# Iterative Linear Solvers

*Chris Jiang*

Year 4 Project
School of Mathematics
University of Edinburgh
March 2020

# Contents

# Abstract

Until now, we already have a variety of different iterative methods for solving sparse systems of linear equations. However, these methods have great difference either in the complexity or efficiency. Most article focus on a specific iterative method which is insightful on that topic. So, In this report, I will introduce a series of different iterative methods for a specific Poisson problem starting with the simplest to the complicated ones.

The algorithm and convergence of methods including Jacobi, Gauss-Seidel, Richardson, Conjugate gradient and Multigrid methods with different cycles will be discussed. Also, the computational cost as well as the speed of convergence is included. I have tried not to go deep in a specific topic and cited the fully explanation if reader is interested.

# Chapter 1

# Introduction

The final goal of this project is to use an initial guess to generate a sequence of improving approximate solutions for a class of problems, in which the $n^{th}$ approximation is derived from the previous ones.

In particular, solving a linear system of equations

$$Ax = b \tag{1.0.1}$$

## 1.1 Solve a differential equation

First of all, considering the second ODE below,

$$u''(x) = f(x) \quad \text{for} \quad 0 < x < 1 \tag{1.1.1}$$

with some given boundary conditions

$$u(0) = \alpha, \quad u(1) = \beta \tag{1.1.2}$$

The function $f(x)$ is given and defined as a function respect to $x$. Our goal is to determine $u(x)$ between the interval $0 < x < 1$. Hence, we plan to compute a series of grid function values of $U_0, U_1, ..., U_m, U_{m+1}$ , where $U_j$ is the approximation value of the solution $u(x_j)$. Note that we have $x_j = jh$ here and $h$ represents the mesh width $h = 1/(m + 1)$ which is the distance between each grid points. If we substitute the boundary conditions into the function, we can obtain that $U_0 = \alpha$ and $U_{m+1} = \beta$ so that $m$ unknown values $U_1, ..., U_m$ are remained for us to compute. If we replace $u''(x)$ in equation 1.1.1 by the centered difference approximation

$$D^2U_j = \frac{1}{h^2}(U_{j-1} - 2U_j + U_{j+1}) \tag{1.1.3}$$

then we can generate a set of linear algebraic equations

$$\frac{1}{h^2}(U_{j-1} - 2U_j + U_{j+1}) = f(x_j) \quad \text{for} \quad j = 1, 2, ..., m \tag{1.1.4}$$

Thus it is obvious that we have a linear system of $m$ equations for the $m$ unknowns, which can be written in matirx multiplication form

$$AU = F \tag{1.1.5}$$

given vector $U$ to be the vector of unknowns $U = [U_1, U_2, ..., U_m]^T$ and

$$A = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{bmatrix}, \; F = \begin{bmatrix} f(x_1) - \frac{\alpha}{h^2} \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{m-1}) \\ f(x_m) - \frac{\beta}{h^2} \end{bmatrix}. \tag{1.1.6}$$

This tridiagonal linear system is nonsingular and can be easily solved for $U$ from any right-hand side $F$ .

## 1.2 Why we need different linear solvers?

So far, most of the direct approaches represents A as $n \times n$ full rank matrix which involves $O(n^2)$ calculations in steps of addition or elimination and will quickly cause the linear system run out of memory. Algorithms using direct methods such as Lower-Upper,Cholesky factorization and QR decomposition all need $\mathcal{O}(n^3)$ of time. Note that for this specific system 1.1.6 it is a kind of tri-diagonal matrix, could be solved using Thomas algorithm which is a simplified form of Gaussian elimination, the solution can be obtained in $\mathcal{O}(n)$ operations instead of $\mathcal{O}(n^3)$.
As our goal is to reduce the $\mathcal{O}(n^3)$ run-time of those standard decomposition methods using iterative schemes. By making matrix A sparse and restricting the operations on it thus A can only apply to vectors. This operation can be done in time proportional to the number of nonzero values in a matrix.
Moreover, if an iterative scheme can obtain a fairly accurate solution to the system (1.0.1) in a few iterations, run-time can be reduced to a large extent. So our aim is clear that we need to find a suitable iterative scheme such it takes few iterations and involves less operations in one iteration.

# Chapter 2

# Modified Helmholtz equation

Now considering the modified Helmholtz equation which is

$$u''(x) - \theta\, u(x) = f(x) \quad \text{for} \quad 0 < x < 1 \tag{2.0.1}$$

with the same boundary conditions and a constant $\theta > 0$

$$u(0) = \alpha, \quad u(1) = \beta \tag{2.0.2}$$

Note that the Poisson problem we focus is just a special case of Helmholtz equation with $\theta = 0$ and we only introduce a little about this general case. We know the mesh width is same as solving 1.1.1. Using the same approximation $U_j$ which is the solution to $u(x_j)$, so that we still have $m$ unknown values $U_1, U_2, ..., U_m$ to solve. Using the same centered difference approximation to replace $u''(x)$ we have:

$$D^2 U_j = \frac{1}{h^2}(U_{j-1} - (2 + h^2\theta)U_j + U_{j+1}) \tag{2.0.3}$$

generating the set of equations:

$$\frac{1}{h^2}(U_{j-1} - (2 + h^2\theta)U_j + U_{j+1}) = f(x_j) \quad \text{for} \quad j = 1, 2, ..., m \tag{2.0.4}$$

Given the boundary condition $U_m = 1$ and $U_1 = 0$. Hence we can compute the matrix of linear system of those m equations writing in the form of $AU = F$. where

$$A = \frac{1}{h^2}\begin{bmatrix} -2 - h^2\theta & 1 & & & & & \\ 1 & -2 - h^2\theta & 1 & & & & \\ & 1 & -2 - h^2\theta & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 - h^2\theta & 1 & \\ & & & & 1 & -2 - h^2\theta & \end{bmatrix} \tag{2.0.5}$$

F is the same as the one written in 2.0.6 and $U$ is the vector of unknowns $U = [U_1, U_2, ..., U_m]^T$.

## 2.1 Discrete Eigenvalues

Here we can try to solve the eigenvalues of the Helmholtz equation 2.0.1, which is finding $\lambda$ in the equation below:

$$(\frac{\partial^2}{\partial x^2} - \theta)u = \lambda u \tag{2.1.1}$$

given the boundary condition $u(0) = \alpha, u(1) = \beta$. Recall the **Sturm-Liouville equation**[14] which is a second order linear differential equation that can be written in the form:

$$(p(x)y')' + (q(x) + \lambda r(x))y = 0 \tag{2.1.2}$$

where $p, q$ and $r$ are specific functions, and $\lambda$ is a parameter. Clearly the equation 2.1.1 is in Sturm-Liouville form if we take $p(x) = 1, q(x) = -\theta, r(x) = -1$. Solving this we can get the eigenvalue and eigenfunction value of the equation as we already know that nonzero solutions occur only when:

$$\begin{aligned} \lambda = \lambda_n &= \frac{n^2\pi^2}{\beta - \alpha} - \theta \\ u = u_n &= \frac{sin(n\pi x)}{\beta - \alpha} \end{aligned} \tag{2.1.3}$$

where $n \in N$. Also, we could solve equation 2.1.1 in an alternative way which is using the matrix of linear system 2.0.5 and find the eigenvalues in the equation below:

$$AU = \lambda U \tag{2.1.4}$$

where $U = [U_1, U_2, ..., U_m]^T$. However, as the matrix is an approximation of the second derivative of $u(x)$, there do involves the error, let's named $\epsilon$ here. Thus we can obtain the results:

$$\begin{aligned} U_m &= \sin(n\pi m\Delta x) \\ \lambda &= -n^2\pi^2 - \theta + \epsilon \end{aligned} \tag{2.1.5}$$

Here $\Delta x$ means the step size which is $\frac{\beta-\alpha}{m}$ in this case as we have $m$ steps in the range between two boundary conditions.

# Chapter 3

# Different Iteration Methods

Now in this section, we can introduce a variety of iteration methods from baseline to improved ones to solve the linear system $Ax = b$. Also we need to discuss the limiting behavior according to the number of iterations and time computing the result.

## Why iterative methods?

In general, we tend to choose those algorithms which have a proportional running time to the number of non-zero entries in the matrix A such that no more than the space that used to store A is needed.
Iterative methods are designed to solve linear systems with multiple iterations while each iteration only performs matrix multiplications and a few vector operations. Compared with the direct methods which are based on elimination, typically Gaussian elimination, those iterative algorithms will not generate the exact solutions. Instead, the result we obtained from iterative methods get closer to the solution once we complete one iteration. The advantage of these methods is the efficiency which save a huge amount of time when dealing with giant problems. Also, the storage needed for saving information is significantly small.

## 3.1 Richardson Iteration

Starting with Richardson Iteration which is chosen as the baseline of this project. Richardson iteration is

$$x^{(k+1)} = x^{(k)} + \omega(b - Ax^{(k)}) \tag{3.1.1}$$

where $\omega$ is a scalar parameter that has to be chosen such that the sequence $x^{(k)}$ converges.

## Convergence of Richardson Iteration

Introducing the notation for the error

$$e^{(k)} = x^{(k)} - x \tag{3.1.2}$$

which is given by subtracting the exact solution x. Thus we got the equality for errors

$$e^{(k+1)} = e^{(k)} - \omega A e^{(k)} = (I - \omega A)e^{(k)} \tag{3.1.3}$$

Hence we can generate the equation below

$$\|e^{(k+1)}\| = \|(I - \omega A)e^{(k)}\| \leq \|(I - \omega A)\|\|e^{(k)}\| \tag{3.1.4}$$

for any vector norm and the corresponding induced matrix norm. Thus, we will show that if $\|I - \omega A\| < 1$, the method converges and the convergence rate depends on how much the norm is less than 1. As we are assuming A is symmetric, $I - \omega A$ is symmetric as well, and so its norm is the maximum absolute value of its eigenvalues. Let $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$ be the set of eigenvalues of matrix $A$. Hence we can obtain the set of eigenvalues of $I - \omega A$ which are $1 - \alpha\lambda_i$ for $1 \leq i \leq n$. Hence the norm of $I - \omega A$ is

$$\|I - \omega A\| = \max_i \|1 - \alpha\lambda_i\| = \|\max(1 - \alpha\lambda_1, 1 - \alpha\lambda_n)\| \tag{3.1.5}$$

This is minimized by taking

$$\alpha = \frac{2}{\lambda_1 + \lambda_n}$$

and in this case, the smallest and largest eigenvalues of $I - \omega A$ become

$$\pm\frac{\lambda_n - \lambda_1}{\lambda_1 + \lambda_n}$$

Thus the norm of $I - \omega A$ can be obtained by

$$\|I - \omega A\| = 1 - \frac{2\lambda_1}{\lambda_1 + \lambda_n} \tag{3.1.6}$$

In order to show that $x^{(k+1)}$ converge to the exact solution, let's name $x^*$, considering the error between them which is $x^* - x^{(k+1)}$, we can observe that:

$$\begin{aligned} x^* - x^{(k+1)} &= ((I - \omega A)x^* + \omega b) - ((I - \omega A)x^{(k)} + \omega b) \\ &= (I - \omega A)(x^* - x^(k)) \end{aligned} \tag{3.1.7}$$

repeat this process we could obtain the equation below:

$$x^* - x^{(k+1)} = (I - \omega A)^{k+1} x^* \tag{3.1.8}$$

taking norm to both sides of this equation we can get:

$$\begin{aligned} \|x^* - x^{(k+1)}\| &= \|(I - \omega A)^{k+1} x^*\| \\ &\leq \|(I - \omega A)^{k+1}\| \|x^*\| \\ &= \|(I - \omega A)\|^{k+1} \|x^*\| \\ &\leq (1 - \frac{2\lambda_1}{\lambda_1 + \lambda_n})^{k+1} \|x^*\| \\ &\leq e^{-2\lambda_1 (k+1)/\lambda_1 + \lambda_n} \|x^*\| \end{aligned} \tag{3.1.9}$$

Hence we can make this conclusion: If we want to get a solution $x^{(k)}$ such that

$$\frac{\|x^* - x^{(k)}\|}{\|x^*\|} \leq \epsilon$$

for some $\epsilon \in \mathbb{R}$, it is sufficient to operate

$$\frac{\lambda_1 + \lambda_n}{2\lambda_1} \ln(\frac{1}{\epsilon}) = (\frac{\lambda_n}{2\lambda_1} + \frac{1}{2}) \ln(\frac{1}{\epsilon})$$

iterations. Note that the fraction $\lambda_n / \lambda_1$ is named the **condition number**[16] of matrix $A$ if $A$ is symmetric. It is often written as $\kappa(A)$, and the running time of iterative algorithms is often represented in terms of this quantity. It is easy to see that if the condition number is relatively small, this algorithm will converge to a solution pretty fast.

## 3.2 Jacobi Iteration

The Jacobi method will be introduced as the second iterative method which is modified from the previous baseline Richardson method. Jacobi is an representative iterative algorithm for approximating the solutions of a strictly diagonally dominant system of linear equations. During one iteration, each diagonal element is solved and an approximate value will substitute the original initial guess of the solution. This process will be processed until it converges.
Let $Ax = b$ be a square system with $n$ linear equations where

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}. \tag{3.2.1}$$

Then A can be decomposed into a matrix $D$ of all diagonal component, and the remained components $R$

$$A = D + R \qquad \text{where} \tag{3.2.2}$$

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \text{ and } R = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix} \tag{3.2.3}$$

The solution is then obtained by iterative method

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - R\mathbf{x}^{(k)}) \tag{3.2.4}$$

where $x^{(k)}$ is the $k^{th}$ approximation of $x$ and $x^{(k+1)}$ is the next iteration of $x$. Thus the formula can be conclude by

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n. \tag{3.2.5}$$

## Computational Cost of Jacobi iteration

The sequence $x_{(k+1)}$ where $k \in \mathbb{N}$ from Jacobi iteration is defined by the relation in equation 3.2.4. Hence for each iteration, we need to calculate

1. The product of $Rx^{(k)}$
2. The difference between $b$ and the product $Rx^{(k)}$ above, this step is just $n$ subtractions.
3. The product $D^{-1}$ and the result from last step. As $D$ is diagonal, we only need $n$ divisions for this step.

