The Conjugate Gradient Method with Even Less Pain

Alasdair Paren

March 2025

Abstract

The Conjugate Gradient Method is the most prominent iterative method for solving sparse systems of linear equations. Many textbook treatments of the topic are written with neither illustrations nor intuition, their victims can be found to this day babbling senselessly in the corners of dusty libraries [1]. Fortunately a fantastic monograph [1] makes the elegant ideas of Conjugate Gradient Method more approachable. This set of notes is aimed to be even more basic and to the point an introduction to that wonderful introduction. Covering the bare essentials, namely the big idea behind The Conjugate Gradient Method, how the updates are derived and informally proving some of its nice properties.

1 The Conjugate Gradient Method

1.1 Problem

The Conjugate Gradient Method (CG) in its most basic form is designed to solve problems of the form:

$$A\mathbf{x}^* = \boldsymbol{b} \tag{1}$$

where $\mathbf{x}^*, \boldsymbol{b} \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ is a positive semi-definite matrix. CG is especially well suited for sparse A, where a high fraction of the entries are zero, however sparsity will be one of the last aspects of this algorithm that we cover. CG can equivalently be used to find the optimal point of well posed unconstrained quadratic problems:

$$\mathbf{x}^* \operatorname*{argmin}_{\mathbf{x}} \quad Q(\mathbf{x}) \triangleq_{\mathbf{x}} \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \boldsymbol{b}^\top \mathbf{x}. \tag{2}$$

The gradient of (2) at a point $\mathbf{x}$ has the form:

$$\nabla_{\mathbf{x}} Q(\mathbf{x}) = A\mathbf{x} - \boldsymbol{b}, \tag{3}$$

and hence finding $\nabla_{\mathbf{x}} Q(\mathbf{x}) = 0$ requires the same computation as solving (1) directly.

1.2 Notation

As CG often uses the negative gradient direction let us define:

$$\boldsymbol{g}_t \triangleq -\nabla_{\mathbf{x}} Q(\mathbf{x}) = \boldsymbol{b} - A\mathbf{x}_t. \tag{4}$$

Any point $\mathbf{x}$ that satisfies the first order optimality conditions must have:

$$0 = A\mathbf{x}^* - \boldsymbol{b}, \tag{5}$$

$$\boldsymbol{b} = A\mathbf{x}^*. \tag{6}$$

Combining this with (3) and (6) gives:

$$\boldsymbol{g}_t = -\nabla_{\mathbf{x}} Q(\mathbf{x}) = -A(\mathbf{x}_t - \mathbf{x}^*) = -A\mathbf{e}_t, \tag{7}$$

where the last equality comes from defining the difference or error between a point $\mathbf{x}_t$ and $\mathbf{x}^*$ as:

$$\mathbf{e}_t \triangleq \mathbf{x}_t - \mathbf{x}^*. \tag{8}$$

1.3 Inverting A

The most naive way to solve a problem of the form of (1) is to simply compute the inverse of A giving:

$$\mathbf{x}^* = A^{-1}\boldsymbol{b}. \tag{9}$$

However, without specialised matrix inversion algorithms[1] this approach has computational complexity $O(n^3)$, and thus is very slow for large problems. The Conjugate Gradient Method aims to do better, in fact it offers an iterative algorithms that takes T steps to compute the exact solution, where T is dependent on the sparsity, under some mild assumptions such as infinite precision. In the worst case $T = n$, this is in contrast to most iterative approaches that only asymptotically approach x^*.

1.4 A-orthogonality

Two vectors $\mathbf{x}$ and $\mathbf{y}$ are said to be A-orthogonal if for a positive definite matrix A they satisfy:

$$\mathbf{x}^T A\mathbf{y} = 0 \tag{10}$$

1.5 Big Idea

The big idea behind CG is to consider that the solution can be written in the following form:

$$\mathbf{e}_0 = \mathbf{x}_0 - \mathbf{x}^* = \sum_{i=0}^{k-1} \delta_i \boldsymbol{d}_i, \tag{11}$$

where for any $i \neq j$, $\boldsymbol{d}_i, \boldsymbol{d}_j, \in \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_k\}$ are non-zero A-orthogonal vectors, δ_i are unknown scalars nad k is rank of A. In this introduction to CG we don't formally prove that such a decomposition exists given any choice of $\boldsymbol{d}_1$.

