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 an iterative method, so it can be applied to sparse systems that are too large to be handled by direct methods such as the Cholesky decomposition. Such systems often arise when numerically solving partial differential equations.

The conjugate gradient method can also be used to solve unconstrained optimization problems such as energy minimization. It was developed by Magnus Hestenes and Eduard Stiefel.[1]

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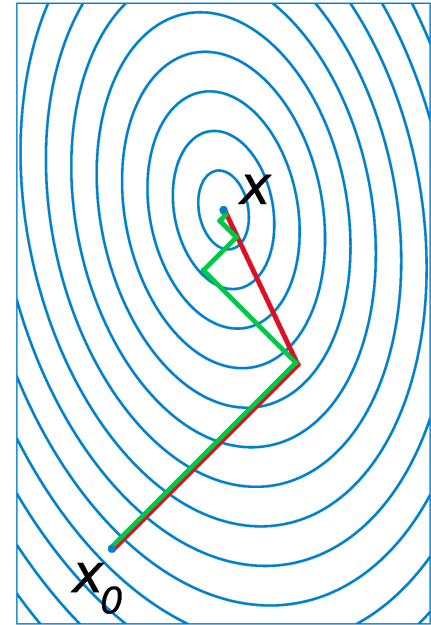
Description of the method

Suppose we want to solve the following system of linear equations

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

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

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 as a direct method

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

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

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

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

Two vectors are conjugate 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 $\mathbf{u}$. (Note: This notion of *conjugate* is not related to the notion of complex conjugate.)

Suppose that $\{\mathbf{p}_k\}$ is a sequence of n mutually conjugate directions. Then the $\mathbf{p}_k$ form a basis of $\mathbf{R}^n$, so we can expand the solution $\mathbf{x}_*$ of $\mathbf{A}\mathbf{x} = \mathbf{b}$ in this basis:

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

and we see that

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

The coefficients are given by

$$\mathbf{p}_k^{\mathrm{T}}\mathbf{b} = \mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{x}_* = \sum_{i=1}^{n} \alpha_i \mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_i = \alpha_k \mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k.$$

(because $\forall i \neq k, p_i, p_k$ are mutually conjugate)

$$\alpha_k = \frac{\mathbf{p}_k^{\mathrm{T}}\mathbf{b}}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k} = \frac{\langle \mathbf{p}_k, \mathbf{b} \rangle}{\langle \mathbf{p}_k, \mathbf{p}_k \rangle_{\mathbf{A}}} = \frac{\langle \mathbf{p}_k, \mathbf{b} \rangle}{\|\mathbf{p}_k\|_{\mathbf{A}}^2}.$$

This result is perhaps most transparent by considering the inner product defined above.

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

The conjugate gradient method 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 $\mathbf{x}_0 = 0$ (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; so if f($\mathbf{x}$) becomes smaller in an iteration it means that we are closer to $\mathbf{x}_*$.

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

This suggests taking the first basis vector $\mathbf{p}_1$ to be the negative of the gradient of f at $\mathbf{x} = \mathbf{x}_0$. The gradient of f equals $\mathbf{A}\mathbf{x}_0 - \mathbf{b}$. Starting with a "guessed solution" $\mathbf{x}_0$ (we can always guess that $\mathbf{x}_*$ is $\mathbf{0}$ and set $\mathbf{x}_0$ to $\mathbf{0}$ if we have no reason to guess for anything else), this means we take $\mathbf{p}_1 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$. The other vectors in the basis will be conjugate to the gradient, hence the name *conjugate gradient method*.

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

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

Note that $\mathbf{r}_k$ is the negative gradient of f at $\mathbf{x} = \mathbf{x}_k$, so the gradient descent method would be to move in the direction $\mathbf{r}_k$. Here, we insist that the directions $\mathbf{p}_k$ be conjugate to each other. We also require that the next search direction be built out of the current residue and all previous search directions, which is reasonable enough in practice.

The conjugation constraint is an orthonormal-type constraint and hence the algorithm bears resemblance to Gram-Schmidt orthonormalization.

