Reducing Memory Access Latencies using Data Compression in Sparse, Iterative Linear Solvers

An All-College Thesis

College of Saint Benedict/Saint John's University

by Neil Lindquist
April 2018

Project Title: Reducing Memory Access Latencies using Data Compression in Sparse, Iterative Linear Solvers
Approved by:

Mike Heroux
Scientist in Residence

Robert Hesse
Associate Professor of Math

Jeremy Iverson
Assistant Professor of Computer Science

Bret Benesh
Chair, Department of Mathematics

Imad Rahal
Chair, Department of Computer Science

Director, All College Thesis Program

Abstract

Contents

1 Introduction

2 Background

2.1 Conjugate Gradient

Conjugate Gradient is the iterative solver used by HPCG [1]. Symmetric, positive definite matrices will guarantee the converge of Conjugate Gradient to the correct solution within n iterations, where n is the number of dimensions, when using exact algebra [4]. More importantly, Conjugate Gradient can be used as in iterative method, providing a solution, $\vec{x}$, where $\left\|\mathbf{A}\vec{x} - \vec{b}\right\|$ is within some tolerance, after significantly fewer than n iterations, allowing it to find solutions to problems where even n iterations is infeasible [5].

To understand the Conjugate Gradient, first consider the quadratic form of $\mathbf{A}\vec{x} = \vec{b}$. The quadratic form is a function $f : \mathbb{R}^n \to \mathbb{R}$ where

$$f(\vec{x}) = \tfrac{1}{2}\vec{x}^T\mathbf{A}\vec{x} - \vec{b}\cdot\vec{x} + c \tag{1}$$

for some $c \in \mathbb{R}$. Note that

$$\nabla f(\vec{x}) = \tfrac{1}{2}\left(\mathbf{A} + \mathbf{A}^T\right)\vec{x} - \vec{b}$$

Then, when $\mathbf{A}$ is symmetric,

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

So, the solution to $\mathbf{A}\vec{x} = \vec{b}$ is the sole critical point of f [3]. Since $\mathbf{A}$ is the Hessian matrix of f at the point, if $\mathbf{A}$ is positive definite, then that critical point is a minimum. Thus, if $\mathbf{A}$ is a symmetric, positive definite matrix, then the minimum of f is the solution to $\mathbf{A}\vec{x} = \vec{b}$ [5].

The method of Steepest Decent is useful for understanding Conjugate Gradient, because they both use a similar approach to minimize Equation 1, and thus solve $\mathbf{A}\vec{x} = \vec{b}$. This shared approach is to take an initial $\vec{x}_0$ and move downwards in the steepest direction, within certain constraints, of the surface defined by Equation 1 [3]. Because the gradient at a point is the direction of maximal increase, $\vec{x}$ should be moved in the opposite direction of the gradient. Thus, to compute the next value of $\vec{x}$, use

$$\vec{x}_{i+1} = \vec{x}_i + \alpha_i\vec{r}_i \tag{2}$$

for some $\alpha_i > 0$ and where $\vec{r}_i = -\nabla f(\vec{x}_i) = \vec{b} - \mathbf{A}\vec{x}_i$ is the residual of $\vec{x}_i$. Since $\mathbf{A}\vec{x} = \vec{b}$ is the only critical point and a minimum of the quadratic function, f, the ideal value of α_i is the one that minimizes $f(\vec{x}_{i+1})$. Thus, choose α_i such that

$$\begin{aligned} 0 &= \tfrac{\mathrm{d}}{\mathrm{d}\alpha_i} f(\vec{x}_{i+1}) \\ &= \tfrac{\mathrm{d}}{\mathrm{d}\alpha_i} f(\vec{x}_i + \alpha\vec{r}_i) \\ \alpha_i &= \frac{\vec{r}_i \cdot \vec{r}_i}{\vec{r}_i \cdot \mathbf{A}\vec{r}_i}. \end{aligned}$$

