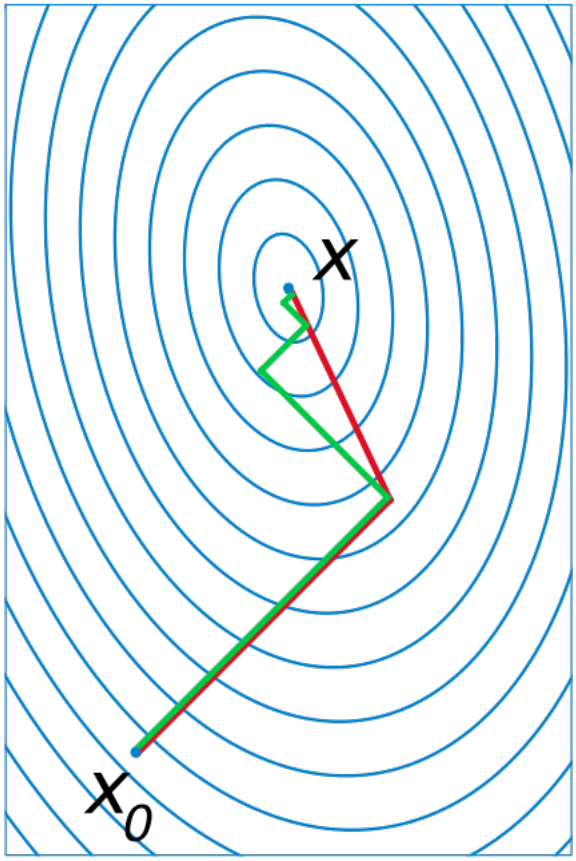
Conjugate gradient method

In mathematics, the **conjugate gradient method** is an algorithm for the numerical solution of particular systems of linear equations, namely those whose matrix is positive-definite. The conjugate gradient method is often implemented as an iterative algorithm, applicable to sparse systems that are too large to be handled by a direct implementation or other direct methods such as the Cholesky decomposition. Large sparse systems often arise when numerically solving partial differential equations or optimization problems.

The conjugate gradient method can also be used to solve unconstrained optimization problems such as energy minimization. It is commonly attributed to Magnus Hestenes and Eduard Stiefel,[1][2] who programmed it on the Z4,[3] and extensively researched.[4][5]

The biconjugate gradient method provides a generalization to non-symmetric matrices. Various nonlinear conjugate gradient methods seek minima of nonlinear equations and black-box objective functions.



A comparison of the convergence of gradient descent with optimal step size (in green) and conjugate vector (in red) for minimizing a quadratic function associated with a given linear system. Conjugate gradient, assuming exact arithmetic, converges in at most n steps, where n is the size of the matrix of the system (here $n = 2$).

Contents

Description of the problem addressed by conjugate gradients

Suppose we want to solve the system of linear equations

$$\mathbf{Ax} = \mathbf{b}$$

for the vector $\mathbf{x}$, where the known $n \times n$ matrix $\mathbf{A}$ is symmetric (i.e., $\mathbf{A}^{\mathrm{T}} = \mathbf{A}$), positive-definite (i.e. $\mathbf{x}^{\mathrm{T}}\mathbf{Ax} > 0$ for all non-zero vectors $\mathbf{x}$ in $\mathbf{R}^n$), and real, and $\mathbf{b}$ is known as well. We denote the unique solution of this system by $\mathbf{x}_*$.

Derivation as a direct method

The conjugate gradient method can be derived from several different perspectives, including specialization of the conjugate direction method for optimization, and variation of the Arnoldi/Lanczos iteration for eigenvalue problems. Despite differences in their approaches, these derivations share a common topic—proving the orthogonality of the residuals and conjugacy of the search directions. These two properties are crucial to developing the well-known succinct formulation of the method.

We say that two non-zero vectors $\mathbf{u}$ and $\mathbf{v}$ are conjugate (with respect to $\mathbf{A}$) if

$$\mathbf{u}^{\mathsf{T}}\mathbf{A}\mathbf{v} = 0.$$

Since $\mathbf{A}$ is symmetric and positive-definite, the left-hand side defines an inner product

$$\mathbf{u}^{\mathsf{T}}\mathbf{A}\mathbf{v} = \langle \mathbf{u}, \mathbf{v} \rangle_{\mathbf{A}} := \langle \mathbf{A}\mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{u}, \mathbf{A}^{\mathsf{T}}\mathbf{v} \rangle = \langle \mathbf{u}, \mathbf{A}\mathbf{v} \rangle.$$