For dense matrices, the first step needs $\Theta(n^2)$ operations and clearly it dominates the computational cost of other two steps. As for the sparse matrix, the matrix-vector multiplication in the first step can be performed using fewer operations when the matrix contains a lot of zeros. One step of Jacobi iteration using sparse matrix cost $\Theta(n)$ as $n \to \infty$ if each row of matrix $A$ contains at most $l > 0$ non-zero entries outside the diagonal.

## Numerical Test on Jacobi method

Here we implement the Jacobi iteration in python using sparse matrix in order to testing the number of iterations needed to converge against the dimension of matrix. The code is placed in appendix and the relation is shown below as figure 3.1. The given example tested the Jacobi iteration with a range of dimensions of matrix $A$ in the form of 1.1.6 from 10 to 300 with step size 10. The matrix is saved in sparse matrix form and the number of iterations needed is calculated for each size. Figure 3.1 shows that we need much more iterations to converge when the size of matrix grows larger. And figure 3.2 is the log-log plot for figure 3.1, this shows that the relation between dimensions and number of iterations fits a natural log relation. Thus the increase of iterations needed for Jacobi is exponential not linear. This is not what we expect because this may take too long for a large matrix. The implementation of Jacobi method in python can be seen in Appendix.

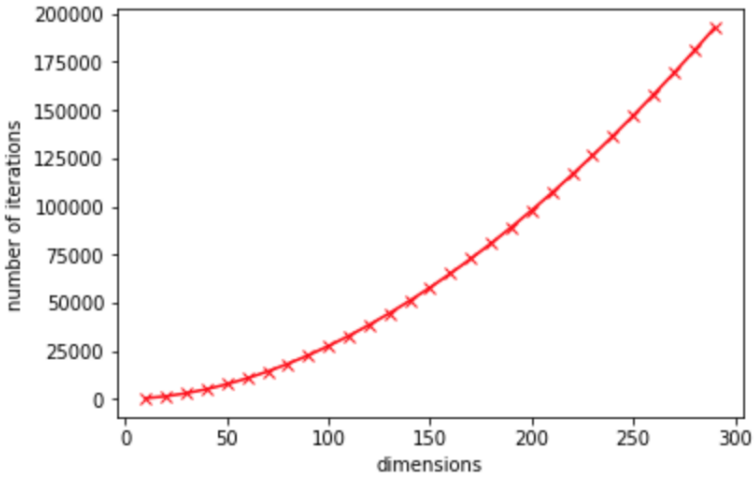



Figure 3.1: Figures illustrate the relation between number of iteration and size of matrix. It is clear to see that the iterations needed for Jacobi method to converge is not linear but require much more calculations when the size of matrix increases.

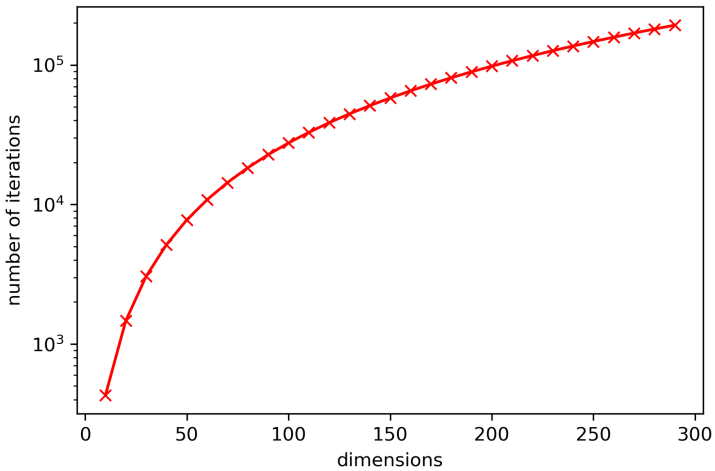



Figure 3.2: Log-log plot for figure 3.1, fits a natural log relation which shows that the original plot have a exponential relation between dimensions of matrix and number of iterations.

## 3.3 Gauss-Seidel method

The Gauss–Seidel method is an iterative technique for solving a square system of $n$ linear equations with unknown $x$ which is similar to the Jacobi method above defined by the iteration below with same $Ax = b$ as 3.2.1

$$L\mathbf{x}^{(k+1)} = \mathbf{b} - U\mathbf{x}^{(k)} \tag{3.3.1}$$

The decomposition of A into its lower triangula component and its strictly upper triangular component is given by:

$$A = L + U \text{ where } L = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \ U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{3.3.2}$$

we can rewrite the linear equations system as:

$$L\mathbf{x} = \mathbf{b} - U\mathbf{x} \implies \mathbf{x}^{(k+1)} = L^{-1}(\mathbf{b} - U\mathbf{x}^{(k)}) \tag{3.3.3}$$

the elements of $x^{(k+1)}$ can be computed sequentially using forward substitution

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n. \tag{3.3.4}$$

### Computational Cost of Gauss-Seidel method

As we have shown above, Gauss-Seidel method result in an update of forward substitution[2] which also need $\Theta(n^2)$ operations. However, Gauss-Seidel improves over Jacobi but not by an order of magnitude. The test for Gauss-Seidel shows a similar result as Jacobi method hence I only focus on Jacobi method.

## 3.4 Convergence Analysis of Jacobi and Gauss-Seidel methods

Considering the one-dimensional case of the Poisson problem, equation 1.1.1. Both Jacobi and Gauss-Seidel can be analyzed by splitting A into $A = L + D + U$ where L,D and U are all $m \times m$ matrices. Then the system $Ax = b$ can be rewritten by substitute the decomposition of $A$ which gives:

$$Lx + Dx + Nx = b \implies Dx = b - Lx - Ux \tag{3.4.1}$$

where

$$D = \frac{1}{h^2}\begin{bmatrix} -2 & 0 & & & & \\ 0 & -2 & 0 & & & \\ & 0 & -2 & 0 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 0 & -2 & 0 \\ & & & & 0 & -2 \end{bmatrix} = -\frac{2}{h^2} I \tag{3.4.2}$$

is the diagonal of matrix A and

$$L = \frac{1}{h^2}\begin{bmatrix} 0 & 0 & & & & \\ 1 & 0 & 0 & & & \\ & 1 & 0 & 0 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 0 & 0 \\ & & & & 1 & 0 \end{bmatrix},\ U = \frac{1}{h^2}\begin{bmatrix} 0 & 1 & & & & \\ 0 & 0 & 1 & & & \\ & 0 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 0 & 0 & 1 \\ & & & & 0 & 0 \end{bmatrix} \tag{3.4.3}$$

is the lower and upper triangular part of matrix A respectively which suggests the iterative method:

$$Dx^{(k+1)} = b - (L+U)x^{(k)} \tag{3.4.4}$$

If we assume that the $x^{(k)}$ is known in each iteration and solving the linear system with matrix D to get $x^{(k+1)}$.
The system can be written as

$$x^{(k+1)} = \frac{h^2}{2}(L+U)x^{(k)} - \frac{h^2}{2}b \tag{3.4.5}$$

where

$$\frac{h^2}{2}(L+U) = \frac{1}{2}\begin{bmatrix} 0 & 1 & & & & \\ 1 & 0 & 1 & & & \\ & 1 & 0 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & 0 & 1 \\ & & & & 1 & 0 \end{bmatrix} \tag{3.4.6}$$

To analysis both Jacobi and Gauss-Seidel methods, we can write the iteration formula in the same way derived from 4.5.4 thus

$$x^{(k+1)} = M\,x^{(k)} + c \tag{3.4.7}$$

where M is the iteration matrix and c is constant values. Now let $\hat{x}$ be the actual value of solution such that

$$\hat{x} = M\,\hat{x} + c \tag{3.4.8}$$

which means the real solution fix the equation 4.5.7 such that when $x^{(k)} = \hat{x}$ then $x^{(k+1)} = \hat{x}$.
Introducing the error $e^{(k)} = x^{(k)} - \hat{x}$, thus when we subtract 4.5.7 by 4.5.8 we

have

$$e^{(k+1)} = M\, e^{(k)} \tag{3.4.9}$$

and then we can derive that

$$e^{(k)} = M^k\, e^0 \tag{3.4.10}$$

Hence we can conclude that this method converge from any initial guess $x^0$ given $M^k \to 0$ when $k \to \infty$. So, what conditions do we need to satisfy this converge method?
Assume M is diagonalizable, using the definition of diagonalizable matrix which can be written as

$$M = PDP^{-1} \tag{3.4.11}$$

where D is a diagonal $n \times n$ matrix with the eigenvalues $\lambda_1, \lambda_2, ..., \lambda_m$ of A as its entries and P is a non-singular $n \times n$ matrix consisting of the eigenvectors corresponding to the eigenvalues in D. Hence

$$M^k = PD^kP^{-1} \tag{3.4.12}$$

where

$$D^k = \begin{bmatrix} \lambda_1^k & & & \\ & \lambda_2^k & & \\ & & \ddots & \\ & & & \lambda_m^k \end{bmatrix} \tag{3.4.13}$$

Hence this method converge for all $|\lambda_i| < 1$ where $i = 1, 2, ..., m$ which is the same meaning as the spectral radius of matrix M is smaller than 1, i.e. $\rho(M) < 1$.

## 3.5 Rate of Convergence

Now we can think about the question how fast these two methods (Jacobi and Gauss-Seidel) are expected to converge given the condition that it is convergent. This problem can be solved from equation 3.4.10 which gives a relation between the initial guess and final result. Combining with the discretization of iteration matrix $M^k$ from equation 3.4.12, if we take the 2-norm of both sides we can obtain the following relation:

$$\begin{aligned} ||e^{(k)}||_2 &= ||M^k\, e^0||_2 \\ &= ||PD^kP^{-1}\, e^0||_2 \\ &\leq ||P||_2||D^k||_2||P^{-1}||_2\, ||e^0||_2 \\ &= \rho(M)^k \kappa_2(P)||e^0||_2 \end{aligned} \tag{3.5.1}$$

where we know that the condition number[16] of the eigen-vector matrix $P$ is $\kappa_2(P) = ||P||_2||P^{-1}||_2$.
Apply the definition from LeVeque[8] appendix C.6 which states that if matrix A commutes with its adjoint, $AA^H = A^HA$, then A is said to be a normal matrix. Note that any normal matrix is diagonalizable and $P$ can be chosen to be unitary. Hence, if the matrix $M$ is a normal matrix, then the eigenvectors are orthogonal

and condition number is 1. Simplify the result from 3.5.1 we can get:

$$||e^{(k)}||_2 \leq \rho(M)^k ||e^0||_2 \tag{3.5.2}$$

where the rate of convergence is clearly indicated by the spectral radius factor $\rho(M)$ for each iteration.
In case of $M$ is not normal, the asymptotic rate of convergence can be represented by the spectral radius $\rho(M)$ as $k \to \infty$. However this may a bad indication of the behavior of the error if we choose a small $k$.

# Chapter 4

# Conjugate Gradient Method

Conjugate gradient is the most popular iterative method[15] for solving large linear equations system. CG is effective for systems of the form which we are discussing

$$Ax = b \tag{4.0.1}$$

where $x$ is an unknown vector, $b$ is a known vector, and $A$ is a known symmetric positive-definite matrix.
This system always converges to the exact solution in a finite number of $n$ iterations and we could obtain a sufficient accurate solution in far more less than $n$ iterations. That effective convergence is crucial to the popularity of CG, since the operation count of Jacobi or Gauss-Seidel is far too large for most sparse problems and we wish to use an iterative method that takes less iterations and much quicker.

We can see the pseudo-code of basic CG algorithm here which follows LeVaque, and we will discuss this in the later section:

Choose initial guess $u_0$ (possibly the zero vector)
$r_0 = f - Au_0$
$d_0 = r_0$
for $k = 1, 2, ...$
  $w_{k-1} = Ad_{k-1}$
  $\alpha_{k-1} = (r_{k-1}^T r_{k-1})/(d_{k-1}^T \omega_{k-1})$ [8]
  $u_k = u_{k-1} + \alpha_{k-1} d_{k-1}$
  $r_k = r_{k-1} - \alpha_{k-1}\omega_{k-1}$
  if $||r_k||$ is less than the initialised tolerance then break
  $\beta_{k-1} = (r_k^T r_k)/(r_{k-1}^T r_{k-1})$
  $d_k = r_k + \beta_{k-1} d_{k-1}$
  end

## 4.1 Steepest descent method

First, we can motivate the CG method by considering the steepest descent method in solving a minimization problem. Using the same choice of equation from

LeVeque[8] section 4.3, Consider the function $\phi : \mathbb{R}^m \to \mathbb{R}$ defined by

$$\phi(u) = \frac{1}{2}u^T Au - u^T f \tag{4.1.1}$$

which is a quadratic function with variables $u_1, ..., u_n$. If we take partial derivative of function $\phi$ with respect to each u. By defining the derivative equal to 0 in order to reach the minimum, so we can obtain a set of equations:

$$\frac{\partial \phi}{\partial u_1} = \sum_{i=1}^{n} a_{1i}u_i - f_1$$
$$\frac{\partial \phi}{\partial u_2} = \sum_{i=2}^{n} a_{2i}u_i - f_2$$
$$...$$

This is exactly the linear system $Au = f$ we are focusing on. Thus if we wish to find $u^*$ solving the linear system it can be approached by finding $u^*$ which minimize the function $\phi(u)$. We can notice that it is finding

$$\nabla\phi = 0 = Au - f \tag{4.1.2}$$

and hence $\phi(u)$ has a unique minimum at $u^*$ such that $\nabla\phi(u^*) = 0$. This is generally true given the condition that $A \in \mathbb{R}^m \times \mathbb{R}^m$ is symmetric positive definite.
Same as all other iterative methods, we could start with an arbitrary initial guess $u^0$ and take a series of steps $u^1, u^2...$ until we reach a value $u^k$ which is close enough to the actual solution $u^*$ by moving downhill based on the values of $\phi(u)$. So when we choosing the direction of the next step, we take the one in which function $\phi(u^i)$ decrease most rapidly, same as finding the direction

$$-\nabla\phi(u^i) = f - Au^i \tag{4.1.3}$$

according to equation 5.1.2. Thus we can obtain the error and residual shows how far we are from the solution and $f$:

$$e^i = u^i - u^*$$
$$r^i = f - Au^i$$

and finally we can reach that

$$r^i = -Ae^i \tag{4.1.4}$$

Hence the goal becomes finding how big the step we need to take along the direction of steepest descent, which is finding the optimal $\alpha$ in the equation below:

$$u^{i+1} = u^i + \alpha r^i \tag{4.1.5}$$

minimizing $\phi$ along a line. And clearly, starting with arbitrary $u^0$, the first step we take along the direction of steepest descent gives $u^1$ which is

$$u^1 = u^0 + \alpha r^0 \tag{4.1.6}$$

We could find the $\alpha$ by setting the directional derivative to zero which is

$$\frac{d}{d\alpha} f(u^1) = -(r^1)^T \frac{d}{d\alpha} u^1 = -(r^1)^T r^0 = 0 \tag{4.1.7}$$

by the following steps, we could obtain the step length parameter $\alpha$:

$$\begin{aligned}
(r^1)^T r^0 &= 0 \\
(f - Au^1)^T r^0 &= 0 \\
(f - A(u^0 + \alpha r^0))^T r^0 &= 0 \\
(f - Au^0)^T r^0 - \alpha (Ar^0)^T r^0 &= 0 \\
(f - Au^0)^T r^0 &= \alpha (Ar^0)^T r^0 \\
(r^0)^T r^0 &= \alpha (Ar^0)(r^0)^T \\
\alpha &= \frac{(r^0)^T r^0}{(r^0)^T A r^0}
\end{aligned} \tag{4.1.8}$$