From the definition of the eigenvectors $\{\mathbf{z}_1, \ldots, \mathbf{z}_k\}$ of a matrix A, where the eigenvectors satisfy $A\mathbf{z}_i = \lambda \mathbf{z}_i$, for some scalar λ and using the fact that the eigenvectors form an orthogonal basis it can be shown that the eigenvectors form such a decomposition, by pre-multiplying the definition by a different eigenvector $i \neq j$. However, this is a unique case where the directions are both orthogonal and A-orthogonal, in general a set of A-orthogonal vectors won't be orthogonal. Finally, it is easy to show that two co-linear directions $\boldsymbol{d}_i$ and $\boldsymbol{d}_j$ would not be A-orthogonal for any non-zero $A \succ 0$, considering a single element of the left hand side of (10).

1.6 Why A-orthogonality?

Why not just assume the directions $\boldsymbol{d}$ are orthogonal? In this section we hope to give some motivation behind what makes A-orthogonality, the right choice for problems of the form (1) and (2). Let us consider

[1] These lower the complexity $O(n^{\sim}2.5)$, but normally require, additional "adds" or additional assumptions such as n being a power of 2.

how the function value $Q(\mathbf{x})$ changes when moving along two directions unit $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ by amount α_1 and α_2 respectively:

$$Q(\mathbf{x} + \alpha_1 \boldsymbol{d}_1 + \alpha_2 \boldsymbol{d}_2) = \frac{1}{2}(\mathbf{x} + \alpha_1 \boldsymbol{d}_1 + \alpha_2 \boldsymbol{d}_2)^\top A(\mathbf{x} + \alpha_1 \boldsymbol{d}_1 + \alpha_2 \boldsymbol{d}_2) - \boldsymbol{b}^\top(\mathbf{x} + \alpha_1 \boldsymbol{d}_1 + \alpha_2 \boldsymbol{d}_2) \tag{12}$$

multiplying out the quadratic terms,

$$= \frac{1}{2}\mathbf{x}^\top A\mathbf{x} + \frac{1}{2}\alpha_1^2 \boldsymbol{d}_1^\top A\boldsymbol{d}_1 + \frac{1}{2}\alpha_2^2 \boldsymbol{d}_2^\top A\boldsymbol{d}_2 + \alpha_1 \mathbf{x}^\top A\boldsymbol{d}_1 + \alpha_2 \mathbf{x}^\top A\boldsymbol{d}_2 + \alpha_1\alpha_2 \boldsymbol{d}_1^\top A\boldsymbol{d}_2 - \boldsymbol{b}^\top(\mathbf{x} + \alpha_1 \boldsymbol{d}_1 + \alpha_2 \boldsymbol{d}_2)$$

and taking gradient with respect to α_1 and α_2 gives:

$$\nabla_{\alpha_1} Q(\mathbf{x} + \alpha_1 \boldsymbol{d}_1 + \alpha_2 \boldsymbol{d}_2) = \alpha_1 \boldsymbol{d}_1^\top A\boldsymbol{d}_1 + \mathbf{x}^\top A\boldsymbol{d}_1 + \alpha_2 \boldsymbol{d}_1^\top A\boldsymbol{d}_2 - \boldsymbol{b}^\top \boldsymbol{d}_1,$$
$$\nabla_{\alpha_2} Q(\mathbf{x} + \alpha_1 \boldsymbol{d}_1 + \alpha_2 \boldsymbol{d}_2) = \alpha_2 \boldsymbol{d}_2^\top A\boldsymbol{d}_2 + \mathbf{x}^\top A\boldsymbol{d}_2 + \alpha_1 \boldsymbol{d}_1^\top A\boldsymbol{d}_2 - \boldsymbol{b}^\top \boldsymbol{d}_2.$$

Notice how the penultimate term in each gradient expression depends on the other variable, with ∇_{α_1} being a function of α_2 and vice versa. In order to remove this dependance we need the term $\boldsymbol{d}_1^\top A\boldsymbol{d}_2$ to equal zero, which is exactly the A-orthogonality definition. With A-orthogonal directions $\boldsymbol{d}_1^\top A\boldsymbol{d}_1 = 0$, the optimal distance to move in one direction $\boldsymbol{d}_1$ is not affected by how far one might have already moved, or will later move in direction $\boldsymbol{d}_2$. Hopefully, it is clear why a decomposition of this form might be particularly useful, as we can hopefully move in each direction exactly once when finding the minimum of problem (2). This is in contrast to gradient methods such as SGD that often move in similar directions at different iterations, such as when zig-zagging down a valley, as seen in Figure 1a. While the example here only considers two directions it should be easy to see that this behaviour extends to a general set of n A-orthogonal directions. Finally, notice that the basis vectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \dots \boldsymbol{u}_n$ would only have this property for diagonal A, as if there is a non-zero off-diagonal element a_{ij}, then $\boldsymbol{u}_i^\top A\boldsymbol{u}_j = a_{ij} \neq 0$.

