Conjugate gradient methods

Author: Alexandra Roberts, Anye Shi, Yue Sun (CHEME/SYSEN 6800, Fall 2021)

Contents

Introduction

The conjugate gradient method (CG) was originally invented to minimize a quadratic function:

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}\mathbf{x}$$

where $\mathbf{A}$ is an $n \times n$ symmetric positive definite matrix, $\mathbf{x}$ and $\mathbf{b}$ are $n \times 1$ vectors. The solution to the minimization problem is equivalent to solving the linear system, i.e. determining $\mathbf{x}$ when $\nabla F(x) = 0$, i.e. $\mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}$.

The conjugate gradient method is often implemented as an iterative algorithm and can be considered as being between Newton's method (https://en.wikipedia.org/wiki/Newton%27s_method), a second-order method that incorporates Hessian and gradient, and the method of steepest descent, a first-order method that uses gradient. Newton's Method usually reduces the number of iterations needed, but the calculation of the Hessian matrix and its inverse increases the computation required for each iteration. Steepest descent takes repeated steps in the opposite direction of the gradient of the function at the current point. It often takes steps in the same direction as earlier ones, resulting in slow convergence (Figure 1). To avoid the high computational cost of Newton's method and to accelerate the convergence rate of steepest descent, the conjugate gradient method was developed.

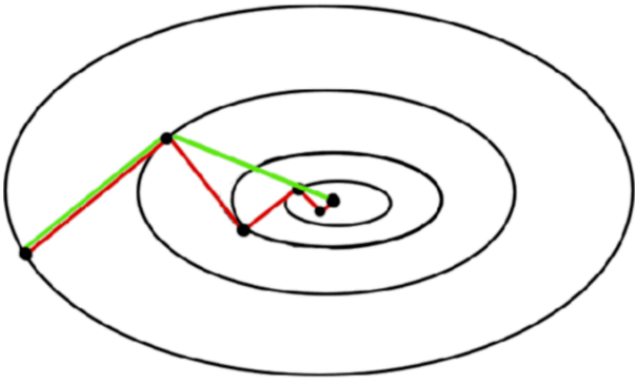

Figure 1. A comparison of the convergence of gradient descent (in red) and conjugate vector (in green) for minimizing a quadratic function. In theory, the conjugate gradient method converges in at most n steps, where n is the size of the matrix of the system (here $n = 2$).[1]

The idea of the CG method is to pick n orthogonal search directions first and, in each search direction, take exactly one step such that the step size is to the proposed solution x at that direction. The solution is reached after n steps [2] as, theoretically, the number of iterations needed by the CG method is equal to the number of different eigenvalues of $\mathbf{A}$, i.e. at most n. This makes it attractive for large and sparse problems. The method can be used to solve least-squares problems and can also be generalized to a minimization method for general smooth functions [2].

Theory

The definition of A-conjugate direction

Let $\mathbf{A}$ be a symmetric positive definite matrix. $\{\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}\}, \mathbf{d}_i \in \mathbf{R}^n, \mathbf{d}_i \neq 0$ are the search directional vectors that are orthogonal (conjugate) to each other with respect to $\mathbf{A}$ if $\mathbf{d}_i^T \mathbf{A} \mathbf{d}_j = 0, \forall i \neq j$. Note that if $\mathbf{A} = 0$, any two vectors will be conjugated to each other. If $\mathbf{A} = \mathbf{I}$, conjugacy is equivalent to the conventional notion of orthogonality. If $\{\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}\}$ are $\mathbf{A}$-conjugated to each other, then the set of vectors $\{\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}\}$ are linearly independent.

To illustrate the idea of conjugacy, the coordinate axes can be specified as search directions (Figure 2). The first step (horizontal) leads to the correct x coordinate of the solution $\mathbf{x}^*$. The second step (vertical) will reach the correct y coordinate of the solution $\mathbf{x}^*$ and then finish searching. We define

that the error $\mathbf{e}_i = \mathbf{x}^* - \mathbf{x}_i$ is a vector that indicates how far $\mathbf{x}_i$ is from the solution. The residual $\mathbf{g}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_i$ indicates how far $\mathbf{A}\mathbf{x}_i$ is from the correct value of $\mathbf{b}$. Notice that $\mathbf{e}_1$ is orthogonal to $\mathbf{d}_0$. In general, for each step we choose a point