Hence if we summarise we can generate the method of Steepest Descent which is

$$\begin{aligned}
r^i &= f - Au^i \\
\alpha^i &= \frac{(r^i)^T r^i}{(r^i)^T A r^i} \\
u^{i+1} &= u^i + \alpha^i r^i
\end{aligned} \tag{4.1.9}$$

and the following algorithm can be concluded:

**Algorithm SD (steepest descent).**
input: $A \in \mathbb{C}^{n \times n}$ Hermitian, positive definite, $b \in \mathbb{C}^n$, $x_0 \in \mathbb{C}^n$, $\varepsilon_r > 0$
output: $x_k \in \mathbb{C}^n$ with $Ax_k \approx b$

1: **for** $k = 1, 2, \ldots$ **do**
2: $\quad r_{k-1} = b - Ax_{k-1}$
3: $\quad$ **if** $\|r_{k-1}\| \leq \varepsilon_r$ **then**
4: $\quad\quad$ return $x_{k-1}$
5: $\quad$ **end if**
6: $\quad \alpha_{k-1} = \|r_{k-1}\|^2 / \|r_{k-1}\|_A^2$
7: $\quad x_k = x_{k-1} + \alpha_{k-1} r_{k-1}$
8: **end for**

modified from **Steepest Descent**[10] by Juan C. Meza.

## Convergence Analysis of Steepest Descent

The steepest descent method have a nice convergence theory[12]. It is fairly simple to show the linear rate of convergence of steepest descent method. However, this linear convergence rate is still too slow for large scale problem hence it is not

suitable for any practical application.
It is easiest to consider the quantity $\phi(u^k) - \phi(u^*)$ for the analysis of covergence where where $u^*$ denotes the global minimizer of equation 4.1.1. Note the the unique minimizer $u^*$ is given by the solution of linear system $Au^* = f$. We can do the following operation:

$$\begin{aligned}\phi(u^k) - \phi(u^*) &= \frac{1}{2}((u^k)^T A u^k - f^T u^k) - \frac{1}{2}((u^*)^T A u^* - f^T u^*) \\ &= \frac{1}{2}((u^k)^T A u^k - (Au^*)^T u^k) - \frac{1}{2}((u^*)^T A u^* - (Au^*)^T u^*) \\ &= \frac{1}{2}(u^k - u^*)^T A (u^k - u^*)\end{aligned} \tag{4.1.10}$$

To compute a bound, we could use the lemma due to Kantorivich, which can be found in Luenberger[9] which gives the following result

$$\phi(u^{k+1}) - \phi(u^*) \leq [\frac{\kappa(A) - 1}{\kappa(A) + 1}]^2 \phi(u^k) - \phi(u^*) \tag{4.1.11}$$

where $\kappa(A)$ is the condition number of matrix A and $\kappa(A) = \lambda_n/\lambda_1$. A similar bound can be derived for the case of a general nonlinear objective function, if we assume that $\alpha^k$ is the global minimizer along the search direction.

## 4.2 Conjugate directions

Steepest Descent often finds itself taking steps in the same direction as earlier step. So this comes the idea that if we pick a set of orthogonal search directions $d_0, d_1, .., d_{n-1}$. In each search direction, we'll take exactly one step and that step will be just the right length to line up evenly with $x$, finish the search after $n$ steps. In general, for each step we choose a point

$$u^{i+1} = u^i + \alpha^i d_i \tag{4.2.1}$$

To find the value of $\alpha^i$ use the fact that $e^{i+1}$ should be orthogonal to $d_i$, so that we will never step in the direction of $d_i$ again. The figure 4.1 illustrates this idea, using the coordinate axes as search directions. As we know $e^1$ is orthogonal to $d_0$.The first horizontal step leads to the correct $x^1$ coordinate and the second vertical step will hit $x$. By apply this condition, we can modify equation 4.2.1 by:

$$\begin{aligned}(d_i)^T e^i &= 0 \\ (d_i)^T (e^i + \alpha^i d_i) &= 0 \\ \alpha^i &= -\frac{(d_i)^T e^i}{(d_i)^T d_i}\end{aligned} \tag{4.2.2}$$

However, we can not solve $\alpha^i$ without knowing the value of $e^i$. Hence, an alternative way to reach the solution is to make the search directions A-orthogonal. Let's claim the A-orthogonal first, if there exists two vectors $v_1, v_2$ A-orthogonal

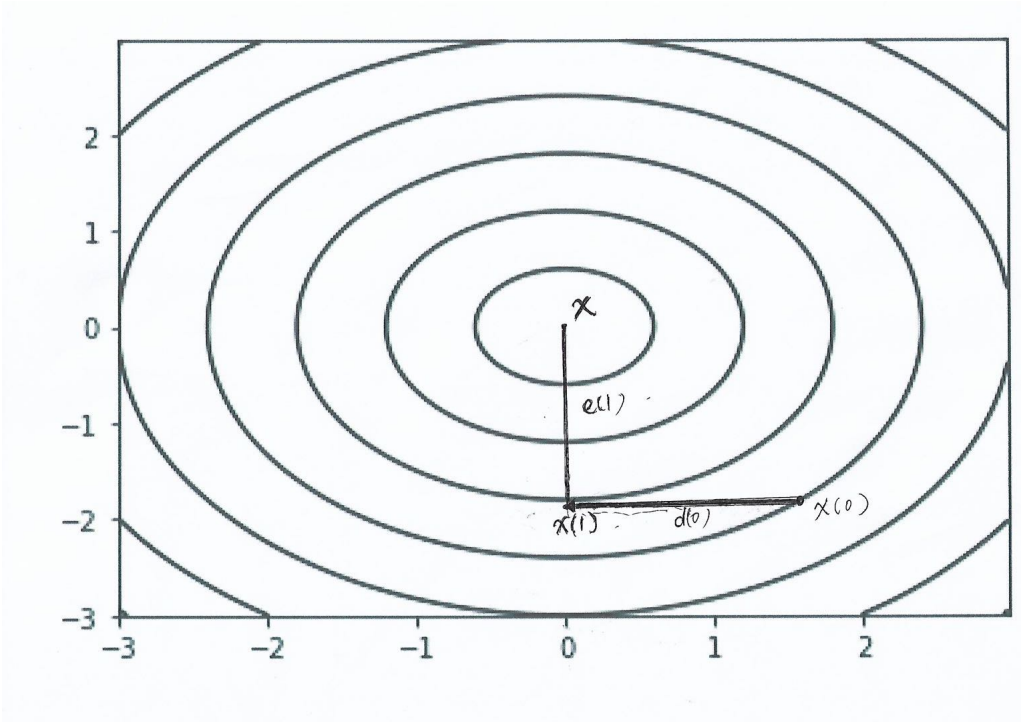



Figure 4.1: The Method of conjugate Directions. Unfortunately, we can not solve this question unless we know the answer because we can't compute $\alpha^i$ without knowing $e^i$ and if we knew $e^i$, the problem would already be solved.

if they have the relation below

$$v_1 A v_2 = 0$$

If $A = I$ , this just means the vectors are orthogonal, and A-orthogonal is a natural generalization of the notion of orthogonality. So in this case, we need $e^{i+1}$ to be A-orthogonal with $d_i$ and this orthogonality condition is exactly the same as finding the minimum point along the search direction $d_i$ as in the steepest decent. We can see this conclusion by setting the directional derivative to zero:

$$\begin{aligned} \frac{d}{d\alpha} f(u^{i+1}) &= 0 \\ f'(u^{i+1})^T \frac{d}{d\alpha} u^{i+1} &= 0 \\ -(r^{i+1})^T d_i &= 0 \\ (d_i)^T A e^{i+1} &= 0 \end{aligned} \tag{4.2.3}$$

Following the derivation of equation 4.2.2, here is the expression for $\alpha^i$ when the search directions are A-orthogonal:

$$\begin{aligned} \alpha^i &= -\frac{(d_i)^T A e^i}{(d_i)^T A d_i} \\ &= \frac{(d_i)^T r^i}{(d_i)^T A d_i} \end{aligned} \tag{4.2.4}$$

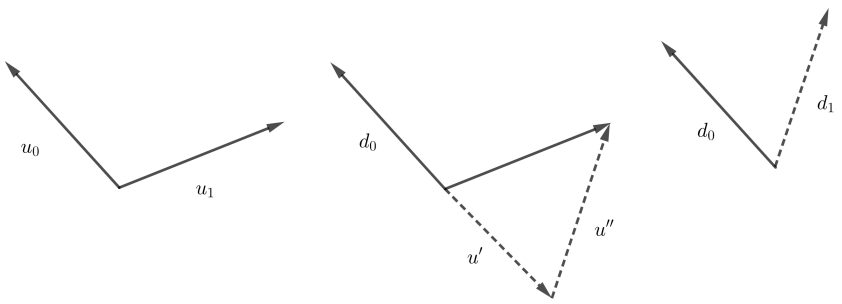



Figure 4.2: Gram-Schmidt conjugation of two linear independent vectors $u_0, u_1$, setting $d_0 = u_0$, let $u', i''$ be the A-orthogonal components of $d_0$ with $u'$ parallel. After conjugation we only leave the $u''$ component and let this vector to be $d_1$.

and this time the expression for $\alpha^i$ is able to calculate.

## 4.3 Gram-Schmidt Conjugation

So all we needed now is a set of A-orthogonal search direction and we could generate them in a simple way which is named **conjugate Gram-Schmidt process**. The Gram–Schmidt process takes a finite, linearly independent set $S = u_1, ..., u_k$ for $k \leq n$ and generates an orthogonal set $S' = d_1, ..., d_k$ that spans the same k-dimensional subspace of $\mathbb{R}^n$ as S. To construct $d_i$ , take $u_i$ and subtract out any components that are not A-orthogonal to the previous $d$ vectors. In other words, set $d_0 = u_0$, and for $i > 0$ , set

$$d_i = u_i + \sum_{k=0}^{i-1} \beta_{ik} d_k \tag{4.3.1}$$

where $\beta_{ik}$ are defined for $i > k$. Following the process below we could obtain the expression for $\beta$:

$$(d_i)^T A d_j = (u_i)^T A d_j + \sum_{k=0}^{i-1} \beta_{ik} d_k A d_j \tag{4.3.2}$$

and by the orthogonality of $d$ vectors we can get:

$$\begin{aligned} 0 &= (u_i)^T A d_j + \beta_{ij} (d_j)^T A d_j \quad \text{for} \quad i > j \\ \beta_{ij} &= -\frac{(u_i)^T A d^i}{(d_j)^T A d_j} \end{aligned} \tag{4.3.3}$$

we can also illustrate this by a example in figure 4.2 below, begin with two linearly independent vectors $u_0, u_1$, set $d_0 = u_0$.

The vector $u_1$ is composed of two component $u', u''$ where $u'$ is parallel to $d_0$ and $u''$ is A-orthogonal to $d_0$. . After conjugation, only the A-orthogonal portion remains which is $u''$ and $d_1 = u''$.

However, even we could successfully get a set of orthogonal searching directions through Gram-Schmidt process, there is still some difficulty in the implementation. If we use Gram-Schmidt conjugation in the method of Conjugate Direction, we need to save all the old search vector in memory to avoid duplication and construct the new one. Hence a operational cost of $\mathcal{O}(n^3)$ is needed to generate the full searching direction set. This method is useful until we discover the CG algorithm which avoid this difficulty of memorization.

## 4.4 Conjugate Gradient Method

Now we could step into the CG method. In fact, CG is simply the method of Conjugate Directions where the search directions are constructed by conjugation of the residuals, by setting $u_i = r_i$.
First, as we already know that the residual works well for steepest decent so this will also work for conjugate gradients. Also, residual has an important property that it is orthogonal to the previous search direction so that a new, linearly independent search direction will be produced unless the residual is zero which in this case the question has already been solved.
Now we could start with the subspace span $\{r_0, r_1, ..., r_{k-1}\}$ as all searching directions are built from residuals and each of them is orthogonal to all previous searching directions and residuals.
Thus we can generate the following properties of CG algorithm given $r_k \neq 0$:

1. $d_k$ is A-conjugate to all previous searching directions which means for $j = 0, 1, ..., k-1$, $p_k^T A p_j = 0$.