This gives the following expression:

$$\mathbf{p}_{k+1} = \mathbf{r}_k - \sum_{i \leq k} \frac{\mathbf{p}_i^{\mathrm{T}}\mathbf{A}\mathbf{r}_k}{\mathbf{p}_i^{\mathrm{T}}\mathbf{A}\mathbf{p}_i}\mathbf{p}_i$$

(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

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

with

$$\alpha_{k+1} = \frac{\mathbf{p}_{k+1}^{\mathrm{T}}\mathbf{b}}{\mathbf{p}_{k+1}^{\mathrm{T}}\mathbf{A}\mathbf{p}_{k+1}} = \frac{\mathbf{p}_{k+1}^{\mathrm{T}}(\mathbf{r}_k + \mathbf{A}\mathbf{x}_k)}{\mathbf{p}_{k+1}^{\mathrm{T}}\mathbf{A}\mathbf{p}_{k+1}} = \frac{\mathbf{p}_{k+1}^{\mathrm{T}}\mathbf{r}_k}{\mathbf{p}_{k+1}^{\mathrm{T}}\mathbf{A}\mathbf{p}_{k+1}},$$

where the last equality holds because $\mathbf{p}_{k+1}$ and $\mathbf{x}_k$ are conjugate.

The resulting algorithm

The above algorithm gives the most straightforward explanation of the conjugate gradient method. However, it requires storage of all previous searching directions and residue vectors, as well as many matrix vector multiplications, and thus can be computationally expensive. In practice, one slightly modifies the condition obtaining the last residue vector, not to minimize the metric following the search direction, but instead to make it orthogonal to the previous residue. Minimization of the metric along the search direction will be obtained automatically in this case. One can then obtain an algorithm which only requires storage of the last two residue vectors and the last search direction, and only one matrix vector multiplication. Note that the algorithm described below is equivalent to the previously discussed straightforward procedure.

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 $\mathbf{0}$.

$\mathbf{r}_0 := \mathbf{b} - \mathbf{Ax}_0$

$\mathbf{p}_0 := \mathbf{r}_0$

$k := 0$

repeat

$\qquad \alpha_k := \dfrac{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{Ap}_k}$

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

$\qquad \mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{Ap}_k$

$\qquad$ if r_{k+1} is sufficiently small then exit loop

$\qquad \beta_k := \dfrac{\mathbf{r}_{k+1}^{\mathrm{T}}\mathbf{r}_{k+1}}{\mathbf{r}_k^{\mathrm{T}}\mathbf{r}_k}$

$\qquad \mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$

$\qquad k := k + 1$

end repeat

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

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

Example code in Matlab

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

    for i=1:10^(6)
        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

To illustrate the conjugate gradient method, we will complete a simple example.

Considering 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

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

$$r_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} - \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -8 \\ -3 \end{bmatrix}.$$

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 α_0 using the relationship

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

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 $\mathbf{p}_1$.

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

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.

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

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 of the conjugate gradient method

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 is $\kappa(A)$, the slower the improvement.[]

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

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

$\qquad \alpha_k := \dfrac{\mathbf{r}_k^{\mathrm{T}}\mathbf{z}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k}$

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

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

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

$\qquad \mathbf{z}_{k+1} := \mathbf{M}^{-1}\mathbf{r}_{k+1}$

$\qquad \beta_k := \dfrac{\mathbf{z}_{k+1}^{\mathrm{T}}\mathbf{r}_{k+1}}{\mathbf{z}_k^{\mathrm{T}}\mathbf{r}_k}$

$\qquad \mathbf{p}_{k+1} := \mathbf{z}_{k+1} + \beta_k\mathbf{p}_k$

$\qquad k := k + 1$

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]

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

where $\mathbf{E}\mathbf{E}^{\mathrm{T}} = \mathbf{M}$ and $\hat{\mathbf{x}} = \mathbf{E}^{\mathrm{T}}\mathbf{x}$.