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{d}_i \ .$$

To find the value of α_i, use the fact that $\mathbf{e}_i$ should be orthogonal to $\mathbf{d}_i$ so that no need to step in the direction of $\mathbf{d}_i$ again in the later search. Using this condition, we have

$$\mathbf{d}_i^T \mathbf{e}_{i+1} = 0$$

$$\mathbf{d}_i^T (\mathbf{e}_i + \alpha_i \mathbf{d}_i) = 0$$

$$\alpha_i = -\frac{\mathbf{d}_i^T \mathbf{e}_i}{\mathbf{d}_i^T \mathbf{d}_i}.$$

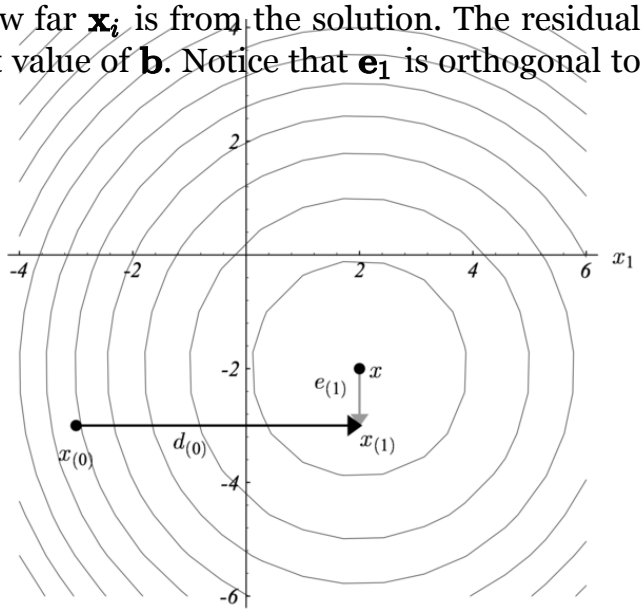

Figure 2. The method of conjugate directions. Here, we use the coordinate axes as search directions.[3]

The motivation of A-conjugacy[4]

Notice from the above equation that the error e_i should be given first to get α_i. If e_i is known, it would mean that the solution $\mathbf{x}^*$ is already known, which is problematic. So, to compute α_i without knowing $\mathbf{e}_i$, let $D = \{\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}\}$ be a set of n $\mathbf{A}$-conjugate vectors, then D can be used as a basis and express the solution $\mathbf{x}^*$ to $\nabla F(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}$ is:

$$\mathbf{x}^* = \sum_{i=0}^{n-1} \alpha_i \mathbf{d}_i$$

Then conduct transformation $\mathbf{A}$ on both sides

$$\mathbf{A}\mathbf{x}^* = \sum_{i=0}^{n-1} \alpha_i \mathbf{A}\mathbf{d}_i$$

Then multiply $\mathbf{d}_k^T$ on both sides

$$\mathbf{d}_k^T \mathbf{A}\mathbf{x}^* = \sum_{i=0}^{n-1} \alpha_i \mathbf{d}_k^T \mathbf{A}\mathbf{d}_i$$

Because $\mathbf{A}\mathbf{x} = \mathbf{b}$ and the $\mathbf{A}$-conjugacy of D, i.e. $\mathbf{d}_k^T \mathbf{A}\mathbf{d}_i = 0, \forall k \neq i$, the multiplication will cancel out all the terms except for term k

$$\mathbf{d}_k^T \mathbf{b} = \alpha_k \mathbf{d}_k^T \mathbf{A}\mathbf{d}_k$$

So α_k can be obtained as the following:

$$\alpha_k = \frac{\mathbf{d}_k^T \mathbf{b}}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k}$$

Then the solution $\mathbf{x}^*$ will be

$$\mathbf{x}^* = \sum_{i=0}^{n-1} \alpha_i \mathbf{d}_i = \sum_{i=0}^{n-1} \frac{\mathbf{d}_i^T \mathbf{b}}{\mathbf{d}_i^T \mathbf{A}\mathbf{d}_i} \mathbf{d}_i \,.$$

It takes n steps to reach $\mathbf{x}^*$. Additionally, because $\mathbf{A}$ is a symmetric and positive-definite matrix, so the term $\mathbf{d}_i^T \mathbf{A}\mathbf{d}_i$ defines an inner product and, therefore, no need to calculate the inversion of matrix $\mathbf{A}$.

Conjugate Direction Theorem

To demonstrates that the above procedure does compute $\mathbf{x}^*$ in n steps, we introduce the conjugate direction theorem. Similar as the above, let $D = \{\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}\}$ be a set of n $\mathbf{A}$-conjugate vectors, $\mathbf{x}_0 \in \mathbf{R}^n$ be a random starting point. Then the theorem is as the following:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$$

$$\mathbf{g}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$$

$$\alpha_k = \frac{\mathbf{g}_k^T \mathbf{d}_k}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k} = \frac{(\mathbf{b} - \mathbf{A}\mathbf{x}_k)^T \mathbf{d}_k}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k}$$

After n steps, $\mathbf{x}_n = \mathbf{x}^*$.