Note that by using Equation 2, we can derive

$$\vec{r}_{i+1} = \vec{r}_i - \alpha \mathbf{A}\vec{r}_i. \tag{3}$$

Because $\mathbf{A}\vec{r}_i$ is already computed to find α_i, using Equation 3 to compute the residual results in one less matrix-vector product per iteration. The steps for the Method of Steepest Decent are

$$\begin{aligned}
\vec{r}_0 &= \vec{b} - \mathbf{A}\vec{x}_0 \\
\alpha_i &= \frac{\vec{r}_i \cdot \vec{r}_i}{\vec{r}_i \cdot \mathbf{A}\vec{r}_i} \\
\vec{x}_{i+1} &= \vec{x}_i + \alpha_i \vec{r}_i \\
\vec{r}_{i+1} &= \vec{r}_i - \alpha \mathbf{A}\vec{r}_i
\end{aligned}$$

until $\|\vec{r}_i\|$ is less than some tolerance [5].

Example 1. Consider the linear system

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}, \quad \vec{b} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

and use $c = 0$. Note that the solution is

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

When starting at the origin, the iteration of Method of Steepest Decent becomes

$$\begin{array}{lll}
\vec{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \vec{r}_0 = \begin{bmatrix} 5 \\ 5 \end{bmatrix} & \alpha_0 = 2/7 \\
\vec{x}_1 = \begin{bmatrix} 10/7 \\ 10/7 \end{bmatrix} & \vec{r}_1 = \begin{bmatrix} 5/7 \\ -5/7 \end{bmatrix} & \alpha_1 = 2/3 \\
\vec{x}_2 = \begin{bmatrix} 40/21 \\ 20/21 \end{bmatrix} & \vec{r}_2 = \begin{bmatrix} 5/21 \\ 5/21 \end{bmatrix} & \alpha_2 = 2/7 \\
\vec{x}_3 = \begin{bmatrix} 290/147 \\ 50/49 \end{bmatrix} & \vec{r}_3 = \begin{bmatrix} 5/147 \\ -5/147 \end{bmatrix} & \alpha_3 = 2/3 \\
\vdots & \vdots & \vdots
\end{array}$$

The $\vec{x}_i$'s are plotted with a contour graph of the quadratic form in Figure 1. □

The Conjugate Directions family of linear solvers, of which Conjugate Gradient is a member of, attempts to improve on the number of iterations needed by Steepest Decent. [5]. Note that, in Example 1, the directions of $\vec{r}_0$ and $\vec{r}_2$ are the same and the directions of $\vec{r}_1$ and $\vec{r}_3$ are the same. Thus, the same direction has to be traversed multiple times. Additionally, note that the two sets of residual directions are perpendicular to each other. Conjugate Directions attempts

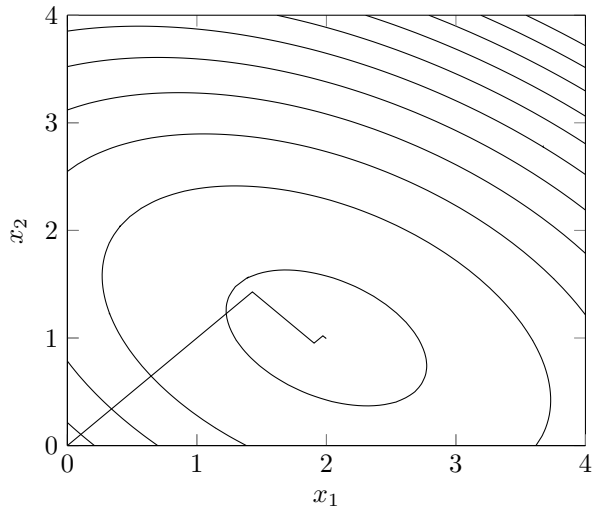

Figure 1: Contour graph of the quadratic function and the first six values of $\vec{x}$ produced by steepest decent for Example 1

to improve on this, by making the search directions, $\vec{d}_0, \vec{d}_1, \ldots$, **A**-orthogonal to each other and only moving $\vec{x}$ once in each search direction. Two vectors, $\vec{u}, \vec{v}$ are **A**-orthogonal, or conjugate, if $\vec{u}^T \mathbf{A} \vec{v} = 0$. The requirement for Conjugate Directions is to make $\vec{e}_{i+1}$ **A**-orthogonal to $\vec{d}_i$, where $\vec{e}_i = \vec{x}_i - \mathbf{A}^{-1}\vec{b}$ is the error of $\vec{x}_i$. The computation of α_i changes to find the minimal value along $\vec{d}_i$ instead of $\vec{r}_i$.