Two vectors are conjugate if and only if they are orthogonal with respect to this inner product. Being conjugate is a symmetric relation: if $\mathbf{u}$ is conjugate to $\mathbf{v}$, then $\mathbf{v}$ is conjugate to $\mathbf{u}$. Suppose that

$$P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$$

is a set of n mutually conjugate vectors (with respect to $\mathbf{A}$). Then P forms a basis for $\mathbb{R}^n$, and we may express the solution $\mathbf{x}_*$ of $\mathbf{Ax} = \mathbf{b}$ in this basis:

$$\mathbf{x}_* = \sum_{i=1}^{n} \alpha_i \mathbf{p}_i.$$

Based on this expansion we calculate:

$$\mathbf{A}\mathbf{x}_* = \sum_{i=1}^{n} \alpha_i \mathbf{A}\mathbf{p}_i.$$

Left-multiplying by $\mathbf{p}_k^\mathsf{T}$:

$$\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{x}_* = \sum_{i=1}^{n} \alpha_i \mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_i,$$

substituting $\mathbf{A}\mathbf{x}_* = \mathbf{b}$ and $\mathbf{u}^\mathsf{T}\mathbf{A}\mathbf{v} = \langle \mathbf{u}, \mathbf{v} \rangle_\mathbf{A}$:

$$\mathbf{p}_k^\mathsf{T}\mathbf{b} = \sum_{i=1}^{n} \alpha_i \langle \mathbf{p}_k, \mathbf{p}_i \rangle_\mathbf{A},$$

then $\mathbf{u}^\mathsf{T}\mathbf{v} = \langle \mathbf{u}, \mathbf{v} \rangle$ and using $\forall i \neq k : \langle \mathbf{p}_k, \mathbf{p}_i \rangle_\mathbf{A} = 0$ yields

$$\langle \mathbf{p}_k, \mathbf{b} \rangle = \alpha_k \langle \mathbf{p}_k, \mathbf{p}_k \rangle_\mathbf{A},$$

which implies

$$\alpha_k = \frac{\langle \mathbf{p}_k, \mathbf{b} \rangle}{\langle \mathbf{p}_k, \mathbf{p}_k \rangle_\mathbf{A}}.$$

This gives the following method[4] for solving the equation $\mathbf{A}\mathbf{x} = \mathbf{b}$: find a sequence of n conjugate directions, and then compute the coefficients α_k.

As an iterative method

If we choose the conjugate vectors $\mathbf{p}_k$ carefully, then we may not need all of them to obtain a good approximation to the solution $\mathbf{x}_*$. So, we want to regard the conjugate gradient method as an iterative method. This also allows us to approximately solve systems where n is so large that the direct method would take too much time.

We denote the initial guess for $\mathbf{x}_*$ by $\mathbf{x}_0$ (we can assume without loss of generality that $\mathbf{x}_0 = \mathbf{0}$, otherwise consider the system $\mathbf{A}\mathbf{z} = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ instead). Starting with $\mathbf{x}_0$ we search for the solution and in each iteration we need a metric to tell us whether

we are closer to the solution $\mathbf{x}_*$ (that is unknown to us). This metric comes from the fact that the solution $\mathbf{x}_*$ is also the unique minimizer of the following quadratic function

$$f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} - \mathbf{x}^\mathsf{T}\mathbf{b}, \qquad \mathbf{x} \in \mathbf{R}^n .$$

The existence of a unique minimizer is apparent as its second derivative is given by a symmetric positive-definite matrix

$$\nabla^2 f(\mathbf{x}) = \mathbf{A} ,$$

and that the minimizer (use D$f(\mathbf{x})$=0) solves the initial problem is obvious from its first derivative

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b} .$$

This suggests taking the first basis vector $\mathbf{p}_0$ to be the negative of the gradient of f at $\mathbf{x} = \mathbf{x}_0$. The gradient of f equals $\mathbf{A}\mathbf{x} - \mathbf{b}$. Starting with an initial guess $\mathbf{x}_0$, this means we take $\mathbf{p}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$. The other vectors in the basis will be conjugate to the gradient, hence the name *conjugate gradient method*. Note that $\mathbf{p}_0$ is also the residual provided by this initial step of the algorithm.

Let $\mathbf{r}_k$ be the residual at the kth step:

$$\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k .$$

As observed above, $\mathbf{r}_k$ is the negative gradient of f at $\mathbf{x} = \mathbf{x}_k$, so the gradient descent method would require to move in the direction $\mathbf{r}_k$. Here, however, we insist that the directions $\mathbf{p}_k$ be conjugate to each other. A practical way to enforce this is by requiring that the next search direction be built out of the current residual and all previous search directions. The conjugation constraint is an orthonormal-type constraint and hence the algorithm can be viewed as an example of Gram-Schmidt orthonormalization. This gives the following expression:

$$\mathbf{p}_k = \mathbf{r}_k - \sum_{i<k} \frac{\mathbf{p}_i^\mathsf{T}\mathbf{A}\mathbf{r}_k}{\mathbf{p}_i^\mathsf{T}\mathbf{A}\mathbf{p}_i}\mathbf{p}_i$$