Proof:
The error e_0 from the starting point to the solution can be formulated as the following:

$$\mathbf{x}^* - \mathbf{x}_0 = \alpha_0 \mathbf{d}_0 + \alpha_1 \mathbf{d}_1 + \ldots + \alpha_{n-1} \mathbf{d}_{n-1}$$

And the traversed steps from the starting point to the point $\mathbf{x}_k$ can be expressed as the following:

$$\mathbf{x}_k - \mathbf{x}_0 = \alpha_0 \mathbf{d}_0 + \alpha_1 \mathbf{d}_1 + \ldots + \alpha_{k-1} \mathbf{d}_{k-1}$$

The residual term from the solution point to the point $\mathbf{x}_k$ can be expressed as the following:

$$\mathbf{g}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k = \mathbf{A}\mathbf{x}^* - \mathbf{A}\mathbf{x}_k = \mathbf{A}(\mathbf{x}^* - \mathbf{x}_k)$$

Therefore, the α_k is as the following:

$$\mathbf{d}_k^T \mathbf{A}(\mathbf{x}^* - \mathbf{x}_0) = \mathbf{d}_k^T \mathbf{A}(\alpha_0 \mathbf{d}_0 + \alpha_1 \mathbf{d}_1 + \ldots + \alpha_{n-1} \mathbf{d}_{n-1}) = \alpha_k \mathbf{d}_k^T \mathbf{A} \mathbf{d}_k$$

$$\alpha_k = \frac{\mathbf{d}_k^T \mathbf{A}(\mathbf{x}^* - \mathbf{x}_0)}{\mathbf{d}_k^T \mathbf{A} \mathbf{d}_k}$$

But, still, the solution $\mathbf{x}^*$ has to be known first to calculate α_k. So, the following manipulations can get rid of using $\mathbf{x}^*$ in the above expression:

$$\mathbf{d}_k^T \mathbf{A}(\mathbf{x}_k - \mathbf{x}_0) = \mathbf{d}_k^T \mathbf{A}(\alpha_0 \mathbf{d}_0 + \alpha_1 \mathbf{d}_1 + \ldots + \alpha_{k-1} \mathbf{d}_{k-1}) = 0$$

$$\mathbf{d}_k^T \mathbf{A}(\mathbf{x}^* - \mathbf{x}_0) = \mathbf{d}_k^T \mathbf{A}(\mathbf{x}^* - \mathbf{x}_k + \mathbf{x}_k - \mathbf{x}_0) = \mathbf{d}_k^T \mathbf{A}(\mathbf{x}^* - \mathbf{x}_k)$$

Therefore

$$\alpha_k = \frac{\mathbf{d}_k^T \mathbf{A}(\mathbf{x}^* - \mathbf{x}_0)}{\mathbf{d}_k^T \mathbf{A} \mathbf{d}_k} = \frac{\mathbf{d}_k^T \mathbf{A}(\mathbf{x}^* - \mathbf{x}_k)}{\mathbf{d}_k^T \mathbf{A} \mathbf{d}_k} = \frac{\mathbf{d}_k^T \mathbf{g}_k}{\mathbf{d}_k^T \mathbf{A} \mathbf{d}_k}$$

The conjugate gradient method

The conjugate gradient method is a conjugate direction method in which selected successive direction vectors are treated as a conjugate version of the successive gradients obtained while the method progresses. The conjugate directions are not specified beforehand but rather are determined sequentially at each step of the iteration [4]. If the conjugate vectors are carefully chosen, then not all

the conjugate vectors may be needed to obtain the solution. Therefore, the conjugate gradient method is regarded as an iterative method. This also allows approximate solutions to systems where n is so large that the direct method requires too much time [2].

