An-Najah National University
Faculty of Graduate Studies

Multigrid Methods for Elliptic Partial Differential Equations

By
Rania Taleb Mohammad Wannan

Supervisor
Dr. Anwar Saleh

Submitted in Partial Fulfillment of the Requirements for the Degree of Master in Science in Mathematics, Faculty of Graduate Studies, at An- Najah National University, Nablus, Palestine.

2010

Multigrid Methods for Elliptic Partial Differential Equations

By
Rania Taleb Mohammad Wannan

Supervisor
Dr. Anwar Saleh

This Thesis was defended successfully on 28/4/2010 and approved by:

Committee Members		Signature
1. Dr. Anwar Saleh	Supervisor	.....
3. Dr. Saed Mallak	External Examiner	.....
4. Dr. Samir Mater	Internal Examiner	.....

Dedication

I dedicate this thesis to my parents and my husband Khalid, without their patience, understanding, support and most of all love, this work would not have been possible.

Acknowledgement

I am heartily thankful to my supervisor, Dr. Anwar Saleh, whose encouragement, guidance and support from the initial to the final level enabled me to develop and understanding the subject.

My thanks and appreciation goes to my thesis committee members Dr. Saed Mallak and Dr. Samir Matar for their encouragement, support, interest and valuable hints.

I acknowledge An-Najah National University for supporting this work, and I wish to pay my great appreciation to all respected teachers and staff in department of mathematics.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of this thesis.

إقـــرار

أنا الموقع أدناه مقدم الرسالة التي تحمل العنوان:

Multigrid Methods for Elliptic Partial Differential Equations

اقر بأن ما اشتملت عليه هذه الرسالة إنما هي نتاج جهدي الخاص، باستثناء ما تمـــت الإشـــارة إليه حيثما ورد، وأن هذه الرسالة ككل، أو أي جزء منها لم يقدم من قبل لنيل أية درجة علمية أو بحث علمي أو بحثي لدى أية مؤسسة تعليمية أو بحثية أخرى.

Declaration

The work provided in this thesis, unless otherwise referenced, is the researcher's own work, and has not been submitted elsewhere for any other degree or qualification.

Student's name: **اسم الطالب:**

Signature: **التوقيع:**

Date: **التاريخ:**

Table of Contents

List of Tables

Multigrid methods for Elliptic Partial Differential Equations

By
Rania Taleb Mohammad Wannan
Supervisor
Dr. Anwar Saleh

Abstract

Partial differential equations appear in mathematical models that describe natural phenomena. Various methods can be used for solving such equations. In this thesis, an overview of classical iterative methods, as well as, the most recent multigrid methods is given. The classical iterative methods used are; the Jacobi, the Gauss-Seidel, and the SOR methods. Jacobi and Gauss-Seidel methods are efficient in smoothing the error but not in reducing it. The smoothing property of some classical methods motivated the work done on multigrid methods. Poisson's problem in one and two dimensions has been used as model problem in the study of multigrid methods. The study shows that the rate of convergence of multigrid methods does not depend on the mesh size, a feature that makes multigrid methods good accelerator of classical methods like Gauss-Seidel.

Chapter one

Introduction

1.1 Discretization

1.2 A Brief history of multigrid

1.3 Grid structure

1.4 Stencil notation

Chapter 1

Introduction

Many physical problems, such as fluid flow problems, are represented by mathematical models that consist of Partial Differential Equation (PDE) or system of PDE's together with a set of boundary conditions. In most cases, such PDE's are of order two. Linear second-order PDE's are classified in three categories: parabolic, hyperbolic, and elliptic. The general second-order linear PDE in two independent variables x and y can be written as:

$$Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu = G$$

where $A, B, C, D, E, F,$ and G are given functions of x and y. This equation is said to be parabolic if $B^2 - 4AC = 0$, hyperbolic if $B^2 - 4AC > 0$ and elliptic if $B^2 - 4AC < 0$. For example, in one dimension, the diffusion equation; $u_t - ku_{xx} = 0$ is parabolic. The wave equation; $u_{tt} - c^2 u_{xx} = 0$ is hyperbolic, while Laplace's equation in two dimensions; $u_{xx} + u_{yy} = 0$ is elliptic. The PDE is incomplete without boundary and initial conditions. There are three types of boundary conditions:

- *Dirichlet boundary conditions* where the solution is specified at the boundaries.

- *Neumann boundary conditions* where the normal derivative at the boundaries is given.
- *Robin boundary conditions* where the solution and its normal derivative is given in a mixed way.

In this thesis, only the Dirichlet boundary conditions are considered.

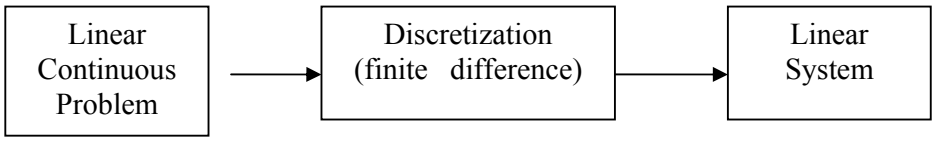
Exact (continuous) solutions of such models are not always available. In fact, for some models, it is not known whether an analytic solution exists or not. For this reason, approximate solutions are needed. Elliptic boundary value problems are the type of the problems to which multigrid methods can be applied very efficiently. Other examples of successful applications are parabolic problems, hyperbolic problems, optimization problems**.**

In this thesis, multigrid methods based on finite difference discritization is considered. First, the problem is discretized leading to a system of linear equations if the PDE is linear and a system of nonlinear equations if the PDE is nonlinear. Then the algebraic system is solved using the most efficient techniques. The result is the discrete solution of the boundary value problem.

1.1 Discretization

There are several methods to discretize a PDE some of these methods are the finite difference methods and the finite element methods. The finite difference is simple and is the most popular when the boundaries

are rectangular such as in numerical wheather prediction. Finite element methods, are most popular when the boundaries are irregular or moving like in simulation of the forces acting on an airplane or in a car accident.



Suppose that u is the exact solution of the elliptic PDE with independent variables x and y, where $a \le x \le b, \mathrm{c} \le \mathrm{y} \le \mathrm{d}$, and we need to find the approximate solution. First, we discretize the PDE. Choosing integers n and m, and define step sizes $h_1 = \frac{b-a}{n}$ and $h_2 = \frac{d-c}{m}$. Partitioning the interval $[a,b]$ into n equal subintervals each of width h_1 and the interval $[c,d]$ into m equal subintervals of width h_2 as in Figure1.1. The result is a grid on the rectangle $[a,b]\times[c,d]$ obtained by drawing vertical and horizontal lines through the points with coordinates (x_i, y_j) where:

$$x_i = a + ih_1, i = 0,1,...,n \ .$$

and $y_j = c + jh_2, j = 0,1,...,m$.

The lines $x = x_i$ and $y = y_j$ are called grid lines, and their intersections are called grid points (mesh points). Numerical differentiation formulas are used to replace the derivatives in the elliptic PDE, converting the elliptic

PDE into an algebraic equation for each grid point. For simplicity, we use the following second-order centered-difference formulas:

$$\frac{\partial u}{\partial x} \approx \frac{u_{i+1j} - u_{i-1j}}{2h}$$

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i+1j} - 2u_{ij} + u_{i-1j}}{h^2}$$

Similarly, for other derivatives.

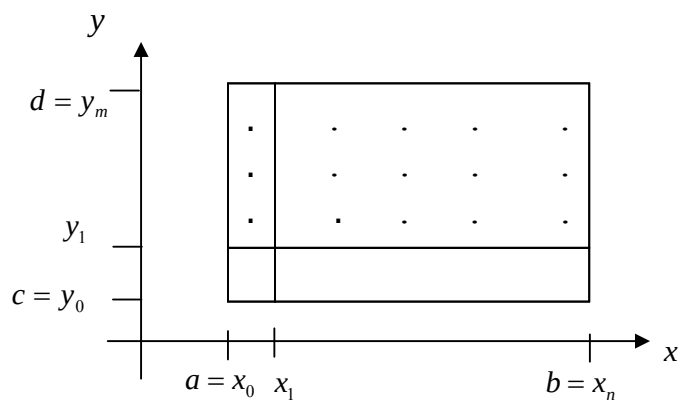


Figure1.1

Example 1.1

Consider the Poisson equation:

$$u_{xx} + u_{yy} = 1$$

In the square region $\Omega = [-1,1] \times [-1,1]$ with boundary condition, $u(x, y) = 0, \forall (x, y) \in \partial\Omega$. Using second–order formulas for the derivatives with $h_1 = h_2 = 0.5$, give the difference equation:

$$u_{i+1j} - 4u_{ij} + u_{i-1j} + u_{ij+1} + u_{ij-1} = 0.25, \quad 1 \le i,\ j \le 3$$

The linear system associated with this problem has the form

$$\begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix} \times \begin{bmatrix} u_{11} \\ u_{12} \\ u_{13} \\ u_{21} \\ u_{22} \\ u_{23} \\ u_{31} \\ u_{32} \\ u_{33} \end{bmatrix} = \begin{bmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{bmatrix}.$$

Several solution methods can be used to solve the linear system resulted from the discretization process. Direct methods such as Gaussian elimination can be used, other more efficient direct methods also can be used. In real problems, The systems are very large systems , and the direct methods become inefficient, since they lead to the formation of intermediate matrices, making the number of arithmetic operations necessary for the solution too large. For this reason, iterative methods are used for solving such systems. Several classical iterative methods exist. Some of such methods are:

- Jacobi method.
- Gauss-Seidel method.
- Successive over relaxation (SOR) method.

