as rapidly as possible **at the current point**.

Solution by Steepest Descent

- We saw above that $f'(\alpha) = p^T(Ax^k - b) + \alpha p^T Ap$.
- This gives the rate of change at distance α along the search vector.
- So, $f'(0)$ gives rate of change along p at the *current position* (i.e., x^k).
- The idea is to pick p to make $f'(0)$ as negative as possible.
- This gives the direction of fastest decrease.
- We have that $f'(0) = p^T(Ax^k - b) = p^T \nabla F(x^k)$, so $f'(0)$ is minimized for

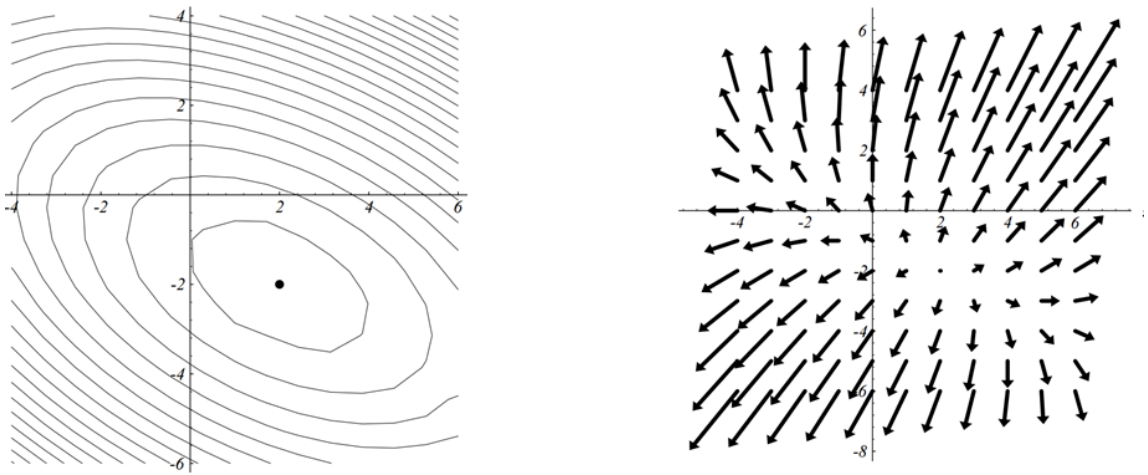
$$p = -\frac{\nabla F(x^k)}{\|\nabla F(x^k)\|}, \quad \text{(assuming we want unit } p, \|p\| = 1)$$
$$= \frac{b - Ax^k}{\|b - Ax^k\|} = \frac{r^k}{\|r^k\|}.$$

Solution by Steepest Descent

- It is actually not necessary to normalize the search direction vector.
- Therefore we take $p = r^k$, which gives the iteration $x^{k+1} = x^k + \alpha r^k$ and optimal α as

$$\alpha = \frac{p^T r^k}{p^T A p} = \frac{(r^k)^T r^k}{(r^k)^T A r^k}.$$

- Intuitively, we are finding the negative gradient of F (i.e., residual) and following it "downhill" as shown in Figure 2.



Contours of F Gradient Vectors

Figure: Example contours and gradient vectors of a function F.

Solution by Steepest Descent

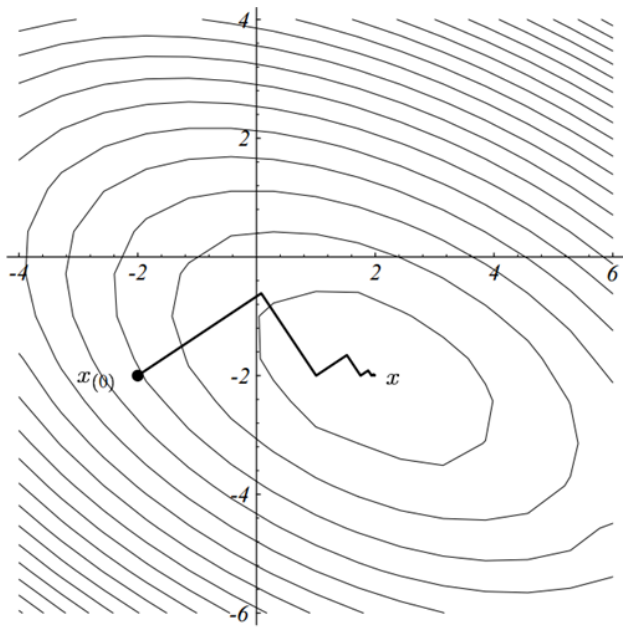
- The steepest descent algorithm is given in the Lecture Notes.
- For efficiency, instead of recomputing the residual from scratch at each step we can derive a simple **update rule**

$$\begin{aligned} r^{k+1} &= b - Ax^{k+1}, \\ &= b - A(x^k + \alpha r^k), \\ &= b - Ax^k - \alpha Ar^k, \\ &= r^k - \alpha Ar^k. \end{aligned}$$