2 Update

Given this insight at a time step t we aim to use the update:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha_t \boldsymbol{d}_t, \tag{13}$$

where we aim to choose the directions $\boldsymbol{d}_t$ such that $\forall t, j : t > j \geq 0$, $\boldsymbol{d}_t^\top A\boldsymbol{d}_j = 0$, or, in words, we aim to choose the directions so that all $\boldsymbol{d}_t$ are A-orthogonal. We will cover how we select these directions later. We first assume we get the directions $\boldsymbol{d}_t$ from an oracle and consider how to select the step size α_t at each time step.

In order to minimise $Q(x)$ on each step we want to find where $\frac{\partial}{\partial \alpha_t} Q(\mathbf{x}_{t+1}) = \frac{\partial}{\partial \alpha_t} Q(\mathbf{x}_t + \alpha_t \boldsymbol{d}_t) = 0$. By chain rule: $\frac{\partial}{\partial \alpha_t} Q(\mathbf{x}_{t+1}) = \frac{\partial \mathbf{x}_{t+1}}{\partial \alpha_t} \cdot \frac{\partial}{\partial \mathbf{x}_{t+1}} Q(\mathbf{x}_{t+1}) = -\boldsymbol{d}_t \boldsymbol{g}_{t+1}$. Hence the minimum is found when $\boldsymbol{g}_{t+1}^\top \boldsymbol{d}_t = 0$, or in words when the gradient at the next point is orthogonal to the current search direction $\boldsymbol{d}_t$. *Note, this is worth remembering as it will prove useful later on.* Starting from this fact, we can derive the following update for α_t:

$$\begin{aligned}
\boldsymbol{g}_{t+1}^\top \boldsymbol{d}_t &= 0, \\
(b - A\mathbf{x}_{t+1})^\top \boldsymbol{d}_t &= 0, \\
(b - A(\mathbf{x}_t + \alpha_t \boldsymbol{d}_t))^\top \boldsymbol{d}_t &= 0, \\
(b - A\mathbf{x}_t + \alpha_t A\boldsymbol{d}_t)^\top \boldsymbol{d}_t &= 0, \\
(\boldsymbol{g}_t + \alpha_t A\boldsymbol{d}_t)^\top \boldsymbol{d}_t &= 0, \\
\boldsymbol{g}_t^\top \boldsymbol{d}_t + \alpha_t \boldsymbol{d}_t^\top A\boldsymbol{d}_t &= 0.
\end{aligned}$$

Rearranging gives:

$$\alpha_t = \frac{\boldsymbol{d}_t^\top \boldsymbol{g}_t}{\boldsymbol{d}_t^\top A \boldsymbol{d}_t}. \tag{14}$$

Alternatively:

$$\begin{aligned}
Q(\mathbf{x} + \alpha_t \boldsymbol{d}_t) &= \frac{1}{2}(\mathbf{x} + \alpha_t \boldsymbol{d}_t)^\top A(\mathbf{x} + \alpha_t \boldsymbol{d}_t) - \boldsymbol{b}^\top(\mathbf{x} + \alpha_t \boldsymbol{d}_t), \\
Q(\mathbf{x} + \alpha_t \boldsymbol{d}_t) &= \frac{1}{2}\mathbf{x}^\top A\mathbf{x} + \frac{1}{2}\alpha_t^2 \boldsymbol{d}_t^\top A \boldsymbol{d}_t + \alpha_t \boldsymbol{d}_t^\top A\mathbf{x} - (\mathbf{x} + \alpha_t \boldsymbol{d}_t)^\top \boldsymbol{b}, \\
\frac{\partial}{\partial \alpha_t} Q(\mathbf{x} + \alpha_t \boldsymbol{d}_t) &= \alpha_t \boldsymbol{d}_t^\top A \boldsymbol{d}_t - \boldsymbol{d}_t^\top (A\mathbf{x} - \boldsymbol{b}) = \alpha_t \boldsymbol{d}_t^\top A \boldsymbol{d}_t - \boldsymbol{d}_t^\top \boldsymbol{g}_t.
\end{aligned}$$

Setting $\frac{\partial}{\partial \alpha_t} Q(\mathbf{x} + \alpha_t \boldsymbol{d}_t) = 0$ and rearranging recovers the same result $\alpha_t = \frac{\boldsymbol{d}_t^\top \boldsymbol{g}_t}{\boldsymbol{d}_t^\top A \boldsymbol{d}_t}$.

2.1 Consequences of this step size

We now show that $\alpha_t = -\delta_t$ as defined in (11).