Iterative methods begin with an initial approximation of the solution, and generate a sequence of approximations assumed to converge to the

exact solution. The error in such approximations is the result of machine (rounding) error and the number of iterations used. Classical iterative methods are easy to implement and may be successfully applied to more general systems than most direct methods. However, iterative methods suffer some limitations. They are characterized by slow error reduction, but they provide rapid damping, leaving smooth error. For this reason, these methods are called smoothers. Multigrid methods have been developed through attempts to overcome these limitations. They use these classical methods as smoothers.

1.2 A Brief History of Multigrid Methods

First studies investigating multigrid methods are given by Fedorenko from 1962 to 1964, who developed the first multigrid scheme of the Poisson equation in a unit square. His work was generalized to the general linear elliptic PDE with variable smooth coefficients by Bachvalov in 1966. The actual efficiency of multigrid methods was reported in a paper by Brandt in 1973, who presented another paper in 1977, clearly outlining the main principles and practical utility of multigrid methods. Brandt's work drew attention and marked the beginning of rapid development. During 1975 and 1976, Hackbusch developed the fundamental elements of multigrid methods, Hackbusch's first systematic report in 1976 contained many theoretical and practical investigations, which were taken up and

developed further by several authors. Since the early 1980s, the field of multigrid extended and many researchers have contributed to this field. Two series of conferences dedicated to multigrid methods were set up: the European Multigrid Conference (EMG) held at Cologne in 1981 and 1985, Bonn in 1991, Amsterdam in 1993, Stuttgart in 1996 and Ghent in 1999. In the US, the Copper Mountain Conferences on Multigrid is held every two year since 1983. An essential contribution to development of the multigrid community is the MGNET website maintained by Craig C. Douglas: http://www.mgnet.org , this is a large communication platform and a resource on everything related to multigrid methods.

1.3 Grid structure

While, classical iterative methods use a single grid, multigrid methods use more than one grid. In one dimension, let $\Omega = [a,b]$ be a domain. A grid Ω_h is defined by:

$$\Omega_h = \left\{ x \in [a,b] : x = a + ih, i = 0,1,...n, h = \frac{b-a}{n} \right\} \qquad 1.3.1$$

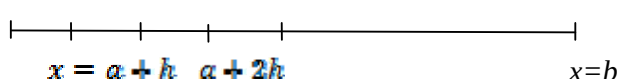
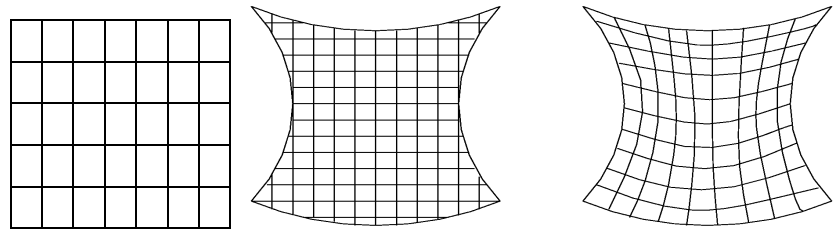


Figure 1.2

Domains in two dimensions may be rectangular, circular, or irregular. And the grid may be Cartesian grid, boundary-fitted curvilinear grid. However, only Cartesian grids will be considered.

In this thesis, only rectangular domains with Cartesian grid are considered.



(a) (b) (c)

Figure1.3 (a) Cartesian grid on rectangular domain
(b) Cartesian grid on irregular domain
(c) boundary-fitted curvilinear grid

If $\Omega = [a,b] \times [c,d]$ is rectangular domain then the grid is:

$$\Omega_{h,k} = \left\{ (x,y) \in \Omega : x = a + ih_1, y = c + jh_2 . h_1 = \frac{b-a}{n}, h_2 = \frac{d-c}{m} \right\} \qquad 1.3.2$$

Consider Ω_h as in equation 1.3.1. A coarser grid can be obtained by deleting all grid points with odd index i, then we obtain:

$$\Omega_H = \left\{ x \in [a,b] : x = a + iH, H = 2h, i = 0,1,....\frac{n}{2} \right\}.$$

The number of subintervals n need to be divisible by 2. Ω_H is called coarse grid, and Ω_h is called fine grid and the process is called coarsening. Coarsening can be done in a different way, by deleting every other grid point or reducing subintervals by 0.5. However, dividing by two is the most popular. Coarser grids $\Omega_{h_l}, l = 0,...,k$, can be obtained by repeating the process taking into account that the member of subintervals *n* must be in

the form 2^k. The coarsest grid is Ω_{h_0}, and the finest grid Ω_{h_k}. For simplicity we replace h_l by l.

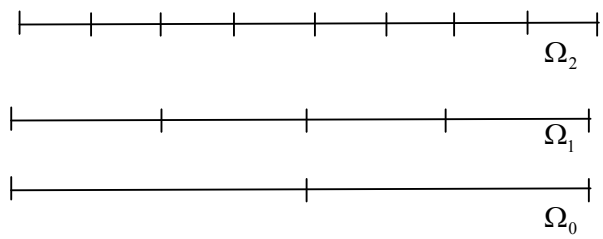


Figure 1.4 coarsening with *n=8* at finest grid

In two dimensions the coarse grid is:

$$\Omega_{H,K} = \left\{ (x,y) \in \Omega : x = a + iH_1, y = c + jH_2, i = 0,1,\ldots \frac{n}{2}, j = 0,1,\ldots \frac{m}{2} \right\}$$

n, m are in power of two, $H_1 = 2h_1, H_2 = 2h_2$

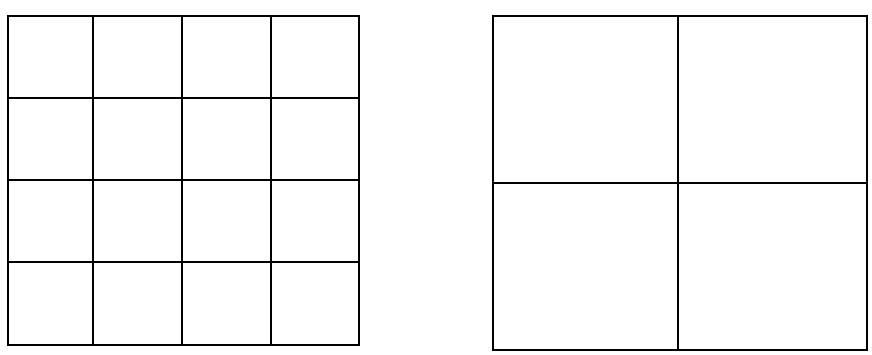


Figure 1.5 coarsening with *n=m=4* in the finer grid

1.4 Stencil notation

Using stencil notation is important in describing the moving between grids operators which will be studied later.

Let $u_h : \Omega_h \to \Re$, be a grid function. We can define an operator on the set of grid function by:

$[S_k]_h u_h(x) = \sum_k s_k u_h(x+kh)$, where $[S_k]_h = [. \quad . \quad s_{-1} \quad s_0 \quad s_1 \quad . \quad .]$

is the stencil.

In two dimension, the stencil is:

$$[S_{k_1,k_2}]_{h_1,h_2} = \begin{bmatrix} . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & s_{-1,1} & s_{0,1} & s_{1,1} & . & . \\ . & . & s_{-1,0} & s_{0,0} & s_{1,0} & . & . \\ . & . & s_{-1,-1} & s_{0,-1} & s_{1,-1} & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \end{bmatrix}$$

And the operator on the set of grid function is defined by :

$$[S_{k_1,k_2}]_{h_1,h_2} u_{h_1,h_2}(x,y) = \sum_{(k_1,k_2)} s_{k_1,k_2} u(x+k_1h_1, y+k_2h_2)$$

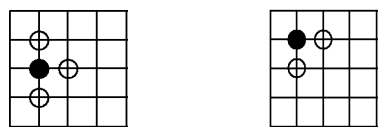
Assume that the only finite number of coefficients s_{k_1,k_2} are nonzero. Many of the stencils considered are five-point or compact nine-point stencils.

$$\begin{bmatrix} & s_{0,1} & \\ s_{-1,0} & s_{0,0} & s_{1,0} \\ & s_{0,-1} & \end{bmatrix}_h \qquad \begin{bmatrix} s_{-1,1} & s_{0,1} & s_{1,1} \\ s_{-1,0} & s_{0,0} & s_{1,0} \\ s_{-1,-1} & s_{0,-1} & s_{1,-1} \end{bmatrix}_h$$

Five-point stencil. Compact nine-point stencil

Near the boundary points the stencils may have to be modified on the domain. In Figure (1.6 a) the point is at the west boundary, so it is known.

The modified five point stencil is $\left[S_{k_1,k_2}\right]_h = \begin{bmatrix} & s_{0,1} & \\ 0 & s_{0,0} & s_{1,0} \\ & s_{0,-1} & \end{bmatrix}_h$. In Figure (1.6 b), the five point stencil for the northwest corner can be modified as $\left[S_{k_1,k_2}\right]_h = \begin{bmatrix} & 0 & \\ 0 & s_{0,0} & s_{1,0} \\ & s_{0,-1} & \end{bmatrix}_h$.



(a) **(b)**

Figure 1.6

Chapter two

Classical Iterative Methods

2.1 Introduction

2.2 Classical iterative methods

2.3 Convergence of classical iterative methods

Chapter 2

Classical Iterative Methods

2.1 Introduction

Direct methods for the linear system proceed through a finite number of steps and produce the exact solution to the level of rounding error. An iterative method starts with an initial approximation and produces a sequence of approximations (vectors) of the solution that is supposed to converge to the exact solution. The error in the approximate solution is due to the machine (rounding error) and to the number of terms in the sequence (iterations) used.

For large linear systems iterative methods often have advantages over direct methods in terms of speed and demands on computer memory. Accuracy is proportional to the number of iterations. When the sequence is convergent, iterations will suffice to produce an acceptable solution. This means higher accuracy needs more iterations. The number of iterations needed for a specific accuracy depends on the speed of convergence of the method. Another advantage of the classical iterative methods is that they are usually stable, and they will damp errors as process continues.