2. The residual $r_k$ is orthogonal to all previous residuals which means for $j = 0, 1, ..., k-1$, $r_k^T r_j = 0$.

3. The three subspaces of $\mathbb{R}^m$ is identical:

$$\begin{aligned} &\text{span}\{r_0, r_1, ..., r_{k-1}\} \\ &\text{span}\{r_0, Ar_0, A^2 r_0, ..., A^{k-1} r_0\} \\ &\text{span}\{d_0, Ad_0, A^2 d_0, ..., A^{k-1} d_0\} \end{aligned} \tag{4.4.1}$$

This subspace is called a Krylov subspace, a subspace created by repeatedly applying a matrix to a vector.
The iterate $u_k$ is formed by adding multiples of the search directions $d_j$ to the initial guess $u_0$ and hence must lie in the affine spaces $u_0 + \mathcal{K}_k$.
Thus we can see that CG algorithm can be interpreted as minimizing the function $\phi(u)$ over the space $u_0 + \mathcal{K}_k$ in the $k^{th}$ iteration.
Now we could come back to the pseudo-code of CG algorithm at the beginning of this chapter which generally follows what we discussed above, including one matrix multiplication in each iteration to compute $\omega_{k-1}$. Rewriting equation of $\beta_{ij}$ in 4.3.3, given the property that

$$d_{k-1}^T r_{k-1} = r_{k-1}^T r_{k-1}$$

combining with equation 4.2.4, it could be simplified as follow:

$$\begin{aligned}\beta_i &= \frac{r_i^T r_i}{d_{i-1}^T r_{i-1}} \\ &= \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}\end{aligned} \tag{4.4.2}$$

and these are all we need to finish up the algorithm.

$$d_0 = r_0 = f - Au_0 \tag{4.4.3}$$

$$\alpha_i = \frac{(d_i)^T r^i}{(d_i)^T A d_i} \quad \text{from equation 4.2.4} \tag{4.4.4}$$

$$u_{i+1} = u_i + \alpha_i d_i \tag{4.4.5}$$

$$r_{i+1} = r_i - \alpha_i A d_i \tag{4.4.6}$$

$$\beta_{i+1} = \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i} \tag{4.4.7}$$

$$d_{i+1} = r_{i+1} + \beta_{i+1} d_i \tag{4.4.8}$$

Put all these together and we could form the algorithm of conjugate gradients. The importance of CG is because both the space complexity and time complexity per iteration are reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(m)$, where $m$ is the number of nonzero entries of matrix $A$.

# Convergence of Conjugate Gradient

As we already know that CG is complete after $n$ iterations, actually we don't need to care about the convergence. The detailed proof could be find in LeVeque section 4.3.4. The general idea for the convergence theory is related to the fact that $u_k$ minimizes $\phi(u)$ over the affine space $u_0 + \mathcal{K}_\kappa$ defined in the previous section.
Since A is Symmetric, positive-definite, matrix $A$ can be decomposed to $Q\lambda Q^T$ where $\Lambda$ is the diagonal matrix with set of eigenvalues $\lambda_1, ..., \lambda_n$ on its entries.
So we can define the new parameters

$$\begin{aligned}y &= Q^T u \\ \bar{f} &= Q^T f \\ y^* &= Q^T u^*\end{aligned}$$

thus we can rewrite $\phi(u)$ in terms of $y$:

$$\begin{aligned}\phi(u) = \bar{\phi}(y) &= \frac{1}{2}u^T Q\Lambda Q^T u - f^T Q Q^T u \\ &= \frac{1}{2}y^T \Lambda y - \bar{f}^T y \\ &= \sum_{i=1}^{n}(\frac{1}{2}\lambda_i y_i^2 - \bar{f}_i y_i)\end{aligned} \tag{4.4.9}$$

Hence we can see that

$$\begin{aligned}y_i^* &= \frac{\bar{f}_i}{\lambda_i} \\ \phi^* &= -\frac{1}{2}\sum_{i=1}^{n}\frac{\bar{f}_i^{\,2}}{\lambda_i}\end{aligned} \tag{4.4.10}$$

recall the Krylov subspace we defined before which is

$$\begin{aligned}\mathcal{K}_\kappa &= \text{span}\{f, Af, ..., A^{k-1}f\} \\ &= \{p(A)f \mid \text{polynomial } p, \deg p < k\}\end{aligned} \tag{4.4.11}$$

we could define the Krylov sequence $u^1, u^2, ...$ as

$$u^k = \text{argmin}_{u\in\mathcal{K}_\kappa} f(u) = \text{argmin}_{u\in\mathcal{K}_\kappa}||u - u^*||_A^2 \tag{4.4.12}$$

and the conjugate gradient methods actually generates the Krylov sequence. We could rewrite the Krylov sequence in terms of $y$ which gives

$$\begin{aligned}y^k &= \text{argmin}_{y\in\bar{\mathcal{K}}_\kappa}\bar{f}(y) \\ \bar{\mathcal{K}}_\kappa &= \text{span}\{\bar{f}, \Lambda\bar{f}, ..., \Lambda^{k-1}\bar{f}\} \\ y_i^k &= p_k(\lambda_i)\bar{f}_i \quad \text{where} \quad \deg p_k < k \\ \text{given} \quad p_k &= \text{argmin}_{\deg p<k}\sum_{i=1}^{n}\bar{f}_i^2(\frac{1}{2}\lambda_i p(\lambda_i)^2 - p(\lambda_i))\end{aligned} \tag{4.4.13}$$

To make this easy, considering the term $\phi(u^k) - \phi^*$, replace $u$ with y and

substitute the result we obtained above:

$$\begin{aligned}
\phi(u^k) - \phi^* &= \bar{\phi}(y^k) - \phi^* \\
&= \min_{\deg p<k} \frac{1}{2} \sum_{i=1}^{n} \bar{f}_i^2 \frac{(\lambda_i p(\lambda_i) - 1)^2}{\lambda_i} \\
&= \min_{\deg p<k} \frac{1}{2} \sum_{i=1}^{n} \bar{y}_i^{*2} \lambda_i (\lambda_i p(\lambda_i) - 1)^2 \\
&= \min_{\deg q\leq k, q(0)=1} \frac{1}{2} \sum_{i=1}^{n} \bar{y}_i^{*2} \lambda_i q(\lambda_i)^2 \\
&= \min_{\deg q\leq k, q(0)=1} \frac{1}{2} \sum_{i=1}^{n} \bar{f}_i^2 \frac{q(\lambda_i)^2}{\lambda_i}
\end{aligned} \tag{4.4.14}$$

and finally we could obtain the rate of convergence:

$$\begin{aligned}
\tau_k &= \frac{\min_{\deg q\leq k, q(0)=1} \sum_{i=1}^{n} \bar{y}_i^{*2} \lambda_i q(\lambda_i)^2}{\sum_{i=1}^{n} \bar{y}_i^{*2} \lambda_i} \\
&\leq \min_{\deg q\leq k, q(0)=1} (\max_{i=1,\ldots,n} q(\lambda_i)^2)
\end{aligned} \tag{4.4.15}$$

with the following property that if there is a polynomial q of degree k, with $q(0) = 1$, that is small on the spectrum[6] of A, then $\phi(u^k) - \phi^*$ is small. Also, if eigenvalues are clustered in $k$ groups, then $y^k$ is a good approximate solution. Note that if solution $u^*$ is approximately a linear combination of $k$ eigenvectors of A, then $y^k$ is a good approximate solution.
A bound on convergence rate can be generated by taking the polynomial $q$ as Chebyshev polynomial[1] of degree $k$, that is small on interval $[\lambda_{\min}, \lambda_{\max}]$, we can get

$$\tau_k \leq (\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1})^k, \quad \text{where} \quad \kappa = \frac{\lambda_{\max}}{\lambda_{\min}} \tag{4.4.16}$$

the convergence of CG method can be much faster than this, if spectrum[6] of $A$ is spread but clustered.
This is an abbreviation of analysis of the CG convergence, the more specific proof and discuss can be found in LeVeque[8] chapter 4.3.4.

## 4.5 Numerical test for Conjugate Gradient

In order to see the effectiveness of Conjugate Gradient, we could test the performance by plotting the number of iterations needed for CG to converge while the dimension of matrix is increasing. In this specific test, the set matrices are chosen to be in range 0 to 300 with step size 10 and the number of iterations needed are calculated for each dimensions. It is clearly that CG is a way more efficient and fast iterative method which takes much less iterations compared with Jacobi method. The implementation of conjugate gradient can be seen in appendix A.

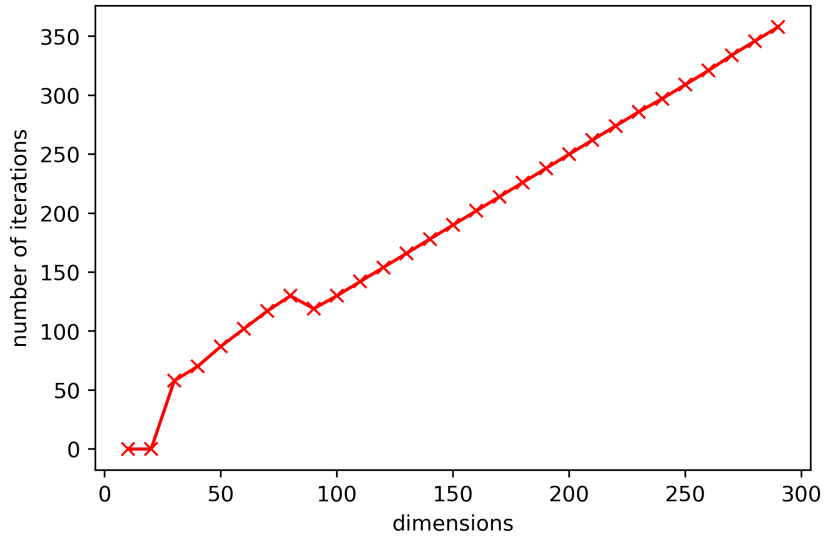



Figure 4.3: Numerical test between of conjugate gradient which shows that the relation between number of iterations needed for CG to converge against dimension of matrix is linear. We can observe that the number of iterations needed for CG is always about the level of dimensions which gives a linear relation.

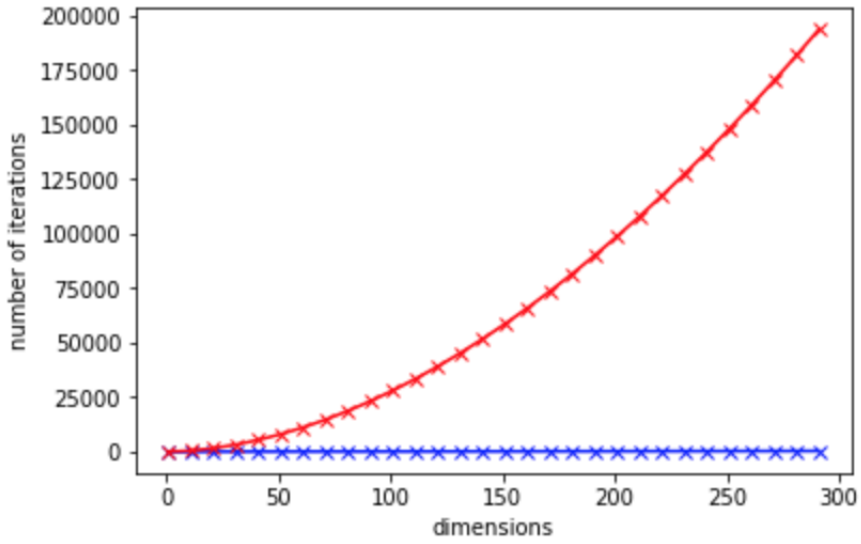



Figure 4.4: This figure put the number of iterations needed for Jacobi method(red line) together with CG method(blue line at the bottom). It is clear that the iterations needed to converge is for these two methods is not in a same level. CG needs only a extremely small number of iterations(shown is figure 4.3) to converge compared with Jacobi. And it is much more effective than a exponential relation for Jacobi method which shows the effectiveness of CG.

# Chapter 5

# Multigrid methods

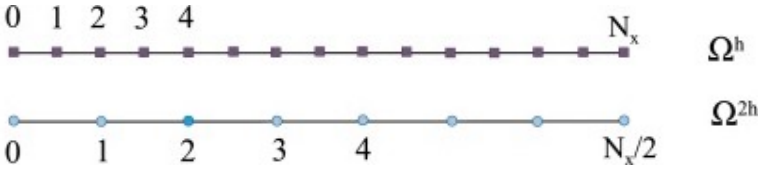
As Jacobi and Gauss-Seidel iterations need a significantly large number of iterations, here we introduce a much more effective way: Multigrid method which change to coarser grid. This idea avoid the case that error vector $e$ removed most of its high frequencies in a few iterations and low frequencies reduced extremely slow. Thus low frequencies act exactly like high frequencies.

## 5.1 Coarse Grid Systems

Consider a grid with grid size $h$ with the corresponding linear system:

$$A^h u^h = f^h \tag{5.1.1}$$

Here we introduce the method to solve this system efficiently which use coarser grids. Given a grid $\Omega^h$, a uniform refinement step remove half of all nodes of $\Omega^h$ leading to $\Omega^{2h}$. Hence, the nodes of $\Omega^{2h}$ are the nodes of $\Omega^h$ with even numbers(see the figure below).




## 5.2 Two-Grid Correction Scheme

As we have the original equation 5.1.1. We can also investigate that the iterative method can be also applied to the error equation which is also named as residual equation. Let $u^i$ be the $i^{th}$ approximation of $u^*$, thus the error $e^i = u^* - u^i$ need to satisfy the equation:

$$Ae^i = A(u^* - u^i) = f - Au^i =: r^i \tag{5.2.1}$$

Now we can introduce the procedure of this two-level method using the residual equation, general idea follows Briggs et al.(2000) page 37 [5]:

**1.** Smooth $v_1$ times on $A^h u^h = f^h$ on $\Omega^h$ with initial guess $v^h$ which gives an approximation of solution and still need to be updated later. Thus we can compute the residual $r^h = f^h - A^h v^h$.
**2.** Do restriction to the residual to $\Omega^{2h}$ named the result $R(r^h)$.
**3.** Solve $A^{2h}e^{2h} = R(r^h)$ on $\Omega^{2h}$. Here we can get the approximation of the error $e^{2h}$.
**4.** Prolongate the coarse-grid error $e^{2h}$ to fine grid $\Omega^h$ and we named the result $P(e^{2h})$.
**5.** Update the solution on $\Omega^h$ by $v^h := v^h + P(e^{2h})$.
**6.** Smooth $v_2$ times on $A^h u^h = f^h$ on $\Omega^h$ with initial guess $v^h$.
This procedure is also named Coarser grid correction or two-level method. Using this strategy, we can compute the approximation of error on $\Omega^{2h}$. The first step of this method is called pre smoothing and the last step is post smoothing. However, there are still a lot of things need to be decide. How to define the coarser grid? Which smoother we need to use? How to do the restriction of residual to coarser grid and prolongation of correction to fine grid? We will discuss all those questions later.

## 5.3 Restriction

First we need to introduce the procedure of mapping fine grid data on the next coarser grid which is known as restriction. For the two-level method, the transfer of residual from $\Omega^h$ to $\Omega^{2h}$ before the coarse grid equation can be solved is called restriction.
The values on the fine grid may be transferred to the coarse grid by two different methods: **Injection**[3] and **full weighting**[17] for finite difference schemes.

