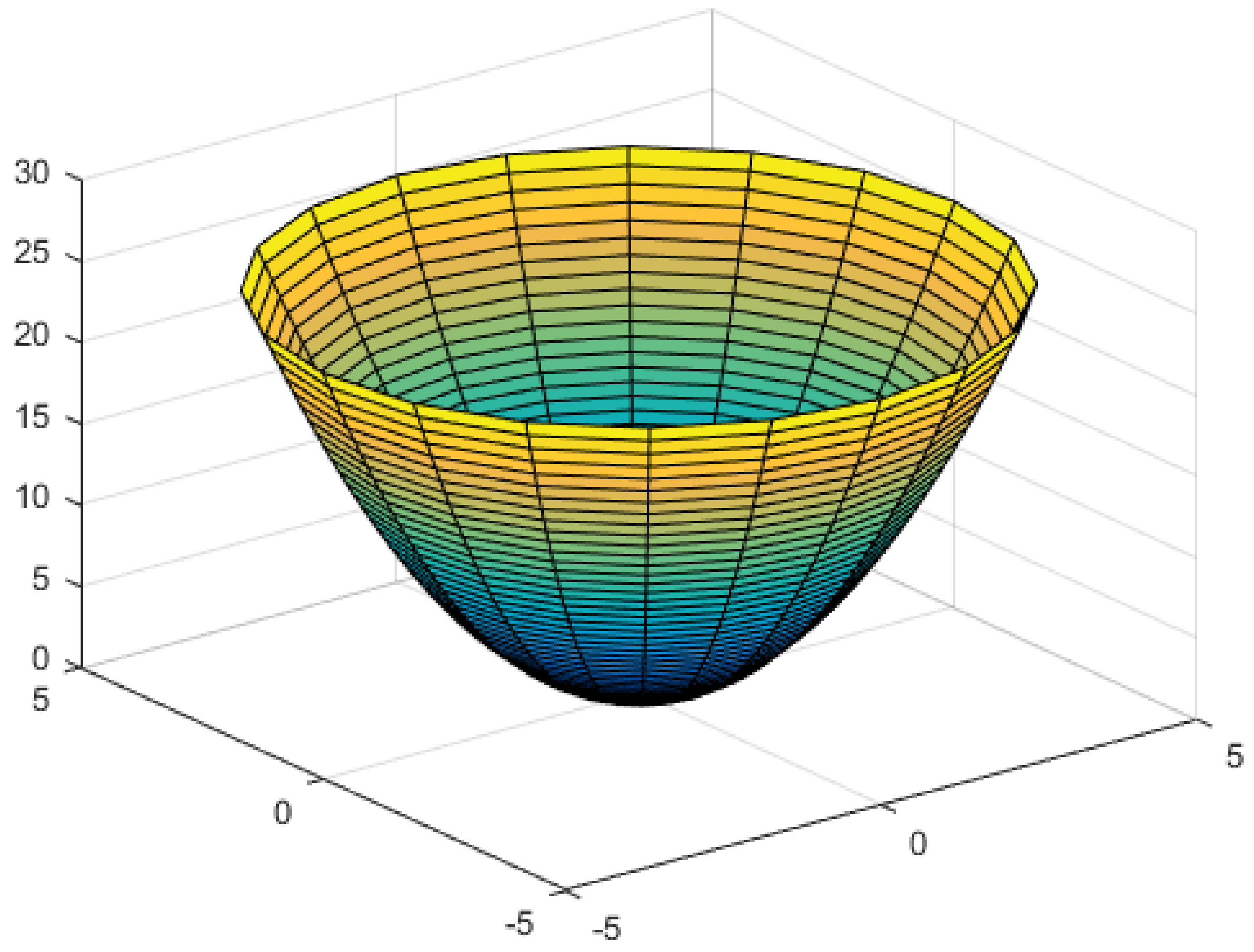
The Conjugate Gradient Method for Solving Linear Systems of Equations

Mike Rambo

Mentor: Hans de Moor



May 2016

Department of Mathematics, Saint Mary's College of California

Contents

Abstract

The Conjugate Gradient Method is an iterative technique for solving large sparse systems of linear equations. As a linear algebra and matrix manipulation technique, it is a useful tool in approximating solutions to linearized partial differential equations. The fundamental concepts are introduced and utilized as the foundation for the derivation of the Conjugate Gradient Method. Alongside the fundamental concepts, an intuitive geometric understanding of the Conjugate Gradient Method's details are included to add clarity. A detailed and rigorous analysis of the theorems which prove the Conjugate Gradient algorithm are presented. Extensions of the Conjugate Gradient Method through preconditioning the system in order to improve the efficiency of the Conjugate Gradient Method are discussed.

1 Introduction

The Method of Conjugate Gradients (cg-method) was initially introduced as a direct method for solving large systems of n simultaneous equations and n unknowns. Eduard Stiefel of the Institute of Applied Mathematics at Zürich and Magnus Hestenes of the Institute for Numerical Analysis, met at a symposium held at the Institute for Numerical Analysis in August 1951 where one of the topics discussed was solving systems of simultaneous linear equations. Hestenes and Stiefel collaborated to write the first papers on the method, published as early as December 1952. During the next two decades, the cg-method was utilized heavily, but not accepted by the numerical analysis community due to a lack of understanding of the algorithms and their speed. Hestenes and Stiefel had suggested the use of the cg-method as an iterative technique for well-conditioned problems; however, it was not until the early 1970's that researchers were drawn to the cg-method as an iterative method. In current research the cg-method is understood as an iterative technique with a focus on improving and extending the cg-method to include different classes of problems such as unconstrained optimization or singular matrices.

2 Relevant Definitions and Notation

Throughout this paper, the following definitions and terminology will be utilized for the cg-method.

Defintion 2.1. Partial Differential Equations (PDEs): an equation in which the solution is a function which has two or more independent variables.

Remark. The cg-method is utilized to estimate solutions to linear PDEs. Linear PDEs are PDEs in which the unknown function and its derivatives all have power 1.

A solution to a PDE can be found by linearizing into a system of linear equations:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots \qquad \vdots \qquad \ddots \qquad \vdots\ &= \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

This system of equations will be written in matrix form $Ax = b$, where

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \qquad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

The matrix A is referred to as the **Coefficient Matrix**, and x the **Solution Matrix**.