Classical iterative methods have the disadvantage, of smoothing errors. After few iterations, the error become smooth. and the result is slower convergence.

Consider the linear system:

$$A\mathbf{u} = \mathbf{f} \qquad 2.1.1$$

We will use $\mathbf{u} = (u_1, u_2, ..., u_n)$ to denote to the exact solution of this system, and $\mathbf{v} = (v_1, v_2, ..., v_n)$ to denote the approximation of the exact solution.

Definition 2.1.2:

Let $\mathbf{v}$ be the approximation of the exact solution $\mathbf{u}$ of the linear system $A\mathbf{u} = \mathbf{f}$ .The error in $\mathbf{v}$ is:

$$\mathbf{e} = \mathbf{u} - \mathbf{v} \qquad 2.1.2$$

The residual is:

$$\mathbf{r} = \mathbf{f} - A\mathbf{v} \qquad 2.1.3$$

As a result:

$$\mathbf{r} = A\mathbf{e} \ .$$

These two measures can be computed by any standard vector norm.

Definition 2.1.2[11] vector norm:

Let $\Re^n$ be a real vector space. A function $\|.\| : \Re^n \to \Re$ with the properties:

0. $\|\mathbf{u}\| \geq 0$

1. $\|\mathbf{u}\| = 0$ if and only if $\mathbf{u} = \mathbf{0}$

2. $\|\alpha\mathbf{u}\| = |\alpha|\|\mathbf{u}\|$ for any real scalar α

3. $\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\|$

For all $\mathbf{u}, \mathbf{v} \in \Re^n$, is called a vector norm. The most common norms are

$\| \mathbf{u} \|_p = \left(\sum_{i=1}^{n} |u_i|^p \right)^{\frac{1}{p}}$, $1 \leq p < \infty$ called the p-norm. If $p = \infty$ then

$\|\mathbf{u}\|_\infty = \max_{1 \leq i \leq n} |u_i|$ is called infinite norm. For $p = 2$, the norm is called the Euclidean norm.

An iterative method generate a sequence of approximations $\{\mathbf{u}^m\}_{m=0}^{\infty}$ using the iteration:

$$\mathbf{u}^{m+1} = T\mathbf{u}^m + \mathbf{C} \qquad 2.1.4$$

where $\mathbf{u}^m$ is the approximation solution after *m* iterations and T is called the iteration matrix of the iterative method. Different iterative methods have different iteration matrices. Convergence of an iterative method depends on the iteration matrix T for the method.

2.2 Basic iterative methods

We will consider the following three most popular classical iterative methods:

- Jacobi method
- Gauss-Seidel method
- SOR method

Consider the linear system $A\mathbf{u} = \mathbf{f}$. If we can split A as $A = M - N$ with M nonsingular, then the linear system is:

$$(M - N)\mathbf{u} = \mathbf{f}$$

$$M\mathbf{u} = N\mathbf{u} + \mathbf{f}$$

and the iterative method is:

$$M\mathbf{u}^{m+1} = N\mathbf{u}^{m} + \mathbf{f} \qquad m = 0,1,..$$

so

$$\mathbf{u}^{m+1} = T\mathbf{u}^{m} + \mathbf{C} \qquad m = 0,1,.. \qquad 2.2.1$$

where $T = M^{-1}N$ and $\mathbf{C} = M^{-1}\mathbf{f}$.

Now, consider the splitting $A = D - L - U$ where D denotes the diagonal part of the matrix A. The matrices $-L$ *and* $-U$ are strictly lower and upper parts of A, respectively. Based on this splitting, many choices for M and N are possible leading to different iterative methods.

Jacobi iterative method uses the splitting $M = D,$ and $N = L + U$. The iteration is given by:

$$\mathbf{u}^{m+1} = \left[D^{-1}(L+U) \right] \mathbf{u}^{m} + D^{-1}\mathbf{f} \quad m = 0,1,2.. \qquad 2.2.2$$

$$= T_j \mathbf{u}^m + \mathbf{C}_j$$

This is the matrix form of the Jacobi method, where the iteration matrix of Jacobi method is:

$$T_j = D^{-1}(L+U)$$

and

$$\mathbf{C}_j = D^{-1}\mathbf{f}$$

This formula is important in the study of the convergence of the Jacobi method. However, computationally, the iteration is carried out simply by solving equation *i* for the unknown u_i:

$$u_i^{m+1} = \frac{1}{a_{ii}} \left[f_i - \sum a_{ij} u_j^m \right] \quad i = 1,\ldots,n \qquad 2.2.3$$

Jacobi method starts with initial approximation $\mathbf{u}^0$ to compute a new approximation $\mathbf{u}^1$ using equation 2.2.3, then $\mathbf{u}^1$ is used to compute $\mathbf{u}^2$, and the process is repeated until a maximum number of iterations, or a given tolerance (maximum error norm allowed) is reached.

The actual error $\mathbf{u} - \mathbf{u}^m$ in the mth approximation $\mathbf{u}^m$ is not computable since the exact solution $\mathbf{u}$ is unknown. However, the estimated error $\mathbf{u}^{m+1} - \mathbf{u}^m$ can be easily computed.

The error norm $\| \mathbf{u}^{m+1} - \mathbf{u}^m \|$, for any norm, is compared with a given

tolerance to stop the iteration process.

A variation of Jacobi iterative method is the damped (weighted) Jacobi iterative method. The iteration of the damped Jacobi iterative method is given by:

$$\mathbf{u}^{m+1} = T_{dj}\mathbf{u}^{m} + \omega D^{-1}\mathbf{f}\,,\ 0 < \omega < 1$$

where:

$$T_{dj} = \left[(1-\omega)I + \omega T_j\right]$$

Gauss-Seidel method is similar to Jacobi method but it uses the most recent values to update the unknowns. The iteration is:

$$u_i^{m+1} = \frac{1}{a_{ii}}\left[f_i - \sum_{j=1}^{i-1} a_{ij}u_j^{m+1} - \sum_{j=i+1}^{n} a_{ij}u_j^{m}\right]\ ,\ i = 1,\ldots,n \qquad 2.2.4$$

Splitting $M=D-L$ and $N=U$ gives:

$$(D-L)\mathbf{u}^{m+1} = U\mathbf{u}^{m} + \mathbf{f}$$

$$\begin{aligned}\mathbf{u}^{m+1} &= \left[(D-L)^{-1}U\right]\mathbf{u}^{m} + (D-L)^{-1}\mathbf{f} \\ &= T_g\mathbf{u}^{m} + \mathbf{I}_g\end{aligned} \qquad 2.2.5$$

where the iteration matrix for Gauss-Siedel method is:

$$T_g = (D-L)^{-1}U$$

and

$$\mathbf{I}_g = (D-L)^{-1}\mathbf{f}$$

This is the matrix form of the Gauss-Seidel method.

The idea of Jacobi and Gauss-Seidel methods is to generate a sequence of approximations that converges to the solution of the system. A corresponding sequence of residuals converges to the zero vector.

Let $\mathbf{u}_i^{m+1} = \left(u_1^{m+1},....,u_{i-1}^{m+1},u_i^m,....,u_n^m\right)^T$ be the approximate solution vector after *m+1* iterations. With residual $\mathbf{r}_i^{m+1} = \left(r_{1i}^{m+1},.....r_{ni}^{m+1}\right)^T$. Gauss-Seidel method can be characterized by choosing u_i^{m+1} that satisfy

$$u_i^{m+1} = u_i^m + \frac{r_{ii}^{m+1}}{a_{ii}} \qquad 2.2.6$$

Gauss-Seidel method can be modified by taking the form of a weighted average of the last two iterations as:

$$u_i^{m+1} = u_i^m + \omega\frac{r_{ii}^{m+1}}{a_{ii}} \qquad 2.2.7$$

Choices of positive ω will leads to faster convergence. If $0 < \omega < 1$,the method is called *under relaxation method*, and if $\omega > 1$ the method is called *over relaxation method*. These methods are used to accelerate the convergence for the systems that are convergent by Gauss-Seidel technique. This method is called successive over relaxation (SOR), and is given by:

$$u_i^{m+1} = (1-\omega)u_i^m + \frac{\omega}{a_{ii}}\left(f_i - \sum_{j=1}^{i-1} a_{ij}u_j^{m+1} - \sum_{j=i+1}^{n} a_{ij}u_j^m\right) \qquad 2.2.8$$

The matrix form of the SOR method which is important in theoretical analysis is given by:

$$\mathbf{u}^{m+1} = \left[(D - \omega L)^{-1} [(1-\omega)D - \omega U] \right] \mathbf{u}^{m} + \left[(D - \omega L)^{-1} \omega \right] \mathbf{f} \qquad 2.2.9$$

$$= T_{SOR} \mathbf{u}^{m} + \mathbf{I}_{SOR}$$

Where

$$T_{SOR} = \left[(D - \omega L)^{-1} [(1-\omega)D - \omega U] \right]$$

is the iteration matrix for SOR method and

$$\mathbf{I}_{SOR} = (D - \omega L)^{-1} \omega \mathbf{f}$$

Note that if $\omega = 1$ the SOR method simplifies to the Gauss-Seidel method.

2.3 Convergence of classical iterative methods

Starting with an initial vector, an iterative method generates a sequence of vectors that approximates of the solution of the given linear system. The sequence may converge or diverge. Convergence and divergence of the method depends on the nature of the coefficient matrix.

In this section we will perform convergence analysis for the three iterative methods discussed in the previous section. To study the convergence of these methods we need some theorems and definitions.