Derivation of Algorithm[2]

Given $D = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_n\}$ be a set of n $\mathbf{A}$-conjugate vectors, then $F(\mathbf{x}_0 + \alpha_1 \mathbf{d}_1 + \alpha_2 \mathbf{d}_2 + \ldots + \alpha_n \mathbf{d}_n)$ can be minimized by stepping from $\mathbf{x}_0$ along $\mathbf{d}_1$ to the minimum $\mathbf{x}_1$, stepping from $\mathbf{x}_1$ along $\mathbf{d}_2$ to the minimum $\mathbf{x}_2$, etc. And let $\mathbf{x}_0 \in \mathbf{R}_n$ be randomly chosen, then the algorithm is the following:

$Algo\ 1: Pick\ \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_n\}\ mutually\ \mathbf{A}-conjugate,\ and\ start\ from\ a\ random\ \mathbf{x}_0$

Set a starting point $\mathbf{x}_0$

Set $k = 1$

Set the maximum iteration n

While $k \leq n$**:**

$\alpha_k = \mathbf{d}_k^T(\mathbf{b} - \mathbf{A}\mathbf{x}_{k-1}) / \mathbf{d}_k^T \mathbf{A} \mathbf{d}_k$

$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{d}_k$

$k = k + 1$

End while

Return $\mathbf{x}_n$

Here *Alg 1* is with a particular choice of $\{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_n\}$. Let $\mathbf{g}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$ be the residual (equivalent to the negative of the gradient) at $\mathbf{x}_k$. A practical way to enforce this is by requiring that the next search direction be built out of the current gradient and all previous search directions. The CG method picks $\mathbf{d}_{k+1}$ as the component of $\mathbf{g}_k$ $\mathbf{A}$-conjugate to $\{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_k\}$:

$$\mathbf{d}_{k+1} = \mathbf{g}_k - \sum_{i=1}^{k} \frac{\mathbf{g}_k^T \mathbf{A} \mathbf{d}_i}{\mathbf{d}_i^T \mathbf{A} \mathbf{d}_i} \mathbf{d}_i$$

As $\mathbf{g}_k^T \mathbf{A}\mathbf{d}_i = 0$, for $i = 1, \ldots, k$, giving the following CG algorithm:

Alg 2 : Revised algorithm from Alg 1 without particular choice of D

Set a starting point $\mathbf{x}_0$

Set $k = 1$

Set the maximum iteration n

While $k \leq n$**:**

$\mathbf{g}_{k-1} = \mathbf{b} - \mathbf{A}\mathbf{x}_{k-1}$

if $\mathbf{g}_{k-1} = 0$ return $\mathbf{x}_{k-1}$

if (*k > 1*) $\beta_k = \mathbf{d}_{k-1}^T \mathbf{A}\mathbf{g}_{k-1} / \mathbf{d}_{k-1}^T \mathbf{A}\mathbf{d}_{k-1}$

if (*k = 1*) $\mathbf{d}_k = \mathbf{g}_0$

else $\mathbf{d}_k = \mathbf{g}_{k-1} - \beta_k \mathbf{d}_{k-1}$

$\alpha_k = \mathbf{d}_k^T \mathbf{g}_{k-1} / \mathbf{d}_k^T \mathbf{A}\mathbf{d}_k$

$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{d}_k$

$k = k + 1$

End while

Return $\mathbf{x}_n$

The formulas in the *Alg 2* can be simplified as the following:

$$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i \mathbf{d}_i$$

$$\mathbf{b} - \mathbf{A}\mathbf{x}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_{i-1} - \alpha_i \mathbf{A}\mathbf{d}_i$$

$$\mathbf{g}_i = \mathbf{g}_{i-1} - \alpha_i \mathbf{A}\mathbf{d}_i$$

Then β_i and α_i can be simplified by multiplying the above gradient formula by $\mathbf{g}_i$ and $\mathbf{g}_{i-1}$ as the following:

$$\mathbf{g}_i^T \mathbf{g}_i = -\alpha_i \mathbf{g}_i^T \mathbf{A}\mathbf{d}_i$$

$$\mathbf{g}_{i-1}^T \mathbf{g}_{i-1} = \alpha_i \mathbf{g}_{i-1}^T \mathbf{A}\mathbf{d}_i$$

As $\mathbf{g}_{i-1} = \mathbf{d}_i + \beta_i \mathbf{d}_{i-1}$, so we have

$$\mathbf{g}_{i-1}^T \mathbf{g}_{i-1} = \alpha_i \mathbf{g}_{i-1}^T \mathbf{A}\mathbf{d}_i = \alpha_i \mathbf{d}_i^T \mathbf{A}\mathbf{d}_i$$

Therefore

$$\beta_{i+1} = -\frac{\mathbf{g}_i^T \mathbf{g}_i}{\mathbf{g}_{i-1}^T \mathbf{g}_{i-1}}$$

This gives the following simplified version of **Alg 2**:

Alg 3 : Simplified version from Alg 2

Set a starting point $\mathbf{x}_0$

Set $\mathbf{g}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$

Set $k = 1$

Set the maximum iteration n

While $k \leq n$**:**

if $\mathbf{g}_{k-1} = 0$ return $\mathbf{x}_{k-1}$

if ($k > 1$) $\beta_k = -(\mathbf{g}_{k-1}^T \mathbf{g}_{k-1})/(\mathbf{g}_{k-2}^T \mathbf{g}_{k-2})$

if ($k = 1$) $\mathbf{d}_k = \mathbf{g}_0$

else $\mathbf{d}_k = \mathbf{g}_{k-1} - \beta_k \mathbf{d}_{k-1}$

$\alpha_k = (\mathbf{g}_{k-1}^T \mathbf{g}_{k-1})/(\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k)$

$\mathbf{x}_k = \mathbf{x}_{i-1} + \alpha_i \mathbf{d}_i$

$\mathbf{g}_i = \mathbf{g}_{i-1} - \alpha_i \mathbf{A}\mathbf{d}_i$

$k = k + 1$

End while

Return $\mathbf{x}_n$

Numerical example

Consider the linear system $\mathbf{Ax} = \mathbf{b}$

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} 5 & 1 \\ 1 & 8 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \mathbf{b}\,.$$

The initial starting point is set to be

$$\mathbf{x}_0 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

Implement the conjugate gradient method to approximate the solution to the system.

Solution:
The exact solution is given below for later reference:

$$\mathbf{x}_* = \begin{bmatrix} 22/39 \\ 7/39 \end{bmatrix} \approx \begin{bmatrix} 0.5641 \\ 0.1794 \end{bmatrix}.$$

Step 1: since this is the first iteration, use the residual vector $\mathbf{g}_0$ as the initial search direction $\mathbf{d}_1$. The method of selecting $\mathbf{d}_k$ will change in further iterations.

$$\mathbf{g}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 = \begin{bmatrix} 3 \\ 2 \end{bmatrix} - \begin{bmatrix} 5 & 1 \\ 1 & 8 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -8 \\ -8 \end{bmatrix} = \mathbf{d}_1$$

Step 2: the scalar α_1 can be calculated using the relationship

$$\alpha_1 = \frac{\mathbf{g}_0^T \mathbf{g}_0}{\mathbf{d}_1^T \mathbf{A} \mathbf{d}_1} = \frac{\begin{bmatrix} -8 & -8 \end{bmatrix} \begin{bmatrix} -8 \\ -8 \end{bmatrix}}{\begin{bmatrix} -8 & -8 \end{bmatrix} \begin{bmatrix} 5 & 1 \\ 1 & 8 \end{bmatrix} \begin{bmatrix} -8 \\ -8 \end{bmatrix}} = \frac{2}{15}$$

Step 3: so $\mathbf{x_1}$ can be reached by the following formula, and this step finishes the first iteration, the result being an "improved" approximate solution to the system, $\mathbf{x_1}$:

$$\mathbf{x}_1 = \mathbf{x}_0 + \alpha_1 \mathbf{d}_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \frac{2}{15} \begin{bmatrix} -8 \\ -8 \end{bmatrix} = \begin{bmatrix} 0.9333 \\ -0.0667 \end{bmatrix}$$

Step 4: now move on and compute the next residual vector $\mathbf{g_1}$ using the formula

$$\mathbf{g}_1 = \mathbf{g}_0 - \alpha_1 \mathbf{A} \mathbf{d}_1 = \begin{bmatrix} -8 \\ -8 \end{bmatrix} - \frac{2}{15} \begin{bmatrix} 5 & 1 \\ 1 & 8 \end{bmatrix} \begin{bmatrix} -8 \\ -8 \end{bmatrix} = \begin{bmatrix} -1.6 \\ 1.6 \end{bmatrix}$$

Step 5: compute the scalar β_2 that will eventually be used to determine the next search direction $\mathbf{d_2}$.

$$\beta_2 = -\frac{\mathbf{g}_1^T \mathbf{g}_1}{\mathbf{g}_0^T \mathbf{g}_0} = -\frac{\begin{bmatrix} -1.6 & 1.6 \end{bmatrix} \begin{bmatrix} -1.6 \\ 1.6 \end{bmatrix}}{\begin{bmatrix} -8 & -8 \end{bmatrix} \begin{bmatrix} -8 \\ -8 \end{bmatrix}} = -0.04$$

Step 6: use this scalar β_2 to compute the next search direction $\mathbf{d_2}$:

$$\mathbf{d}_2 = \mathbf{g}_1 - \beta_2 \mathbf{d}_1 = \begin{bmatrix} -1.6 \\ 1.6 \end{bmatrix} + 0.04 \begin{bmatrix} -8 \\ -8 \end{bmatrix} = \begin{bmatrix} -1.92 \\ 1.28 \end{bmatrix}$$