Defintion 2.2. A matrix is defined to be **sparse** if a large proportion of its entries are zero.

Defintion 2.3. A $n \times n$ matrix A is defined to be symmetric if $A = A^T$.

Defintion 2.4. A $n \times n$ matrix A is **Positive Definite** if $\mathbf{x}^{\mathbf{T}}A\mathbf{x} > 0$ for every n-dimensional column matrix $\mathbf{x} \neq 0$, where $\mathbf{x}^{\mathbf{T}}$ is the **transpose** of vector $\mathbf{x}$.

Remark. The notation $\langle \mathbf{x}, A\mathbf{x} \rangle$ is commonly used. This notation is called the **inner product**, and $\langle \mathbf{x},\ A\mathbf{x} \rangle = \mathbf{x}^{\mathbf{T}}A\mathbf{x}$.

Typically systems of equations arising from linearizing PDEs yield coefficient matrices which are sparse. However, when discussing large sparse matrix solutions, it is not sufficient to know that the coefficient matrix is sparse. Systems arising from linear PDEs have a coefficient matrix which have non-zero diagonals and all other entries zero. When programming the cg-method, a programmer can utilize the sparse characteristic and only program the non-zero diagonals to reduce storage and improve efficiency. In order for the cg-method to yield a solution the coefficient matrix needs to be symmetric and positive definite. Because the inner product notation is utilized throughout the rest of this paper, it is useful to know some of the basic properties of the inner product.

Theorem 2.1. [4] *For any vectors* $\mathbf{x}, \mathbf{y}$, *and* $\mathbf{z}$ *and any real number* α, *the following are true:*

1. $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$.
2. $\langle \alpha\mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{x}, \alpha\mathbf{y} \rangle = \alpha\langle \mathbf{x}, \mathbf{y} \rangle$
3. $\langle \mathbf{x} + \mathbf{z}, \mathbf{y} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{z}, \mathbf{y} \rangle$
4. $\langle \mathbf{x}, \mathbf{x} \rangle \geq \mathbf{0}$
5. $\langle \mathbf{x}, \mathbf{x} \rangle = \mathbf{0}$ *if and only if* $\mathbf{x} = 0$ $\square$

Defintion 2.5. The Gradient is defined as ∇F where ∇ is called **the Del operator** and is the sum of the partial derivatives with respect to each variable of F.

Eg. *For a function $F(x, y, z)$, then ∇ is $\frac{\partial}{\partial x}\hat{x} + \frac{\partial}{\partial y}\hat{y} + \frac{\partial}{\partial z}\hat{z}$, and ∇F can be written as*

$$\begin{aligned}\nabla F &= \Big(\frac{\partial}{\partial x}\hat{x} + \frac{\partial}{\partial y}\hat{y} + \frac{\partial}{\partial z}\hat{z}\Big)F \\ &= \frac{\partial F}{\partial x}\hat{x} + \frac{\partial F}{\partial y}\hat{y} + \frac{\partial F}{\partial z}\hat{z}.\end{aligned}$$

Remark. The gradient will be used to find the direction in which a scalar-valued function of many variables increases most rapidly.

Theorem 2.2. [9] *Let the scalar σ be defined by*

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

where $\mathbf{b}$ and $\mathbf{x}$ are $n \times 1$ matrices and $\mathbf{b}$ is independent of $\mathbf{x}$, then

$$\frac{\partial \sigma}{\partial \mathbf{x}} = \mathbf{b}$$

□

Theorem 2.3. [9] *Let σ be defined by*

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

where A is a $n \times n$ matrix, $\mathbf{x}$ is a $n \times 1$ matrix, and A is independent of $\mathbf{x}$, then

$$\frac{\partial \sigma}{\partial \mathbf{x}} = (A^T + A)\mathbf{x}$$

□

3 Linearization of Partial Differential Equations

There are several established methods to linearize PDEs. For the following example for linearizing the one-dimensional heat equation, the **Forward Difference Method** is utilized. Note that this process will work for all linear PDEs. Consider the one-dimensional heat equation:

$$\frac{\partial \phi}{\partial t}(x,t) = \alpha \frac{\partial^2 \phi}{\partial x^2}(x,t); \qquad 0 < x < l, \quad t > 0$$

This boundary value problem has the properties that the endpoints remain constant throughout time at 0, so $\phi(0,t) = \phi(l,t) = 0$. Also at time = 0, $\phi(x,0) = f(x)$, which describes the whole system from $0 \leq x \leq l$.

The forward difference method utilizes Taylor series expansion. A Taylor expansion yields infinitely many terms, so the utilization of **Taylor′s formula with remainder** which allows the truncation of the Taylor series at n derivatives with an error R is also needed.

Theorem 3.1. [6] **Taylor′s formula with remainder** *If f and its first n derivatives are continuous in a closed interval $I : \{a \leq x \leq b\}$, and if $f^{n+1}(x)$ exists at each point of the open interval $J : \{a < x < b\}$, then for each x in I there is a β in J for which*

$$R_{n+1} = \frac{f^{n+1}(\beta)}{(n+1)!}(x-a)^{n+1}.$$

□

First, the x-axis is split into equal segments of length s, and a step size of $\Delta x = \frac{l}{s}$. For a time step size Δt, note that $(x_i, t_j) = (i\Delta x, j\Delta t)$ for $0 \leq i \leq s$ and $j \geq 0$.
By expanding ϕ using a Taylor series expansion in t,

$$\phi(x_i, t_i + \Delta t) = \phi(x_i, t_i) + \Delta t \frac{\partial \phi}{\partial t}\Big|_{t_j} + \frac{\Delta t^2}{2!}\frac{\partial^2 \phi}{\partial t^2}\Big|_{t_j} + \cdots$$

Solving for $\frac{\partial \phi}{\partial t}$,

$$\frac{\partial \phi}{\partial t}\Big|_{t_i} = \frac{\phi(x_i, t_j + \Delta t) - \phi(x_i, t_j)}{\Delta t} - \frac{\Delta t}{2!}\frac{\partial^2 \phi}{\partial t^2}\Big|_{t_j} - \cdots$$