Definition 2.3[11] matrix norm:

$\|.\|$ is a matrix norm on n$\times$n matrices if:

1. $\|A\| \geq 0$
2. $\|A\| = 0$ if and only if $A = 0$

3. $\|\alpha A\| = |\alpha|\|A\|$ for any real scalar α

4. $\|A+B\| \le \|A\| + \|B\|$

For $n \times n$ matrix A, some of known matrix norms are:

- **The operator norm** $\|A\| = \max\limits_{\substack{u \ne 0 \\ u \in \Re^n}} \dfrac{\|Au\|}{\|u\|}$

- **The infinite norm** $\|A\|_\infty = \max\limits_{u \ne 0} \dfrac{\|Au\|_\infty}{\|u\|_\infty} = \max\limits_{1 \le i \le n} \sum\limits_{j=1}^{n} |a_{ij}|$

- **The Euclidean norm** $\|A\| = \sqrt{\sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n} |a_{ij}|^2}$

Definition 2.4[16] spectral radius:

The spectral radius of a square matrix T is $\rho(T) = \max|\lambda|$ where the maximum is taken over all eigenvalues λ of T .

Theorem 2.1 [11]:

For each norm and each matrix we have that $\rho(T) \le \|T\|$, conversely, for each matrix T and each $\varepsilon > 0$, there exists a norm such that:

$$\|T\| \le \rho(T) + \varepsilon .$$

Proof:

Let $|\lambda| = \rho(T)$ and u be the eigenvector for λ then:

$$\|T\| = \max_{v \ne 0} \frac{\|Tv\|}{\|v\|} \ge \frac{\|Tu\|}{\|u\|} = \frac{\|\lambda u\|}{\|u\|} = |\lambda| .$$

To construct $\|.\|$ such that $\|T\| \le \rho(T) + \varepsilon$, let

$S^{-1}TS = J$ be Jordan form and $D_\varepsilon =$ diagonal $(1\ ,\varepsilon,\varepsilon^2,.....,\varepsilon^{n-1})$ then

$$(SD_\varepsilon)^{-1}T(SD_\varepsilon) = D_\varepsilon^{-1}JD_\varepsilon = \begin{bmatrix} \lambda_1 & \varepsilon & 0 & . & . & . & . & . & . & . & 0 \\ 0 & \lambda_1 & . & . & . & . & . & . & . & . & . \\ . & . & . & \varepsilon & . & . & . & . & . & . & . \\ . & . & 0 & \lambda_1 & . & . & . & . & . & . & . \\ . & . & . & . & . & \varepsilon & . & . & . & . & . \\ . & . & . & . & . & \lambda_k & . & . & . & . & . \\ . & . & . & . & . & . & . & \varepsilon & . & . & . \\ . & . & . & . & . & . & . & \lambda_m & \varepsilon & . & . \\ . & . & . & . & . & . & . & 0 & \lambda_m & . & 0 \\ . & . & . & . & . & . & . & . & . & . & \varepsilon \\ 0 & . & . & . & . & . & . & . & . & 0 & \lambda_m \end{bmatrix}$$

which mean a Jordan form with ε's above the diagonal. If we use the vector norm

$$\|u\| = \left\|(SD_\varepsilon)^{-1}u\right\|_\infty \quad \ldots\ldots\ (*)$$

to generate the operator norm, then

$$\begin{aligned} \|T\| &= \max_{u\neq 0}\frac{\|Tu\|}{\|u\|} = \max_{u\neq 0}\frac{\left\|(SD_\varepsilon)^{-1}Tu\right\|_\infty}{\left\|(SD_\varepsilon)^{-1}u\right\|_\infty} \\ &= \max_{v\neq 0}\frac{\left\|(SD_\varepsilon)^{-1}T(SD_\varepsilon)v\right\|_\infty}{\|v\|_\infty} \\ &= \left\|(SD_\varepsilon)^{-1}T(SD_\varepsilon)\right\|_\infty \\ &\leq \rho(T)+\varepsilon \end{aligned}$$

Theorem 2.2[11]:

The successive approximation $\mathbf{u}^{m+1} = T\mathbf{u}^m + \mathbf{C}$, $m = 0,1,2....$ converges if and only if $\rho(T) < 1$.

Proof:

Suppose the method converges and $\rho(T) \geq 1$ then there exist an eigenvalue λ of T with $|\lambda| \geq 1$. Let $\mathbf{u}^0 - \mathbf{u}$ be an associated eigenvector then:

$$\mathbf{u}^{m+1} - \mathbf{u} = T\left(\mathbf{u}^m - \mathbf{u}\right) = \ldots\ldots = T^{m+1}\left(\mathbf{u}^0 - \mathbf{u}\right) = \lambda^{m+1}\left(\mathbf{u}^0 - \mathbf{u}\right)$$

which is not approach to zero, and this contradicts the assumption. Conversely, suppose that $\rho(T) < 1$, then $\|T\| < 1$ from previous theorem and $\mathbf{u}^{m+1} - \mathbf{u} = T\left(\mathbf{u}^m - \mathbf{u}\right)$ we have:

$$\left\|\mathbf{u}^{m+1} - \mathbf{u}\right\| \leq \|T\|\left\|\mathbf{u}^m - \mathbf{u}\right\| \leq \|T\|^{m+1}\left\|\mathbf{u}^0 - \mathbf{u}\right\|$$

which converges to zero.

Definition 2.5[16]:

The matrix A of dimension $n \times n$ is strictly diagonally dominant if $|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|$ for each $i = 1,...n$

Theorem 2.3 [11]:

Consider the linear system $A\mathbf{u} = \mathbf{f}$. If A is strictly diagonally dominant, then the Jacobi method converges.

Proof:

The iteration matrix of the Jacobi method is:

$$T_j = D^{-1}(L+U) = \begin{bmatrix} 0 & \frac{-a_{12}}{a_{11}} & . & . & . & \frac{-a_{1n}}{a_{11}} \\ \frac{-a_{21}}{a_{22}} & 0 & . & . & . & \frac{-a_{2n}}{a_{22}} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ \frac{-a_{n1}}{a_{nn}} & \frac{-a_{n2}}{a_{nn}} & . & . & . & 0 \end{bmatrix}$$

Since $\left\| D^{-1}(L+U) \right\|_\infty = \max_{1 \le n \le j} \sum_{\substack{k=1 \\ k \neq j}}^{n} \left| \frac{a_{jk}}{a_{jj}} \right| < 1$, $\rho(T_j) < 1$ so the Jacobi method converges.

Theorem 2.4[6]:

If A is strictly diagonally dominant then the Gauss-Seidel method converges.

Proof:

Let λ be any eigenvalue of the iteration matrix of the Gauss-Seidel method $T_g = (D-L)^{-1}U$ and let $\mathbf{u}$ be the corresponding eigenvector. Without loss of generality, assume $\|\mathbf{u}\|_\infty = 1$

we have:

$$\left((D-L)^{-1}U\right)\mathbf{u} = \lambda\mathbf{u},$$

$$U\mathbf{u} = D\lambda\mathbf{u} - L\lambda\mathbf{u}$$

which mean:

$$-\sum_{j=i+1}^{n} a_{ij}u_j = a_{ii}\lambda u_i - \lambda\sum_{j=1}^{i-1} a_{ij}u_j \ , \ 1 \le i \le n.$$

So

$$\lambda a_{ii}u_i = -\lambda\sum_{j=1}^{i-1} a_{ij}u_j - \sum_{j=i+1}^{n} a_{ij}u_j \ , \ 1 \le i \le n,$$

Now select an index i such that $|u_i| = 1 \ge |u_j|$ for all j then:

$$|\lambda||a_{ii}| \le |\lambda|\sum_{j=1}^{i-1}|a_{ij}| + \sum_{j=i+1}^{n}|a_{ij}| \ ,$$

solving for λ and using the diagonally dominance of A, we get:

$$|\lambda| \le \left(\sum_{j=i+1}^{n}|a_{ij}|\right)\left(|a_{ii}| - \sum_{j=1}^{i-1}|a_{ij}|\right)^{-1} < 1$$

then $\rho(T_g) < 1$ so Gauss-Seidel converges .

The following theorem gives conditions on the convergence of the SOR

method.

Theorem 2.5[11] kahan:

For arbitrary $n \times n$ matrix A, $\rho(T_{SOR}) \geq |\omega - 1|$ therefore $\rho(T_{SOR}) < 1$ only if $0 < \omega < 2$ where T_{SOR} is the iteration matrix for the SOR method

Proof:

Write the characteristic polynomial of T_{SOR} as:

$$\varphi(\lambda) = \det(\lambda I - T_{SOR}) = \det\left(\left(I - \omega D^{-1} L\right)\left(\lambda I - T_{SOR}\right)\right).$$

Because $I - \omega D^{-1} L$ is lower triangular matrix with 1 on the diagonal ,

$\det\left(I - \omega D^{-1} L\right) = 1$

then

$$\varphi(\lambda) = \det\left(DD^{-1}\left(I - \omega D^{-1} L\right)\left[\lambda I - (D - \omega L)^{-1}\left[(1-\omega)D + \omega U\right]\right]\right)$$

$$= \det\left((\lambda + \omega - 1)I - \omega\lambda D^{-1} L - \omega D^{-1} U\right)$$

Since $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of T_{SOR}, the constant coefficient of the characteristic polynomial:

$$\varphi(0) = \pm \prod_{i=1}^{n} \lambda_i = \pm \det\left((\omega - 1)I\right) = \pm(\omega - 1)^n,$$

Now:

$$\max_{1 \leq i \leq n} |\lambda_i| \geq |\omega - 1|$$

which implies:

$$\rho(T_{SOR}) \geq |\omega - 1|$$

for convergences we need:

$$\rho(T_{SOR}) < 1$$

then:

$$|\omega - 1| < 1$$

which leads to:

$$0 < \omega < 2 .$$

Chapter three

Multigrid Methods

3.1 Introduction

3.2 Two-grid methods

3.3 Moving between grids: restriction and prolongation

3.4 The multigrid cycle

3.5 The full multigrid

3.6 Multigrid iteration operator

Chapter 3

Multigrid methods

3.1 Introduction

Jacobi and Gauss-Seidel methods are characterized by their slow rate of convergence [1]. They are efficient in smoothing the error but not in reducing it. By smoothing, we mean damping the error components with short wave length, which is done after very few iterations (relaxation sweeps). To reduce smooth error, it takes many relaxation sweeps, which means slow rate of convergence. If we analyze this error into components of wavelengths, the error will have components of many different wavelengths, there will be short wavelength error components and long wavelength error components. For short wavelength error components, Jacobi and Gauss-Seidel methods provide rapid damping leaving behind longer wavelength error components (smooth). Long wavelength error components (smooth) are responsible for the slow convergence. The basic idea behind multigrid methods is to reduce long wavelength error components.