- The term Ar^k is already needed in the algorithm, so we can save one matrix-vector product per iteration by storing its result.
- Note that you can view steepest descent as a **nonlinear** iterative method with the iteration matrix $M = M^k = \frac{1}{\alpha_k} I$ that *changes* on each iteration.

Solution by Steepest Descent

- The steepest descent algorithm actually behaves quite poorly in terms of convergence.
- Since we assumed A was SPD steepest descent will indeed converge.
- However, steepest descent can be "shortsighted" and will often yield slow (zig-zag-like convergence towards the solution) as seen below.



Solution by Steepest Descent

- For a SPD matrix A the error vectors $e^k = x^k - x^*$ for steepest descent satisfy

$$||e^{k+1}||_A \leq \left(\frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} \right) ||e^k||_A,$$

  where $||\cdot||_A$ indicates the "A-norm" or **energy norm**: $||x||_A = \sqrt{x^T A x}$.
- For a proof of this result see Saad textbook, Section 5.3.1 or [Shewchuk 1994].
- Next we will look at the **conjugate gradient** method, which chooses a different sequence of steps that can sometimes perform much better.

Solution by Steepest Descent - Towards the Conjugate Gradient Method

- Recall the **steepest descent** method.
- We considered finding the x that gives the minimum of

$$F(x) = \frac{1}{2}x^T Ax - b^T x, \quad x \in \mathbb{R}^n,$$

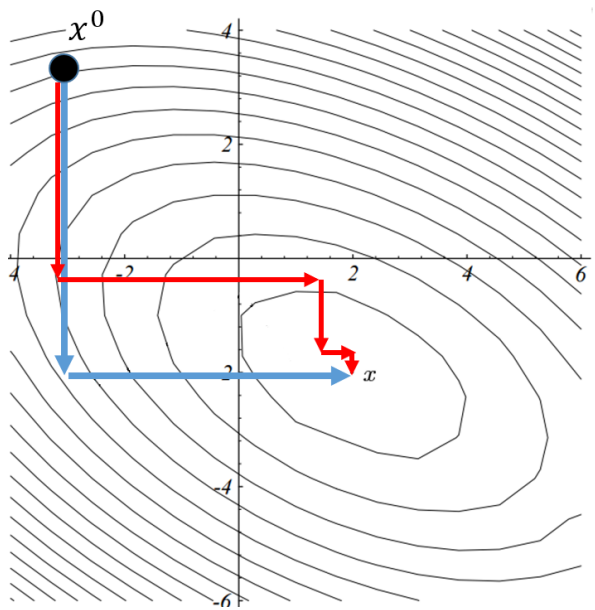
  which also is the solution to $Ax = b$.
- We developed an iteration $x^{k+1} = x^k + \alpha p^k$ with:
  - search direction $p^k = r^k = b - Ax^k$,
  - step length $\alpha = \frac{(r^k)^T p^k}{(p^k)^T A p^k} = \frac{(r^k)^T r^k}{(r^k)^T A r^k}$,
- The search direction gives a locally fastest decrease, but not the globally "best" direction.
- This leads to zig-zag like convergence towards the solution.
- We now discuss how we can do better, starting with the **conjugate directions** method, then refining it to finally arrive at the **conjugate gradient** method.

Another Search Direction Idea

- Imagine an approach that (somehow) finds the solution in the x_1 axis, then in the x_2 axis, ..., then in the x_n axis.
- It would complete in n iterations, touching each **orthogonal** axis **once**.
- Can we achieve something like this?

Another Search Direction Idea

- Choosing step $p^k = \text{axis}_k$ does not actually work.
- The lowest point (minimum value) along each axis in *not* the solution for that axis.
- The figure below depicts this idea.
- The blue vectors are what we would like.
- But, the red is the (bad) result if we use axes as search directions.