By Taylor's formula with remainder, the higher order derivatives can be replaced with $\frac{\Delta t^2}{2}\frac{\partial^2 \phi}{\partial t^2}(x_i, \tau_j)$, for some $\tau_j \in (t_{j-1}, t_{t+1})$ so $\frac{\partial \phi}{\partial t}$ is reduced to

$$\frac{\partial \phi}{\partial t}\Big|_{t_j} = \frac{\phi(x_i, t_j + \Delta t) - \phi(x_i, t_j)}{\Delta t} - \frac{\Delta t^2}{2}\frac{\partial^2 \phi}{\partial t^2}\Big|_{\tau}$$

By a similar Taylor expansion,

$$\frac{\partial^2 \phi}{\partial x^2} = \frac{\phi(x_i + \Delta x, t_j) - 2\phi(x_i, t_j) + \phi(x_i - \Delta x, t_j)}{\Delta x^2} - \frac{\Delta x^2}{12}\frac{\Delta^4 \phi}{\Delta x^4}(\chi, t_j),\ \chi \in (x_{i-1}, x_{x+1})$$

By substituting back into the heat equation and writing $\phi(x_i, t_j)$ as $\Phi_{i,j}$,

$$\frac{\Phi_{i,j+1} - \Phi_{i,j}}{\Delta t} = \alpha \frac{\Phi_{i+1,j} - 2\Phi_{i,j} + \Phi_{i-1,j}}{\Delta x^2}$$

with an error of

$$\xi = \frac{\Delta t}{2}\frac{\partial^2 \phi}{\partial t^2}(x_i, \tau_j) - \alpha \frac{\Delta^2}{12}\frac{\partial^4 \phi}{\partial x^4}(\chi_i, t_j)$$

Solving for $\Phi_{i,j+1}$ as a function of $\Phi_{i,j}$, which is found to be:

$$\Phi_{i,j+1} = (1 - \frac{2\alpha\Delta t}{\Delta x^2})\Phi_{i,j} + \alpha\frac{\Delta t}{\Delta x^2}(\Phi_{i+1,j} + \Phi_{i-1,j}) \text{ for each } i = 0, 1, \cdots, s-1 \text{ and } j = 0, 1, \cdots$$

Now by choosing initial values $i = 0$ and $j = 0$, the first vector for the given system is:

$$\Phi_{0,0} = f(x_0), \quad \Phi_{1,0} = f(x_1), \ . \ . \ . \ \Phi_{s,0} = f(x_s)$$

Building the next vectors,

$$\begin{aligned}
\Phi_{0,1} &= \phi(0, t_1) = 0 \\
\Phi_{1,1} &= (1 - \frac{2\alpha\Delta t}{\Delta x^2})\Phi_{1,0} + \alpha\frac{\Delta t}{\Delta x^2}(\Phi_{2,0} + \Phi_{0,0}) \\
&\vdots \\
\Phi_{s-1,1} &= (1 - \frac{2\alpha\Delta t}{\Delta x^2})\Phi_{s-1,0} + \alpha\frac{\Delta t}{\Delta x^2}(\Phi_{s,0} + \Phi_{s-2,0}) \\
\Phi_{s,1} &= \phi(s, t_1) = 0
\end{aligned}$$

By continuing to build these vectors in s, there is a pattern which can be written as the matrix equation $\theta^{(j)} = A\theta^{(j-1)}$, for each successive $j > 0$ and where A is the $(s-1)$ x $(s-1)$ matrix

$$\begin{bmatrix}
(1-2\beta) & \beta & 0 & \cdots & 0 \\
\beta & (1-2\beta) & \ddots & \ddots & \vdots \\
0 & \ddots & \ddots & \ddots & 0 \\
\vdots & \ddots & \ddots & \ddots & \beta \\
0 & \cdots & 0 & \beta & (1-2\beta)
\end{bmatrix}$$

where $\beta = \alpha\frac{\Delta t}{\Delta x^2}$. A acts on the vector $\theta^{(j)} = (\Phi_{1,j}, \Phi_{2,j}, \cdots, \Phi_{s-1,j})^T$ and produces $\theta^{(j+1)}$, the next vector after a time step of j. A is a tri-diagonal matrix with an upper and lower triangular section of zeroes. For any linear PDE, the matrix equations derived using finite difference methods will have a coefficient matrix A which is symmetric, positive definite and sparse. Recall that because the non-zero terms of A are the three diagonals and the rest of the terms are zero, a programmer will be able to just program the diagonals as vectors which makes the space-complexity of this matrix for the cg-method small.

4 The Mechanisms of the Conjugate Gradient Method

4.1 Steepest Descent

There are a few fundamental techniques utilized to find solutions to simultaneous systems of equations derived from linear PDEs. A first attempt may be to try direct methods, such as Gaussian elimination, which would yield exact solutions to the system. However, for large systems, Gaussian elimination on a sparse matrix system eliminates the sparse characteristic of the matrix, making this very computation and memory intensive. The next theorem illustrates a geometry of the method. A proof will be stated later.

Theorem 4.1. [4] *The solution vector x^* to the system $Ax = b$ where A is positive definite and symmetric, is the minimal value of the quadratic form $f(x) = \frac{1}{2}\langle x, Ax\rangle - \langle x, b\rangle$.* □

As an illustration, consider a simple 2×2 system. The example provided by equation (1) is pictured in ($figure$ 1).

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \tag{1}$$

This system represents the lines

$$2x + y = 3$$
$$x + y = 2$$

For this example, the function $f(x)$ from $Theorem$ 4.1, will project a paraboloid into a three dimensional graph (figure 2). Note that this does not lose generality for higher dimensions. The cg-method applies in orders that cannot be represented geometrically.

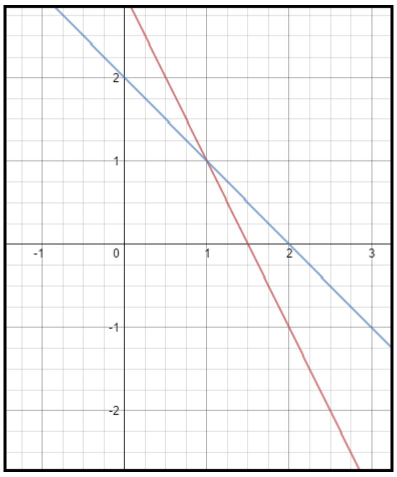


Figure 1: *A system of two equations and two unknowns; the red equation is $2x+y=3$, the blue equation is $x+y=2$*

Note also that *Theorem* 4.1, leads to the fact that the *minimization* of $f(x)$ is the solution to the system of interest. For a 2×2 system, the vertex is the minimum value of the paraboloid (figure 3). As noted earlier, direct methods are very time consuming, however, the paraboloid yields potential solutions through less direct or iterative solutions. Several iterative algorithms produce the minimal value. The method presented here utilizes the method of *Steepest Decent*.