The rate of convergence of classical iterative methods can be improved with multigrid methods. A multigrid method begins with Jacobi or Gauss-Seidel iterations, for the one job that they do well, removing short

wavelength error components to leave a smooth error. The central idea is to move to a coarse grid where transferred error is not smooth.

We illustrate this method using the simplest case a two grid method.

3.2 Two-grid method

We can introduce the two-grid method by starting from the general iteration based on approximate solution of the defect (residual) equation. If we discretize the PDE on uniform grid with mesh size h, we can write the resulting set of linear equations as:

$$A_h \mathbf{u}_h = \mathbf{f}_h \qquad 3.2.1$$

Let u_h be the exact solution of equation 3.2.1. let $\mathbf{u}_h^m$ be the approximate solution after m relaxation sweeps with error:

$$\mathbf{e}_h^m = \mathbf{u}_h - \mathbf{u}_h^m$$

and residual:

$$\mathbf{r}_h^m = \mathbf{f}_h - A_h \mathbf{u}_h^m$$

This leads to the following defect equation:

$$\mathbf{r}_h^m = A_h \mathbf{e}_h^m \qquad 3.2.2$$

If we approximate A_h by any simpler operator $\hat{A}_h$ where $\hat{A}_h^{-1}$ exists, for example $\hat{A}_h$ is the diagonal part of A_h in Jacobi iteration, and the lower triangular part of A_h for Gauss-Seidel iteration. Then the solution $\hat{\mathbf{e}}_h^m$ of the defect equation

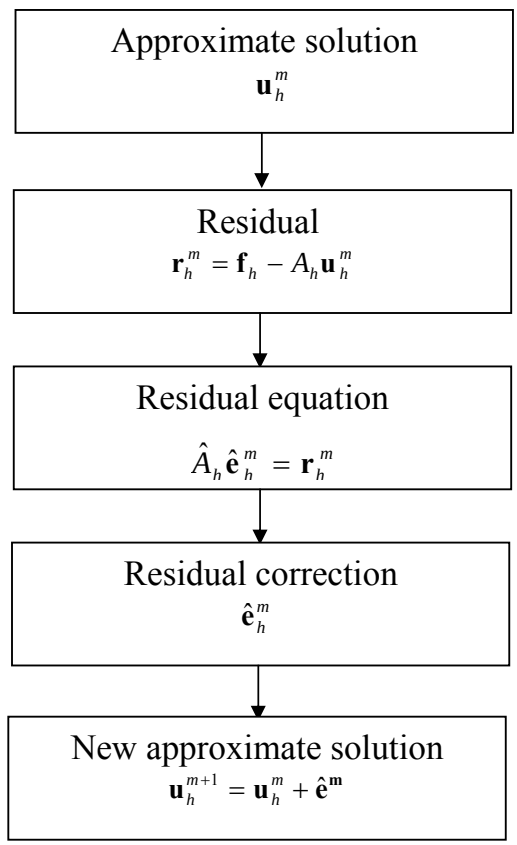
$$\hat{A}_h \hat{\mathbf{e}}_h^m = \mathbf{r}_h^m$$

is added to the old approximation $\mathbf{u}_h^m$ giving a new approximation $\mathbf{u}_h^{m+1}$.

This means:

$$\mathbf{u}_h^{m+1} = \mathbf{u}_h^m + \hat{\mathbf{e}}_h^m$$

We can describe the previous steps by the following flowchart:



The iteration operator of this method is given by:

$$M_h = I - \hat{A}_h^{-1} A_h$$

Then we have:

$$\mathbf{u}_h^{m+1} = M_h \mathbf{u}_h^m + \hat{A}_h^{-1} \mathbf{f}_h$$

Another type of approximation for A_h is to coarsify rather than simplify. i.e. we form a suitable approximation A_H of A_h on coarse grid with mesh size $H = 2h$, and then the defect equation 3.2.2 is replaced by:

$$A_H \mathbf{e}_H^m = \mathbf{r}_H^m \qquad 3.2.3$$

Because A_H has smaller order, equation 3.2.3 is easier to solve than equation 3.2.2. The residual $\mathbf{r}_H^m$ and the error $\mathbf{e}_H^m$ are grid functions on the coarser grid Ω_H, therefore two linear transfer operators to move between grids are needed. The first operator is a restriction from the fine grid to the coarse grid:

$$I_h^H : g(\Omega_h) \rightarrow g(\Omega_H)$$

This operator is used to transfer the residual $\mathbf{r}_h^m$ from Ω_h to Ω_H (i.e. $\mathbf{r}_H^m = I_h^H \mathbf{r}_h^m$).

The second operator is a prolongation from the coarse grid to the fine grid:

$$I_H^h : g(\Omega_H) \rightarrow g(\Omega_h)$$

This operator is used to transfer the error $\mathbf{e}_H^m$ from Ω_H to Ω_h (i.e. $\mathbf{e}_h^m = I_H^h \mathbf{e}_H^h$).

Finally, the new approximation $\mathbf{u}_h^{m+1}$ is computed by adding a coarse grid correction $\mathbf{e}_h^m = I_H^h \mathbf{e}_H^h$ to $\mathbf{u}_h^m$ replacing a new relaxation sweep on the fine grid by a new and cheaper one on the coarse grid. This process is called coarse grid correction, and it can be described as follows:

- Compute the residual: $\mathbf{r}_h^m = \mathbf{f}_h - A_h \mathbf{u}_h^m$.
- Transfer the residual to the coarse grid: $\mathbf{r}_H^m = I_h^H \mathbf{r}_h^m$.
- Solve the residual equation: $A_H \, \mathbf{e}_H^m = \mathbf{r}_H^m$.
- Transfer the error $\mathbf{e}_H^m$ to the fine grid: $\mathbf{e}_h^m = I_H^h \mathbf{e}_H^h$.
- Compute a new approximation: $\mathbf{u}_h^{m+1} = \mathbf{u}_h^m + \mathbf{e}_h^m$.

The high frequency components can be reduced by smoothing on the fine grid using iterative methods like Jacobi and Gauss-Seidel. The low frequency components of the error are effectively reduced by coarse grid correction procedure. But the high frequency components of the error are not even representable on the coarse grid see Figure [3.1] and so cannot be reduced to zero. This leads us to combine the two processes of smoothing and the coarse grid correction to get the two grid method.

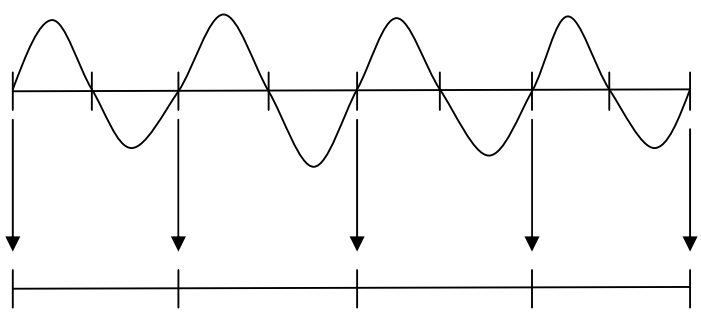


Figure [3.1]: high frequency components errors are not representable (not visible) on the coarse grid.

Each iteration step of a two-grid method consists of presmoothing, coarse grid correction and postsmoothing part as follows:

- Pre-smoothing: compute $\overline{\mathbf{u}}_h^m$ by applying $v_1 \geq 0$ steps of a given

smoothing procedure to $\mathbf{u}_h^m$.

- Coarse grid correction: use $\overline{\mathbf{u}}_h^m$ to get $\mathbf{u}_h^{m(new)}$.
- Post-smoothing: compute $\mathbf{u}_h^{m+1}$ by applying $v_2 \geq 0$ steps of the given smoothing procedure to $\mathbf{u}_h^{m(new)}$.

Two-grid procedure can be presented by:

$$\begin{array}{ccc}
\mathbf{u}_h^m \xrightarrow{smoothing} \overline{\mathbf{u}}_h^m \rightarrow \overline{\mathbf{r}}_h^m = \mathbf{f}_h - A_h \overline{\mathbf{u}}_h^m & & \mathbf{e}_h^m \rightarrow \overline{\mathbf{u}}_h^m + \mathbf{e}_h^m \xrightarrow{smoothing} \mathbf{u}_h^{m+1} \\
I_h^H \downarrow & & \uparrow I_H^h \\
\overline{\mathbf{r}}_H^m \rightarrow A_H \mathbf{e}_H^m = \overline{\mathbf{r}}_H^m & &
\end{array}$$

But two-grid methods are of little practical significance due to the still large complexity of the coarse grid problem. However, they serve as the basis for the multigrid methods. Instead of solving the coarse grid residual equation exactly, we can get an approximate solution of it by introducing an even coarser grid, and using the two-grid iteration method. This idea can be applied using coarser and coarser grids, down to some coarsest grid where any solution method can be used.