Proof. We start from our definition of $\mathbf{e}_0 = \sum_{i=0}^{n-1} \delta_i \boldsymbol{d}_i$ and premultiply by $\boldsymbol{d}_j^\top A$:

$$\begin{aligned}
\boldsymbol{d}_j^\top A \mathbf{e}_0 &= \sum_{i=0}^{n-1} \delta_i \boldsymbol{d}_j^\top A \boldsymbol{d}_i, \\
\boldsymbol{d}_j^\top A \mathbf{e}_0 &= \delta_j \boldsymbol{d}_j^\top A \boldsymbol{d}_j, \\
\delta_j &= \frac{\boldsymbol{d}_j^\top A \mathbf{e}_0}{\boldsymbol{d}_j^\top A \boldsymbol{d}_j}, \\
\delta_j &= \frac{\boldsymbol{d}_j^\top A \mathbf{e}_0 + \sum_{i=0}^{j-1} \alpha_i \boldsymbol{d}_j^\top A \boldsymbol{d}_i}{\boldsymbol{d}_j^\top A \boldsymbol{d}_j},
\end{aligned}$$

Note, we can add $\sum_{i=0}^{j-1} \alpha_i \boldsymbol{d}_j^\top A \boldsymbol{d}_i$ in the last line as it is is equal to 0 due to A-orthogonality. Factorising gives:

$$\delta_j = \frac{\boldsymbol{d}_j^\top A(\mathbf{e}_0 + \sum_{i=0}^{j-1} \alpha_i \boldsymbol{d}_i)}{\boldsymbol{d}_j^\top A \boldsymbol{d}_j}.$$

Notice from (8) $\mathbf{e}_t = \mathbf{x}_t - \mathbf{x}^*$ thus given $\mathbf{x}_t = \mathbf{x}_0 + \sum_{i=0}^{t-1} \alpha_i \boldsymbol{d}_i$, we get $\mathbf{e}_t = \mathbf{x}_0 + \sum_{i=0}^{t-1} \alpha_i \boldsymbol{d}_i - \mathbf{x}^*$ using $\mathbf{e}_0 = \mathbf{x}_0 - \mathbf{x}^*$ we get $\mathbf{e}_t = \mathbf{e}_0 + \sum_{i=0}^{t-1} \alpha_i \boldsymbol{d}_i$. Plugging this in gives:

$$\delta_j = \frac{\boldsymbol{d}_j^\top A \mathbf{e}_j}{\boldsymbol{d}_j^\top A \boldsymbol{d}_j} = -\frac{\boldsymbol{d}_j^\top \boldsymbol{g}_j}{\boldsymbol{d}_j^\top A \boldsymbol{d}_j}.$$

Where the last equality is a result of (7). Comparing α_t and δ_j it is clear that $\alpha_t = -\delta_t$. □

Thus if we are given one of the A-orthogonal directions at each time step by using (13) we simply remove one element from the sum on the right hand side of (11). After after taking k steps using we get $\mathbf{x}_t = \mathbf{x}_0 + \sum_{t=1}^{k-1} \alpha_t \boldsymbol{d}_t$ plugging this into $\mathbf{x}_0 = \mathbf{x}^* + \sum_{i=0}^{k-1} \delta_i \boldsymbol{d}_i$, (11) we get:

$$\mathbf{x}_t = \mathbf{x}^* + \sum_{i=0}^{k-1} \delta_i \boldsymbol{d}_i + \sum_{t=0}^{k-1} \alpha_t \boldsymbol{d}_t, = \mathbf{x}^* + \sum_{i=0}^{k-1} \delta_i \boldsymbol{d}_i - \sum_{t=0}^{k-1} \delta_t \boldsymbol{d}_t, \tag{15}$$

and hence on the k^{th} step we return a point $\mathbf{x}_k = \mathbf{x}^*$ and $\mathbf{e}_k = 0$.

3 Finding A-orthogonal Directions

One way of finding a set of A-orthogonal directions is Gram-Schmidt Conjugation. Gram-Schmidt Conjugation starts with a set of orthogonal direction $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_t\}$, say the set of axis aligned unit vectors. Then, at each step we simply select the next search direction $\boldsymbol{d}_t$ by removing the projection of all previous directions onto the next orthogonal vector $\boldsymbol{u}_t$, via an update of the form:

$$\boldsymbol{d}_t = \boldsymbol{u}_t + \sum_{k=1}^{t-1} \beta_{tk} \boldsymbol{d}_k, \tag{16}$$