### Injection for Restriction

Injection is considered as the simplest way of restriction which is defined as:

$$R_h^{2h} : \mathbb{R}^{N-1} \implies \mathbb{R}^{N/2-1} \text{ where } \quad \phi_i^{2h} = \phi_{2i}^h, \quad \phi^{2h} = R_h^{2h}\phi^h, \quad i = 1, ..., \frac{N}{2} - 1 \tag{5.3.1}$$

In this type of restriction, we take the values for each coarse grid simply the value of grid function at corresponding node on fine grid thus the values are transferred between nodes directly(See the figure 5.1 below).

If we ignore half of the nodes on $\Omega^h$, then the values of the residual and error in these nodes which is the value of $\phi_i$ can not pass on correct information to the system of coarse grid. Hence injection is not an efficient way of restriction and those errors will generally be incorrect.

### Full weighting for Restriction

Full weighting utilizes two additional neighboring nodes on the fine grid thus this weighted restriction uses all nodes on find grid. Hence we can define this average

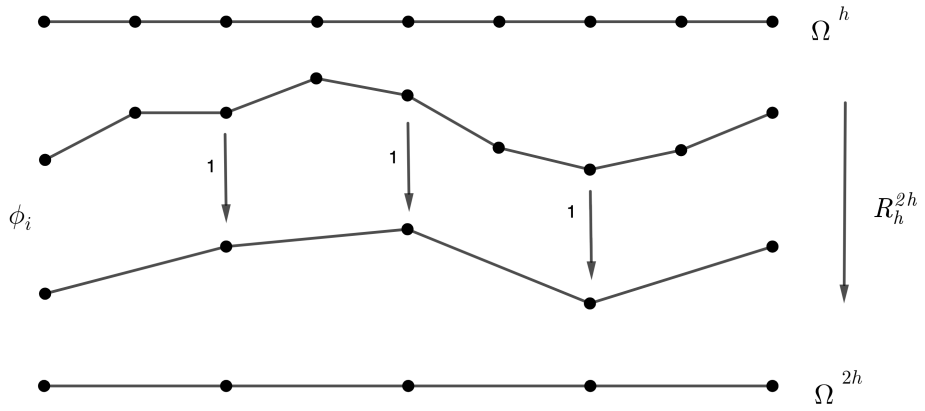



Figure 5.1: An example of injection of restriction between two grid levels the information of grids in fine grid $\Omega^h$ is transferred to coarse grid $\Omega^{2h}$ by directly given the value of the grid at the position in fine grid to the one in coarse grid.

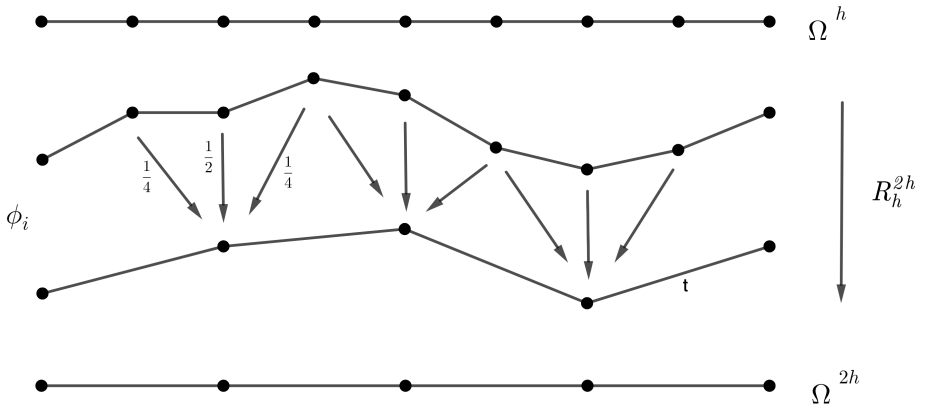



Figure 5.2: An example of full-weighting of restriction. Full weighting is done by taking the half of value itself and the 1/4 of two neighbour grids in the fine grid and give this weighted value to the coarse grid in corresponding position.

method by the equation below:

$$\begin{aligned} R_h^{2h} : \mathbb{R}^{N-1} \implies \mathbb{R}^{N/2-1} \text{ where } \quad \phi^{2h} = R_h^{2h}\phi^h, \\ \phi_i^{2h} = \frac{1}{4}(\phi_{2i-1}^h + 2\phi_{2i}^h + \phi_{2i+1}^h), \quad i = 1, ..., \frac{N}{2} - 1 \end{aligned} \tag{5.3.2}$$

and this could be represented by figure 5.2.

Also, if the spaces of domain and range of restriction function $I_h^{2h}$ are both natural basis, then the full weighting restriction operator can be represent as a

matrix form:

$$R_h^{2h} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 & & & & & & \\ & & 1 & 2 & 1 & & & & \\ & & & & 1 & 2 & 1 & & \\ & & & & & & \ddots & & \\ & & & & & & 1 & 2 & 1 \end{bmatrix} \in \mathbb{R}^{N/2-1} \times \mathbb{R}^{N-1} \tag{5.3.3}$$

Let's make a simple example which do full weighting restriction of $Ru = v$ with fine grid u and coarse grid v which have 9 $u's$ and 5 $v's$ forming a 5 by 9 matrix R.

$$\frac{1}{4}\begin{bmatrix} 1 & 2 & 1 & & & & & & \\ & & 1 & 2 & 1 & & & & \\ & & & & 1 & 2 & 1 & & \\ & & & & & & 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} \tag{5.3.4}$$

This matrix performs the full weighting restriction on the figure above.

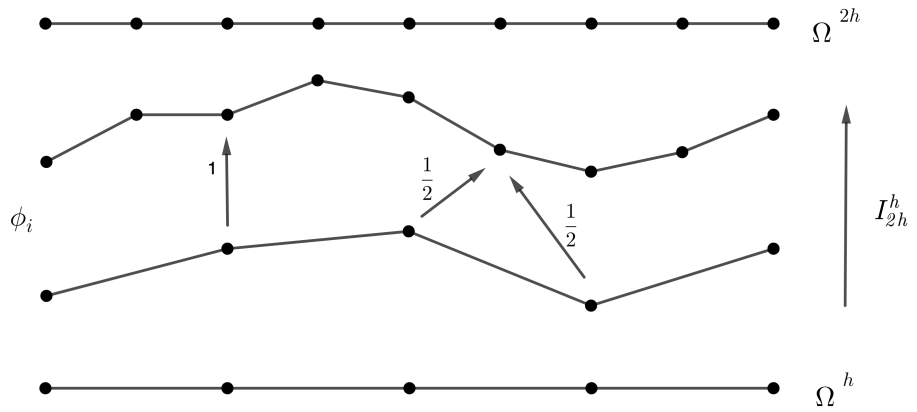



Figure 5.3: An example of linear interpolation between two grid levels, this is the backward procedure of restriction which interpolate information back from coarse grid to fine grid. This is done by directly give the value to fine grids if it is also in coarse grid level. If it is not then take the average of the value of two neighbour grids in coarse grid.

## 5.4 Prolongation

The project from coarse grid to fine grid is named Prolongation or interpolation. Let $N_x$ be the number of nodes on the fine grid $\Omega^h$, and $\frac{N_x}{2}$ the number of nodes on the coarse grid $\Omega^{2h}$. Thus the node i on $\Omega^{2h}$ can be mapped to the node 2i on $\Omega^h$ where $0 \leq i \leq N/2$. Generally, we use the simplest approach which is the linear prolongation and we may only consider this type of interpolation here.

For even nodes of $\Omega^h$, we takes directly the value of the corresponding node of $\Omega^{2h}$ and for odd nodes of $\Omega^h$ we take the arithmetic mean of the two neighbor nodes. Thus we can define the prolongation equation by:

$$\begin{aligned} I_{2h}^h : \mathbb{R}^{N/2-1} &\implies \mathbb{R}^{N-1} \text{ where } \quad \phi^h = I_{2h}^h \phi^{2h}, \\ \phi_{2i}^h &= \phi_i^{2h} \qquad \text{for } i = 1, ..., N/2-1 \\ \phi_{2i+1}^h &= \frac{1}{2}(\phi_i^{2h} + \phi_{i+1}^{2h}) \qquad \text{for } i = 1, ..., N/2-1 \end{aligned} \tag{5.4.1}$$

And see the figure 5.3 which represent the linear interpolation.

The linear prolongation is also a linear operator between two finite-dimensional

spaces $\mathbb{R}^{N/2-1}$ and $\mathbb{R}^{N-1}$. Thus we can give the matrix form of this operator.

$$I_{2h}^{h} = \frac{1}{2}\begin{bmatrix} 1 & & & & & \\ 2 & & & & & \\ 1 & 1 & & & & \\ & 2 & & & & \\ & 1 & 1 & & & \\ & & 2 & \ddots & & \\ & & 1 & \ddots & & \\ & & & & 1 & \\ & & & & 2 & \\ & & & & 1 & \end{bmatrix} \in \mathbb{R}^{N-1} \times \mathbb{R}^{N/2-1} \tag{5.4.2}$$

With this representation, we can see an important relation between full weighting operator and linear interpolation operator:

$$\frac{1}{2} I_{2h}^{h} = (R_{h}^{2h})^{T}$$

Also, considering the effect of the prolongation on different error modes. Assume that the unknown error is a smooth function on the fine grid $\Omega^h$ and the coarse grid approximation on $\Omega^{2h}$ is computed which is exactly in the nodes of the coarse grid. The interpolation of this coarse grid approximation is still a smooth function on the fine grid because there will be no new oscillations if we simply take the midpoint of the line connecting two neighbour nodes on coarse grid. Hence we can expect a relatively good approximation of the smooth error on the fine grid. If the error on fine grid is oscillating, then each interpolation of a coarse grid approximation to the fine grid is a smooth function and we cannot expect that the error on the fine grid is approximated well, see the figure 5.4 below.

In conclusion, prolongation gives the best result if the error on the fine grid is smooth. Thus, the prolongation is an appropriate complement to the smoother which works most efficiently if the error is oscillating.

## 5.5 Coarse grid Matrix

Now we can think about how we can get the coarse grid matrix $A^2h$ which we need to use in the two-level method.
Starting from the residual equation

$$A^h e^h = r^h \tag{5.5.1}$$

If we assumed $e^h$ is in the range of prolongation operator $I_{2h}^{h}$ thus there exists an error defined on coarse grid such that

$$e^h = I_{2h}^{h}(e^{2h}) \tag{5.5.2}$$

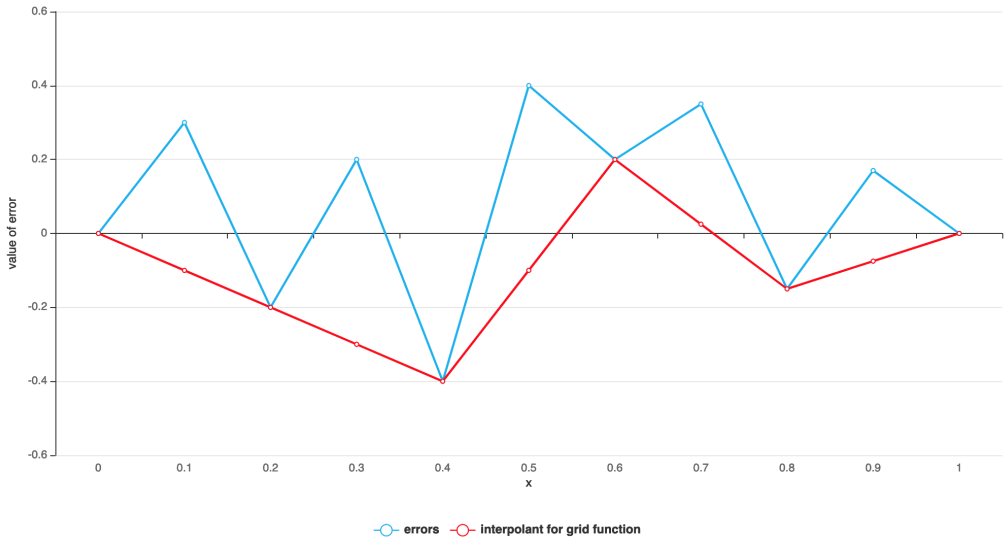



Figure 5.4: An example model of error, by computing the value of error against x, we can see the oscillation of error between two grid levels, clearly in the coarse grid, some information can not be represented.

And if we substitute the error on fine grid to the original residual equation and multiply the restriction operator $R_h^{2h}$ to both sides of the equation we can get

$$R_h^{2h} A^h I_{2h}^h (e^{2h}) = R_h^{2h} r^h = R(r^h) \tag{5.5.3}$$

Hence we can observe that the transformed equation have the same form as the original residual equation thus we have

$$A^{2h} := R_h^{2h} A^h I_{2h}^h \tag{5.5.4}$$

and this typical kind of definition to coarse grid matrix named **Galerkin projection**[11].

Now, as we have already obtained the matrix form of those 3 matrices on in the equation 5.5.4, matrix form of restriction and prolongation operator from last 2 sections, we can generate the matrix form of coarse grid matrix $A^{2h}$ which

is

$$A^{2h} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 & & & & & & \\ & & 1 & 2 & 1 & & & & \\ & & & & 1 & 2 & 1 & & \\ & & & & & & \ddots & & \\ & & & & & & 1 & 2 & 1 \end{bmatrix} \frac{1}{h^2} \begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 & 1 & \\ & & & & 1 & -2 \end{bmatrix}$$

$$\frac{1}{2}\begin{bmatrix} 1 & & & & & \\ 2 & & & & & \\ 1 & 1 & & & & \\ & 2 & & & & \\ & 1 & 1 & & & \\ & & 2 & \ddots & & \\ & & 1 & \ddots & & \\ & & & & 1 & \\ & & & & 2 & \\ & & & & 1 \end{bmatrix}$$

$$= \frac{1}{4h^2}\begin{bmatrix} 0 & -2 & 0 & 1 & & & & & \\ 0 & 1 & 0 & -2 & & & & & \\ 0 & 0 & 0 & 1 & & & & & \\ & & & & \ddots & & & & \\ & & & & & 1 & 0 & 0 & 0 \\ & & & & & -2 & 0 & 1 & 0 \\ & & & & & 1 & 0 & -2 & 0 \end{bmatrix} \frac{1}{2}\begin{bmatrix} 1 & & & & & \\ 2 & & & & & \\ 1 & 1 & & & & \\ & 2 & & & & \\ & 1 & 1 & & & \\ & & 2 & \ddots & & \\ & & 1 & \ddots & & \\ & & & & 1 & \\ & & & & 2 & \\ & & & & 1 \end{bmatrix}$$

$$= \frac{1}{8h^2}\begin{bmatrix} -4 & 2 & & & & & \\ 2 & -4 & 2 & & & & \\ & 2 & -4 & 2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 2 & -4 & 2 & \\ & & & & 2 & -4 \end{bmatrix} = \frac{1}{4h^2}\begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & -2 & 1 & \\ & & & & 1 & -2 \end{bmatrix}$$

Here it is obvious to find that the coarse matrix $A^{2h}$ has the same form as the matrix A in equation 1.1.6 with its $h$ replaced by $2h$. Thus in case of the Poisson equation we discussed, the matrix $A^{2h}$ defined above is the same as the original matrix A (1.1.6) defined by discretizing the differential operator on coarse grid $\Omega^{2h}$. This relation between Galerkin projection and discretization problem on coarse grid will not hold all the time but it can be found often.