(see the picture at the top of the article for the effect of the conjugacy constraint on convergence). Following this direction, the next optimal location is given by

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

with

$$\alpha_k = \frac{\mathbf{p}_k^\mathsf{T}(\mathbf{b} - \mathbf{A}\mathbf{x}_k)}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k} = \frac{\mathbf{p}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k},$$

where the last equality follows from the definition of $\mathbf{r}_k$. The expression for α_k can be derived if one substitutes the expression for $\mathbf{x}_{k+1}$ into f and minimizing it w.r.t. α_k

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) =: g(\alpha_k)$$
$$g'(\alpha_k) \stackrel{!}{=} 0 \quad \Rightarrow \quad \alpha_k = \frac{\mathbf{p}_k^\mathsf{T}(\mathbf{b} - \mathbf{A}\mathbf{x}_k)}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}.$$

The resulting algorithm

The above algorithm gives the most straightforward explanation of the conjugate gradient method. Seemingly, the algorithm as stated requires storage of all previous searching directions and residue vectors, as well as many matrix-vector multiplications, and thus can be computationally expensive. However, a closer analysis of the algorithm shows that $\mathbf{r}_i$ is orthogonal to $\mathbf{r}_j$, i.e. $\mathbf{r}_i^\mathsf{T}\mathbf{r}_j = 0$, for i ≠ j. And $\mathbf{p}_i$ is A-orthogonal to $\mathbf{p}_j$, i.e. $\mathbf{p}_i^\mathsf{T} A \mathbf{p}_j = 0$, for i ≠ j. This can be regarded that as the algorithm progresses, $\mathbf{p}_i$ and $\mathbf{r}_i$ span the same Krylov subspace. Where $\mathbf{r}_i$ form the orthogonal basis with respect to standard inner product, and $\mathbf{p}_i$ form the orthogonal basis with respect to inner product induced by A. Therefore, $\mathbf{x}_k$ can be regarded as the projection of $\mathbf{x}$ on the Krylov subspace.

The algorithm is detailed below for solving $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is a real, symmetric, positive-definite matrix. The input vector $\mathbf{x}_0$ can be an approximate initial solution or $\mathbf{0}$. It is a different formulation of the exact procedure described above.

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
if $\mathbf{r}_0$ is sufficiently small, then return $\mathbf{x}_0$ as the result
$\mathbf{p}_0 := \mathbf{r}_0$
$k := 0$
repeat
$\alpha_k := \frac{\mathbf{r}_k^\mathsf{T} \mathbf{r}_k}{\mathbf{p}_k^\mathsf{T} \mathbf{A} \mathbf{p}_k}$
$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k$
if $\mathbf{r}_{k+1}$ is sufficiently small, then exit loop
$\beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T} \mathbf{r}_{k+1}}{\mathbf{r}_k^\mathsf{T} \mathbf{r}_k}$
$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
$k := k + 1$
end repeat
return $\mathbf{x}_{k+1}$ as the result

This is the most commonly used algorithm. The same formula for β_k is also used in the Fletcher–Reeves nonlinear conjugate gradient method.

Restarts

We note that $\mathbf{x}_1$ is computed by the gradient descent method applied to $\mathbf{x}_0$. Setting $\beta_k = 0$ would similarly make $\mathbf{x}_{k+1}$ computed by the gradient descent method from $\mathbf{x}_k$, i.e., can be used as a simple implementation of a restart of the conjugate gradient iterations.[4] Restarts could slow down convergence, but may improve stability if the conjugate gradient method misbehaves, e.g., due to round-off error.

Explicit residual calculation

The formulas $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$ and $\mathbf{r}_k := \mathbf{b} - \mathbf{A}\mathbf{x}_k$, which both hold in exact arithmetic, make the formulas $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k$ and $\mathbf{r}_{k+1} := \mathbf{b} - \mathbf{A}\mathbf{x}_{k+1}$ mathematically equivalent. The former is used in the algorithm to avoid an extra

multiplication by $\mathbf{A}$ since the vector $\mathbf{A}\mathbf{p}_k$ is already computed to evaluate α_k. The latter may be more accurate, substituting the explicit calculation $\mathbf{r}_{k+1} := \mathbf{b} - \mathbf{A}\mathbf{x}_{k+1}$ for the implicit one by the recursion subject to round-off error accumulation, and is thus recommended for an occasional evaluation.[6]