3.3 Moving between grids: restriction and prolongation.

In multigrid methods, it is necessary to move approximations, residual and errors between grids. There are two types of grid transfer: restriction and prolongation. Restriction transfer values from fine grid to the next

coarse grid. Prolongation transfer values from the coarse grid to the next fine grid.

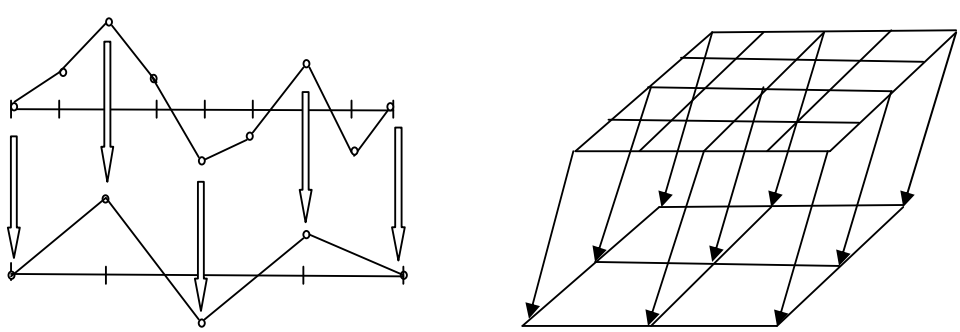
The choice of restriction and prolongation operators I_h^H and I_H^h for intergrid transfer of grid values depends on the choice of the coarse grid. In this thesis, only standard coarsening will be considered.

3.3.1 Restriction

The simplest restriction operator is the injection operator:

$$\begin{aligned}\mathbf{r}_H(p) &= I_h^H \mathbf{r}_h(p) \\ &= \mathbf{r}_h(p)\end{aligned}, \qquad p \in \Omega_H \subset \Omega_h$$

This identifies grid function at coarse grid points by the corresponding grid values at fine grid points as in the following figure:



(a) (b)

Figure [3.2]: (a) Restriction by injection operator in one dimension.

(b) Restriction by injection operator in two dimensions.

Another restriction operator is the Full Weighting (FW) operator. This operator can be illustrated by the following Figure:

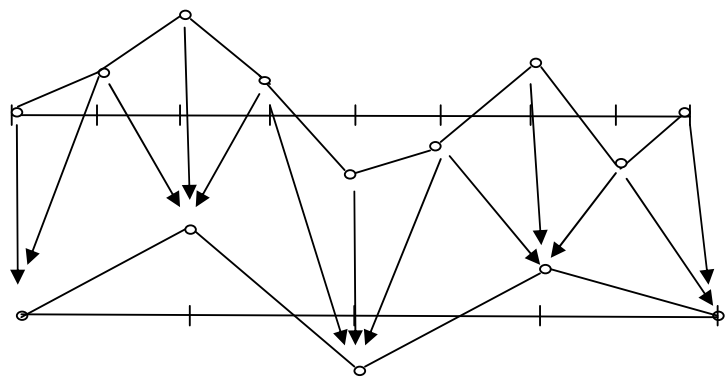


Figure [3.3]: restriction by full weighting operator in one dimension

This restriction operator is represented by stencil notation as:

$$\begin{bmatrix} \frac{1}{4} & \frac{2}{4} & \frac{1}{4} \end{bmatrix}_h^{2h}$$

i.e. $\mathbf{r}_h(x) = I_h^{2h}\mathbf{r}_h(x) = \frac{1}{4}\left(r_h(x-h) + 2r_h(x) + r_h(x+h)\right), x \in \Omega_{2h}$.

But if x is the left boundary point then the stencil is modified by:

$$\begin{bmatrix} 0 & \frac{2}{4} & \frac{1}{4} \end{bmatrix}_h^{2h}$$

If x is the right boundary point then the stencil is:

$$\begin{bmatrix} \frac{1}{4} & \frac{2}{4} & 0 \end{bmatrix}_h^{2h}$$

In two dimensions, the full weighting operator is given by:

$$\begin{bmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{bmatrix}_h^{2h}$$

which means:

$$
\begin{aligned}
r_{2h}(x,y) &= I_h^{2h} r_h(x,y) \\
&= \frac{1}{16}\begin{bmatrix} 4r_h(x,y)+2r_h(x+h,y)+2r_h(x-h,y)+2r_h(x,y+h)+2r_h(x,y-h) \\ +r_h(x+h,y+h)+r_h(x+h,y-h)+r_h(x-h,y+h)+r_h(x-h,y-h) \end{bmatrix}
\end{aligned}
$$

where $(x,y) \in \Omega_{2h}$.

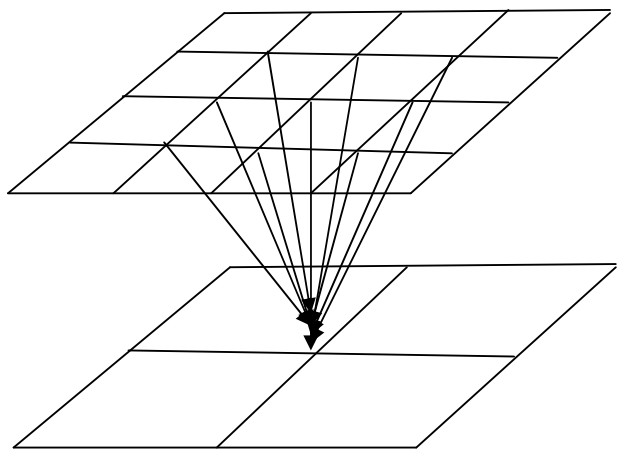


Figure [**3.4**]: restriction by full weighting operator in two dimensions.

If x is a boundary point, full weighting operator is modified as follows:

For a north-west corner, the FW stencil is:

$$
\begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{4}{16} & \frac{2}{16} \\ 0 & \frac{2}{16} & \frac{1}{16} \end{bmatrix}_h^{2h}
$$

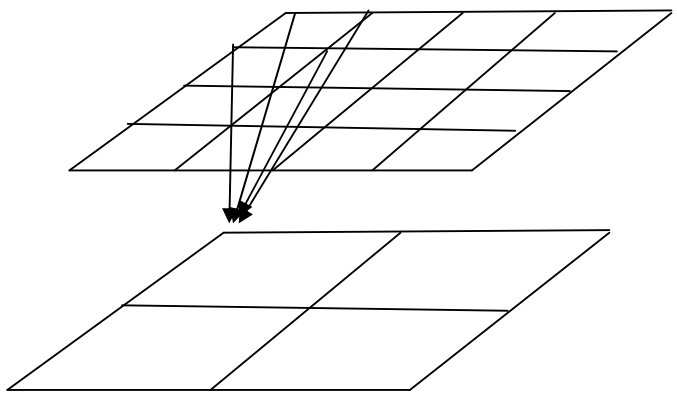


Figure [3.5] : restriction by full weighting operator for corner point

If x is a west boundary point, then the FW stencil is:

$$\begin{bmatrix} 0 & \frac{2}{16} & \frac{1}{16} \\ 0 & \frac{4}{16} & \frac{2}{16} \\ 0 & \frac{2}{16} & \frac{1}{16} \end{bmatrix}_h^{2h}$$

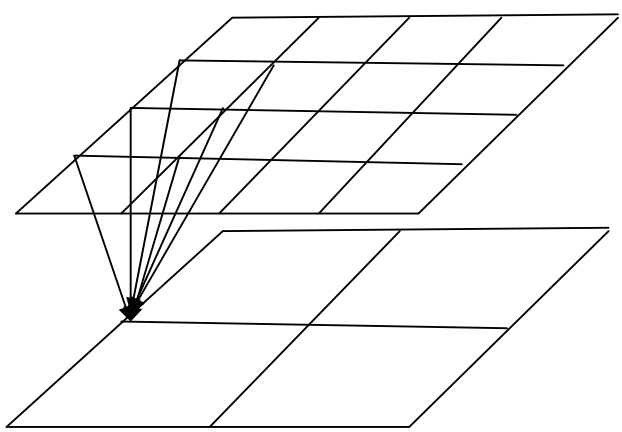


Figure [3.6]: restriction by full weighting operator for boundary point.

Another operator is the Half Weighting (HW) operator. It is a five-weighted average. In stencil notation, the HW reads:

$$\begin{bmatrix} 0 & \frac{1}{8} & 0 \\ \frac{1}{8} & \frac{4}{8} & \frac{1}{8} \\ 0 & \frac{1}{8} & 0 \end{bmatrix}_h^{2h}$$

This means:

$$r_{2h}(x,y) = \frac{1}{8}\left[4r_h(x,y) + r_h(x+h,y) + r_h(x-h,y) + r_h(x,y+h) + r_h(x,y-h)\right]$$

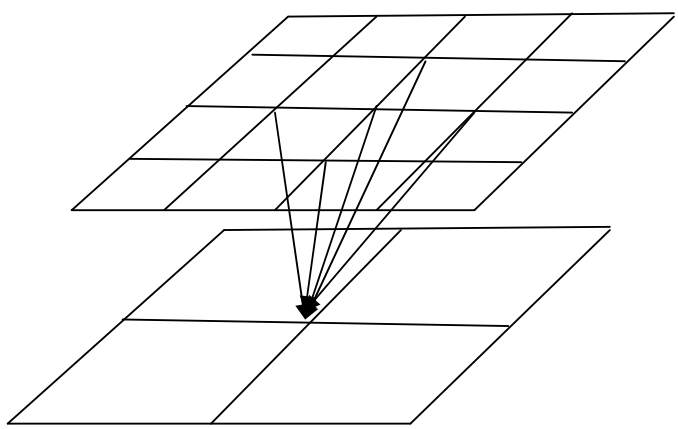


Figure [3.7]: restriction by half weighting operator for an interior point

3.3.2 Prolongation

The prolongation operator maps coarse grid values onto fine grid values. In one dimension, the values at points on the coarse grid are copied to the corresponding fine grid points. The remaining values at the fine grid points are computed by taking the averages of the values of the left and the right coarse grid points.