The concept of steepest decent is simple: start at an arbitrary point anywhere on the paraboloid and follow the steepest curve to the minimized value of the paraboloid. Start at an arbitrary point $x_{(0)}$ and then take a series of iterations, $x_{(1)}, x_{(2)}, \cdots$. Stop when $x_{(i)}$ is sufficiently close to the solution x^*.

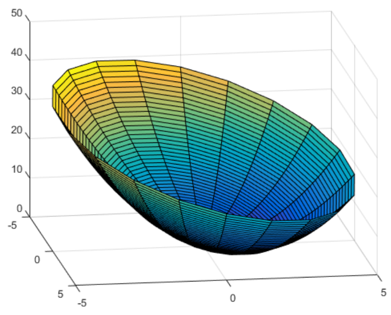


Figure 2: *The paraboloid formed from the quadratic form from Theorem 4.1 for example (1)*

For any estimate $x_{(i)}$, the solution x^* can be found by taking the direction that the quadratic forms decreases most rapidly, that is taking the negative gradient, $-\nabla f(x_i)$. By a direct computation,

$$
\begin{aligned}
-\nabla f(x) &= -\nabla\left(\frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle\right) \\
&= -\nabla\left(\frac{1}{2}x^T * Ax - b^T x\right) \\
&= -\left(\frac{1}{2}A^T x + \frac{1}{2}Ax - b\right) && \text{(by Theorem 2.2 and Theorem 2.3)}
\end{aligned}
$$

Since A is symmetric, it can be simplified further to be

$$= -(Ax - b)$$

$$-\nabla f(x) = b - Ax \tag{2}$$

So for each x_i, $-\nabla f(x_i) = b - Ax_i$. Note that this definition of the direction of steepest descent is exactly equal to the *residual*. A *residual* is the difference between the real value and the estimated

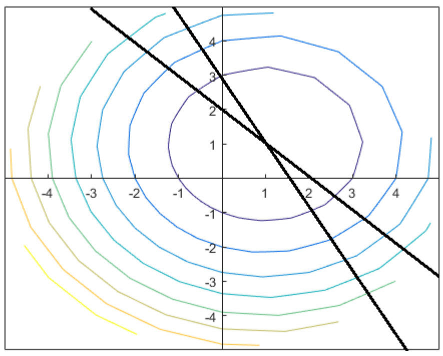


Figure 3: *Contour map of paraboloid with the linear system from example (1) with solutions at the vertex of the paraboloid*

value. In this case the real value is b and the estimated value is Ax_i, so the residual is $r_i = b - Ax_i$. This residual indicates how far away from the value of b the approximate step is.

For the best convergence to the solution it is ideal to travel in the direction of steepest descent. So, each subsequent step is some scalar multiple of the direction of steepest descent. Determining how far needs to be defined. The equation is then

$$x_1 = x_0 + s_0 r_0$$

where s is the scalar for how far in the direction of the residual each iteration descends. For successive steps, this equation is

$$x_{i+1} = x_i + s_i r_i$$

In each successive step the search direction $d_{i+1} = r_i$ is chosen. In order to solve for each s_i, a procedure known as a *line search* evaluates s to minimize the value of f along the line in the direction of steepest descent. First, consider $f(x_{i+1})$ and minimize s_i, that is, take the derivative of $f(x_{i+1})$ with respect to s_i:

$$\begin{aligned} \frac{d}{ds_i} f(x_{i+1}) &= f'(x_{i+1})^T \frac{d}{ds_i} x_{i+1} && \text{(Chain Rule)} \\ &= f'(x_{i+1})^T \frac{d}{ds_i}(x_i + s_i r_i) \\ &= f'(x_{i+1})^T r_i && (3) \end{aligned}$$

To find the minimum of s_i set equation (3) equal to zero

$$f'(x_{i+1})^T r_i = 0$$

Recognize that because $f'(x_{i+1})^T$ and r_i are vectors whose product is 0 then either one vector is zero or they are orthogonal. Since neither vector can be zero, they must be orthogonal. As indicated earlier, $-f'(x_{i+1}) = b - Ax_{i+1}$, so

$$\begin{aligned}
f'(x_{i+1})^T r_i &= 0 \\
(b - Ax_{i+1})^T r_i &= 0 \\
[b - A(x_i + s_i r_i)]^T r_i &= 0 \\
[b - Ax_i - s_i(Ar_i)]^T r_i &= 0 \\
(b - Ax_i)^T r_i - s_i(Ar_i)^T r_i &= 0 \\
(b - Ax_i)^T r_i &= s_i(Ar_i)^T r_i \\
\frac{(b - Ax_i)^T r_i}{(Ar_i)^T r_i} &= s_i \\
\frac{r_i^T r_i}{(Ar_i)^T r_i} &= s_i
\end{aligned}$$

Another way of writing s_i is in inner product notation:

$$s_i = \frac{\langle r_i, r_i \rangle}{\langle Ar_i, r_i \rangle} \tag{4}$$

First, r_0 is calculated, then repeated using $r_i = b - Ax_i$ in order to calculate x_{i+1} which can be computed by $x_{i+1} = x_i + s_i r_i$ where $s_i = \dfrac{\langle r_i, r_i \rangle}{\langle Ar_i, r_i \rangle}$. A quick consideration of the computational algorithm shows that the algorithm is dominated by two matrix products, first Ax_i, then Ar_i. Ax_i can be eliminated by

$$\begin{aligned}
x_{i+1} &= x_i + s_i r_i \\
-A(x_{i+1}) &= -A(x_i + s_i r_i) \\
-Ax_{i+1} &= -Ax_i + s_i(-Ar_i) \\
-Ax_{i+1} + b &= -Ax_i + b - s_i Ar_i \\
r_{i+1} &= r_i - s_i Ar_i
\end{aligned}$$

Now it is only necessary to compute Ar_i for both equations. What the algorithm gains in efficiency it loses in accuracy, because x_i is no longer being computed, so the error may increase. This suggests that computing a new x_i at regular intervals helps reduce the error.

Before moving on to improvements, the proof of *Theorem* 4.1, follows.