## 5.6 Convergence of Two-grid Scheme

The general idea of how to prove the convergence of the two-level multi-grid method follows 'Multigrid Methods' [7]. For proving the convergence of this method, we need to identify the iteration matrix $S_{2grid}$ of this scheme. First, identify the iteration matrix of a chosen smoother $S_m$, and we may ignore the post smoothing for simplicity. The approximation of the solution named $v^n$ before pre smoothing and the result after the update will be $v^{n+1}$. Thus if we apply $v_1$ times of pre smoothing, we can get:

$$e^{v_1} = S_m^{v_1} e^0 \text{ where } e^0 = v^* - v^n, e^{v_1} = v^* - v_{v_1}^n \tag{5.6.1}$$

given $v^*$ to be the actual solution. If we substitute $e^0$ and $e^{v_1}$ into the iterative equation we have:

$$\begin{aligned} v^* - v_{v_1}^n &= S_m^{v_1}(v^* - v^n) \\ v_{v_1}^n &= v^* - S_m^{v_1}(v^* - v^n) \end{aligned} \tag{5.6.2}$$

where $v_{v_1}^n$ here represents the $v^h$ in two-grid correction scheme and we could substitute back to the residual equation:

$$r^h = f^h - A^h v^h = f^h - A^h(v^* - S_m^{v_1}(v^* - v^n)) = A^h S_m^{v_1}(v^* - v^n) \tag{5.6.3}$$

using this equation in the original scheme and follows the step of updating we have:

$$\begin{aligned} v^{n+1} &= v_{v_1}^n + I_{2h}^h e^{2h} \\ &= v^* - S_m^{v_1}(v^* - v^n) + I_{2h}^h (A^{2h})^{-1} R_h^{2h} r^h \\ &= S_m^{v_1} v^n + (I - S_m^{v_1})(A^h)^{-1} f^h + I_{2h}^h (A^{2h})^{-1} R_h^{2h} A^h S_m^{v_1}((A^h)^{-1} f^h - v^n) \\ &= (I - I_{2h}^h (A^{2h})^{-1} R_h^{2h} A^h) S_m^{v_1} v^n + \\ &\quad ((I - S_m^{v_1}) + I_{2h}^h (A^{2h})^{-1} R_h^{2h} A^h S_m^{v_1})(A^h)^{-1} f^h \end{aligned} \tag{5.6.4}$$

Thus it is obvious that the iteration matrix of two-grid correction scheme is :

$$S_{2grid} = (I - I_{2h}^h (A^{2h})^{-1} R_h^{2h} A^h) S_m^{v_1} \tag{5.6.5}$$

Substitute back $v^* = (A^h)^{-1} f^h$ to the equation 5.6.4, we can get that it follows a iteration form of $v^{n+1} = S_{2grid} v^n + k v^*$ which gives that $v^*$ is a fix point. Then if this 2-grid method converge, it converge to $v^*$.
Now, from the Theorem on Iterative Method Convergence[13], a sufficient and necessary condition for the convergence of the fixed point iteration is the spectral radius of iteration matrix less than 1. As we already known the difficulty of computing $S_{2grid}$, $\rho(S_{2grid})$ is even harder. Hence, we may consider doing the spectral norm instead, i.e. $|||S_{2grid}|||$ where we need to show the relation below for some fixed number $\rho$:

$$\rho(S_{2grid}) \leq |||S_{2grid}||| \leq \rho < 1 \tag{5.6.6}$$

If we do some modification to the equation 5.6.5 to the form below:

$$S_{2grid} = ((A^h)^{-1} - I_{2h}^h (A^{2h})^{-1} R_h^{2h}) A^h S_m^{v_1} \tag{5.6.7}$$

$$|||S_{2grid}||| \leq ||||((A^h)^{-1} - I_{2h}^h (A^{2h})^{-1} R_h^{2h})|||| \; |||A^h S_m^{v_1}||| \tag{5.6.8}$$

Thus we could analysis these two components of 2-grid scheme separately. The first factor in (5.6.8) describes the effect of the coarse grid approximation. The smaller the first factor is, the better is the coarse grid solution which approximates the error in fine grid. And by the Approximation property, definition 5.10 on 'Multigrid Methods'[7], states that if there is a constant $C_a$ which is independent of h, such that:

$$||||((A^h)^{-1} - I_{2h}^h (A^{2h})^{-1} R_h^{2h})|||| \leq C_a h^{\alpha} \tag{5.6.9}$$

with some $\alpha > 0$.

The second factor measures the efficiency of the smoothing step. Same, by definition 5.8 of 'Multigrid Methods'[7], which claim that he matrix $S_m$ is said to possess the smoothing property, if $\exists \eta(v)$ and $\bar{v}(t)$ s.t. the definition is independent of $h$, and the same $\alpha$ as described in approximation property, we have

$$|||A^h S_m^{v_1}||| \leq \eta(v) h^{-\alpha} \quad \text{for all } 1 \leq v \leq \bar{v}(h), \tag{5.6.10}$$

where $\eta(v) \to 0$ as $v \to \infty$ and $\bar{v}(h) = \infty$ or $\bar{v}(h) \to 0$ as $h \to 0$. However, this smoothing property does not mean that the smoothing iteration always converge. It is only required that the error is smoothed in a certain way using up to $\bar{v}(h)$ smoothing steps. But we only consider the case that $\bar{v}(h) = \infty$ here, which is same as we have a convergent smoothing iteration.
Thus if we combine these two property together we can get the equation of 2-grid scheme matrix:

$$|||S_{2grid}||| \leq C_a h^{\alpha} \eta(v) h^{-\alpha} = C_a \eta(v) \tag{5.6.11}$$

given the conditions that $\eta(v) \to 0$ as $v \to \infty$, $C_a \eta(v)$ is smaller than any given $\rho > 0$ if $v$ is sufficiently large.
Hence we can deduce the two-grid convergence theorem: Suppose the smoothing property and the approximation property hold. Let $\rho > 0$ be a fixed number, if $\bar{v}(t) = \infty$ for all t, $\exists \underline{v}$ such that

$$|||S_{2grid}||| \leq C_a \eta(v) \leq \rho \tag{5.6.12}$$

whenever $v > \underline{v}$. By showing this theorem, we have achieved our goal of equation 5.6.6 which gives a bound to the spectral norm of iteration matrix. This satisfy the Convergence theorem[13] which states that spectral radius of iteration matrix less than 1 is a sufficient condition.

Also, the convergence theorem[13] says that the two-level method converges with a rate that is independent of $h$ if sufficiently many smoothing steps are applied. In actual cases, most problem only needs a few pre-smoothing iterations to sufficiently converge.

## 5.7 Multigrid Cycles

The main idea behind multigrid methods is to accelerate the convergence of a basic iterative method by solving a coarse problem. In most cases, the problem we need to solve is extremely large hence solving in coarse level $2h$ is still too large and the solving procedure will take too much time. Hence we could do the two-level scheme in multiple times such that the problem is reduced to a solvable extend, in other words, solving the coarse grid problem instead of the fine grid problem where the coarse problem is much cheaper to solve.
As we have discussed the 2-grid scheme which is used to solve the coarse grid equation including one prolongation and one restriction. Here comes the question, what if we have multiple coarse levels? Is it absolutely possible if we want to do more restriction steps and solve the original problem with less mesh points. The idea is simple, the recursive application of two-grid scheme to solve the coarse grid equation forms the general idea of Multigrid method. Moreover, we could decide which kind of trajectory doing restriction and interpolation is the most suitable for specific questions which is the named the **Multigrid Cycles**.

Back to our original problem, we wish to solve the coarse grid equation

$$A^{2h}e^{2h} = R_h^{2h}r^h := r^{2h} \tag{5.7.1}$$

efficiently using two-grid scheme recursively, follows the scheme of Briggs[5] page 40, we could simplify the residual equation by replacing the right-side vector of the residual equation to $f^{2h}$ by $r^{2h}$ because it is just another right-side vector. Also, the solution of the residual equation will be replaced by $u^{2h}$ instead of using $e^{2h}$ as it is just a solution vector. Hence the approximation of solution $u^{2h}$ is now denoted by $v^{2h}$, the solution vector on the finest grid will be denoted by $u^h$ and the current iterate by $v^h$. These modifications simplify the notations and can be used in the implementation of multigrid method.
Note that we could simply choose the initial guess of $v^{2h}$ on the coarse grid $\Omega^{2h}$ to be zero, i.e. $v^{2h} = 0$, as we assumed given no information about the solution.

Here we can introduce the first multigrid cycle, which is as introduced, the two-grid correction scheme embedded with itself. We assume that there are $l > 1$ grids where the finest grid has the grid spacing h and the grid spacing increase by the factor 2 for each coarser grid such that the grid spaces are $Lh$, and $L = 2^{l-1}$.

### Multigrid V-Cycle Scheme

$$v^h \leftarrow V^h(v^h, f^h)$$

- Choose a suitable smoother and relax on $A^h u^h = f^h$ $v_1$ times using this smoother with defined initial guess $v^h$, get the updated result $v^h$.
- Compute $f^{2h} = R_h^{2h}r^h = R_h^{2h}(f^h - A^h v^h)$
  Begin the recursion of restriction steps.

- Relax on $A^{2h}u^{2h} = f^{2h}$ $v_1$ times using the smoother with defined initial guess $v^{2h} = 0$, get the updated result $v^{2h}$.
- Compute $f^{4h} = R^{4h}_{2h} r^{2h} = R^{4h}_{2h}(f^{2h} - A^{2h}v^{2h})$
- Relax on $A^{4h}u^{4h} = f^{4h}$ $v_1$ times using the smoother with defined initial guess $v^{4h} = 0$, get the updated result $v^{4h}$.
- Compute $f^{8h} = R^{8h}_{4h} r^{4h} = R^{8h}_{4h}(f^{4h} - A^{4h}v^{4h})$

  $\vdots$
- Solve $A^{Lh}u^{Lh} = f^{Lh}$
  Begin the recursion of interpolation steps.

  $\vdots$
- Correct the solution by $v^{4h} += I^{4h}_{8h} v^{8h}$
- Relax on $A^{4h}u^{4h} = f^{4h}$ $v_2$ times using the smoother with defined initial guess $v^{4h}$.
- Correct the solution by $v^{2h} += I^{2h}_{4h} v^{4h}$
- Relax on $A^{2h}u^{2h} = f^{2h}$ $v_2$ times using the smoother with defined initial guess $v^{2h}$.
- Correct the solution by $v^{h} += I^{h}_{2h} v^{2h}$
- Relax on $A^{h}u^{h} = f^{h}$ $v_2$ times using the smoother with defined initial guess $v^{h}$.

This algorithm restrict the grid level down to the coarsest grid, which only consist one or a few interior grid points, then prolongate back to the finest grid. And if we plot a figure to see the procedure of the grids in the order in which they are visited, we get the figure 5.5.

And because of the shape of the pattern, this algorithm is named V-Cycle. Also, we could represent the specific steps on a grid level graph 5.6 above.

Moreover, we could describe the V-Cycle algorithm in a recursive way which have the definition:

$$v^h \leftarrow V^h(v^h, f^h)$$

1. Choose a suitable smoother and relax on $A^h u^h = f^h$ $v_1$ times using this smoother with defined initial guess $v^h$, get the updated result $v^h$.
2. If $\Omega^h$ is the coarsest grid, solve the problem.
   Else:
   Do one restriction to the next coarse grid : $f^{2h} \leftarrow R^{2h}_h(f^h - A^h v^h)$
   Set the initial guess of next coarse grid to zero: $v^{2h} = 0$
   Call the V-Cycle scheme again $v^{2h} = V^{2h}(v^{2h}, f^{2h})$
3. Update the solution after prolongation : $v^h \leftarrow v^h + I^h_{2h} v^{2h}$

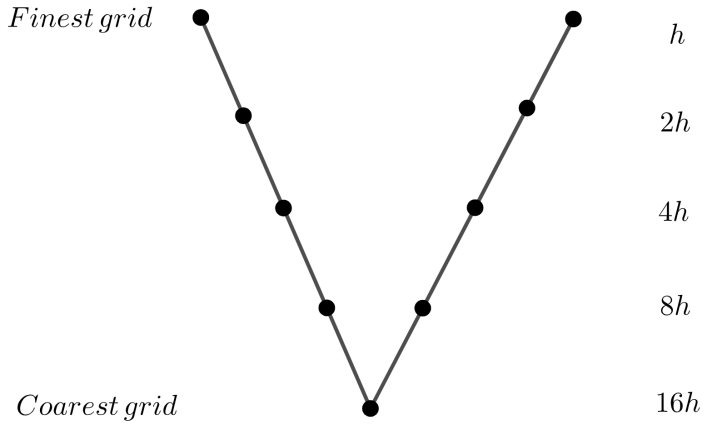



Figure 5.5: An example of a 5-levels v-cycle. The level $h$ represents the fine grid level, and $2h$ means the coarse grid level after one restriction, $4h$ represents another restriction from $2h$ grid level. After 4 restrictions, the fine grid will be reduced to the coarsest grid which is $16h$.