where the scalars β_{tk} are defined for $k < t$. To find the values β_{tk}, we use the same trick as to find α_t transposing and post-multiplying by $A\boldsymbol{d}_j$, $\forall\ 0 < j < t$:

$$\boldsymbol{d}_t^\top A\boldsymbol{d}_j = \boldsymbol{u}_t^\top A\boldsymbol{d}_j + \sum_{k=1}^{t-1} \beta_{tk} \boldsymbol{d}_k^\top A\boldsymbol{d}_j, \tag{17}$$

$$0 = \boldsymbol{u}_t^\top A\boldsymbol{d}_j + \beta_{tj} \boldsymbol{d}_j^\top A\boldsymbol{d}_j, \tag{18}$$

$$\beta_{tj} = -\frac{\boldsymbol{u}_t^\top A\boldsymbol{d}_j}{\boldsymbol{d}_j^\top A\boldsymbol{d}_j}. \tag{19}$$

This procedure would require storing all previous directions $\boldsymbol{d}_j$ and $A\boldsymbol{d}_j$ and thus require $O(n^2)$ memory. Additionally it would also require t dot products be completed at step t making the number of multiplications required in the Gram-Schmidt Conjugation scale quadratically with the iteration count.

3.1 The Gradient is Orthogonal to Previous Search Directions

We now show the gradient at time $\boldsymbol{g}_t$ is perpendicular to all previous directions $\boldsymbol{d}_k$ for $t \geq k$.

Proof. We start from the definition of $\mathbf{e}_t$:

$$\mathbf{e}_t = \mathbf{x}_t - \mathbf{x}^* = \mathbf{x}_0 + \sum_{i=0}^{t-1} \alpha_i \boldsymbol{d}_i - \mathbf{x}^* = \mathbf{e}_0 + \sum_{i=0}^{t-1} \alpha_i \boldsymbol{d}_i = \sum_{i=0}^{n} \delta_i \boldsymbol{d}_i - \sum_{i=0}^{t-1} \delta_i \boldsymbol{d}_i = \sum_{i=t}^{n} \delta_i \boldsymbol{d}_i.$$

This lets us define the following:

$$\mathbf{e}_t = \sum_{i=t}^{n} \delta_i \boldsymbol{d}_i.$$

Pre-multiplying by $\boldsymbol{d}_j^\top A$:

$$\boldsymbol{d}_j^\top A\mathbf{e}_t = \sum_{i=t}^{n} \delta_i \boldsymbol{d}_j^\top A\boldsymbol{d}_i,$$

$$-\boldsymbol{d}_j^\top \boldsymbol{g}_t = -\sum_{i=t}^{n} \delta_i \boldsymbol{d}_j^\top A\boldsymbol{d}_i.$$

If $j \leq t$ all terms in the sum are zero due to the A-orthogonal property and we get:

$$\boldsymbol{d}_j^\top \boldsymbol{g}_t = 0,\ \forall j \leq t. \tag{20}$$

□

3.2 The Second Big Idea - The Gradient Direction form an Orthogonal Basis

The second big idea of the Conjugate Gradient Method is to use the gradient directions $\boldsymbol{g}_t$ as the orthogonal basis of vectors in the Gram-Schmidt Conjugation, instead of the set of axis align vectors $\boldsymbol{u}_t$ described above. We do this by setting:

$$\boldsymbol{d}_0 = \boldsymbol{g}_0, \qquad \boldsymbol{d}_t = \boldsymbol{g}_t + \sum_{k=1}^{t-1} \beta_{t,k} \boldsymbol{d}_k, \tag{21}$$

to construct the next direction $\beta_{t,k}$ is calculated as described in (19) with $\boldsymbol{u}_t = \boldsymbol{g}_t$. *Note, we have relabelled the indexes in the following equation*:

$$\beta_{i,j} = -\frac{\boldsymbol{g}_i^\top A \boldsymbol{d}_j}{\boldsymbol{d}_j^\top A \boldsymbol{d}_j}, \ \ \forall \ \ 0 < j < i. \tag{22}$$

We next show that the $\boldsymbol{g}_t$ vectors do indeed form an orthogonal basis by showing:

$$\boldsymbol{g}_t^\top \boldsymbol{g}_k = 0, \ \forall t > k. \tag{23}$$

Proof. From (21) it follows that the direction at time t can be written as a linear combination of all previous negative gradient directions:

$$\boldsymbol{d}_t = \sum_{k=0}^{t-1} \gamma_k \boldsymbol{g}_k,$$

where γ_k are non-zero scalars. From (20) we have that $\boldsymbol{g}_t^\top \boldsymbol{d}_j = 0, \ \forall j \leq t$, as the past directions $\boldsymbol{d}_j \forall j \leq t$ are all linear combinations of the previous gradients via induction the current negative gradient direction $\boldsymbol{g}_t$ must be orthogonal to the previous gradient directions also. □

While at first the choice to use the gradient directions as the orthogonal basis might just seem like a nice property, in fact this choice leads to $\beta_{tk} = 0$ for $t \geq k+1$, or in words, that at each time step all constants β bar $\beta_{t,t-1}$ in the Gram-Schmidt Conjugation are zero, and thus we only need to store and remove the projection of the last descent direction $\boldsymbol{d}_{t-1}$.