Another Search Direction Idea

- Choosing orthogonal directions (axes) and minimizing along them one at a time clearly does not work.
- Let us try something else, based on **A-orthogonality**.
- We first need to some definitions.

Definition 2.1

Suppose A is SPD, then the A-inner product is defined as

$$(p, q)_A = p^T A q.$$

Definition 2.2

The A-norm is given by

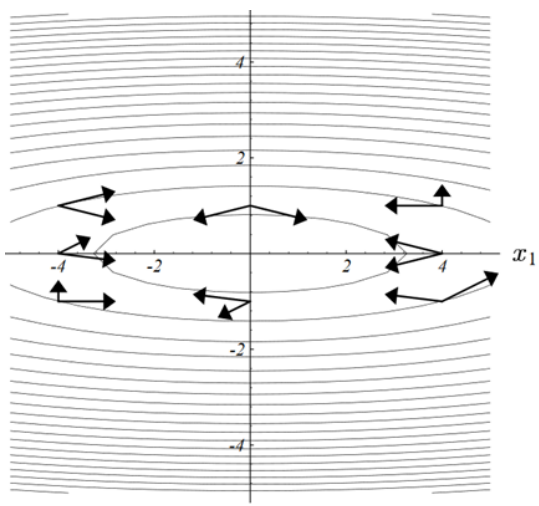
$$||p||_A = \sqrt{(p, p)_A}.$$

Definition 2.3

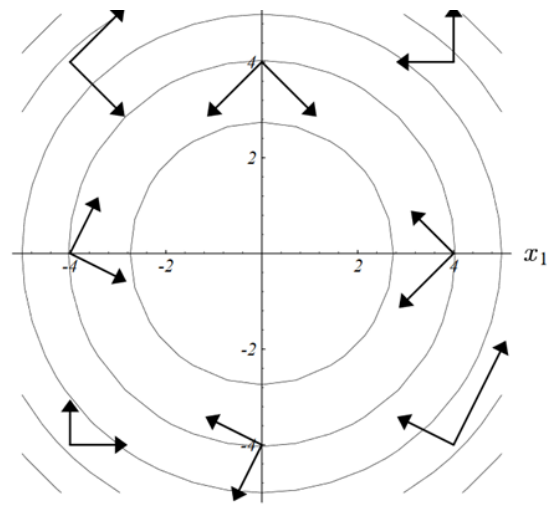
Two vectors p, q are A-orthogonal (or conjugate) if $(p, q)_A = 0$.

Another Search Direction Idea

- Figure 3 visualizes vectors that are A-orthogonal.
- In general, A-orthogonal vectors are not orthogonal in the standard Euclidean space.
- The vectors are instead orthogonal when you transform to the new space by multiplying by A.



A-orthogonal vector pairs (are not orthogonal!)



Corresponding orthogonal vector pairs after stretching the space according to A

Figure: Visualization of the concept of A-orthogonality.

Another Search Direction Idea

- Intuitively, A-orthogonality is a kind of orthogonality that respects the properties of A.
- We build a new search direction method based on A-orthogonality.
- We choose each search direction to be A-orthogonal to **all** previous search directions.
- This will avoid searching redundant directions repeatedly.
- So we now need an algorithm to construct A-orthogonal vectors.
- We will use Gram-Schmidt A-orthogonalization.

Another Search Direction Idea - Gram-Schmidt (A-)orthogonalization

- Gram-Schmidt A-orthogonalization construct a set of A-orthogonal vectors, incrementally.
- Suppose the previous search directions $p^0, p^1, p^2, \ldots, p^{k-1}$ are all mutually A-orthogonal.
- Given a new proposed direction u^k, we convert it into a p^k that is A-orthogonal to all prior p^i.
- The idea is to subtract out the components of u^k that are *not* A-orthogonal to earlier p^i's, leaving behind a vector that *is*.

Another Search Direction Idea - Gram-Schmidt (A-)orthogonalization

- Figure 4 gives a visualization of one step of Gram-Schmidt in 2D (for regular orthogonality).
- Consider some vector u, and a (previous) basis vector a.
- Then the vector $u - (u^T a)a$ will be orthogonal to a.
- Let's derive a Gram-Schmidt process to construct A-orthogonal vectors.