The linear prolongation is defined as:

$$\hat{e}_h(x) = I_{2h}^{h}\hat{e}_{2h}(x)$$

$$= \begin{cases} \hat{e}_{2h}(x) & \text{, for coarse grid points} \\ \frac{1}{2}\left[\hat{e}_{2h}(x-h) + \hat{e}_{2h}(x+h)\right] & \text{, for points locates between two coarse grid points.} \end{cases}$$

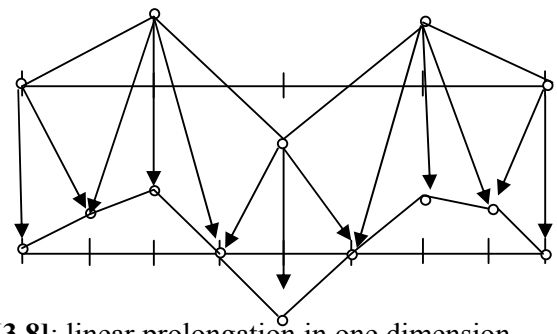


Figure [3.8]: linear prolongation in one dimension.

In two dimensions, the most used prolongation is bilinear, which is given by:

$$\hat{e}_h(x,y) = I_{2h}^{h}\hat{e}_{2h}(x,y)$$

$$= \begin{cases} \hat{e}_{2h}(x,y) & \text{, for coarse grid points} \\ \frac{1}{2}\left[\hat{e}_{2h}(x,y+h) + \hat{e}_{2h}(x,y-h)\right] & \text{for points located between two coarse grid points vertically} \\ \frac{1}{2}\left[\hat{e}_{2h}(x+h,y) + \hat{e}_{2h}(x-h,y)\right] & \text{for points located between two coarse grid points horizentally} \\ \frac{1}{4}\begin{bmatrix} \hat{e}_{2h}(x+h,y+h) + \hat{e}_{2h}(x+h,y-h) + \\ \hat{e}_{2h}(x-h,y+h) + \hat{e}_{2h}(x-h,y-h) \end{bmatrix} & \text{for points located in the center of square whose vertex are coarse grid points.} \end{cases}$$

This can be illustrated by the following figure:

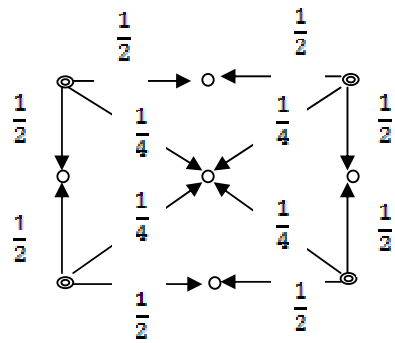
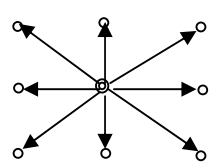


Figure [3.9]: bilinear prolongation operator: (◎) coarse grid point, (∘) fine grid point

In stencil notation we write the bilinear interpolation operator I_{2h}^{h} as:

$$I_{2h}^{h} = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_{2h}^{h}$$

The brackets are reversed, since the stencil entries correspond to weights in a distribution process as:



Another prolongation operator is a linear operator which takes place in triangles as illustrated in the following figure:

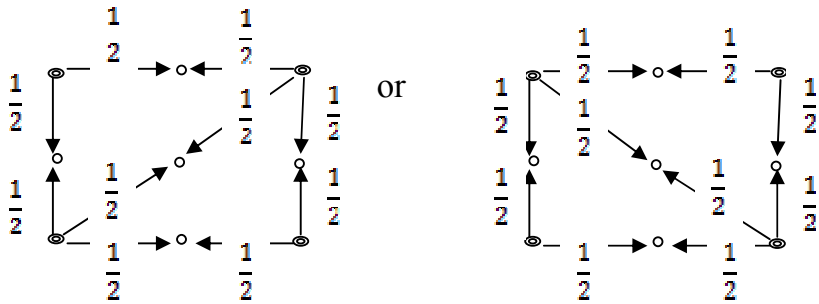


Figure [3.10]: linear prolongation operator, (⊗) coarse grid points and (○) fine grid points

This linear prolongation is given by:

$$\hat{e}_h(x,y) = I_{2h}^{h}\hat{e}_{2h}(x,y)$$

$$= \begin{cases} \hat{e}_{2h}(x,y) & \text{, for coarse grid points} \\ \frac{1}{2}\left[\hat{e}_{2h}(x,y+h)+\hat{e}_{2h}(x,y-h)\right] & \text{for points located between two coarse grid points vertically.} \\ \frac{1}{2}\left[\hat{e}_{2h}(x+h,y)+\hat{e}_{2h}(x-h,y)\right] & \text{for points located between two coarse grid points horizentally.} \\ \frac{1}{2}\left[\hat{e}_{2h}(x+h,y+h)+\hat{e}_{2h}(x-h,y-h)\right] \text{ or} & \\ \frac{1}{2}\left[\hat{e}_{2h}(x+h,y-h)+\hat{e}_{2h}(x-h,y+h)\right] & \text{for points locate in the center of square whose vertex are coarse grid points.} \end{cases}$$

3.4 The Multigrid cycles.

A two grid cycle consists of three steps: presmoothing, coarse grid correction and postsmoothing. A Multigrid cycle can be obtained by performing a number of two grid cycles, say γ, at each intermediate stage to obtain a better approximation:

$$A_H e_H^m = r_H^m$$

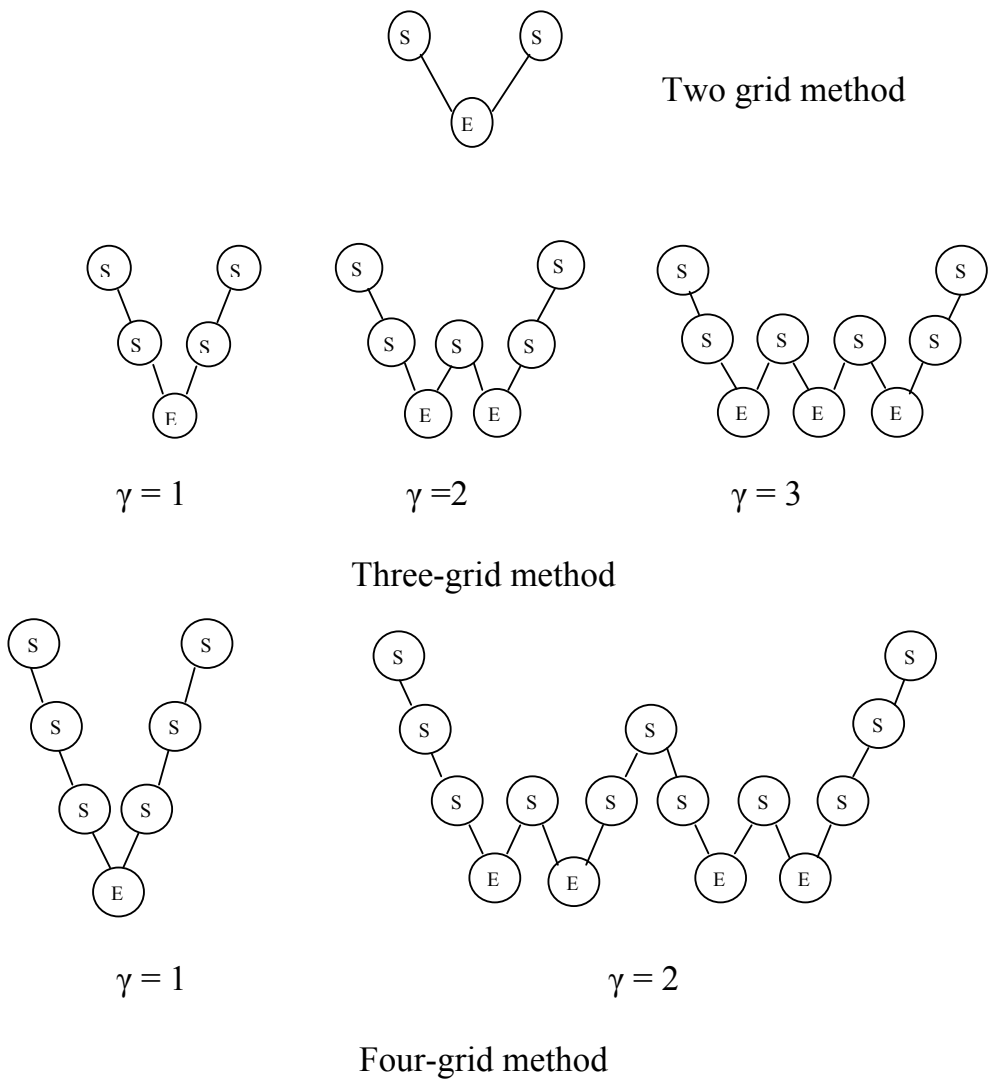


Figure [3.11]: structure of one multigrid cycle for different grids and different values of γ, where (S) for smoothing, (E) for exact solution, \ for fine to coarse, / for coarse to fine transfer.

The two cases $\gamma = 1$, and $\gamma = 2$ are particularly interesting. In the case $\gamma = 1$, the cycle is called *V*-cycle, and if $\gamma = 2$, then the cycle is called *W*-cycle , and the number γ is called the cycle index.