Step 7: now compute the scalar α_2 using newly acquired $\mathbf{d_2}$ by the same method as that used for α_1.

$$\alpha_2 = \frac{\mathbf{g}_1^T \mathbf{g}_1}{\mathbf{d}_2^T \mathbf{A} \mathbf{d}_2} = \frac{\begin{bmatrix} -1.6 & 1.6 \end{bmatrix} \begin{bmatrix} -1.6 \\ 1.6 \end{bmatrix}}{\begin{bmatrix} -1.92 & 1.28 \end{bmatrix} \begin{bmatrix} 5 & 1 \\ 1 & 8 \end{bmatrix} \begin{bmatrix} -1.92 \\ 1.28 \end{bmatrix}} = 0.1923$$

Step 8: finally, find $\mathbf{x_2}$ using the same method as that used to find $\mathbf{x_1}$.

$$\mathbf{x}_2 = \mathbf{x}_1 + \alpha_2 \mathbf{d}_2 = \begin{bmatrix} 0.9333 \\ -0.0667 \end{bmatrix} + 0.1923 \begin{bmatrix} -1.92 \\ 1.28 \end{bmatrix} = \begin{bmatrix} 0.5641 \\ 0.1794 \end{bmatrix}$$

Therefore, $\mathbf{x_2}$ is the approximation result of the system.

Application

Conjugate gradient methods have often been used to solve a wide variety of numerical problems, including linear and nonlinear algebraic equations, eigenvalue problems and minimization problems. These applications have been similar in that they involve large numbers of variables or dimensions. In these circumstances any method of solution which involves storing a full matrix of this large order, becomes inapplicable. Thus recourse to the conjugate gradient method may be the only alternative [5]. Here we demonstrate three application examples of CG method.

Iterative image reconstruction

The conjugate gradient method is used to solve for the update in iterative image reconstruction problems. For example, in the magnetic resonance imaging (MRI) contrast known as quantitative susceptibility mapping (QSM), the reconstructed image χ is iteratively solved for from magnetic field data $\mathbf{b}$ by the relation[6]

$$\mathbf{b} = \mathbf{D}\chi$$

Where D is the matrix expressing convolution with the dipole kernel in the Fourier domain. Given that the problem is ill-posed, a physical prior is used in the reconstruction, which is framed as a constrained L1 norm minimization

$$min_{\chi}\|f(\chi)\|_1$$

$$s.t.\ \|g(\chi) - c\|_2^2$$

A detailed treatment of the function $f(\chi)$ and $g(\chi)$ can be found at [6]. This problem can be expressed as an unconstrained minimization problem via the Lagrange Multiplier Method

$$min_{\chi,\lambda}E(\chi,\lambda)$$

Where

$$E(\chi,\lambda) \equiv \|f(\chi)\|_1 + \lambda(\|g(\chi) - c\|_2^2 - \varepsilon)$$

The first-order conditions require $\nabla_{\chi}E(\chi,\lambda) = 0$ and $\nabla_{\lambda}E(\chi,\lambda) = 0$. These conditions result in $\nabla_{\chi}f(\chi) + \nabla_{\chi}g(\chi) - \tilde{c} = 0$ and $\|g(\chi) - c\|_2^2 \approx \varepsilon$, respectively. The update can be solved for $\nabla_{\chi}f(\chi) + \nabla_{\chi}g(\chi) - \tilde{c} = L(\chi)\chi - \tilde{c} = 0$ via fixed point iteration [6].

$$\chi_{n+1} = L^{-1}(\chi^n)\tilde{c}$$

And expressed as the quasi-Newton problem, more robust to round-off error [6]

$$\chi_{n+1} = \chi_n - L^{-1}(\chi_n)\nabla E(\chi_n, \lambda)$$

Which is solved with the CG method until the residual $\|\chi_{n+1} - \chi\|_2 / \|\chi_n\|_2 \leq \theta$ where θ is a specified tolerance, such as 10^{-2}.

Additional L1 terms, such as a downsampled term [7] can be added, in which case the cost function is treated with penalty methods and the CG method is paired with Gauss-Newton to address nonlinear terms.