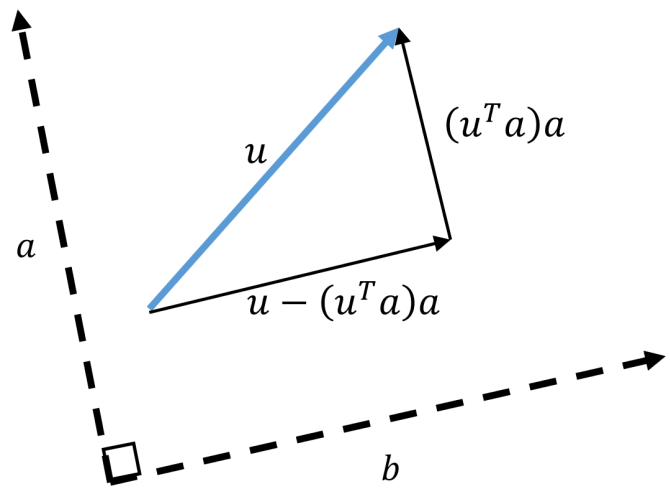


Figure: Orthogonalizing a vector u with respect to a.

Another Search Direction Idea - Gram-Schmidt (A-)orthogonalization

- We start with a vector u^k and subtract all prior p^i components to form p^k, so

$$p^k = u^k + \sum_{i=0}^{k-1} \beta_i p^i.$$

- Now we just need to find the coefficients β_i.
- For each p^j for $j = 0, \ldots, k-1$ use A-orthogonality against p^k:

$$\begin{aligned} 0 &= (p^k, p^j)_A, \\ &= \left(u^k + \sum_{i=0}^{k-1} \beta_i p^i, p^j \right)_A, \\ &= (u^k, p^j)_A + \sum_{i=0}^{k-1} \beta_i (p^i, p^j)_A. \end{aligned}$$

Another Search Direction Idea - Gram-Schmidt (A-)orthogonalization

- Earlier p^i's were mutually A-orthogonal, so $(p^i, p^j)_A = 0$ for $i \neq j$.
- Therefore,

$$\begin{aligned} 0 &= (u^k, p^j)_A + \beta_j (p^j, p^j)_A, \\ \Rightarrow \beta_j &= -\frac{(u^k, p^j)_A}{(p^j, p^j)_A}. \end{aligned} \tag{1}$$

- Using this strategy we can construct an A-orthogonal set of p^k vectors spanning $\mathbb{R}^n$, when given input u^k's.

Another Search Direction Idea - Conjugate Directions Method

- To summarize, the conjugate directions method starts with a given set of vectors u^k spanning $\mathbb{R}^n$.
- We then A-orthogonalize them by Gram-Schmidt to get A-orthogonal search directions p^k.
- The same basic iteration as steepest descent is then performed to compute the solution x to $Ax = b$.
- The changes to the steepest descent iteration are:
  - use new p^k as search directions (instead of residuals r^k),
  - use our original step length expression $\alpha = \frac{(r^k)^T p^k}{(p^k)^T A p^k}$ (i.e., not $\frac{(r^k)^T r^k}{(p^k)^T A r^k}$).

Another Search Direction Idea - Conjugate Directions Method

The drawbacks of conjugate directions (w/ Gram-Schmidt) are as follows.

- With respect to memory we need to keep *all* prior search vectors p^i.
- In terms of computational cost we need to perform complete Gram-Schmidt at each step, which takes $O(n^3)$ flops.
- There is also the lingering question about how to choose the input (non-A-orthogonal) u^k vectors?
- A fun fact is that if the proposed search vectors u^k at each step (before A-orthogonalization) are the axes, this gives Gaussian Elimination again!

Conjugate Gradient Method

- The conjugate gradient method is a variation on the conjugate directions method that improves in two ways:
  1. choose the vectors u^k at each step to be the residual r^k (to be A-orthogonalized),
  2. carefully exploit (A-)orthogonality to avoid storing all prior search vectors.
- Item 1. immediately turns our earlier Gram-Schmidt process into

$$p^k = r^k + \sum_{i=0}^{k-1} \beta_i p^i = r^k - \sum_{i=0}^{k-1} \frac{(r^k, p^i)_A}{(p^i, p^i)_A} p^i, \tag{2}$$

  which gives the first version of the conjugate gradient algorithm in the Lecture Notes.
- This first version is still costly, do not implement this version!