Theorem 4.1 [4] The solution vector x^* to the system $Ax = b$ where A is positive definite and symmetric, is the minimal value of the quadratic form $f(x) = \frac{1}{2}\langle x, Ax \rangle - \langle x, b \rangle$.

Proof: Let x and d be fixed vectors not equal to 0 and let $m \in \mathbb{R}$. Then,

$$\begin{aligned} f(x+md) &= \frac{1}{2}\langle x+md, A(x+md)\rangle - \langle x+md, b\rangle \\ &= \frac{1}{2}\langle x, Ax\rangle + \frac{m}{2}\langle d, Ax\rangle + \frac{m}{2}\langle x, Ad\rangle + \frac{m^2}{2}\langle d, Ad\rangle - \langle x, b\rangle - m\langle d, b\rangle \\ &= \frac{1}{2}\langle x, Ax\rangle - \langle x, b\rangle + m\langle d, Ax\rangle - m\langle d, b\rangle + \frac{m^2}{2}\langle d, Ad\rangle \end{aligned}$$

Notice that $\frac{1}{2}\langle x, Ax\rangle - \langle x, b\rangle = f(x)$ and $m\langle d, Ax\rangle - m\langle d, b\rangle = m\langle d, Ax-b\rangle$, so this can be rewritten as

$$f(x+md) = f(x) - m\langle d, b-Ax\rangle + \frac{m^2}{2}\langle d, Ad\rangle$$

Because x and d are fixed vectors, the quadratic function $g(m)$ is defined by

$$g(m) = f(x+md)$$

The minimum value of $g(m)$ is when $g'(m) = 0$, because the m^2 coefficient is positive. Then

$$g'(m) = 0 = -\langle d, b-Ax\rangle + m\langle d, Ad\rangle$$

solving for m,

$$m' = \frac{\langle d, b-Ax\rangle}{\langle d, Ad\rangle}$$

Substituting m' into $g(m)$,

$$\begin{aligned} g(m') &= f(x+m'd) \\ &= f(x) - m'\langle d, b-Ax\rangle + \frac{(m')^2}{2}\langle d, Ad\rangle \\ &= f(x) - \frac{\langle d, b-Ax\rangle}{\langle d, Ad\rangle}\langle d, b-Ax\rangle + \frac{1}{2}\left(\frac{\langle d, b-Ax\rangle}{\langle d, Ad\rangle}\right)^2\langle d, Ad\rangle \\ &= f(x) - \frac{1}{2}\frac{\langle d, b-Ax\rangle^2}{\langle d, Ad\rangle} \end{aligned}$$

Because $\frac{\langle d, b-Ax\rangle^2}{\langle d, Ad\rangle}$ is always positive or zero, for any vectors $d \neq 0$, $f(x+m'd) < f(x)$, or when $\langle d, b-Ax\rangle = 0$, $f(x+m'd) = f(x)$.

Suppose that x^* satisfies, $Ax = b$. Then $\langle d, b-Ax^*\rangle = 0$, and $f(x)$ cannot be made any smaller than $g(x^*)$. Thus, x^* minimizes $f(x)$.

Next, suppose that x^* minimizes $f(x)$. Then for any vector d, $f(x^*, t_m d) \geq g(x^*)$. Thus, $\langle d, b-Ax^*\rangle = 0$. Since d is non-zero, this implies that $b - Ax^* = 0$. Therefore $Ax^* = b$.

$\square$

4.2 Conjugate Directions

The algorithm just derived is the steepest descent algorithm. This algorithm is not sufficient for the cg-method because the convergence is slow. The convergence is slow because by following the negative gradient the search directions are often chosen in the same directions as previously searched directions. In order to improve the convergence, the condition that the search directions are never in the same direction as any of the previous search directions is imposed on the algorithm. This condition is to choose a set of nonzero vectors $\{d^{(1)}, \cdots, d^{(n)}\}$ such that $\langle d^{(i)}, Ad^{(j)} \rangle = 0$ if $i \neq j$. This is known as the **A-orthogonality condition**.

Simply choosing vectors to be orthogonal is not sufficient because in choosing a starting point and an initial search direction, there would only be a single vector orthogonal to the search direction which would result in the solution. Unfortunately, in order to choose the correct orthogonal vector and both scalars it would be necessary to know the solution. Clearly, this is not a viable solution.

Imposing A-orthogonality on the steepest descent algorithm and utilizing *Theorem* 4.1, the new s_i are given by

$$s_i = \frac{\langle d^{(i)}, r^{(i)} \rangle}{\langle d^{(i)}, Ad^{(i)} \rangle} \qquad \text{for } r_i = b - Ax^{(i)}$$

The next step is still calculated by

$$x^{(i+1)} = x^{(i)} + s_i d^{(i)}$$

The next theorem shows the full power of this choice of search directions.

Theorem 4.2. [4] *Let $\{d^{(1)}, \cdots, d^{(n)}\}$ be an A-orthogonal set of nonzero vectors associated with the positive definite symmetric matrix A, and let $x^{(0)}$ be arbitrary. Define*

$$s_k = \frac{\langle d^{(k)}, b - Ax^{(k-1)} \rangle}{\langle d^{(k)}, Ad^{(k)} \rangle} \qquad x^{(k)} = x^{(k-1)} + s_k d^{(k)}$$

for $k = 1, 2, \cdots, n$. Then, assuming exact arithmetic, $Ax^{(n)} = b$.

Proof: Consider the A-orthogonal set associated with a positive definite symmetric matrix A, $\{d^{(1)}, \cdots, d^{(n)}\}$. Let $x^{(0)}$ be arbitrary. So, for the n^{th} vector,

$$x^{(n)} = x^{(n-1)} + s_n d^{(n)} \tag{5}$$

The $x^{(n-1)}$ in equation (5) can be expanded to $x^{(n-1)} = x^{(n-2)} + s_{n-1}d^{(n-1)}$. By continuing this process, the $x^{(k)}$ terms can be continually replaced with the $x^{(k-1)} + s_k d^{(k)}$ to obtain

$$x^{(n)} = x^{(0)} + s_1 d^{(1)} + s_2 d^{(2)} + \cdots + s_n d^{(n)}$$

multiplying both sides by matrix A, and subtracting b from both sides,

$$Ax^{(n)} - b = Ax^{(0)} - b + s_1 Ad^{(1)} + s_2 Ad^{(2)} + \cdots + s_n Ad^{(n)}$$