Facial recognition

The realization of face recognition can be achieved by the implementation of conjugate gradient algorithms with the combination of other methods. The basic steps is to decompose images into a set of time-frequency coefficients using discrete wavelet transform (DWT) (Figure 3)[8]. Then use basic back propagation (BP) to train a neural network (NN). And to overcome the slow convergence of BP using the steepest gradient descent, conjugate gradient methods are introduced. Generally, there are four types of CG methods for training a feed-foward NN, namely, Fletcher-Reeves CG, Polak-Ribikre CG, Powell-Beale CG, and scaled CG. All the CG methods include the steps demonstrated in *Alg 3* with their respective modifications[8][10].

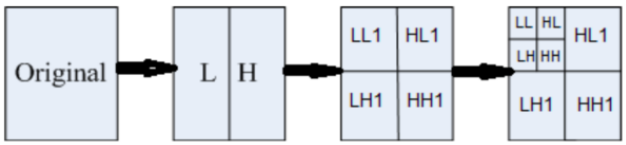

Figure 3. The process of decomposing an image. DWT plays a significant role in reducing the dimension of an image and extract the features by decomposing an image in frequency domain into sub-bands at different scales. The DWT of an image is created as follows: in the first level of decomposition, the image is split into four sub-bands, namely HH1, HL1, LH1, and LL1, as shown in the figure. The HH1, HL1 and LH1 sub-bands represent the diagonal details, horizontal features and vertical structures of the image, respectively. The LL1 sub-band is the low resolution residual consisting of low frequency components and it is this sub-band which is further split at higher levels of decomposition.[8][9]

Notice that β_k is a scalar that varies in different versions of CG methods. β_k for Fletcher-Reeves CG is defined as the following:

$$\beta_{i+1} = -\frac{\mathbf{g}_i^T \mathbf{g}_i}{\mathbf{g}_{i-1}^T \mathbf{g}_{i-1}}$$

where $\mathbf{g}_{i-1}$ and $\mathbf{g}_i$ are the previous and current gradients, respectively. Fletcher Reeves suggested restarting the CG after every n or n+1 iterations. This suggestion can be very helpful in practice[10].

The second version of the CG, namely, Polak-Ribiere CG, was proposed in 1977[11]. In this algorithm, the scalar at each iteration is computed as[12]:

$$\beta_{i+1} = -\frac{\Delta \mathbf{g}_{i-1}^T \mathbf{g}_i}{\mathbf{g}_{i-1}^T \mathbf{g}_{i-1}}$$

Powell has showed that Polak-Ribiere CG method has better performance compared with Fletcher-Reeves CG method in many numerical experiments[10].

The mentioned CGs, Fletcher-Reeves and Polak-Ribitre algorithms, periodically restart by using the steepest descent direction every n or $n+1$ iterations. A disadvantage of the periodical restarting is that the immediate reduction in the objective function is usually less than it would be without restart. Therefore, Powell proposed a new method of restart based on an earlier version[10][12]. As stated by this method, when there is negligibility orthogonality left between the current gradient and the previous gradient (the following condition is satisfied), the restart occurs[10][12].

Another version of the CG, scaled CG, was proposed by Moller. This algorithm uses a step size scaling mechanism that avoids a time consuming line-search per learning iteration. Moller[13] has showed that the scaled CG method has superlinear convergence for most problems.

Conjugate gradient method in deep learning

The conjugate gradient method introduced hyperparameter optimization in deep learning algorithm can be regarded as something intermediate between gradient descent and Newton's method, which does not require storing, evaluating, and inverting the Hessian matrix, as it does Newton's method. Optimization search in conjugate gradient is performed along with conjugate directions. They generally produce faster convergence than gradient descent directions. Training directions are conjugated concerning the Hessian matrix.

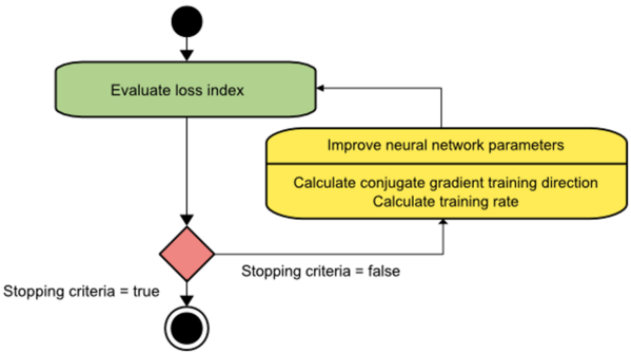

Figure 4. Activity diagram for the training process with the conjugate gradient.[14]

Let's denote $\mathbf{d}$ as the training direction vector. Starting with an initial parameter vector $\mathbf{w_0}$ and an initial training direction vector $\mathbf{d_0} = -\mathbf{g_0}$, the conjugate gradient method constructs a sequence of training direction as:

$$\mathbf{d}_{i+1} = \mathbf{g}_{i+1} + \gamma_i \mathbf{d}_i \qquad for\ i = 0, 1, \ldots$$