Conjugate Gradient Method - Efficient Conjugate Gradient Method

- Let's work towards a more efficient algorithm.
- We want concise recursive expressions for
  - the step lengths α,
  - step directions p, and
  - Gram-Schmidt coefficients β.
- We will need to consider the spaces involved and their relationships, as well as, repeatedly exploit orthogonality and A-orthogonality.

Conjugate Gradient Method - Efficient Conjugate Gradient Method

- Consider the space spanned by vectors p^i for $i = 0, \ldots, k-1$.

$$\begin{aligned} & \text{span}\{p^0, p^1, \ldots, p^{k-1}\} \\ = \; & \text{span}\{r^0, r^1, \ldots, r^{k-1}\}, \quad (\text{by Gram-Schmidt}) \\ = \; & \text{span}\{r^0, Ar^0, A^2r^0, \ldots, A^{k-1}r^0\} \quad (\text{See Lecture Notes}). \end{aligned}$$

- A space constructed this way (powers of A times a vector) is called a (k-dimensional) **Krylov subspace**, denoted $\mathcal{K}_k(A, r^0)$.

Conjugate Gradient Method - Efficient Conjugate Gradient Method

- It can also be shown (See Lecture Notes) that $r^k \perp \text{span}\{p^0, p^1, \ldots, p^{k-1}\} = \text{span}\{r^0, r^1, \ldots, r^{k-1}\}$, i.e., $(r^k, r^j) = 0$ for $j = 0, 1, \ldots, k-1$.
- That is, the current residual is orthogonal to the prior search directions and residuals.
- Currently, the **step length** is computed as $\alpha_k = \frac{(r^k, p^k)}{(p^k, p^k)_A}$.
- Observe however that

$$\begin{aligned} (r^k, p^k) &= \left(r^k, r^k + \sum_{j=0}^{k-1} \beta_i p^i \right), \quad \text{(since } (r^k, p^i) = 0 \text{ for } i < k\text{)} \\ &= (r^k, r^k), \end{aligned} \tag{3}$$

  which gives a new way of computing α_k as

$$\alpha_k = \frac{(r^k, r^k)}{(p^k, p^k)_A}.$$

Conjugate Gradient Method - Efficient Conjugate Gradient Method

- To make computing the search direction more efficient we need the identity

$$(r^k, p^i)_A = 0 \text{ for } i = 0, 1, \ldots, k-2. \tag{4}$$

Proof.

See the Lecture Notes. □

Conjugate Gradient Method - Efficient Conjugate Gradient Method

- Now we can construct **search direction** p^k without storing all prior p^i.
- Starting from (2), Gram-Schmidt gives

$$\begin{aligned} p^k &= r^k - \sum_{i=0}^{k-1} \frac{(r^k, p^i)_A}{(p^i, p^i)_A} p^i, \\ &= r^k - \frac{(r^k, p^{k-1})_A}{(p^{k-1}, p^{k-1})_A} p^{k-1}, \end{aligned}$$

  by (4).
- Hence, only the current residual and previous step direction are needed to compute p^k.
- This saves us storage and flops.

Conjugate Gradient Method - Efficient Conjugate Gradient Method

- To arrive at the standard conjugate gradient method we further simplify β_{k-1}.
- Equation (1) gave $\beta_{k-1} = -\frac{(u^k, p^{k-1})_A}{(p^{k-1}, p^{k-1})_A}$.
- Later replacing u^k by r^k gave

$$\beta_{k-1} = -\frac{(r^k, p^{k-1})_A}{(p^{k-1}, p^{k-1})_A}. \tag{5}$$

- In the numerator we have $(r^k, p^{k-1})_A$.
- By the residual update rule, $r^k = r^{k-1} - \alpha_{k-1} A p^{k-1}$, so applying $(r^k, \cdot)$ to this rule, we get that

$$(r^k, r^k) = \underbrace{(r^k, r^{k-1})}_{=0 \text{ since the } r\text{'s are orthogonal}} - \alpha_{k-1}(r^k, Ap^{k-1}),$$

- Rearranging, we get that the numerator is

$$(r^k, p^{k-1})_A = (r^k, Ap^{k-1}) = -\frac{1}{\alpha_{k-1}}(r^k, r^k).$$

Conjugate Gradient Method - Efficient Conjugate Gradient Method

- Now consider the denominator $(p^{k-1}, p^{k-1})_A$.
- We have that $(r^k, p^{k-1}) = 0$ by orthogonality.
- Applying $(\cdot, p^{k-1})$ to the residual update rule gives

$$\begin{aligned} \underbrace{(r^k, p^{k-1})}_{=0} &= (r^{k-1}, p^{k-1}) - \alpha_{k-1}(Ap^{k-1}, p^{k-1}) \\ 0 &= (r^{k-1}, r^{k-1}) - \alpha_{k-1}(p^{k-1}, p^{k-1})_A, \end{aligned}$$

  because earlier (3) we showed $(r^{k-1}, p^{k-1}) = (r^{k-1}, r^{k-1})$.
- Rearranging, gives the denominator as $(p^{k-1}, p^{k-1})_A = \frac{1}{\alpha_{k-1}}(r^{k-1}, r^{k-1})$.