Computation of alpha and beta

In the algorithm, α_k is chosen such that $\mathbf{r}_{k+1}$ is orthogonal to $\mathbf{r}_k$. The denominator is simplified from

$$\alpha_k = \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{r}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k} = \frac{\mathbf{r}_k^\mathsf{T}\mathbf{r}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$$

since $\mathbf{r}_{k+1} = \mathbf{p}_{k+1} - \beta_k \mathbf{p}_k$. The β_k is chosen such that $\mathbf{p}_{k+1}$ is conjugated to $\mathbf{p}_k$. Initially, β_k is

$$\beta_k = -\frac{\mathbf{r}_{k+1}^\mathsf{T}\mathbf{A}\mathbf{p}_k}{\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k}$$

using

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k$$

and equivalently

$$\mathbf{A}\mathbf{p}_k = \frac{1}{\alpha_k}(\mathbf{r}_k - \mathbf{r}_{k+1}),$$

the numerator of β_k is rewritten as

$$\mathbf{r}_{k+1}^\mathsf{T}\mathbf{A}\mathbf{p}_k = \frac{1}{\alpha_k}\mathbf{r}_{k+1}^\mathsf{T}(\mathbf{r}_k - \mathbf{r}_{k+1}) = -\frac{1}{\alpha_k}\mathbf{r}_{k+1}^\mathsf{T}\mathbf{r}_{k+1}$$

because $\mathbf{r}_{k+1}$ and $\mathbf{r}_k$ are orthogonal by design. The denominator is rewritten as

$$\mathbf{p}_k^\mathsf{T}\mathbf{A}\mathbf{p}_k = (\mathbf{r}_k + \beta_{k-1}\mathbf{p}_{k-1})^\mathsf{T}\mathbf{A}\mathbf{p}_k = \frac{1}{\alpha_k}\mathbf{r}_k^\mathsf{T}(\mathbf{r}_k - \mathbf{r}_{k+1}) = \frac{1}{\alpha_k}\mathbf{r}_k^\mathsf{T}\mathbf{r}_k$$

using that the search directions $\mathbf{p}_k$ are conjugated and again that the residuals are orthogonal. This gives the β in the algorithm after cancelling α_k.

Example code in MATLAB / GNU Octave

```
function x = conjgrad(A, b, x)
    r = b - A * x;
    p = r;
    rsold = r' * r;

    for i = 1:length(b)
        Ap = A * p;
        alpha = rsold / (p' * Ap);
        x = x + alpha * p;
        r = r - alpha * Ap;
        rsnew = r' * r;
        if sqrt(rsnew) < 1e-10
              break
        end
        p = r + (rsnew / rsold) * p;
        rsold = rsnew;
    end
end
```

Numerical example

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

$$\mathbf{Ax} = \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix},$$

we will perform two steps of the conjugate gradient method beginning with the initial guess

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

in order to find an approximate solution to the system.

Solution

For reference, the exact solution is

$$\mathbf{x} = \begin{bmatrix} \frac{1}{11} \\ \\ \frac{7}{11} \end{bmatrix} \approx \begin{bmatrix} 0.0909 \\ \\ 0.6364 \end{bmatrix}$$

Our first step is to calculate the residual vector $\mathbf{r}_0$ associated with $\mathbf{x}_0$. This residual is computed from the formula $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, and in our case is equal to

$$\mathbf{r}_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -8 \\ -3 \end{bmatrix} = \mathbf{p}_0.$$

Since this is the first iteration, we will use the residual vector $\mathbf{r}_0$ as our initial search direction $\mathbf{p}_0$; the method of selecting $\mathbf{p}_k$ will change in further iterations.

We now compute the scalar α_0 using the relationship

$$\alpha_0 = \frac{\mathbf{r}_0^{\mathsf{T}} \mathbf{r}_0}{\mathbf{p}_0^{\mathsf{T}} \mathbf{A} \mathbf{p}_0} = \frac{\begin{bmatrix} -8 & -3 \end{bmatrix} \begin{bmatrix} -8 \\ -3 \end{bmatrix}}{\begin{bmatrix} -8 & -3 \end{bmatrix} \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} -8 \\ -3 \end{bmatrix}} = \frac{73}{331}.$$

We can now compute $\mathbf{x}_1$ using the formula

$$\mathbf{x}_1 = \mathbf{x}_0 + \alpha_0 \mathbf{p}_0 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \frac{73}{331} \begin{bmatrix} -8 \\ -3 \end{bmatrix} = \begin{bmatrix} 0.2356 \\ 0.3384 \end{bmatrix}.$$

This result completes the first iteration, the result being an "improved" approximate solution to the system, $\mathbf{x}_1$. We may now move on and compute the next residual vector $\mathbf{r}_1$ using the formula

$$\mathbf{r}_1 = \mathbf{r}_0 - \alpha_0 \mathbf{A} \mathbf{p}_0 = \begin{bmatrix} -8 \\ -3 \end{bmatrix} - \frac{73}{331} \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} -8 \\ -3 \end{bmatrix} = \begin{bmatrix} -0.2810 \\ 0.7492 \end{bmatrix}.$$

Our next step in the process is to compute the scalar β_0 that will eventually be used to determine the next search direction $\mathbf{p}_1$.

