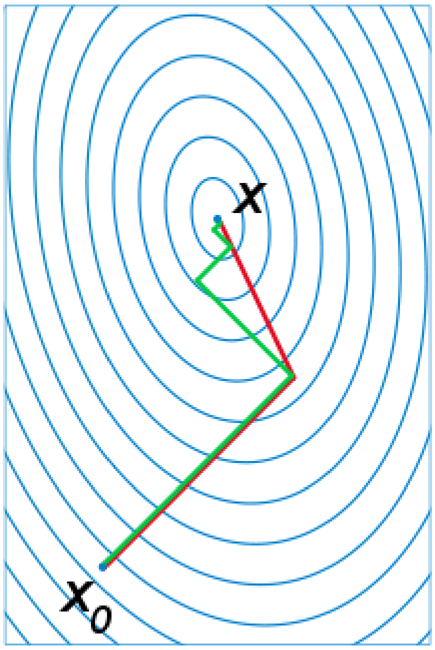
WIKIPEDIA

# 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 symmetric and 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.




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 $n = 2$).

The conjugate gradient method can also be used to solve unconstrained optimization problems such as energy minimization. It was mainly developed by Magnus Hestenes and Eduard Stiefel[1] who programmed it on the Z4.[2]

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

## Contents

# Description of the problem addressed by conjugate gradients

Suppose we want to solve the system of linear equations

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

for the vector $\mathbf{x}$, where the known $n \times n$ matrix $\mathbf{A}$ is symmetric (i.e., $\mathbf{A}^\mathsf{T} = \mathbf{A}$), positive-definite (i.e. $\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} > 0$ 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 $\mathbf{x}_*$.

# As a direct 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

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

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 $\mathbf{A}\mathbf{x} = \mathbf{b}$ 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 for solving the equation $\mathbf{Ax} = \mathbf{b}$: find a sequence of $n$ conjugate directions, and then compute the coefficients $\alpha_k$.

# 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 $\mathbf{Az} = \mathbf{b} - \mathbf{Ax}_0$ 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 $Df(\mathbf{x})=0$) solves the initial problem is obvious from its first derivative

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

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 $\mathbf{Ax} - \mathbf{b}$. Starting with an initial guess $\mathbf{x}_0$, this means we take $\mathbf{p}_0 = \mathbf{b} - \mathbf{Ax}_0$. 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 $k$th step:

$$\mathbf{r}_k = \mathbf{b} - \mathbf{Ax}_k.$$

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.[3] 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 $\alpha_k$ can be derived if one substitutes the expression for $\mathbf{x}_{k+1}$ into $f$ and minimizing it w.r.t. $\alpha_k$

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

## 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 $\beta_k$ is also used in the Fletcher–Reeves nonlinear conjugate gradient method.

### Computation of alpha and beta

In the algorithm, $\alpha_k$ 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 $\beta_k$ is chosen such that $\mathbf{p}_{k+1}$ is conjugated to $\mathbf{p}_k$. Initially, $\beta_k$ 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 $\beta_k$ 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 $\beta$ in the algorithm after cancelling $\alpha_k$.

### 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 **Ax** = **b** given by

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

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 $\mathbf{r}_0$ = **b** - $\mathbf{A}\mathbf{x}_0$, and in our case is equal to

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

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 $\alpha_0$ 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 $\beta_0$ 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 $\beta_0$, 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 $\alpha_1$ using our newly acquired $\mathbf{p}_1$ using the same method as that used for $\alpha_0$.

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

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 it produces the exact solution after a finite number of iterations, which is not larger than the size of the matrix, in the absence of round-off error. However, the conjugate gradient method is unstable with respect to even small perturbations, e.g., most directions are not in practice conjugate, and the exact solution is never obtained. Fortunately, the conjugate gradient method can be used as an iterative method as it provides monotonically improving approximations $\mathbf{x}_k$ to the exact solution, which 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.[4]

If $\kappa(A)$ is large, preconditioning is used to replace the original system $\mathbf{Ax} - \mathbf{b} = 0$ with $\mathbf{M}^{-1}(\mathbf{Ax} - \mathbf{b}) = 0$ 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 $\Pi_k$ 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 [5]

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

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 $\infty$

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

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

# 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:

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

**repeat**

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

**if** $\mathbf{r}_{k+1}$ is sufficiently small **then** exit loop **end if**

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

**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[1]

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

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.