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

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

instead of the Fletcher–Reeves formula

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

may dramatically improve the convergence in this case.[2] This version of the preconditioned conjugate gradient method can be called[3] **flexible,** as it allows for variable preconditioning. The implementation of the flexible version requires storing an extra vector. For a fixed preconditioner, $\mathbf{z}_{k+1}^{\mathrm{T}}\mathbf{r}_k = 0,$ 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 converges not slower than the locally optimal steepest descent method.[4]

The conjugate gradient method vs. the locally optimal steepest descent method

In both the original and the preconditioned conjugate gradient methods one only needs to always set $\beta_k := 0$ in order to turn them into 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 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 $\mathbf{A}^T\mathbf{A}$ and right-hand side vector $\mathbf{A}^T\mathbf{b}$, since $\mathbf{A}^T\mathbf{A}$ is a symmetric positive-semidefinite matrix for any $\mathbf{A}$. The result is conjugate gradient on the normal equations (CGNR).

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

As an iterative method, it is not necessary to form $\mathbf{A}^T\mathbf{A}$ 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 $\kappa(\mathbf{A}^T\mathbf{A})$ 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 $\mathbf{A}$ is ill-conditioned, i.e., $\mathbf{A}$ has a large condition number.

Notes

References

The conjugate gradient method was originally proposed in

- Hestenes, Magnus R.; Stiefel, Eduard (December 1952). "Methods of Conjugate Gradients for Solving Linear Systems" (http://nvl.nist.gov/pub/nistpubs/jres/049/6/V49.N06.A08.pdf) (PDF). *Journal of Research of the National Bureau of Standards* **49** (6).

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 0-471-50023-2.
- Avriel, Mordecai (2003). *Nonlinear Programming: Analysis and Methods*. Dover Publishing. ISBN 0-486-43227-0.
- Golub, Gene H.; Van Loan, Charles F.. "Chapter 10". *Matrix computations* (3rd ed.). Johns Hopkins University Press. ISBN 0-8018-5414-8.
- Saad, Yousef. "Chapter 6". *Iterative methods for sparse linear systems* (2nd ed.). SIAM. ISBN 978-0-89871-534-7.

External links

- An Introduction to the Conjugate Gradient Method Without the Agonizing Pain (http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf) by Jonathan Richard Shewchuk.
- Iterative methods for sparse linear systems (http://www-users.cs.umn.edu/~saad/books.html) by Yousef Saad
- LSQR: Sparse Equations and Least Squares (http://www.stanford.edu/group/SOL/software/lsqr.html) by Christopher Paige and Michael Saunders.

Article Sources and Contributors

Conjugate gradient method *Source*: http://en.wikipedia.org/w/index.php?oldid=542340993 *Contributors*: 2andrewknyazev, A. Pichler, Abeliano, Akinoame, Akulo, Alexander Chervov, Aurum1987, Ben pcc, BenFrantzDale, Bgwhite, BigDix56, Bowiki, Bviktor, CES1596, CWii, Caltas, Carthu15, Chris Howard, David Schaich, Dcoetzee, Dheevatsa, Fungus4ever, Gaoyi, Giftlite, Hare, Instinct, Isheden, Jitse Niesen, Jj137, Jrennie, K.menin, Kri, Kxx, Lambiam, Leftynm, Lerichard, Luc4, Magioladitis, Mange01, Mardebikas, Martpol, Michael Hardy, Mschures, Mythealias, Nancy, Ndgoalby, Nimur, Oleg Alexandrov, Osck, PAR, Paranoidhumanoid, PerryTachett, Philefou, Pilloff, Pupdike, Raul654, Rjwilmsi, Sabamo, Salih, SeldonPlan, Simiprof, Smarchesini, Tdoune, Tgdwyer, Thegeneralguy, TomyDuby, Tong, Trisapeace, Troy 07, Xlicolts613, Zweiblum, 170 anonymous edits

Image Sources, Licenses and Contributors

File:Conjugate gradient illustration.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Conjugate_gradient_illustration.svg *License*: Public Domain *Contributors*: Oleg Alexandrov

License