$$\beta_0 = \frac{\mathbf{r}_1^{\mathsf{T}} \mathbf{r}_1}{\mathbf{r}_0^{\mathsf{T}} \mathbf{r}_0} = \frac{\begin{bmatrix} -0.2810 & 0.7492 \end{bmatrix} \begin{bmatrix} -0.2810 \\ 0.7492 \end{bmatrix}}{\begin{bmatrix} -8 & -3 \end{bmatrix} \begin{bmatrix} -8 \\ -3 \end{bmatrix}} = 0.0088.$$

Now, using this scalar β_0, we can compute the next search direction $\mathbf{p}_1$ using the relationship

$$\mathbf{p}_1 = \mathbf{r}_1 + \beta_0 \mathbf{p}_0 = \begin{bmatrix} -0.2810 \\ 0.7492 \end{bmatrix} + 0.0088 \begin{bmatrix} -8 \\ -3 \end{bmatrix} = \begin{bmatrix} -0.3511 \\ 0.7229 \end{bmatrix}.$$

We now compute the scalar α_1 using our newly acquired $\mathbf{p}_1$ using the same method as that used for α_0.

$$\alpha_1 = \frac{\mathbf{r}_1^{\mathsf{T}} \mathbf{r}_1}{\mathbf{p}_1^{\mathsf{T}} \mathbf{A} \mathbf{p}_1} = \frac{\begin{bmatrix} -0.2810 & 0.7492 \end{bmatrix} \begin{bmatrix} -0.2810 \\ 0.7492 \end{bmatrix}}{\begin{bmatrix} -0.3511 & 0.7229 \end{bmatrix} \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} -0.3511 \\ 0.7229 \end{bmatrix}} = 0.4122.$$

Finally, we find $\mathbf{x}_2$ using the same method as that used to find $\mathbf{x}_1$.

$$\mathbf{x}_2 = \mathbf{x}_1 + \alpha_1 \mathbf{p}_1 = \begin{bmatrix} 0.2356 \\ 0.3384 \end{bmatrix} + 0.4122 \begin{bmatrix} -0.3511 \\ 0.7229 \end{bmatrix} = \begin{bmatrix} 0.0909 \\ 0.6364 \end{bmatrix}.$$

The result, $\mathbf{x}_2$, is a "better" approximation to the system's solution than $\mathbf{x}_1$ and $\mathbf{x}_0$. If exact arithmetic were to be used in this example instead of limited-precision, then the exact solution would theoretically have been reached after $n = 2$ iterations (n being the order of the system).

Convergence properties

The conjugate gradient method can theoretically be viewed as a direct method, as in the absence of round-off error it produces the exact solution after a finite number of iterations, which is not larger than the size of the matrix. In practice, the exact solution is never obtained since the conjugate gradient method is

unstable with respect to even small perturbations, e.g., most directions are not in practice conjugate, due to a degenerative nature of generating the Krylov subspaces.

As an iterative method, the conjugate gradient method monotonically (in the energy norm) 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. The improvement is typically linear and its speed is determined by the condition number $\kappa(A)$ of the system matrix A: the larger $\kappa(A)$ is, the slower the improvement.[7]

If $\kappa(A)$ is large, preconditioning is commonly used to replace the original system $\mathbf{A}\mathbf{x} - \mathbf{b} = 0$ with $\mathbf{M}^{-1}(\mathbf{A}\mathbf{x} - \mathbf{b}) = 0$ such that $\kappa(\mathbf{M}^{-1}\mathbf{A})$ is smaller than $\kappa(\mathbf{A})$, see below.

Convergence theorem

Define a subset of polynomials as

$$\Pi_k^* := \{\, p \in \Pi_k \ : \ p(0) = 1 \,\} ,$$

where Π_k is the set of polynomials of maximal degree k.

Let $(\mathbf{x}_k)_k$ be the iterative approximations of the exact solution $\mathbf{x}_*$, and define the errors as $\mathbf{e}_k := \mathbf{x}_k - \mathbf{x}_*$. Now, the rate of convergence can be approximated as [4][8]

$$\begin{aligned}
\|\mathbf{e}_k\|_{\mathbf{A}} &= \min_{p \in \Pi_k^*} \|p(\mathbf{A})\mathbf{e}_0\|_{\mathbf{A}} \\
&\leq \min_{p \in \Pi_k^*} \max_{\lambda \in \sigma(\mathbf{A})} |p(\lambda)| \, \|\mathbf{e}_0\|_{\mathbf{A}} \\
&\leq 2\left(\frac{\sqrt{\kappa(\mathbf{A})} - 1}{\sqrt{\kappa(\mathbf{A})} + 1}\right)^k \|\mathbf{e}_0\|_{\mathbf{A}} ,
\end{aligned}$$

where $\sigma(\mathbf{A})$ denotes the spectrum, and $\kappa(\mathbf{A})$ denotes the condition number.