Proof. We start by showing that the current gradient $\boldsymbol{g}_t$ can be calculated from the previous using:

$$\boldsymbol{g}_{t+1} = -A\mathbf{e}_{t+1} = -A(\mathbf{x}_{t+1} - \mathbf{x}^*) = -A(\mathbf{x}_t + \alpha_t \boldsymbol{d}_t - \mathbf{x}^*) = -A(\mathbf{e}_t + \alpha_t \boldsymbol{d}_t) = \boldsymbol{g}_t - \alpha_t A \boldsymbol{d}_t. \tag{24}$$

Now we pre-multiply (24) by $\boldsymbol{g}_i^\top$ and arrearage, giving:

$$\boldsymbol{g}_i^\top \boldsymbol{g}_{t+1} = \boldsymbol{g}_i^\top \boldsymbol{g}_t - \alpha_t \boldsymbol{g}_i^\top A \boldsymbol{d}_t, \tag{25}$$

$$\boldsymbol{g}_i^\top A \boldsymbol{d}_t = \frac{1}{\alpha_t} \boldsymbol{g}_i^\top \boldsymbol{g}_t - \frac{1}{\alpha_t} \boldsymbol{g}_i^\top \boldsymbol{g}_{t+1}. \tag{26}$$

However form (23) we have $\boldsymbol{g}_i^\top \boldsymbol{g}_j = 0, \qquad i \neq j$, Hence we now investigate the value of $\boldsymbol{g}_i^\top A \boldsymbol{d}_t$ using (26) for different values i:

$$\boldsymbol{g}_i^\top A \boldsymbol{d}_t = \begin{cases} \frac{1}{\alpha_t} \boldsymbol{g}_i^\top \boldsymbol{g}_t, & i = t, \\ -\frac{1}{\alpha_{t-1}} \boldsymbol{g}_i^\top \boldsymbol{g}_t, & i = t+1, \\ 0, & \text{otherwise.} \end{cases} \tag{27}$$

From equation (22) we have:

$$\beta_{i,j} = -\frac{\boldsymbol{g}_i^\top A\boldsymbol{d}_j}{\boldsymbol{d}_j^\top A\boldsymbol{d}_j}, \;\; \forall \;\; 0 < j < i. \tag{28}$$

This lets us show:

$$\beta_{i,t} = \begin{cases} \frac{1}{\alpha_{t-1}} \frac{\boldsymbol{g}_i^\top \boldsymbol{g}_i}{\boldsymbol{d}_{t-1}^\top A\boldsymbol{d}_{t-1}}, & i = t+1, \\ 0, & i \geq t+1. \end{cases} \tag{29}$$

Thus the only no zero β is $\beta_{t,t-1}$:

$$\beta_{t,t-1} = \frac{1}{\alpha_{t-1}} \frac{\boldsymbol{g}_i^\top \boldsymbol{g}_i}{\boldsymbol{d}_{t-1}^\top A\boldsymbol{d}_{t-1}}.$$

□

3.3 Simplifying Alpha and Beta

While we have demonstrated that we only need to calculate $\beta_{t,t-1}$ as all other betas are zero, we can simplify its form significantly. We start by recalling the definition of α_t form:(14):

$$\alpha_t = \frac{\boldsymbol{d}_t^\top \boldsymbol{g}_t}{\boldsymbol{d}_t^\top A\boldsymbol{d}_t}.$$

Inverting and writing in terms of $t-1$ gives:

$$\frac{1}{\alpha_{t-1}} = \frac{\boldsymbol{d}_{t-1}^\top A\boldsymbol{d}_{t-1}}{\boldsymbol{d}_{t-1}^\top \boldsymbol{g}_{t-1}}.$$