$$\alpha_i = \frac{\vec{d}_i^T \vec{r}_i}{\vec{d}_i^T \mathbf{A} \vec{d}_i}.$$

Conjugate Gradient is a form of Conjugate Directions where the residuals are made to be **A**-orthogonal to each other. This is done using the Conjugate Gram-Schmidt Process. To do this, each search direction, $\vec{d}_i$ is computed by taking $\vec{r}_i$ and removing any components that are not **A**-orthogonal to the previous $\vec{d}$'s. So, let $\vec{d}_0 = \vec{r}_0$ and for $i > 0$ let

$$\vec{d}_i = \vec{r}_i + \sum_{k=0}^{i-1} \beta_{(i,k)} \vec{d}_k$$

with $\beta_{(i,k)}$ defined for $i > k$. Then, solving for $\beta_{(i,k)}$ gives

$$\beta_{(i,k)} = -\frac{\vec{r}_i \cdot \mathbf{A}\vec{d}_i}{\vec{d}_j \cdot \mathbf{A}\vec{d}_j}.$$

Note that each residual is orthogonal to the previous search directions, and thus the previous residuals. So, it can be shown that $\vec{r}_{i+1}$ is **A**-orthogonal to

all previous search directions, except $\vec{d_i}$ [5]. Then, $\beta_{(i,k)} = 0$ for $i - 1 \neq k$. To simplify notation, let $\beta_i = \beta_{(i,i-1)}$. So, each new search direction can then be computed by

$$\vec{d_i} = \vec{r_i} + \beta_i \vec{d}_{i-1}.$$

This results in the following steps for Conjugate Gradient

$$\begin{aligned}
\vec{d_0} = \vec{r_0} &= \vec{b} - \mathbf{A}\vec{x}_0 \\
\alpha_i &= \frac{\vec{r_i} \cdot \vec{r_i}}{\vec{d_i} \cdot \mathbf{A}\vec{d_i}} \\
\vec{x}_{i+1} &= \vec{x_i} + \alpha_i \vec{d_i} \\
\vec{r}_{i+1} &= \vec{r_i} - \alpha \mathbf{A}\vec{d_i} \\
\beta_{i+1} &= \frac{\vec{r}_{i+1} \cdot \vec{r}_{i+1}}{\vec{r_i} \cdot \vec{r_i}} \\
\vec{d}_{i+1} &= \vec{r}_{i+1} + \beta_{i+1} \vec{d_i}
\end{aligned}$$

continuing the iteration until $\|\vec{r_i}\|$ is less than the specified tolerance.

Example 2. Consider the linear system used in Example 1 where

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}, \quad \vec{b} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}.$$

The result of applying Conjugate Gradient is

$$\begin{array}{llllll}
\vec{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \vec{r}_0 = \begin{bmatrix} 5 \\ 5 \end{bmatrix} & & \vec{d_0} = \begin{bmatrix} 5 \\ 5 \end{bmatrix} & \alpha_0 = 2/7 \\
\vec{x}_1 = \begin{bmatrix} 10/7 \\ 10/7 \end{bmatrix} & \vec{r}_1 = \begin{bmatrix} 5/7 \\ -5/7 \end{bmatrix} & \beta_1 = 1/49 & \vec{d}_1 = \begin{bmatrix} 40/49 \\ -30/49 \end{bmatrix} & \alpha_1 = 7/10 \\
\vec{x}_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} & \vec{r}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} & & &
\end{array}$$

Note that after two iterations, $\vec{x}$ reaches the exact solution, compared to the iterations of Steepest Decent in Example 1. Figure 2 shows the values of $\vec{x}$ with the contour graph of the quadratic function. □

One way to improve the Conjugate Gradient method is to precondition the system [4]. Instead of solving the original system, $\mathbf{A}\vec{x} = \vec{b}$, Conjugate Gradient solves $\mathbf{M}^{-1}\left(\mathbf{A}\vec{x} - \vec{b}\right) = 0$ instead, where $\mathbf{M}^{-1}$ is the preconditioner. Note that $\mathbf{M}$ should be similar to $\mathbf{A}$, but $\mathbf{M}^{-1}$ should be easier to compute than $\mathbf{A}^{-1}$. Algorithm 1 shows the preconditioned variant of the Conjugate Gradient.