Note, the important limit when $\kappa(\mathbf{A})$ tends to ∞

$$\frac{\sqrt{\kappa(\mathbf{A})}-1}{\sqrt{\kappa(\mathbf{A})}+1} \approx 1-\frac{2}{\sqrt{\kappa(\mathbf{A})}} \quad \text{for} \quad \kappa(\mathbf{A}) \gg 1\,.$$

This limit shows a faster convergence rate compared to the iterative methods of Jacobi or Gauss–Seidel which scale as $\approx 1-\frac{2}{\kappa(\mathbf{A})}$.

No round-off error is assumed in the convergence theorem, but the convergence bound is commonly valid in practice as theoretically explained[5] by Anne Greenbaum.

Practical convergence

If initialized randomly, the first stage of iterations is often the fastest, as the error is eliminated within the Krylov subspace that initially reflects a smaller effective condition number. The second stage of convergence is typically well defined by the theoretical convergence bound with $\sqrt{\kappa(\mathbf{A})}$, but may be super-linear, depending on a distribution of the spectrum of the matrix A and the spectral distribution of the error.[5] In the last stage, the smallest attainable accuracy is reached and the convergence stalls or the method may even start diverging. In typical scientific computing applications in double-precision floating-point format for matrices of large sizes, the conjugate gradient method uses a stopping criteria with a tolerance that terminates the iterations during the first or second stage.

The preconditioned conjugate gradient method

In most cases, preconditioning is necessary to ensure fast convergence of the conjugate gradient method. The preconditioned conjugate gradient method takes the following form:[9]

$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$
$\mathbf{z}_0 := \mathbf{M}^{-1}\mathbf{r}_0$
$\mathbf{p}_0 := \mathbf{z}_0$
$k := 0$
repeat

$\alpha_k := \dfrac{\mathbf{r}_k^\mathsf{T} \mathbf{z}_k}{\mathbf{p}_k^\mathsf{T} \mathbf{A} \mathbf{p}_k}$
$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k$
if $\mathbf{r}_{k+1}$ is sufficiently small **then** exit loop **end if**
$\mathbf{z}_{k+1} := \mathbf{M}^{-1} \mathbf{r}_{k+1}$
$\beta_k := \dfrac{\mathbf{r}_{k+1}^\mathsf{T} \mathbf{z}_{k+1}}{\mathbf{r}_k^\mathsf{T} \mathbf{z}_k}$
$\mathbf{p}_{k+1} := \mathbf{z}_{k+1} + \beta_k \mathbf{p}_k$
$k := k + 1$

end repeat
The result is $\mathbf{x}_{k+1}$

The above formulation is equivalent to applying the conjugate gradient method without preconditioning to the system[10]

$$\mathbf{E}^{-1} \mathbf{A} (\mathbf{E}^{-1})^\mathsf{T} \hat{\mathbf{x}} = \mathbf{E}^{-1} \mathbf{b}$$

where

$$\mathbf{E}\mathbf{E}^\mathsf{T} = \mathbf{M}, \qquad \hat{\mathbf{x}} = \mathbf{E}^\mathsf{T} \mathbf{x}.$$

The preconditioner matrix $\mathbf{M}$ has to be symmetric positive-definite and fixed, i.e., cannot change from iteration to iteration. If any of these assumptions on the preconditioner is violated, the behavior of the preconditioned conjugate gradient method may become unpredictable.

An example of a commonly used preconditioner is the incomplete Cholesky factorization.[11]

The flexible preconditioned conjugate gradient method

In numerically challenging applications, sophisticated preconditioners are used, which may lead to variable preconditioning, changing between iterations. Even if the preconditioner is symmetric positive-definite on every iteration, the fact that it

may change makes the arguments above invalid, and in practical tests leads to a significant slow down of the convergence of the algorithm presented above. Using the Polak–Ribière formula

$$\beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T} \left(\mathbf{z}_{k+1} - \mathbf{z}_k \right)}{\mathbf{r}_k^\mathsf{T} \mathbf{z}_k}$$

instead of the Fletcher–Reeves formula

$$\beta_k := \frac{\mathbf{r}_{k+1}^\mathsf{T} \mathbf{z}_{k+1}}{\mathbf{r}_k^\mathsf{T} \mathbf{z}_k}$$

may dramatically improve the convergence in this case.[12] This version of the preconditioned conjugate gradient method can be called[13] **flexible,** as it allows for variable preconditioning. The flexible version is also shown[14] to be robust even if the preconditioner is not symmetric positive definite (SPD).

The implementation of the flexible version requires storing an extra vector. For a fixed SPD preconditioner, $\mathbf{r}_{k+1}^\mathsf{T} \mathbf{z}_k = 0$, so both formulas for β_k are equivalent in exact arithmetic, i.e., without the round-off error.