Conjugate Gradient Method - Efficient Conjugate Gradient Method

- Combining the numerator and denominator we have

$$\begin{aligned}\beta_{k-1} &= -\frac{(r^k, p^{k-1})_A}{(p^{k-1}, p^{k-1})_A}, \\ &= -\left(-\frac{(r^k, r^k)}{\cancel{\alpha_{k-1}}}\right)\left(\frac{\cancel{\alpha_{k-1}}}{(r^{k-1}, r^{k-1})}\right), \\ &= \frac{(r^k, r^k)}{(r^{k-1}, r^{k-1})}.\end{aligned}$$

- The more efficient version of the conjugate gradient method is given in the Lecture Notes, based on the simplifications above.
- Now conjugate gradient just needs 1 matrix-vector multiply and 2 inner-products per step:
  - matrix-vector multiply Ap^k,
  - dot products (r^k, r^k) and (p^k, Ap^k).

Conjugate Gradient Method - Error Behaviour

- Note that at most n A-orthogonal vectors are needed to span $\mathbb{R}^n$.
- Therefore conjugate gradient will terminate in (at most) n steps with an exact solution (under exact arithmetic).
- At each iteration, the current conjugate gradient solution's error has the minimum A-norm within the subspace it has already explored,i.e.,

$$x^k = \underset{x \in \mathcal{K}_k}{\arg\min} \, ||e^k||_A^2 = \underset{x \in \mathcal{K}_k}{\arg\min} \, ||x^k - x^*||_A^2.$$

- This is because **at each iteration, conjugate gradient zeroes out one of the error components**.
- To see this let $e^i = x^* - x^i$, where x^* is the true solution.
- We therefore have $r^i = Ae^i$.

Conjugate Gradient Method - Error Behaviour

- Now express e^0 as a linear combination of search directions

$$e^0 = \sum_{j=0}^{n-1} \delta_j p^j, \quad \text{for coefficients } \delta_j.$$

- Left multiplying by $(p^k)^T A$ (to exploit A-orthogonality) gives

$$\begin{aligned}
(p^k)^T A e^0 &= \sum_{j=0}^{n-1} \delta_j (p^k)^T A p^j, \\
(p^k, e^0)_A &= \sum_{j=0}^{n-1} \delta_j (p^k, p^j)_A, \\
&= \delta_k (p^k, p^k)_A, \qquad \text{(by A-orthogonality)} \\
\Rightarrow \delta_k &= \frac{(p^k, e^0)_A}{(p^k, p^k)_A}.
\end{aligned}$$

Conjugate Gradient Method - Error Behaviour

- Continuing the calculation with the generic $e^k = e^0 - \sum_{i=0}^{k-1} \alpha_i p^i$ (because the iteration is $x^{k+1} = x^k + \alpha_k p^k$) gives

$$\begin{aligned} \delta_k &= \frac{(p^k, e^0)_A}{(p^k, p^k)_A} \\ &= \frac{(p^k, e^k + \sum_{i=0}^{k-1} \alpha_i p^i)_A}{(p^k, p^k)_A} \\ &= \frac{(p^k, e^k)_A}{(p^k, p^k)_A}, \end{aligned}$$

by the A-orthogonality of the p^i's.

Conjugate Gradient Method - Error Behaviour

- But recall that

$$\begin{aligned} \alpha_k &= \frac{(p^k)^T r^k}{(p^k, p^k)_A} \\ &= \frac{(p^k)^T A e^k}{(p^k, p^k)_A}, \text{ since } r^k = A e^k \\ &= \frac{(p^k, e^k)_A}{(p^k, p^k)_A}. \end{aligned}$$

- Hence, $\alpha_k = \delta_k$, so conjugate gradient zeroes out one component of the error at each iteration:

$$e^i = e^0 - \sum_{j=0}^{i-1} \alpha_j p^j = \left(\sum_{j=0}^{n-1} \delta_j p^j - \sum_{j=0}^{i-1} \delta_j p^j \right) = \sum_{j=i}^{n-1} \delta_j p^j. \quad (6)$$

- After n steps, all the components of e^0 will be gone.