Now take the inner product of both sides with the vector $d^{(k)}$. The A-orthogonality property says that for any $m \neq k$, such that $0 \leq m \leq n$, the inner product is 0. So,

$$\langle Ax^{(n)} - b, d^{(k)} \rangle = \langle Ax^{(0)} - b, d^{(k)} \rangle + s_k \langle Ad^{(k)}, d^{(k)} \rangle$$

By the fact that A is symmetric this can be rewritten as

$$\langle Ax^{(n)} - b, d^{(k)} \rangle = \langle Ax^{(0)} - b, d^{(k)} \rangle + s_k \langle d^{(k)}, Ad^{(k)} \rangle$$

Since $s_k = \dfrac{\langle d^{(k)}, b - Ax^{(k-1)} \rangle}{\langle d^{(k)}, Ad^{(k)} \rangle}$, then $s_k \langle d^{(k)}, Ad^{(k)} \rangle = \langle d^{(k)}, b - Ax^{(k-1)} \rangle$. By expanding the right hand side,

$$\begin{aligned} \langle d^{(k)}, b - Ax^{(k-1)} \rangle &= \langle d^{(k)}, b - Ax^{(0)} + Ax^{(0)} - Ax^{(1)} + Ax^{(1)} + \cdots - Ax^{(k-2)} + Ax^{(k-2)} + Ax^{(k-1)} \rangle \\ &= \langle d^{(k)}, b - Ax^{(0)} \rangle + \langle d^{(k)}, Ax^{(0)} - Ax^{(1)} \rangle + \cdots + \langle d^{(k)}, Ax^{(k-2)} - Ax^{(k-1)} \rangle \qquad (6) \end{aligned}$$

For any i

$$x^{(i)} = x^{(i-1)} + s_i d^{(i)}$$
$$Ax^{(i)} = Ax^{(i-1)} + s_i Ad^{(i)}$$

$$Ax^{(i-1)} - Ax^{(i)} = -s_i Ad^{(i)} \qquad (7)$$

Substituting equation (7) into equation (6),

$$\langle d^{(k)}, b - Ax^{(k-1)} \rangle = \langle d^{(k)}, b - Ax^{(0)} \rangle - s_1 \langle d^{(k)}, Ad^{(1)} \rangle + \cdots - s_{k-1} \langle d^{(k)}, Ad^{(k-1)} \rangle$$

By A-orthogonality $\langle d^{(k)}, Ad^{(m)} \rangle = 0$ if $k \neq m$, so

$$s_k \langle d^{(k)}, Ad^{(k)} \rangle = \langle d^{(k)}, b - Ax^{(k-1)} \rangle = \langle d^{(k)}, b - Ax^{(0)} \rangle$$

Thus,

$$\begin{aligned} \langle Ax^{(n)} - b, d^{(k)} \rangle &= \langle Ax^{(0)} - b, d^{(k)} \rangle + s_k \langle Ad^{(k)}, d^{(k)} \rangle \\ &= \langle Ax^{(0)} - b, d^{(k)} \rangle + \langle d^{(k)}, b - Ax^{(0)} \rangle \\ &= \langle Ax^{(0)} - b, d^{(k)} \rangle - \langle d^{(k)}, Ax^{(0)} - b \rangle \\ &= \langle Ax^{(0)} - b, d^{(k)} \rangle - \langle Ax^{(0)} - b, d^{(k)} \rangle \\ &= 0 \end{aligned}$$

Therefore $\langle Ax^{(n)} - b, d^{(k)} \rangle = 0$. Because $d^{(k)} \neq 0$, $Ax^{(n)} - b = 0$. Therefore $Ax^{(n)} = b$ and $Ax^{(n)} - b$ is orthogonal to the A-orthogonal set of vectors.

□

4.3 Conjugate Gradient Method

A method utilizing an A-orthogonal set of vectors is called a *Conjugate Direction* method. The cg-method is a method of conjugate directions which chooses the residual vectors to be mutually orthogonal. Recall that the residual is defined by $r^{(k)} = b - Ax^{(k)}$. Hestenes and Stiefel chose their search directions, $d^{(k+1)}$, in the cg-method by choosing the residual vectors, $r^{(k)}$, in a way such that they are mutually orthogonal.

Similar to steepest descent, the algorithm starts with an initial approximation $x^{(0)}$ and takes the first direction to be the residual from steepest descent $d^{(1)} = r^{(0)} = b - Ax^{(0)}$. Each approximation is computed by $x^{(k)} = x^{(k-1)} + s_k d^{(k)}$. Applying the A-orthogonality condition from conjugate directions in order to speed up convergence, $\langle d^{(i)}, Ad^{(j)}\rangle = 0$ and $\langle r^{(i)}, r^{(j)}\rangle = 0$ when $i \neq j$. There are two stopping conditions. The first is calculating a residual to be zero, which means

$$\begin{aligned} r^{(k)} &= b - Ax^{(k)} \\ 0 &= b - Ax^{(k)} \\ Ax^{(k)} &= b \end{aligned}$$

This implies that the solution is $x^{(k)}$.

The second stopping condition is if the residual is sufficiently close to 0, within some specified tolerance. The next search direction $d^{(k+1)}$, is computed using the residual from the previous iteration $r^{(k)}$, by

$$d^{(k+1)} = r^{(k)} + c_k d^{(k)}$$

In order for the A-orthogonal condition to hold, c_k must be chosen in such a way that $\langle d^{(k+1)}, Ad^{(k)}\rangle = 0$. In order to figure out what the choice of c_k should be, multiply by A and take the inner product with the previous search direction $d^{(k)}$, to the left side to obtain:

$$\langle d^{(k)}, Ad^{(k+1)}\rangle = \langle d^{(k)}, Ar^{(k)}\rangle + c_k\langle d^{(k)}, Ad^{(k)}\rangle$$

By the A-orthogonality condition, $\langle d^{(k)}, Ad^{(k+1)}\rangle = 0$, so

$$0 = \langle d^{(k)}, Ar^{(k)}\rangle + c_k\langle d^{(k)}, Ad^{(k)}\rangle$$

Then solving for c_k,

$$c_k = -\frac{\langle d^{(k)}, Ar^{(k)}\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle}$$

Next solving for the scalar s_k, from conjugate directions,

$$s_k = \frac{\langle d^{(k)}, r^{(k-1)}\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle}$$