The mathematical explanation of the better convergence behavior of the method with the Polak–Ribière formula is that the method is **locally optimal** in this case, in particular, it does not converge slower than the locally optimal steepest descent method.[15]

Vs. the locally optimal steepest descent method

In both the original and the preconditioned conjugate gradient methods one only needs to set $\beta_k := 0$ in order to make them locally optimal, using the line search, steepest descent methods. With this substitution, vectors $\mathbf{p}$ are always the same as vectors $\mathbf{z}$, so there is no need to store vectors $\mathbf{p}$. Thus, every iteration of these steepest descent methods is a bit cheaper compared to that for the conjugate gradient methods. However, the latter converge faster, unless a (highly) variable and/or non-SPD preconditioner is used, see above.

Conjugate gradient method as optimal

feedback controller for double integrator

The conjugate gradient method can also be derived using optimal control theory.[16] In this approach, the conjugate gradient method falls out as an optimal feedback controller,

$$u = k(x, v) := -\gamma_a \nabla f(x) - \gamma_b v$$

for the double integrator system,

$$\dot{x} = v, \quad \dot{v} = u$$

The quantities γ_a and γ_b are variable feedback gains.[16]

Conjugate gradient on the normal equations

The conjugate gradient method can be applied to an arbitrary n-by-m matrix by applying it to normal equations $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ and right-hand side vector $\mathbf{A}^{\mathrm{T}}\mathbf{b}$, since $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ is a symmetric positive-semidefinite matrix for any $\mathbf{A}$. The result is conjugate gradient on the normal equations (CGNR).

$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$$

As an iterative method, it is not necessary to form $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ explicitly in memory but only to perform the matrix-vector and transpose matrix-vector multiplications. Therefore, CGNR is particularly useful when A is a sparse matrix since these operations are usually extremely efficient. However the downside of forming the normal equations is that the condition number $\kappa(\mathbf{A}^{\mathrm{T}}\mathbf{A})$ is equal to $\kappa^2(\mathbf{A})$ and so the rate of convergence of CGNR may be slow and the quality of the approximate solution may be sensitive to roundoff errors. Finding a good preconditioner is often an important part of using the CGNR method.

Several algorithms have been proposed (e.g., CGLS, LSQR). The LSQR (https://web.stanford.edu/group/SOL/software/lsqr/) algorithm purportedly has the best numerical stability when $\mathbf{A}$ is ill-conditioned, i.e., $\mathbf{A}$ has a large condition number.

Conjugate gradient method for complex Hermitian matrices

The conjugate gradient method with a trivial modification is extendable to solving, given complex-valued matrix A and vector b, the system of linear equations $\mathbf{Ax} = \mathbf{b}$ for the complex-valued vector x, where A is Hermitian (i.e., A' = A) and positive-definite matrix, and the symbol ' denotes the conjugate transpose using the MATLAB/GNU Octave style. The trivial modification is simply substituting the conjugate transpose for the real transpose everywhere. This substitution is backward compatible, since conjugate transpose turns into real transpose on real-valued vectors and matrices. The provided above Example code in MATLAB/GNU Octave thus already works for complex Hermitian matrices needed no modification.

See also

- Biconjugate gradient method (BiCG)
- Conjugate residual method
- Gaussian belief propagation
- Iterative method: Linear systems
- Krylov subspace
- Nonlinear conjugate gradient method
- Preconditioning
- Sparse matrix-vector multiplication

References

1. Hestenes, Magnus R.; Stiefel, Eduard (December 1952). "Methods of Conjugate Gradients for Solving Linear Systems" (http://nvlpubs.nist.gov/nistpubs/jres/049/6/V49.N06.A08.pdf) (PDF). *Journal of Research of the National Bureau of Standards*. **49** (6): 409. doi:10.6028/jres.049.044 (https://doi.org/10.6028%2Fjres.049.044).
2. Straeter, T. A. (1971). "On the Extension of the Davidon–Broyden Class of Rank One, Quasi-Newton Minimization Methods to an Infinite Dimensional Hilbert Space with Applications to Optimal Control Problems". *NASA Technical Reports Server*. NASA. hdl:2060/19710026200 (https://hdl.handle.net/2060%2F19710026200).
3. Speiser, Ambros (2004). "Konrad Zuse und die ERMETH: Ein weltweiter Architektur-Vergleich" [Konrad Zuse and the ERMETH: A worldwide comparison of architectures]. In Hellige, Hans Dieter (ed.). *Geschichten der Informatik. Visionen, Paradigmen, Leitmotive* (in German). Berlin: Springer. p. 185. ISBN 3-540-00217-0.