2.2 Multigrid Preconditioner with Gauss-Seidel Step

Multigrid solvers are a class of methods designed for solving discretized partial differential equations (PDEs) and take advantage of more information that just

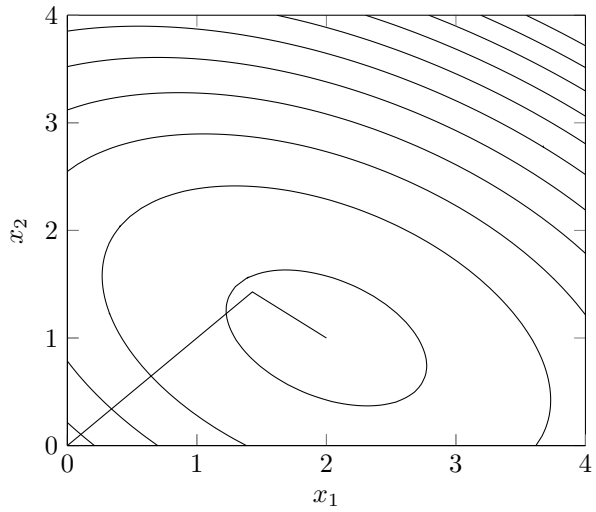

Figure 2: Contour graph of the quadratic function and the each value of $\vec{x}$ produced by Conjugate Gradient for Example 2

the coefficient matrix and the right hand side [4]. In particular, the solvers use discretizations with different mesh sizes to improve performance of relaxation based solvers. In HPCG, a multigrid solver with high tolerance is used as the preconditioner [1]. Because the solver provides an approximation to $\mathbf{A}^{-}1$, the preconditioned matrix is an approximation of $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. This reduces the condition number of the linear system, and so, reduces the number of iterations needed for Conjugate Gradient to converge [4].

The multigrid method works by using meshes of different sizes to allow a relaxation style iterative solver to make the best progress at different levels of meshes [4]. Most relaxation type iterative solvers are able to quickly reduce the components of the residual in the direction of eigenvectors associated with large eigenvalues for the iteration matrix. Such eigenvectors are called high frequency modes. The other components, in the direction of eigenvectors called low frequency modes, are difficult to reduce with standard relaxation. However on a courser mesh, many of these low frequency modes are mapped to high frequency modes [4]. Thus, by applying a relaxation type iterative solver at various mesh sizes, the various components of the residual can be reduced quickly.

In HPCG, a symmetric Gauss-Seidel iteration is used as the relaxation iteration solver at each level of coarseness [1]. The symmetric Gauss-Seidel iteration consists of a forward Gauss-Seidel iteration followed by a backward Gauss-Seidel iteration. Letting $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$ where $\mathbf{L}$ is strictly lower triangular, $\mathbf{D}$ is

Algorithm 1 Preconditioned Conjugate Gradient [4]

$\vec{r}_0 \leftarrow \vec{b} - \mathbf{A}\vec{x}_0$
$\vec{z}_0 \leftarrow \mathbf{M}^{-1}\vec{r}_0$
$\vec{d}_0 \leftarrow \vec{z}_0$
for $i = 0, 1, \ldots$ **until** $\|\vec{r}_i\| \leq \epsilon$ **do**
  $\alpha_i \leftarrow \frac{\vec{r}_i \cdot \vec{z}_i}{\vec{d}_i \cdot \mathbf{A}\vec{d}_i}$
  $\vec{x}_{i+1} \leftarrow \vec{x}_i + \alpha_i \vec{d}_i$
  $\vec{r}_{i+1} \leftarrow \vec{r}_i + \alpha_i \mathbf{A}\vec{d}_i$
  $\vec{z}_{i+1} \leftarrow \mathbf{M}^{-1}\vec{r}_{i+1}$
  $\beta_{i+1} \leftarrow \frac{\vec{r}_{i+1} \cdot \vec{z}_{i+1}}{\vec{r}_i \cdot \vec{z}_i}$
  $\vec{d}_{i+1} \leftarrow \vec{z}_{i+1} + \beta_{i+1}\vec{d}_i$
end for