This gives:

$$\beta_{t,t-1} = \frac{1}{\alpha_{t-1}} \frac{\boldsymbol{g}_i^\top \boldsymbol{g}_i}{\boldsymbol{d}_{t-1}^\top A\boldsymbol{d}_{t-1}} = \frac{\boldsymbol{d}_{t-1}^\top A\boldsymbol{d}_{t-1}}{\boldsymbol{d}_{t-1}^\top \boldsymbol{g}_{t-1}} \frac{\boldsymbol{g}_i^\top \boldsymbol{g}_i}{\boldsymbol{d}_{t-1}^\top A\boldsymbol{d}_{t-1}} = \frac{\boldsymbol{g}_i^\top \boldsymbol{g}_i}{\boldsymbol{d}_{t-1}^\top \boldsymbol{g}_{t-1}} \tag{30}$$

We now show that the search direction $\boldsymbol{d}_t$, satisfies $\boldsymbol{d}_t^\top \boldsymbol{g}_t = \boldsymbol{g}_t^\top \boldsymbol{g}_t$. Starting from (21):

$$\boldsymbol{d}_t = \boldsymbol{g}_t + \sum_{k=1}^{t-1} \beta_{tk} \boldsymbol{d}_k. \tag{31}$$

Post-multipling by $\boldsymbol{g}_t$ gives:

$$\boldsymbol{d}_t^\top \boldsymbol{g}_t = \boldsymbol{g}_t^\top \boldsymbol{g}_t + \sum_{k=1}^{t-1} \beta_{tk} \boldsymbol{d}_k^\top \boldsymbol{g}_t.$$

Using (20) where we showed $\boldsymbol{d}_i^\top \boldsymbol{g}_t = 0, \; \forall \; i \leq t$ we can write:

$$\boldsymbol{d}_t^\top \boldsymbol{g}_t = \boldsymbol{g}_t^\top \boldsymbol{g}_t. \tag{32}$$

Plugging this into the denominator of the right hand side of (30) gives:

$$\beta_{t,t-1} = \frac{\boldsymbol{g}_i^\top \boldsymbol{g}_i}{\boldsymbol{g}_{t-1}^\top \boldsymbol{g}_{t-1}} \tag{33}$$

Finally we can again use (20) to write α_t as:

$$\alpha_t = \frac{\boldsymbol{d}_t^\top \boldsymbol{g}_t}{\boldsymbol{d}_t^\top A\boldsymbol{d}_t} = \frac{\boldsymbol{g}_t^\top \boldsymbol{g}_t}{\boldsymbol{d}_t^\top A\boldsymbol{d}_t}$$

4 Algorithm

We can now put everything together into the CG Algorithm. CG starts at a point $\mathbf{x}_0$ (typically $\mathbf{x}_0 = [0, 0, \ldots, 0]^\top$) and chooses the first direction $\boldsymbol{d}_0$ to be the negative gradient direction $\boldsymbol{d}_0 = \boldsymbol{g}_0 = A\mathbf{x}_0 - \boldsymbol{b}$. CG then proceeds using the following updates:

$$\begin{aligned}
\alpha_t &= \frac{\boldsymbol{g}_t^\top \boldsymbol{g}_t}{\boldsymbol{d}_t^\top A \boldsymbol{d}_t}, \\
\mathbf{x}_{t+1} &= \mathbf{x}_t + \alpha_t \boldsymbol{d}_t, \\
\boldsymbol{g}_{t+1} &= \boldsymbol{g}_t - \alpha_t A \boldsymbol{d}_t, \\
\beta_{t+1} &= \frac{\boldsymbol{g}_{t+1}^\top \boldsymbol{g}_{t+1}}{\boldsymbol{g}_t^\top \boldsymbol{g}_t}, \\
\boldsymbol{d}_{t+1} &= \boldsymbol{g}_{t+1} - \beta_{t+1} \boldsymbol{d}_t, \\
k &= k + 1.
\end{aligned}$$

Alternatively, this can be expressed as the following algorithm:

Algorithm 1 The Conjugate Gradient Method

procedure CONJUGATE GRADIENT($A, \boldsymbol{b}, \mathbf{x}_0$, tolerance, max_iterations)
  $\mathbf{x} \leftarrow \mathbf{x}_0$, $\boldsymbol{g} \leftarrow A\mathbf{x} - \boldsymbol{b}$, $\boldsymbol{d} \leftarrow \boldsymbol{g}$, $k \leftarrow 0$
  while $\|\boldsymbol{g}\| >$ tolerance and $k <$ max_iterations **do**
    $\alpha \leftarrow \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{d}^\top A \boldsymbol{d}}$
    $\mathbf{x} \leftarrow \mathbf{x} + \alpha \boldsymbol{d}$
    $\boldsymbol{g}_{old} \leftarrow \boldsymbol{g}$
    $\boldsymbol{g} \leftarrow \boldsymbol{g} - \alpha A \boldsymbol{d}$
    $\beta \leftarrow \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}_{old}^\top \boldsymbol{g}_{old}}$
    $\boldsymbol{d} \leftarrow \boldsymbol{g} + \beta \boldsymbol{d}$
    $k \leftarrow k + 1$
  return $\mathbf{x}$