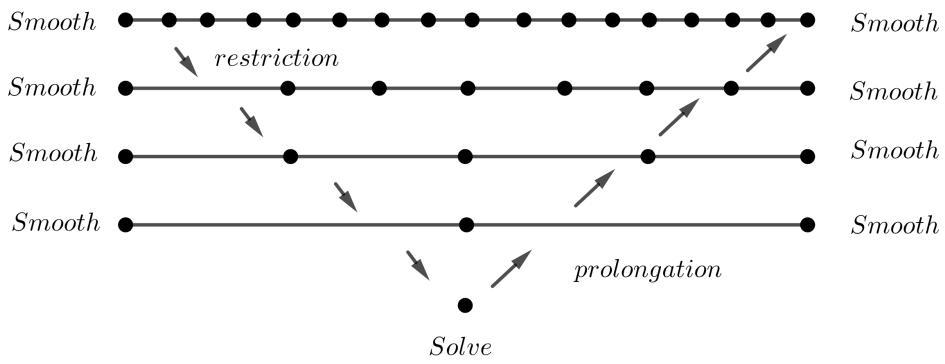



Figure 5.6: A fully explanation of the process of v-cycle,starting with smoothing before each operation and a series of restrictions downward until the coarsest grid, solve the problem at this level, and then prolongate to the finest grid.

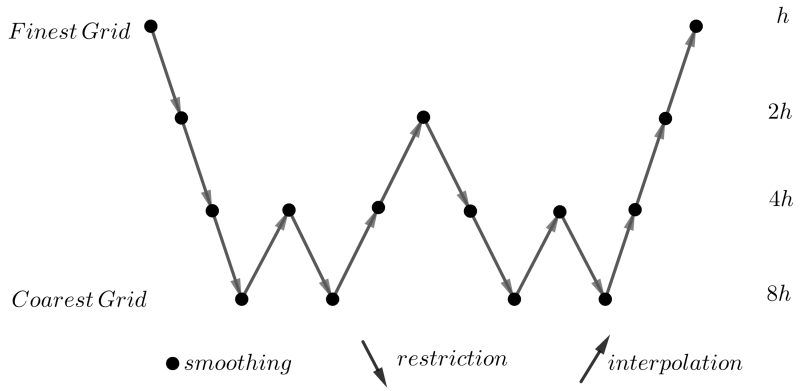



Figure 5.7: An example of w-cycle of 4 grid levels, same as v-cycle before, we need to do smoothing before each restriction and interpolation. The downward arrows represent the restriction and upward ones represent the interpolation.

4. Relax on $A^h u^h = f^h$ $v_2$ times using the smoother with defined initial guess $v^h$.

It is obvious that the procedure of solve at the coarsest grid in V-cycle scheme takes much less time compared with the two-level scheme.
Also, as we have introduced, the V-Cycle is one of the simplest cycle of the family of multigrid methods. The fully defined method is called the $\mu$-cycle method and have the recursive definition below:

## $\mu$-Cycle Scheme

$$v^h \leftarrow M_\mu^h(v^h, f^h)$$

1. Choose a suitable smoother and relax on $A^h u^h = f^h$ $v_1$ times using this smoother with defined initial guess $v^h$, get the updated result $v^h$.

2. If $\Omega^h$ is the coarsest grid, solve the problem.
   Else:
   Do one restriction to the next coarse grid : $f^{2h} \leftarrow R_h^{2h}(f^h - A^h v^h)$
   Set the initial guess of next coarse grid to zero: $v^{2h} = 0$
   If $\Omega^h$ is the finest grid, set $\mu = 1$.
   Call the V-Cycle scheme again $\mu$ times $v^{2h} = M_\mu^{2h}(v^{2h}, f^{2h})$

3. Update the solution after prolongation : $v^h \leftarrow v^h + I_{2h}^h v^{2h}$

4. Relax on $A^h u^h = f^h$ $v_2$ times using the smoother with defined initial guess $v^h$.

Under the definition of $\mu$-Cycle, when $\mu = 1$ is just the V-Cycle and when $\mu = 2$ it becomes **W-Cycle**.

The W-Cycle, like its name, have a hierarchy of grids as the diagram 5.7 above.

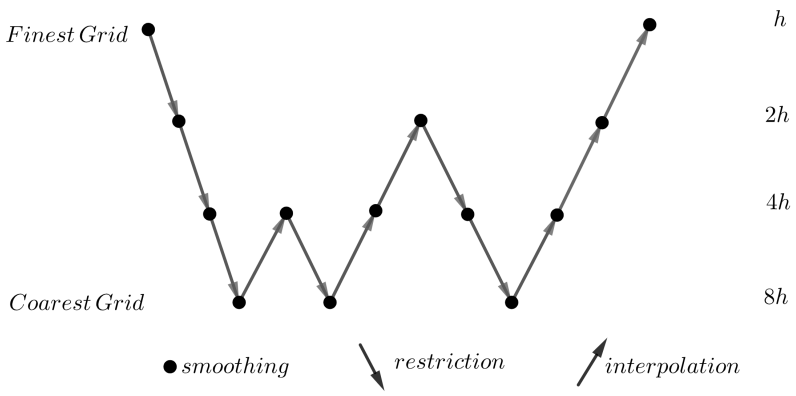



Figure 5.8: An example of f-cycle of 4 grid levels, same as w-cycle but with different shape.

Generally, we only consider cases that $\mu = 1$ or 2. However, there is still modified multigrid cycle in between V-Cycle and W-Cycle which is the **F-Cycle**. The F-cycle starts with the restriction from finest grid to the coarsest grid. While doing the prolongation steps, after having reached each level the first time, again a restriction to the coarsest grid is performed. The hierarchy of a 4-level F-Cycle is shown on the figure 5.8.

Note that we always do $v_1$ times relaxation before correction step and $v_2$ times relaxation after correction step, and we could give a notation to this kind of cycles, for example, a $W(v_1, v_2)$-cycle. Also, when we reach the coarsest grid, using exact solver instead of smoother.

Now we have already get the correction scheme, and we still need to claim that the nested iteration idea which is using coarse grids to obtain improved initial guesses for fine-grid problems. Here gives the algorithm of using nested iteration to solve $A^h u^h = f^h$:

## Nested Iteration

- Using a smoother to solve $A^{h_0} u^{h_0} = f^{h_0}$ on a very coarse grid approximately.
  ⋮
- Smooth $A^{2h} u^{2h} = f^{2h}$ on grid level $\Omega^{2h}$
- solve $A^h u^h = f^h$ on $\Omega^h$ by an iterative method with the initial iterate provided from the coarser grids.

This idea should be the most simple method using coarse grid to improve an iterative method. Recall that the nested iteration stated we could uses coarse grids to generate a better initial guesses for our fine grid problems. Through the V-cycle, we might wondering how could we obtain an informed initial guess for the first fine-grid relaxation? Nested iteration would suggest solving a problem on $\Omega^{2h}$. In the same way, one could obtain a good initial guess for the $\Omega^{2h}$

problem by Nested iteration in solving a problem on $\Omega^{4h}$. Now, it is obvious that this forming a recursive path heading to the coarsest grid. Combining this idea with the correction scheme of V-cycle, we could obtain a new Multigrid cycle named **Full-Multigrid V-cycle**[5], simplified as **FMG**. Here comes the scheme of FMG:

## Full-Multigrid V-cycle

$$v^h \leftarrow FMG^h(f^h)$$

- Initialise $f^h, f^{2h}, f^{4h}, ..., f^H$ by multiply the restriction matrix to the last fine grid level of $f$, i.e. $f^{2h} \leftarrow R_h^{2h} f^h, ...$
- Solve on the coarsest grid $v^H = (A^H)^{-1} f^H$.
  $\vdots$
- interpolate initial guess $v^{4h} \leftarrow I_{8h}^{4h} v^{8h}$
- perform V-Cycle $v^{4h} = V^{4h}(v^{4h}, f^{4h})$ $v_0$ times
- interpolate initial guess $v^{2h} \leftarrow I_{4h}^{2h} v^{4h}$
- perform V-Cycle $v^{2h} = V^{2h}(v^{2h}, f^{2h})$ $v_0$ times
- interpolate initial guess $v^h \leftarrow I_{2h}^h v^{2h}$
- perform V-Cycle $v^h = V^h(v^h, f^h)$ $v_0$ times

The cycling parameter, $v_0$, sets the number of V-cycles done at each level. Generally we pick $v_0 = 1$ in most cases. Of course, FMG have a recursive definition in the form below:

## Recursive Form of Full Multigrid V-Cycle

$$v^h \leftarrow FMG^h(f^h)$$

1. Initialise $f^h, f^{2h}, f^{4h}, ..., f^H$ by multiply the restriction matrix to the last fine grid level of $f$, i.e. $f^{2h} \leftarrow R_h^{2h} f^h, ...$
2. If $\Omega^h$ is the coarsest grid, then solve exactly. Else:
   $v^{2h} \leftarrow FMG(f^{2h})$
3. Set initial guess $v^h \leftarrow I_{2h}^h v^{2h}$.
4. $v^h \leftarrow V(v^h, f^h)$ $v_0$ times.

And the hierarchy of grids of full multigrid V-Cycle is exactly like the diagram 5.9 above.

which shows a FMG with $v_0 = 1$. The full multigrid V-cycle looks like a F-cycle but remove the first restriction step to the coarsest grid and pre smoothing step. In practice, we wish to engage the problem with the best possible initial

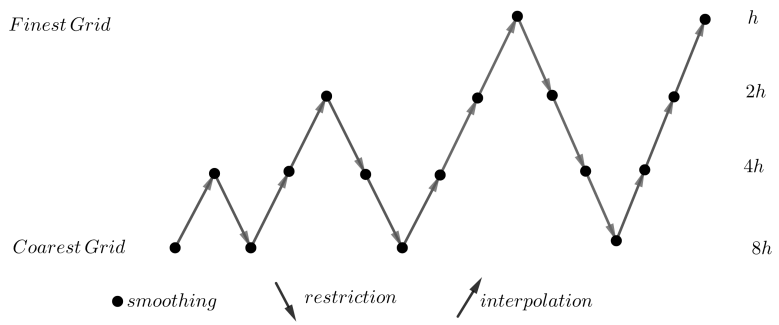



Figure 5.9: An example of Full multigrid V-cycle in 4 grid levels.

guess. FMG is such a suitable approach which solving the problem approximately first on a coarser grid and prolongate this solution back to the fine grid for using as the first approximation. This method avoid the difficulty of complexity and interpret this problem with an effective way.

# Convergence of Multigrid Method

As we have already discussed the convergence of two-grid scheme in section 5.6, we only need to show that this is sufficient to imply that multigrid method with different cycles are convergent. The general idea is the same as all other iterative methods, to show that the spectral radius of the iteration matrix is less than 1. The specific proof of v-cycle and f-cycle is too complicated and we can follow 'Convergence Of Nonconforming V-cycle and F-cycle multigrid Alogorithms'[4] by Susanne C. Brenner for fully explanation. However, We can still proved that the rate of convergence for w-cycle will be bounded by a number $\rho(v) < 1$ which depends on the number of pre smoothing steps and which is independent of the finest step size h and of the number of levels involved in the multigrid scheme. But this technique cannot be applied to the multigrid V-cycle. As we already have the iteration matrix of two-grid scheme

$$S_{2grid} = ((A^h)^{-1} - I_{2h}^h (A^{2h})^{-1} R_h^{2h}) A^h S_m^{v_1}$$

we could obtain the iteration matrix of w-cycle in the same way. Let the levels of the multigrid hierarchy numbered by $0, ..., l$, where level 0 is the coarsest grid. The iteration matrix of the two-level method on level $l$, where the corresponding mesh width should be $h$, is denoted by $S_l$ and it has the form

$$S_l(v) = (I - R_{l-1}^l (A_{l-1}^{-1}) I_l^{l-1}) \tag{5.7.2}$$

Thus the iteration matrix of the multigrid w-cycle scheme is given by

$$\begin{aligned} S_{multigrid,l}(v) &= S_l(v) \quad \text{if } l = 1 \\ S_{multigrid,l}(v) &= S_l(v) + R_{l-1}^l (S_{multigrid,l-1}(v))^2 A_{l-1}^{-1} I_l^{l-1} A_l S_{m,l}^v \quad \text{for } l \geq 2 \end{aligned} \tag{5.7.3}$$

The result can be obtained using the similar way as proving the convergence of two-grid scheme, details of the proof can be found in section 6.2 of 'Multigrid methods'[7]. We could show that the spectral norm of this iteration matrix is smaller than 1 which gives a bound and satisfy the necessary condition of Convergence Theorem.

## 5.8 2-D Model Problem

Of course, the multi-scale algorithm is not restricted to the 1D boundary value problem which we discussed before, i.e.

$$Lu = \frac{d^2u}{dx^2} = f(x), \text{ for } x \text{ in } (0,1). \tag{5.8.1}$$

with some boundary conditions. So let's think about a two dimensional boundary problem:

$$Lu = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y), \text{ where } (x,y) \text{ in } \Omega \tag{5.8.2}$$

given $u = b(x,y)$ and $(x,y)$ on $\partial\Omega$. Here functions $f$ and $b$ are given and as usual, we are looking for a solution $u$. Let the domain to be a unit square for simplicity $\Omega = (0,1)^2$ where the $\partial\Omega$ means the boundary of $\Omega$. This is still the Poisson problem in the first chapter with Dirichlet boundary conditions. And the discretization generally follows 'Why Multigrid Methods Are So Efficient'[18] page 20 by Irad Yavneh. To discretize the problem, we define a uniform grid with mesh height $h = 1/N$ and mesh points given by

$$(x_i, y_j) = (ih, jh) \quad \text{where} \quad i,j = 0,1,...,N \tag{5.8.3}$$

Let $u^h$ denotes the approximation to $u$ and $b^h$ denotes the discrete boundary conditions, and the discrete right-hand side function is denoted $f^h$. The discretization of the derivatives generates a system of equations of $u^h$, which can be written as:

$$\begin{aligned}(L^h u^h)_{i,j} \equiv \frac{u^h_{i-1,j} + u^h_{i+1,j} + u^h_{i,j-1} + u^h_{i,j+1} - 4u^h_{i,j}}{h^2} = f^h_{i,j} \text{ where } i,j = 1,...,N-1 \\ u^h_{i,j} = b^h_{i,j}, i = 0 \text{ or } i = N \text{ or } j = 0 \text{ or } j = N\end{aligned} \tag{5.8.4}$$

We can write this system again more concisely as $L^h u^h = f^h$. Jacobi's algorithm is easily adapted to the 2D problem: we could change our approximation to $u^h_{i,j}$ at each iteration so as to satisfy the $(i,j)$th equation of 5.8.4 with the neighboring values taken from the previous iteration.