Here γ is called the called the conjugate parameter. For all conjugate gradient algorithms, the training direction is periodically reset to the negative of the gradient. The parameters are then improved according to the following expression:

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \eta_i \mathbf{d}_i \qquad for\ i = 0, 1, \ldots$$

The training rate η is usually found by line minimization. Figure 4 depicts an activity diagram for the training process with the conjugate gradient. To improve the parameters, first compute the conjugate gradient training direction. Then, search for a suitable training rate in that direction. This method has proved to be more effective than gradient descent in training neural networks. Also, conjugate gradient performs well with vast neural networks since it does not require the Hessian matrix.

Conclusion

The conjugate gradient method was invented to avoid the high computational cost of Newton's method and to accelerate the convergence rate of steepest descent. As an iterative method, each step only requires $\mathbf{Ad}_i$ multiplication free from the storage of matrix $\mathbf{A}$. And selected direction vectors are treated as a conjugate version of the successive gradients obtained while the method progresses. So it monotonically improves approximations $\mathbf{x}_k$ to the exact solution and may reach the required tolerance after a relatively small (compared to the problem size) number of iterations in the absence of round-off error (https://en.wikipedia.org/wiki/Round-off_error), which makes it widely used for solving large and sparse problems. Because of the high flexibility of the method framework, variants of CG algorithms have been proposed and can be applied to a variety of applications in different fields, such as machine learning and deep learning, in order to enhance the algorithm performance.

Reference

1. Refsnæs, Runar Heggelien. "A Brief introduction to the conjugate gradient method." (2009).
2. W. Stuetzle, "The Conjugate Gradient Method." 2001. [Online]. Available: https://sites.stat.washington.edu/wxs/Stat538-w03/conjugate-gradients.pdf
3. Jonathan Shewchuk, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain," 1994.
4. A. Singh and P. Ravikumar, "Conjugate Gradient Descent." 2012. [Online]. Available: http://www.cs.cmu.edu/~pradeepr/convexopt/Lecture_Slides/conjugate_direction_methods.pdf
5. R. Fletcher, "Conjugate gradient methods for indefinite systems," in Numerical Analysis, Berlin, Heidelberg, 1976, pp. 73–89, doi: 10.1007/BFb0080116

Heidelberg, 1976, pp. 73–89. doi: 10.1007/BFb0080116.
6. T. Liu et al., “Morphology enabled dipole inversion for quantitative susceptibility mapping using structural consistency between the magnitude image and the susceptibility map,” NeuroImage, vol. 59, no. 3, pp. 2560–2568, Feb. 2012, doi: 10.1016/j.neuroimage.2011.08.082.
7. A. Roberts, P. Spincemaille, T. Nguyen, Y. Wang, “International Society for Magnetic Resonance in Medicine,” in MEDI-d: Downsampled Morphological Priors for Shadow Reduction in Quantitative Susceptibility Mapping, 2021.
8. H. Azami, M. Malekzadeh, and S. Sanei, “A New Neural Network Approach for Face Recognition Based on Conjugate Gradient Algorithms and Principal Component Analysis,” Journal of Mathematics and Computer Science, vol. 6, no. 3, pp. 166–175, 2013, doi: 10.22436/jmcs.06.03.01.
9. H. Azami, M. R. Mosavi, S. Sanei, Classification of GPS satellites using improved back propagation training algorithms, Wireless Personal Communications, Springer-Verlog, DOI 10.1007/s11277-012-0844-7 (2012)
10. M. H. Shaheed, Performance analysis of 4 types of conjugate gradient algorithms in the nonlinear dynamic modelling of a TRMS using feedforward neural networks, IEEE Conference on Systems, Man and Cybernetics, (2004), 5985-5990
11. S. Sheel, T. Varshney, R. Varshney, Accelerated learning in MLP using adaptive learning rate with momentum coefficient, International Conference on Industrial and Information Systems, (2007), 307-310
12. Z. Zakaria, N. A. M. Isa, S. A. Suandi, A study on neural network training algorithm for multiface detection in static images, International Conference on Computer, Electrical, Systems Science, and Engineering, (2010), 170-173
13. M. F. Moller, A scaled conjugate gradient algorithm for fast supervised learning, Neural Networks, 6 (1993), 525-533
14. Quesada and Artelnics, “5 Algorithms to Train a Neural Network.” https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network

---