4.1 Computational Complexity and Sparsity

Computing $A\boldsymbol{d}_t$ dominates the per iteration computational complexity, at a cost of $O(n^2)$. Thus the complexity of the algorithm for any A is $O(Tn^2)$, where T is the total number of iterations. In the worst case where n steps are required, we recover the $O(n^3)$ cost of inverting A directly. This might seem discouraging at first, if A is sparse computing $A\boldsymbol{d}_t$ has a cost of $O(\Phi(A))$ where $\Phi(\cdot)$ is a function that returns the number of non-zero elements of its input.

Additionally the number of iterations required is bounded by the rank of A which we denote $\text{rank}(A)$. While we don't prove this formally, after the $\text{rank}(A)^{th}$ iteration of CG we will have found the optimal point in the linear subspace spanned by A as we have explored $\text{rank}(A)$ different directions.

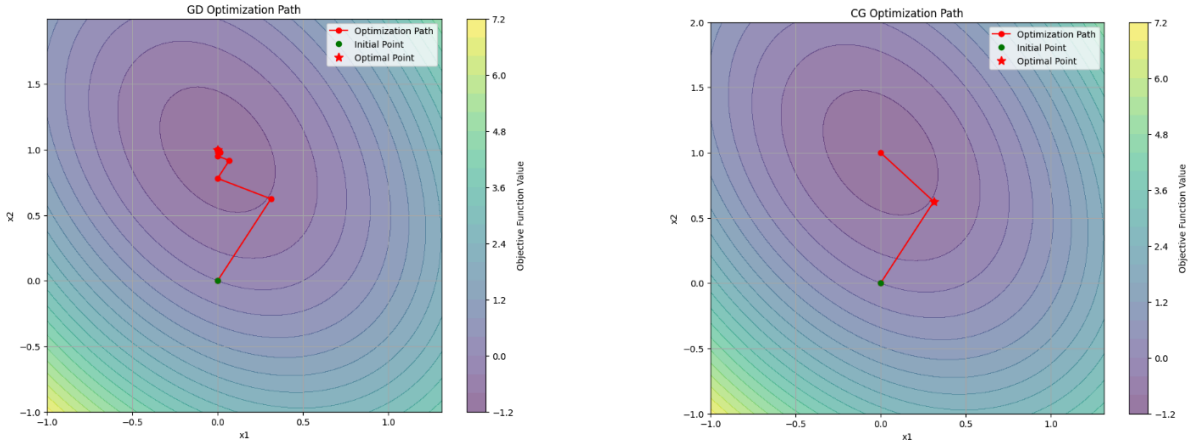
Putting this together we get the computational cost of CG is $O(\Phi(A)\text{rank}(A))$. For high sparsity levels $\frac{\Phi(A)}{n^2} <<< 1$ we get orders of magnitude savings in both computation and memory. This has lead to CG being the go to algorithm for problems of this form.

4.2 Computational Considerations

In Algorithm 1 in order to avoid computing the matrix vector product $A\mathbf{x}_{t+1}$ the gradient $\boldsymbol{g}_{t+1}$ is calculated based on the previous gradient using $\boldsymbol{g}_{t+1} = \boldsymbol{g}_t - \alpha_t A\boldsymbol{d}$, see (24). When the algorithm is run on a device with finite precision small errors can accumulate, and the iterates can drift from their theoretical trajectory. In order to avoid this, it is typical to compute the negative gradient direction exactly $\boldsymbol{g}_t = \boldsymbol{b} - A\mathbf{x}_t$ every few steps [1]. There are other useful tricks and modifications that can be applied to CG, however, these are beyond the scope of this introduction.

4.3 Empirical Comparison to Gradient Descent

While Gradient Descent (GD) and Stochastic Gradient Descent (SGD) have become the most ubiquitous optimisation algorithms. When considering problems of form (2) CG offers a significant advantage both in its theoretical properties and in practice. Figures 1a and 1b shows GD (with an optimal step size α_t) and CG respectively finding the minimum of a two dimensional quadratic, as can be seen CG only requires two steps to find the optimal point while GD approaches it asymptotically. For a more in depth comparison of these two methods we again direct the reader towards [1].



(a) Gradient Descent

(b) Conjugate Gradient Method

Figure 1: GD and CG applied to at two dimensional quadratic minimisation of form (2).

References

[1] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. 1994.