Conjugate Gradient Method - Error Behaviour

- Consider using conjugate gradient on the following example:

$$\begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} x = \begin{bmatrix} 2 \\ -8 \end{bmatrix}$$

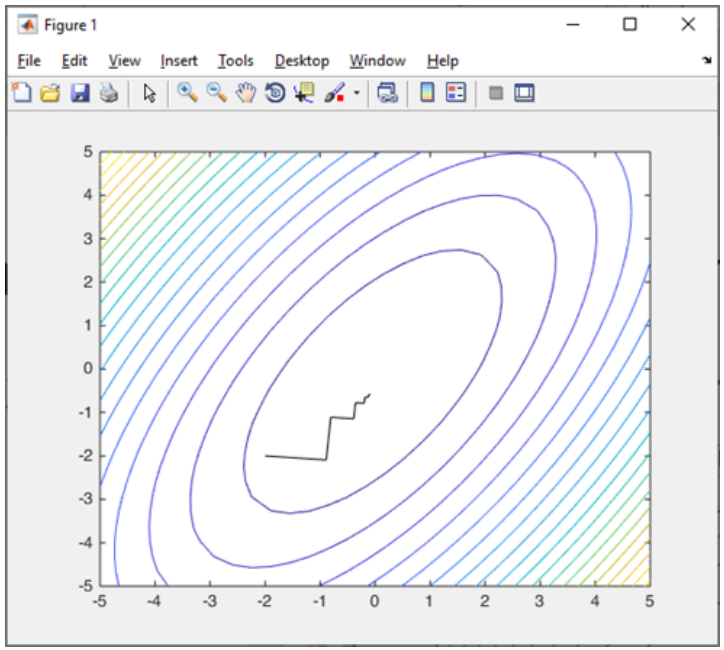
  starting from

$$x^0 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}.$$

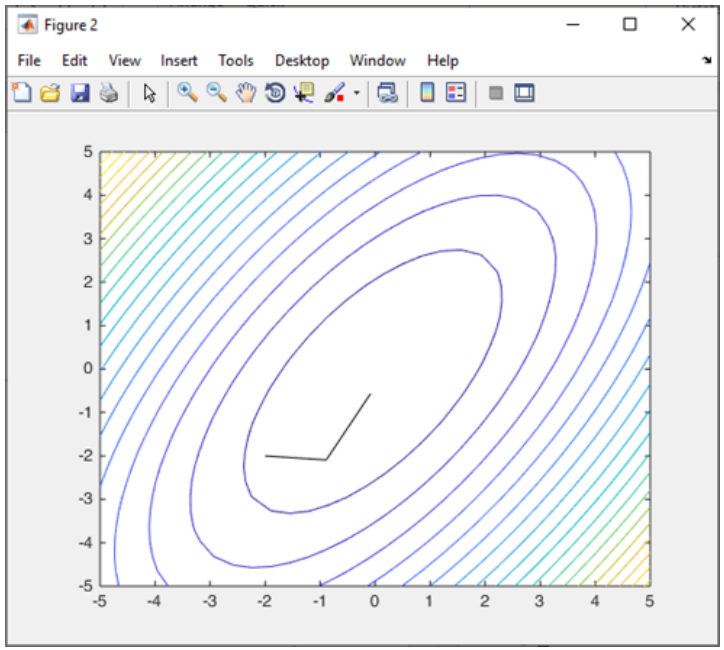
- Conjugate gradient converges in 2 steps since we are in $\mathbb{R}^2$ as shown in Figure 5.

Conjugate Gradient Method - Error Behaviour

- If you are interested in more details, our discussion borrowed heavily from Shewchuk's notes on conjugate gradient.



Steepest Descent



Conjugate Gradient

Figure: Comparison of steepest descent and conjugate gradient methods in $\mathbb{R}^2$.

Conjugate Gradient Method - Error Behaviour

An Introduction to
the Conjugate Gradient Method
Without the Agonizing Pain
Edition $1\frac{1}{4}$

Jonathan Richard Shewchuk
August 4, 1994