diagonal and $\mathbf{U}$ is strictly upper case, the iteration can be represented by

$$\vec{x}_i^* = \mathbf{D}^{-1}\left(\vec{b} - \mathbf{L}\vec{x}_i^* - \mathbf{U}\vec{x}_i\right)$$
$$\vec{x}_{i+1} = \mathbf{D}^{-1}\left(\vec{b} - \mathbf{U}\vec{x}_{i+1} - \mathbf{L}\vec{x}_i^*\right).$$

Note that while $\vec{x}_i^*$ and $\vec{x}_{i+1}$ are on both sides of the equation where they are respectively computed, they can be computed with this formulation by computing the entries in order as they become available for the product with L and U respectively.

2.3 Problem Setup of High Performance Conjugate Gradient

The problem used to create the linear system used by HPCG, and thus by this project, is a three dimensional partial differential equation (PDE) model [1]. This problem is approximating the function $u(x, y, z)$ over the three dimensional rectangular region $\Omega \in \mathbb{R}^3$ such that

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = 0,$$

with $u(x, y, z) = 1$ along the boundaries of Ω. Note that the solution is $u(x, y, z) = 1$ over Ω. The linear system is created by using the finite difference method with a 27-point stencil on the PDE over a rectangular grid with nodes of fixed distance. The matrix's diagonal consists of the value 26, and -1's fill the entries for the row's 26 grid neighbors. The right hand side of the equation has a value of 14 for corner points, 12 for edge points, 9 for side points and 0 for interior points [2]. The solution vector consists of all 1's.

HPCG uses an implementation of Conjugate Gradient algorithm with a multigrid preconditioner variant [1]. As HPCG is designed to emulate the performance characteristics of real world problems with out needing to be a robust

solver, it only uses 3 levels of grid coarseness with only a since smoother pass at the coarsest grid level. The smoother used by the multigrid is based on a symmetric Gauss-Seidel step, however each process uses the old value for entries located on other processes. The restriction operation simply samples half the points in dimension, resulting in a reduction of grid size by a factor of eight in each level of coarseness. To prolong the coarse grids, each coarse point is added to the fine point in was sampled from. The zero vector is used as the overall initial guess for x, as well as the initial guess for each grid level in the multigrid cycle.

2.4 Data Access Patterns of High Performance Conjugate Gradient

The way the matrix and vector data is accessed can provide limitations on attempts to compress the data. The Conjugate Gradient and Multigrid implementations in HPCG do not directly access the matrix and vector values, but instead use low level functions to actually manipulate the data structures [1]. These low level functions include copying a vector, setting a vector to zero, the dot product, a scaled vector sum, the matrix vector product, the symmetric Gauss-Seidel step, the multigrid restriction and the multigrid prolongation. Further data accessing functions exist in HPCG, however, they are not part of the timing. So, any additionally restrictions can be overcome by converting to an uncompressed format, applying the function, then recompressing. The low level functions used in the timed section of the code can be viewed together to produce the overall data access requirements, For the matrices, the matrices do not need to be mutable, the rows need to be readable in both a forward and backwards iteration, the data for a given row has no restriction on its read order, and the diagonal for a given row must be accessible. The vectors, on the other hand, need both random read and write access, with the writes being immediately accessible to future reads.

Copying a vector and setting a vector to zero provide the least data access requirements. Note that a vector's content can by copied by transferring the current representation of the values without any processing. Setting a vector to zero merely requires the ability to write vector values. Both of these functions add little to the data access requirements and are both simple to reimplement with alternative vector representation.