In order to expand the multigrid V-cycle to the 2D model problem, we need to define our coarse grids first. Also, the appropriate grid transfer operators, namely interpolation or restriction matrix, is necessary. We coarsen naturally by selecting all the mesh-points with either $i$ or $j$ odd then delete them, by doing this operation, around 3/4 of the mesh points can be eliminated. As for restriction,

the most general way : direct injection, defined for a fine-grid vector $g^h$ by

$$(R_h^{2h} g^h)_{i,j} = g^h_{2i,2j} \quad \text{where} \quad i,j = 0,1,...,N/2-1 \tag{5.8.5}$$

where the coarse-grid value is simply given by the fine-grid value at the corresponding location. Alternatively, we could using full-weighting which is defined by

$$\begin{aligned}(R_h^{2h} g^h)_{i,j} = \frac{1}{16}(&g^h_{2i-1,2j-1} + g^h_{2i+1,2j-1} + g^h_{2i-1,2j+1} + g^h_{2i+1,2j+1} \\ &+4(g^h_{2i-1,2j} + g^h_{2i+1,2j} + g^h_{2i,2j-1} + g^h_{2i,2j+1}) + 4g^h_{2i,2j})\end{aligned} \tag{5.8.6}$$

given the same $i, j$ in equation 5.8.5. And what this equation do is a local averaging. For coarse-to-fine data transfer, a common choice is bi-linear interpolation, defined for a coarse-grid vector $g^{2h}$ by:

$$(R_h^{2h} g^h)_{i,j} = \begin{cases} g^{2h}_{i/2,j/2} & \text{if i is even, j is even} \\ \frac{1}{2}(g^{2h}_{(i+1)/2,j/2} + g^{2h}_{(i-1)/2,j/2}) & \text{if i is odd, j is even} \\ \frac{1}{2}(g^{2h}_{i/2,(j+1)/2} + g^{2h}_{i/2,(j-1)/2}) & \text{if i is even, j is odd} \\ \frac{1}{4}(g^{2h}_{(i+1)/2,(j+1)/2} + g^{2h}_{(i-1)/2,(j+1)/2} & \\ +g^{2h}_{(i+1)/2,(j-1)/2} + g^{2h}_{(i-1)/2,(j-1)/2}) & \text{if i is odd, j is odd} \end{cases}$$

given $i, j = 0, 1, ..., N/2-1$. This result gives the prolongation matrix and if we substitute this into the previous one dimensional FMG format, we could obtain the 2-D method.

## 5.9 Computational Cost

It is not hard to see that the linear relation between the number of mesh-points and the amount of operations needed in V-cycle on each grid, and this property holds for each of the sub-processes-relaxation, residual computation, restriction, and prolongation. As we already known the number of points at each grid is approximately a constant fraction(1/2) of that of the next finer grid level, generally a half for one-dimensional problem and a quarter for two-dimensional, so that the total number of operations per iteration is just a multiple times the number of operations performed on just the finest grid, which can be represented by a constant factor times the number of operations on finest grid.
We could also observe that this factor could be bounded, in case of one-dimensional problem, the upper bound will be $1 + 1/2 + 1/4 + ... = 2$ and in case of two-dimensional, it becomes $1 + 1/4 + 1/16 + ... = 4/3$.
It is convenient to measure these costs in terms of a work unit (WU)[5] defined by the cost of performing one relaxation sweep on the finest grid. Then, we can measure the work of our V-cycle in units of Wu. Relaxation on the finest grid requires 3 WUs and computing the residual need one. The prolongation and restriction operations require less or equal to one unit of WU. Thus, summing these procedures up, the fine-grid work per V cycle is about 5 WUs. The entire work is therefore about 10 WUs per V-cycle in 1D and about 7 in 2D.

Finally through observation, we can find that on each grid in the hierarchy, the operations (relaxation, prolongation, and restriction) can be executed simultaneously. Hence, the number of sequential steps required by the V-cycle is proportional to the number of grids, which is simply $\mathcal{O}(logN)$.

## 5.10 Numerical Test on Multigrid

To demonstrate the multigrid algorithm's effectiveness, we test the 1-D V-cycle for a range of problem size up to a dimension of 2000 for the number of iterations needed to perform multigrid V-cycle. The result is not surprising that Number of iterations is independent of problem size, see figure 5.10. Also, the plot of error norm against the number of iterations on performing 1-D v-cycle of a 100 dimensional matrix is shown below which gives a comparison between multigrid and the Jacobi method. The V-Cycle's computational cost is in the same level as a few Jacobi iterations. We can see that each V-Cycle reduces the error by roughly an order of magnitude, whereas Jacobi iteration converges slowly. See appendix C for the code of implementation.

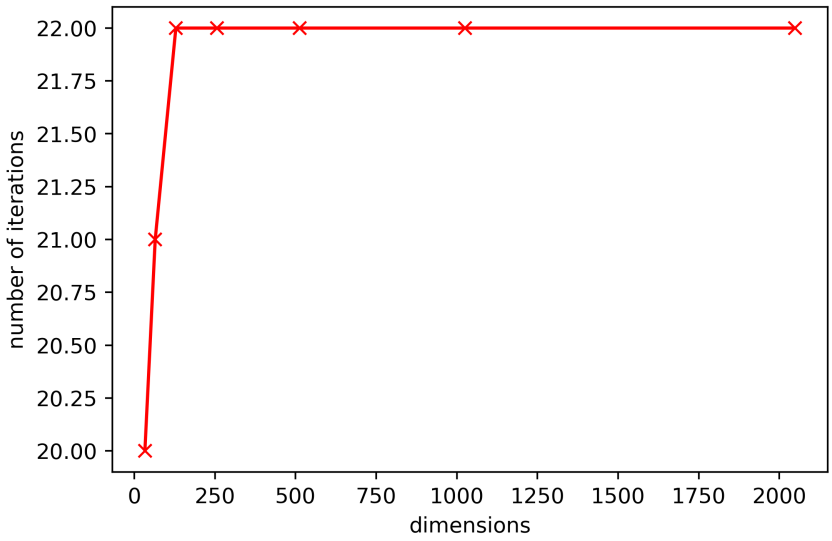



Figure 5.10: to test a set of matrices with dimensions up to 2000 respectively, we can observe that the iterations needed for v-cycle algorithm is independent of the problem size which is reasonable as we always reduced into the coarsest grid and solve the problem at this level.

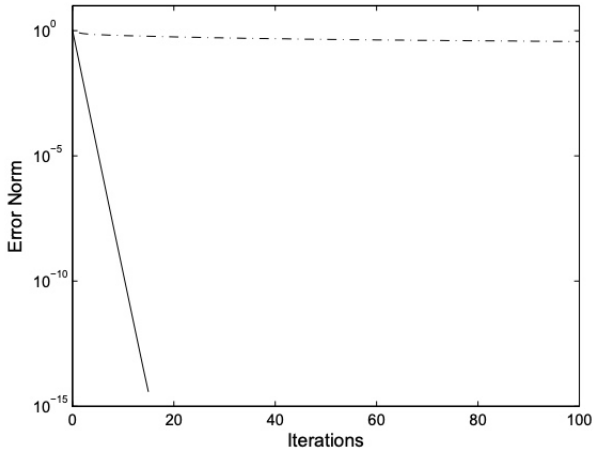



Figure 5.11: The plot shows the error norm against number of iterations compare multigrid v-cycle with Jacobi method. The dotted line is for Jacobi method and the straight line decreasing rapidly is for v-cycle. We can see that each V-Cycle reduces the error by roughly an order of magnitude, whereas Jacobi iteration converges slowly.

# Conclusion

These iterative algorithms I describe in this report provide a glimpse into the machine computational methods. We can make a lot implementations through these algorithms and be proficient on these methods immediately useful solving simple linear equations system on simple domains. The iterative methods I introduced before are starting with the baseline and broaden to the multigrid algorithm which is an advanced algorithm even in this day and age. This topic on improving and discovering more efficient iterative method have been widely researched over last several decades and some of them have been mentioned in this report. Although we have resolved many of the difficulties, exploring these ideas outside the realm of differential equations is still extremely challenging.
The concepts and algorithms I describe in this report are the most common and popular iterative methods which is the basis of my further research and study.

# Bibliography


[1] Irene Ann Abramowitz, Milton; Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover Publications. p. 773, 1983.

[2] A.V.Frolov. Forward substitution. `https://algowiki-project.org/en/Forward_substitution`.

[3] Robert G Bartle. *The Elements of Real Analysis (2nd ed.) p.17*. John Wiley & Sons, 1976.

[4] Susanne C. Brenner. Convergence of nonconforming v -cycle and f-cycle multigrid algorithms for second order elliptic boundary value problems. *MATHEMATICS OF COMPUTATION Volume 73, Number 247, Pages 1041–1066*, 2003.

[5] E. Henson Briggs, W. and S. McCormick. *A multigrid tutorial. 2d ed.* SIAM,Philadelphia, PA., 2000.

[6] Van Loan Charles F Golub, Gene H. *Matrix Computations (3rd ed)*. Johns Hopkins University Press, 1996.

[7] Volker John. Multigrid methods. Technical report, Weierstrass Institute for Applied Analysis and Stochastics, Winter Semester 2013/14.

[8] Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations Steady-State and Time-Dependent Problems. University of Washington Seattle, Washington*. Society for Industrial and Applied Mathematics, 2007.

[9] D. G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. Springer, 2008.

[10] Juan C. Meza. Steepest descent. Technical report, Lawrence Berkeley National Laboratory Berkeley, California 94720, 2010.

[11] Hazewinkel Michiel. *Galerkin Method*. Springer Science+Business Media B.V. / Kluwer Academic Publishers, 1994.

[12] Stephen G. Nash and Ariela Sofer. *Linear and Nonlinear Programming*. McGraw-Hill, 1996.

[13] Ma Changfeng Nie Pu-yan, Huang Na. Convergence analysis and numerical study of a fixed-point iterative method for solving systems of nonlinear equations. *The Scientific World Journal*, 2014.

[14] Bruce A.Watson Paul A.Binding, Patrick J.Browne. *Sturm–Liouville problems with boundary conditions rationally dependent on the eigenparameter, II.* Journal of Computational and Applied Mathematics, 2002.

[15] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213, 1994.

[16] The European Mathematical Society. Encyclopedia of mathematics. `http://www.encyclopediaofmath.org/index.php?title=Condition\_number&oldid=29466`.

[17] Gilbert Strang. *Mathematical Methods for Engineers II chapter 6.3.* massachusetts institute of technology, 2006.

[18] Irad Yavneh. Why multigrid methods are so efficient. *Computing in Science and Engineering*, 2006.

**

# Appendix A

# Implementation of Jacobi iteration

Function to get the matrix of type 1.1.6 in sparse form.

```
def d3(m):
    dok = dok_matrix((m, m), dtype=np.float64)
    h = 1/(m+1)
    for i in range(m):
        dok[i, i] = -2 / (h * h)

        if i < m - 1:
            dok[i, i + 1] = 1 / (h * h)
            dok[i + 1, i] = 1 / (h * h)

    return dok.tocsr()
```

Function to get the diagonal inverse of a matrix.

```
def diaginv(m):
    dok = dok_matrix((m, m), dtype=np.float64)
    h = 1/(m+1)
    for i in range(m):
        dok[i, i] = (h * h) / (-2)

    return dok.tocsr()
```

Function to get the remain of matrix in form 1.1.6 after remove the diagonal inverse.

```
def remain(m):
    dok = dok_matrix((m, m), dtype=np.float64)
    h = 1/(m+1)
    for i in range(m):
        if i < m - 1:
            dok[i, i + 1] = 1 / (h * h)
            dok[i + 1, i] = 1 / (h * h)

    return dok.tocsr()
```

Finally the Jacobi method.

```
def jacobi(m,b,tol,N):
    x = zeros(m)
    start = time.time()
    D = diaginv(m)
    R = remain(m)
    temp = dot(D,R)
    count = 0

    for i in range(N):
        bn = D.dot(b)
        xn = temp.dot(x)
        update = bn - xn

        if(norm(x-update, ord=np.inf)< tol):
            count = i
            break
        #elif(i == N-1):
            #raise Exception("fail to converge")
        else:
            x = update

    end = time.time()
    time1 = end - start

    return x,count,time1
```

# Appendix B

# Implementation of Conjugate Gradient

```
def cg(A, b, x=None):
    count = 0
    n = len(b)
    if not x:
        x = np.ones(n)
    r = b - A.dot(x)
    p = r
    r_k_norm = np.dot(r, r)
    for i in range(2*n):
        Ap = A.dot(p)
        alpha = r_k_norm / p.dot(Ap)
        x += alpha * p
        r -= alpha * Ap
        r_kplus1_norm = np.dot(r, r)
        beta = r_kplus1_norm / r_k_norm
        r_k_norm = r_kplus1_norm
        if r_kplus1_norm < 1e-5:
            # print('Itr:', i)
            count = i
            break
        p = r + beta * p
    return x, count
```

# Appendix C

# Implementation for Multigrid V-cycle

```
def interpolation_matrix(a,b):
    P = dok_matrix((a, b), dtype=np.float64)
    for k in range(b):
        P[2*k,k] = 1; # copy these points
    for k in range(b-1):
        P[2*k+1,k] = .5; # average these points
        P[2*k+1,k+1] = .5;
    return P.tocsr()
```

```
def vcycle(A,f):
    sizeF = np.size(A,axis=0);
    if sizeF < 5:
        v = cg(A,f)
        return v
    N1 = 5   # number of jacobi iteration as smoothing.
    v = jacobi(A,f,1e-10,N1)
    # Restriction
    sizeC = math.ceil(sizeF)
    P = interpolation_matrix(sizeF,sizeC)
    PT = P.transpose()
    residual = f - A.dot(v)
    Rresidual = PT.dot(residual)  # R(r^c)
    Ap = A.dot(P)
    AC = PT.dot(Ap)  # A in coarser grid
    VC = vcycle(AC,Rresidual)
    v += P.dot(VC)
    v = jacobiv(A,f,1e-10,N1,v)
    return v
```

The Jacobi relaxation with restricted N times of iterations.

```
def jacobiv(m,b,tol,N,x):
    start = time.time()
    D = diaginv(m)
    R = remain(m)
    temp = dot(D,R)
    for i in range(N):
        bn = D.dot(b)
        xn = temp.dot(x)
        update = bn - xn
        if(norm(x-update, ord=np.inf)< tol):
            break
        else:
            x = update
    return x
```