Applying the chosen definition of $d^{(k+1)}$,

$$\begin{aligned} s_k &= \frac{\langle d^{(k)}, r^{(k-1)}\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle} \\ &= \frac{\langle r^{(k-1)} + c_{k-1}d^{(k-1)}, r^{(k-1)}\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle} \\ &= \frac{\langle r^{(k-1)}, r^{(k-1)}\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle} + c_{k-1}\frac{\langle d^{(k-1)}, r^{(k-1)}\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle} \end{aligned}$$

Substituting $d^{(k-1)}$ using the definition $d^{(k+1)} = r^{(k)} + c_k d^{(k)}$ to obtain:

$$s_k = \frac{\langle r^{(k-1)}, r^{(k-1)}\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle} + c_{k-1}\frac{\langle r^{(k-2)}, r^{(k-1)}\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle} + c_{k-1}\frac{\langle c_{k-2}d^{(k-2)}, r^{(k-1)}\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle}$$

Because of the orthogonality condition of residuals, $\langle r^{(k-2)}, r^{(k-1)}\rangle = 0$. Clearly, by continuing to expand the $d^{(i)}$ terms none of the terms will contain $r^{(k-1)}$. So, every term except for the $\langle r^{(k-1)}, r^{(k-1)}\rangle$ term is zero. So,

$$s_k = \frac{\langle r^{(k-1)}, r^{(k-1)}\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle}$$

Lastly, calculating the residuals, starting with the definition of $x^{(k)}$,

$$x^{(k)} = x^{(k-1)} + s_{(k)}d^{(k)}$$

By multiplying by $-A$, and adding b the equation becomes,

$$b - Ax^{(k)} = b - Ax^{(k-1)} - s_k Ad^{(k)}$$

Now utilizing the definition of $r^{(k)}$,

$$r^{(k)} = r^{(k-1)} - s_k Ad^{(k)}$$

These formulas complete the Conjugate Gradient method. The initial assumptions are therefore:

$$r^{(0)} = b - Ax^{(0)} \qquad d^{(1)} = r^{(0)}$$

and for each successive step $k = 1, \cdots, n$,

$$s_k = \frac{\langle r^{(k-1)}, r^{(k-1)}\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle} \qquad x^{(k)} = x^{(k-1)} + s_k d^{(k)} \qquad r^{(k)} = r^{(k-1)} - s_k Ad^{(k)}$$

$$c_k = -\frac{\langle d^{(k)}, Ar^{(k)}\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle} \qquad d^{(k+1)} = r^{(k)} + c_k d^{(k)}$$

When running the algorithm, it is clear that there are a few calculations that are done more than once, and are therefore repetitive. By eliminating our dependence on these repetitive calculations, the algorithm becomes more efficient. The matrix multiplications in the c_k formula can both be removed by the following:

Starting with the $r^{(k)}$ formula and taking the inner product with itself on the right,

$$\begin{aligned}
r^{(k)} &= r^{(k-1)} - s_k A d^{(k)} \\
\langle r^{(k)}, r^{(k)} \rangle &= \langle r^{(k-1)}, r^{(k)} \rangle - s_k \langle A d^{(k)}, r^{(k)} \rangle \\
\langle r^{(k)}, r^{(k)} \rangle &= -s_k \langle A d^{(k)}, r^{(k)} \rangle \\
-\frac{1}{s_k} \langle r^{(k)}, r^{(k)} \rangle &= \langle A d^{(k)}, r^{(k)} \rangle \\
-\frac{1}{s_k} \langle r^{(k)}, r^{(k)} \rangle &= \langle d^{(k)}, A r^{(k)} \rangle
\end{aligned}$$

Next considering the s_k formula,

$$\begin{aligned}
s_k &= \frac{\langle r^{(k-1)}, r^{(k-1)} \rangle}{\langle d^{(k)}, A d^{(k)} \rangle} \\
s_k \langle d^{(k)}, A d^{(k)} \rangle &= \langle r^{(k-1)}, r^{(k-1)} \rangle \\
\langle d^{(k)}, A d^{(k)} \rangle &= \frac{1}{s_k} \langle r^{(k-1)}, r^{(k-1)} \rangle
\end{aligned}$$

Substituting into the c_k formula,

$$\begin{aligned}
c_k &= -\frac{\langle d^{(k)}, A r^{(k)} \rangle}{\langle d^{(k)}, A d^{(k)} \rangle} \\
&= -\frac{-\frac{1}{s_k} \langle r^{(k)}, r^{(k)} \rangle}{\frac{1}{s_k} \langle r^{(k-1)}, r^{(k-1)} \rangle} \\
&= \frac{\langle r^{(k)}, r^{(k)} \rangle}{\langle r^{(k-1)}, r^{(k-1)} \rangle}
\end{aligned}$$

This eliminated the redundancy of the algorithm, which increases the efficiency substantially. The following is the pseudo code for the cg-method.

```
$r^{(0)} = b - Ax^{(0)}$
$d^{(1)} = r^{(0)}$
$k = 1$
```

while $(k \leq n)$

$$\begin{aligned} s_k &= \frac{\langle r^{(k-1)}, r^{(k-1)} \rangle}{\langle d^{(k)}, Ad^{(k)} \rangle} \\ x^{(k)} &= x^{(k-1)} + s_k d^{(k)} \\ r^{(k)} &= r^{(k-1)} - s_k A d^{(k)} \end{aligned}$$

if r_k is smaller than a tolerance, quit

$$\begin{aligned} c_k &= \frac{\langle r^{(k)}, r^{(k)} \rangle}{\langle r^{(k-1)}, r^{(k-1)} \rangle} \\ d^{(k+1)} &= r^{(k)} + c_k d^{(k)} \\ k &= k+1 \end{aligned}$$

end

For an example, consider the system

$$\begin{bmatrix} 5 & -2 & 0 \\ -2 & 5 & 1 \\ 0 & 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 10 \\ -10 \end{bmatrix}$$