# 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

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

instead of the Fletcher–Reeves formula

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

may dramatically improve the convergence in this case.[6] This version of the preconditioned conjugate gradient method can be called[7] **flexible,** as it allows for variable preconditioning. The flexible version is also shown[8] 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, $\mathbf{z}_{k+1}^\mathsf{T} \mathbf{r}_k = 0,$ so both formulas for $\beta_k$ 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.[9]

## Example code in MATLAB / GNU Octave

```
function [x, k] = cgp(x0, A, C, b, mit, stol, bbA, bbC)
% Synopsis:
% x0: initial point
% A: Matrix A of the system Ax=b
% C: Preconditioning Matrix can be left or right
% mit: Maximum number of iterations
% stol: residue norm tolerance
% bbA: Black Box that computes the matrix-vector product for A * u
% bbC: Black Box that computes:
%      for left-side preconditioner : ha = C \ ra
%      for right-side preconditioner: ha = C * ra
% x: Estimated solution point
% k: Number of iterations done
%
% Example:
% tic;[x, t] = cgp(x0, S, speye(1), b, 3000, 10^-8, @(Z, o) Z*o, @(Z, o) o);toc
% Elapsed time is 0.550190 seconds.
%
% Reference:
%  Métodos iterativos tipo Krylov para sistema lineales
%  B. Molina y M. Raydan - {{ISBN|908-261-078-X}}
        if nargin < 8, error('Not enough input arguments. Try help.'); end;
        if isempty(A), error('Input matrix A must not be empty.'); end;
        if isempty(C), error('Input preconditioner matrix C must not be empty.'); end;
        x = x0;
        ha = 0;
        hp = 0;
        hpp = 0;
        ra = 0;
        rp = 0;
        rpp = 0;
        u = 0;
        k = 0;

        ra = b - bbA(A, x0); % <--- ra = b - A * x0;
        while norm(ra, inf) > stol
                ha = bbC(C, ra); % <--- ha = C \ ra;
                k = k + 1;
                if (k == mit), warning('GCP:MAXIT', 'mit reached, no conversion.'); return; end;
                hpp = hp;
                rpp = rp;
                hp = ha;
                rp = ra;
```

```
                t = rp' * hp;
                if k == 1
                        u = hp;
                else
                        u = hp + (t / (rpp' * hpp)) * u;
                end;
                Au = bbA(A, u); % <--- Au = A * u;
                a = t / (u' * Au);
                x = x + a * u;
                ra = rp - a * Au;
        end;
```

## 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 **p** are always the same as vectors **z**, so there is no need to store vectors **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.

## Derivation of the 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.

## 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 algorithm purportedly has the best numerical stability when **A** is ill-conditioned, i.e., **A** has a large condition number.

## 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

## Notes


1. Straeter, T. A. "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).
2. Speiser, Ambros "*Konrad Zuse and the ERMETH: A worldwide comparison of architectures*" ("Konrad Zuse und die ERMETH: Ein weltweiter Architektur-Vergleich"), in: Hans Dieter Hellige (ed.): Geschichten der Informatik. Visionen, Paradigmen, Leitmotive. Berlin, Springer 2004, ISBN 3-540-00217-0. p. 185.
3. The conjugation constraint is an orthonormal-type constraint and hence the algorithm bears resemblance to Gram-Schmidt orthonormalization.
4. 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.
5. 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).
6. 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).
7. 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).
8. Henricus Bouwmeester, Andrew Dougherty, Andrew V Knyazev. Nonsymmetric Preconditioning for Conjugate Gradient and Steepest Descent Methods. Procedia Computer Science, Volume 51, Pages 276-285, Elsevier, 2015. https://doi.org/10.1016/j.procs.2015.05.241
9. 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).


## References

The conjugate gradient method was originally proposed in


- Hestenes, Magnus R.; Stiefel, Eduard (December 1952). "Methods of Conjugate Gradients for Solving Linear Systems". *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).


Descriptions of the method can be found in the following text books:

- Atkinson, Kendell A. (1988). "Section 8.9". *An introduction to numerical analysis* (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. (1996-10-15). "Chapter 10". *Matrix computations* (3rd ed.). Johns Hopkins University Press. ISBN 978-0-8018-5414-9.
- 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

- Hazewinkel, Michiel, ed. (2001) [1994], "Conjugate gradients, method of" (https://www.encyclopediaofmath.org/index.php?title=p/c025030), *Encyclopedia of Mathematics*, Springer Science+Business Media B.V. / Kluwer Academic Publishers, ISBN 978-1-55608-010-4