4. Polyak, Boris (1987). *Introduction to Optimization* (https://www.researchgate.net/publication/342978480_Introduction_to_Optimization).
5. Greenbaum, Anne (1997). *Iterative Methods for Solving Linear Systems* (https://doi.org/10.1137/1.9781611970937). doi:10.1137/1.9781611970937 (https://doi.org/10.1137%2F1.9781611970937). ISBN 978-0898713961.
6. Shewchuk, Jonathan R (1994). *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain* (http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf) (PDF).
7. Saad, Yousef (2003). *Iterative methods for sparse linear systems* (https://archive.org/details/iterativemethods0000saad/page/195) (2nd ed.). Philadelphia, Pa.: Society for Industrial and Applied Mathematics. pp. 195 (https://archive.org/details/iterativemethods0000saad/page/195). ISBN 978-0-89871-534-7.
8. Hackbusch, W. (2016-06-21). *Iterative solution of large sparse systems of equations* (2nd ed.). Switzerland: Springer. ISBN 9783319284835. OCLC 952572240 (https://www.worldcat.org/oclc/952572240).
9. Barrett, Richard; Berry, Michael; Chan, Tony F.; Demmel, James; Donato, June; Dongarra, Jack; Eijkhout, Victor; Pozo, Roldan; Romine, Charles; van der Vorst, Henk. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* (http://www.netlib.org/templates/templates.pdf) (PDF) (2nd ed.). Philadelphia, PA: SIAM. p. 13. Retrieved 2020-03-31.
10. Golub, Gene H.; Van Loan, Charles F. (2013). *Matrix Computations* (4th ed.). Johns Hopkins University Press. sec. 11.5.2. ISBN 978-1-4214-0794-4.
11. Concus, P.; Golub, G. H.; Meurant, G. (1985). "Block Preconditioning for the Conjugate Gradient Method". *SIAM Journal on Scientific and Statistical Computing*. **6** (1): 220–252. doi:10.1137/0906018 (https://doi.org/10.1137%2F0906018).
12. Golub, Gene H.; Ye, Qiang (1999). "Inexact Preconditioned Conjugate Gradient Method with Inner-Outer Iteration". *SIAM Journal on Scientific Computing*. **21** (4): 1305. CiteSeerX 10.1.1.56.1755 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.56.1755). doi:10.1137/S1064827597323415 (https://doi.org/10.1137%2FS1064827597323415).
13. Notay, Yvan (2000). "Flexible Conjugate Gradients". *SIAM Journal on Scientific Computing*. **22** (4): 1444–1460. CiteSeerX 10.1.1.35.7473 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.7473). doi:10.1137/S1064827599362314 (https://doi.org/10.1137%2FS1064827599362314).
14. Henricus Bouwmeester, Andrew Dougherty, Andrew V Knyazev. Nonsymmetric Preconditioning for Conjugate Gradient and Steepest Descent Methods (https://doi.org/10.1016/j.procs.2015.05.241). Procedia Computer Science, Volume 51, Pages 276-285, Elsevier, 2015. doi: 10.1016/j.procs.2015.05.241 (https://d

oi.org/10.1016%2Fj.procs.2015.05.241)
15. Knyazev, Andrew V.; Lashuk, Ilya (2008). "Steepest Descent and Conjugate Gradient Methods with Variable Preconditioning". *SIAM Journal on Matrix Analysis and Applications*. **29** (4): 1267. arXiv:math/0605767 (https://arxiv.org/abs/math/0605767). doi:10.1137/060675290 (https://doi.org/10.1137%2F060675290). S2CID 17614913 (https://api.semanticscholar.org/CorpusID:17614913).
16. Ross, I. M., "An Optimal Control Theory for Accelerated Optimization," arXiv:1902.09004 (https://arxiv.org/abs/1902.09004), 2019.

Further reading

- Atkinson, Kendell A. (1988). "Section 8.9". *An introduction to numerical analysis* (https://archive.org/details/introductiontonu0000atki) (2nd ed.). John Wiley and Sons. ISBN 978-0-471-50023-0.
- Avriel, Mordecai (2003). *Nonlinear Programming: Analysis and Methods*. Dover Publishing. ISBN 978-0-486-43227-4.
- Golub, Gene H.; Van Loan, Charles F. (2013). "Chapter 11". *Matrix Computations* (4th ed.). Johns Hopkins University Press. ISBN 978-1-4214-0794-4.
- Saad, Yousef (2003-04-01). "Chapter 6" (https://archive.org/details/iterativemethods0000saad). *Iterative methods for sparse linear systems* (https://archive.org/details/iterativemethods0000saad) (2nd ed.). SIAM. ISBN 978-0-89871-534-7.

External links

- "Conjugate gradients, method of" (https://www.encyclopediaofmath.org/index.php?title=Conjugate_gradients,_method_of), *Encyclopedia of Mathematics*, EMS Press, 2001 [1994]