Where the matrix

$$\mathrm{A} = \begin{bmatrix} 5 & -2 & 0 \\ -2 & 5 & 1 \\ 0 & 1 & 5 \end{bmatrix}$$

is positive definite and symmetric. For the cg-method, first set the starting point

$$x^{(0)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

and then calculate the initial assumptions:

$$r^{(0)} = b - Ax^{(0)} = b = \begin{bmatrix} 20 \\ 10 \\ -10 \end{bmatrix}$$

Next, set $d^{(1)} = r^{(0)}$, so that

$$d^{(1)} = \begin{bmatrix} 20 \\ 10 \\ -10 \end{bmatrix}$$

Then running through the first iteration, $k = 1$,

$$s_1 = \frac{\langle r^{(0)}, r^{(0)} \rangle}{\langle d^{(1)}, Ad^{(1)} \rangle} = \frac{3}{10}$$

$$x^{(1)} = x^{(0)} + s_1 d^{(1)} = \begin{bmatrix} 6 \\ 3 \\ -3 \end{bmatrix}$$

$$r^{(1)} = r^{(0)} - s_1 A d^{(1)} = \begin{bmatrix} -4 \\ 10 \\ 2 \end{bmatrix}$$

$$c_1 = \frac{\langle r^{(1)}, r^{(1)} \rangle}{\langle r^{(0)}, r^{(0)} \rangle} = \frac{1}{5}$$

$$d^{(2)} = r^{(1)} + c_1 d^{(1)} = \begin{bmatrix} 0 \\ 12 \\ 0 \end{bmatrix}$$

Next, for the second iteration $k = 2$,

$$s_2 = \frac{\langle r^{(1)}, r^{(1)} \rangle}{\langle d^{(2)}, Ad^{(2)} \rangle} = \frac{1}{6}$$

$$x^{(2)} = x^{(1)} + s_2 d^{(2)} = \begin{bmatrix} 6 \\ 5 \\ -3 \end{bmatrix}$$

$$r^{(2)} = r^{(1)} - s_2 A d^{(2)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Since the residual $r^{(2)} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$, the solution is exactly $x^{(2)}$.

Since this is a small system it is easy to find the exact solution. Unfortunately for larger systems, the cg-method suffers from a significant weakness: round-off error. The solution to the round-off error is *Preconditioning.*

5 Preconditioning

Typically, mathematicians are used to round-off error when it comes to numerical solutions and usually it is not that problematic. However, round-off error from early steps could accumulate and potentially force the cg-method to *fail*. So what can the mathematician do in order to create a more accurate set of formulas? The main cause for round-off error is in a poorly conditioned matrix A. A new $n \times n$ matrix P which is non-singular is introduced to the system. **Non-singular** means that the determinant is nonzero. Since the cg-method erquires the matrix system to be positive definite, by multiplying P to the left and right of A, the new matrix PAP^T is still positive definite. So define,

$$A' = PAP^T$$

In order for the new matrix equation to make sense set $x' = (P^T)^{-1}x$ and $b' = Pb$. The new matrix equation $A'x' = b'$ is now as follows:

$$A'x' = b' \tag{8}$$

By substituting,

$$\begin{aligned} A'x' &= b' \\ (PAP^T)((P^T)^{-1}x) &= Pb \\ PAx &= Pb \end{aligned}$$

Clearly, the original system is left unchanged.

Utilizing these preconditioned matrices the formulas for the cg-method are as follows:

$$s'_{k+1} = \frac{\langle P^{-1}r^{(k)}, P^{-1}r^{(k)}\rangle}{\langle d^{(k+1)}, Ad^{(k+1)}\rangle} \qquad x^{(k+1)} = x^{(k)} + s'_{k+1}d^{(k+1)} \qquad r^{(k+1)} = r^{(k)} - s'_{k+1}Ad^{(k+1)}$$

$$\lambda'_{k+1} = \frac{\langle P^{-1}r^{(k+1)}, P^{-1}r^{(k+1)}\rangle}{\langle P^{-1}r^{(k)}, P^{-1}r^{(k)}\rangle} \qquad d^{(k+1)} = P^{-t}P^{-1}r^k + \lambda'_k d^{(k)}$$

6 Efficiency

Conjugate Gradient	$\mathcal{O}(m\sqrt{K})$
Gaussian Elimination	$\mathcal{O}(n^3)$

Table 1: *Time complexity of the cg-method vs Gaussian elimination*

In table (1) is a comparison of the time complexity of the cg-method and Gaussian elimination. Where m is the number of non-zero terms in the coefficient matrix, and K is the condition number of the coefficient matrix A and n is the dimension of the coefficient matrix. Calculating the condition number requires knowledge of A^{-1} so it is necessary to estimate it through other means. However, by preconditioning a matrix, it has the effect of reducing the value of K. The goal is for K to be as close to 1 as possible so that it is not a factor in the time-complexity for the cg-method. Most numerical analysits agree that a preconditioner should always be used for the cg-method.

Clearly, by comparing the time complexity, the cg-method is more efficient because there are a small proportion of terms which are zero. To Gaussian elimination, since the system loses its sparse characteristic, no matrix is really sparse so its time complexity is worse for larger systems.

7 Conclusion

The Conjugate Gradient algorithm is used in the fields of engineering optimization problems, neural-net training, and in image restoration. The Conjugate Gradient alogirthm is now a grandfather algorithm for a whole class of methods for solving various matrix systems. However, there is still research into expanding the Conjugate Gradient method to more complex systems such as unconstrained optimization or singular matrices.

8 References

[1] R. Chandra, S.C. Eisenstat, and M.H. Schultz, "Conjugate Gradient Methods for Partial Differential Equations," June 1975.

[2] Gerald W. Recktenwald, "Finite-Difference Approximations to the Heat Equation," 6 March 2011.

[3] Magnus R Hestenes and Eduard Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," *Jounral of Research of the National Bureau of Standards* Vol. 49, No. 6, December 1952.

[4] Richard L. Burden and J. Douglas Faires, "Numerical Analysis", ninth edition 2011.

[5] Jonathan Richard Stewchuk, "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain," 4 August 1994.

[6] Watson Fulks, "Advanced Caluclus: An Introduction To Analysis," John Wiley & Sons, Inc. Hoboken, NJ, 1961.

[7] Mark S. Gockenbach " Partial Differential Equations," Society for Industrial and Applied Mathematics. Philadelphia, PA, 2011.

[8] Gene H. Golub and Dianne P. O'Leary "Some History of the Conjugate Gradient and Lanczos Algorithms: 1948-1976" *Siam Review* Vol. 31, No. 1, March 1989.

[9] Randal J. Barnes, "Matrix Differentiation," University of Minnesota. Spring 2006.