The dot product and sum of scaled vectors are both straightforward functions. Each of them iterates over two or three vectors and applies a few arithmetic operations. The dot product accumulates the sum of the product of the pair of vector entries across the iterations. The sum of scaled vectors computes $w_i = \alpha x_i + \beta y_i$ for each set of entries. Note that the only data iteration between rows in either of these operations is the sum in the dot product, however addition is an associative operation. Thus, both of these functions can be arbitrarily parallelized or have their iteration reordered.

The matrix vector product iterates once over the rows and for each row sums the nonzero entries times the vectors corresponding entries [1]. Both the rows

and the sum in each row may be iterated in any order or in parallel. Thus, the matrix information can be compressed for any iteration order of rows and any iteration order for the values in each row. However, the vector information must be able to be read at an arbitrary index. For each iteration, the matrix information is read only once and the vector entries are read for each nonzero value in the corresponding column (8 to 27 times for the matrix described in Section 2.3). So, assuming the problem is too large for the matrix to fit entirely in the memory caches, the matrix data will always need to be read from main memory, while vector data will be able to utilize caches, resulting in up to 27 fold fewer reads than the matrix data. This hints that the compressing matrix information is more likely to provide an increase in performance of the matrix vector product than compressing the vector information.

The symmetric Gauss-Seidel step is similar to the sparse matrix-vector product, with added complications. First, the step has two iterations, one forward and one backward. Instead of simply summing the row-vector product, each row does the following calculation

$$x_i \leftarrow b_i - \frac{1}{a_{ii}} \sum_{j=1}^{n} a_{ij} x_j$$

with the terms containing nonzero matrix entries removed [1]. Note that each x_i is used immediately in the subsequent rows, this means that any deviation from the base row iteration order or any parallelization of the rows may reduce the effectiveness of the step. Because any delay in writing the new values to $\vec{x}$ results in effectively parallelizing the iteration of the rows, the vector values must be written immediately or within a few iterations. Additionally, the Gauss-Seidel step has the additional requirement that the matrix diagonal of the current row must be accessible.

The restriction and prolongation functions used in the multigrid are the last matrix and vector value accessing functions used in the Conjugate Gradient implementation. Restriction samples points from two fine grid vectors and stores the difference in a coarse grid vector. Prolongation takes a the entries in a coarse grid vector and adds them to select fine grid vectors. So, between these two functions, random read and write access is needed by vectors in all but the coarsest mesh.

2.5 Compression Strategies

Numerous compression strategies were considered for this project. Figure 3 lists the compressions tried for each main data structure. Note that most compression methods were only used with one or two of the data types, even if able to be reasonably used within the constraints of additional data.

Strategy	Vector Values	Matrix Values	Matrix Indices
Single Precision	Yes	Yes	Not Able
Mixed Precision	Yes	Not Able	Not Able
1 Bit	Not Able	Yes	Not Able
Squeeze (SZ)	Yes	Yes	Yes
ZFP	Yes	Yes	No
Elias Gamma	Not Able	Not Able	Yes
Elias Delta	Not Able	Not Able	Yes
Huffman	Not Able	No	Yes
Op Code	Not Able	Not Able	Yes

Figure 3: Overview of Compression Strategies

2.5.1 Restrictions on Compression Strategies

2.5.2 Single and Mixed Precision Floating Point Numbers

2.5.3 1 bit Compression

2.5.4 Squeeze (SZ) Compression

2.5.5 ZFP Compression

2.5.6 Elias Gamma Coding and Delta Coding

2.5.7 Op-Code Compression

2.5.8 Huffman Coding

2.5.9 Combined Compression Strategies

3 Test Results

4 Conclusions and Future Work

5 References

[1] Jack Dongarra, Michael Heroux, and Piotr Luszczek. Hpcg benchmark: a new metric for ranking high performance computing systems. Technical Report UT-EECS-15-736, Electrical Engineering and Computer Sciente Department, Knoxville, Tennessee, November 2015.

[2] David R. Kincaid and E. Ward Cheney. *Numerical Analysis: Mathematics of Scientific Computing*. Pure and applied undergraduate texts. American Mathematical Society, 2002.

[3] J. Nearing. *Mathematical Tools for Physics*. Dover books on mathematics. Dover Publications, 2010.

[4] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[5] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.