We now describe a multigrid *V*-cycle with v_1 and v_2 as the numbers of the presmoothing and postsmoothing iterations respectively. The calculation of a new iterates $\mathbf{u}_h^{m+1}$ from a given approximation is given in the following algorithm [2]:

Let $\mathbf{u}_h^{m+\frac{1}{3}}$ *be the solution after the presmoothing stage,* $\mathbf{u}_h^{m+\frac{2}{3}}$ *be the solution after the coarse grid correction, and* $\mathbf{u}_h^{m+1}$ *be the solution after the postsmoothing stage.*

Step 1: Presmoothing.

Compute $\mathbf{u}_h^{m+\frac{1}{3}}$ *by applying* v_1 *iterations of the smoother*

(Gauss-Siedel, Jacobi) on Ω_h*:* $\mathbf{u}_h^{m+\frac{1}{3}} = S_h^{v_1}\mathbf{u}_h^m$

Where S iteration matrix of the smoother.

Step 2: Coarse grid correction

Compute the residual on Ω_h*:*

$$\mathbf{r}_h = \mathbf{f}_h - A_h\mathbf{u}_h^{m+\frac{1}{3}}$$

Restrict the residual from Ω_h *to* Ω_H *and initialize the coarse grid approximation :*

$$\mathbf{f}_H = I_h^H\mathbf{r}_h, \quad \mathbf{u}_H = 0$$

If Ω_H *is the coarsest grid then solve the coarse grid equation exactly:*

$$A_H\mathbf{u}_H = \mathbf{f}_H, \text{ on } \Omega_H.$$

Else, solve the coarse grid equation:

$$A_H\mathbf{u}_H = \mathbf{f}_H, \text{ on } \Omega_H$$

approximately by applying a multigrid V-cycle starting on Ω_H

End if

Interpolate the coarse grid approximation (error) from Ω_H *to* Ω_h *:*

$$\mathbf{e}_h = I_H^h \mathbf{u}_H$$

Correct the fine grid approximation on Ω_h*:*

$$\mathbf{u}_h^{m+\frac{2}{3}} = \mathbf{u}_h^{m+\frac{1}{3}} + \mathbf{e}_h$$

Step 3: Postsmoothing.

Compute $\mathbf{u}_h^{m+\frac{1}{3}}$ *by applying* v_2 *iterations of the smoother on*

Ω_h*:* $\qquad \mathbf{u}_h^{m+\frac{1}{3}} = S_h^{v_2} \mathbf{u}_h^{m+\frac{2}{3}}.$

Following is the flowchart of a three grid *V*-cycle:

$$\begin{array}{ccc}
\mathbf{u}_h^m \xrightarrow{S_h^{v_1}} \mathbf{u}_h^{m+\frac{1}{3}} & & \mathbf{u}_h^{m+\frac{2}{3}} \xrightarrow{S_h^{v_2}} \mathbf{u}_h^{m+1} \\
I_h^{2h} \downarrow & & \uparrow I_{2h}^h \\
\mathbf{u}_{2h}^m \xrightarrow{S_{2h}^{v_1}} \mathbf{u}_{2h}^{m+\frac{1}{3}} & & \mathbf{u}_{2h}^{m+\frac{2}{3}} \xrightarrow{S_{2h}^{v_1}} \mathbf{u}_{2h}^{m+1} \\
I_{2h}^{4h} \downarrow & & \uparrow I_{4h}^{2h} \\
\mathbf{f}_{4h}^m = I_{2h}^{4h} \mathbf{r}_{2h}^{m+\frac{1}{3}} & \rightarrow & \mathbf{u}_{4h}^{m+1} = A_{4h}^{-1} \mathbf{f}_{4h}^m
\end{array}$$

Figure [**3.12**]: three grids *V*-cycle

3.5 The Full Multigrid Methods

The choice of initial approximation is important in iterative methods. The closer the initial approximation to the exact solution, the better. But

iterative methods are needed when exact solution is unknown. To get a good initial approximation, a procedure called nested iteration can be used as follows:

- Approximate the solution on the coarsest grid
- Transfer the solution to the next fine grid, for example using interpolation.
- Use the transferred solution as an initial approximation on the fine grid.

The process is repeated from coarse to fine grids. Combining the nested iteration method with multigrid method gives the so called Full Multigrid Method (FMG). The FMG starts at the coarsest grid where the equation can be solved exactly. It then proceeds to the next finer grid, performing one or more cycles at each level along the way as shown in Figure [3.13]

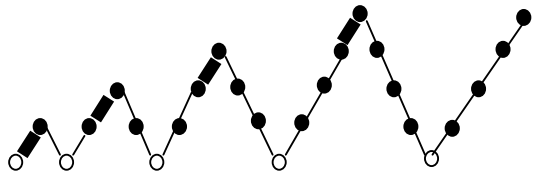


Figure [3.13]: ◆ means transfer of the approximation solution to a finer grid.

Remark: [2]

In general it is not sufficient to start the solution process on a very coarse grid, interpolate the approximation of the coarse grid solution to the next finer grid, smooth the visible error components and so on until the finest grid is reached. Actually the interpolation of the approximation leads to nonnegligible high and low frequency error components on the fine grid that can be reduced efficiently only by a subsequent smoothing of the error on all grid levels. i.e. by revisiting the coarse levels in multigrid cycles.

3.6 Multigrid iteration operator

Discritization of a linear differential equation reduces the equation to a linear system:

$$A_h \mathbf{u}_h = \mathbf{f}_h$$

Given approximation $\mathbf{u}_h^m$, we find $\mathbf{u}_h^{m+1}$ by coarse grid correction method which is given by:

$$\mathbf{u}_h^{m+1} = K_h^H \mathbf{u}_h^m + N\mathbf{f}_h \qquad 3.6.1$$

where

$K_h^H = I - I_H^h A_H^{-1} I_h^H A_h$ is the coarse grid correction matrix

and $N = I_H^h A_H^{-1} I_h^H$.

We can prove equation 3.6.1 using the relation between $\mathbf{u}_h^m$ and $\mathbf{u}_h^{m+1}$:

$$\begin{aligned} \mathbf{u}_h^{m+1} &= \mathbf{u}_h^m + \mathbf{e}_h^m \\ &= \mathbf{u}_\mathrm{h}^\mathrm{m} + I_H^h \mathbf{e}_H^m \end{aligned}$$

But

$$\mathbf{e}_h^{m+1} = A_H^{-1}\mathbf{r}_H^m$$

and

$$\mathbf{r}_H^m = I_H^h(\mathbf{f}_h - A\mathbf{u}_h^m)$$

by substitution we get:

$$\mathbf{u}_h^{m+1} = \mathbf{u}_h^m + I_H^h A_H^{-1} I_H^h(\mathbf{f}_h - A\mathbf{u}_h^m)$$

which completes the proof.

For the error:

$$\mathbf{e}_h^{m+1} = K_h^H \mathbf{e}_h^m \qquad 3.6.2$$

Recall that:

$$\mathbf{e}_h^{m+1} = \mathbf{u}_h - \mathbf{u}_h^{m+1}.$$

If we multiply both sides of equation 3.6.2 by A_h, we get the residual after coarse grid correction:

$$\mathbf{r}_h^{m+1} = A_h K_h^H A_h^{-1}\mathbf{r}_h^m$$

The error after v_1 presmoothing iterations is given by:

$$\mathbf{e}_h^{\frac{1}{3}} = S^{v_1}\mathbf{e}_h^0$$

where $\mathbf{e}_h^0$ is the initial error. After coarse grid correction, the error is:

$$\mathbf{e}_h^{\frac{2}{3}} = K_h^H \mathbf{e}_h^{\frac{1}{3}}.$$

Then the error after two-grid method is given by:

$$\mathbf{e}_h^1 = Q_2 \mathbf{e}_h^0$$

where

$$Q_2 = S^{v_2} K_h^H S^{v_1} \qquad 3.6.3$$

is the two-grid iteration matrix.

Theorem 3.7.1[1]:

The iteration matrix $Q_k(v_1, v_2)$ of the multigrid method satisfies:

$$Q_2(v_1, v_2) = \widetilde{Q}_2(v_1, v_2) \qquad 3.6.4$$

and

$$Q_k(v_1, v_2) = \widetilde{Q}_k(v_1, v_2) + S_k^{v_2} I_{k-1}^k (Q_{k-1})^\gamma A_{k-1}^{-1} I_k^{k-1} A_k S_k^{v_1} \qquad 3.6.5$$

where:

$$\widetilde{Q}_k(v_1, v_2) = S_k^{v_2} \left\{ I - I_{k-1}^k A_{k-1}^{-1} I_k^{k-1} A_k \right\} S_k^{v_1}$$

is the iteration matrix of multigrid method.

Proof:

Equation 3.6.4 follows from equation 3.6.3. Equation 3.6.5 is proved by induction, let the equation be true for *n=k*. we want to prove that it is true at *n=k+1*.

Let $\mathbf{e}_{k+1}^0$ be the error on Ω_{k+1} before multigrid, $\mathbf{e}_{k+1}^{\frac{1}{3}}$ is the error after pre-smoothing, $\mathbf{e}_{k+1}^{\frac{2}{3}}$ is the error after coarse grid correction, and $\mathbf{e}_{k+1}^1$ is the error after post-smoothing then we have:

$$\mathbf{e}_{k+1}^{\frac{1}{3}} = S_{k+1}^{\nu_1}\mathbf{e}_{k+1}^{0} \qquad 3.6.6$$

The coarse grid problem to be solved is:

$$A_k\mathbf{u}_k = -I_{k+1}^{k}A_{k+1}\mathbf{e}_{k+1}^{\frac{1}{3}}$$

with initial guess $\mathbf{u}_k^0 = 0$. Hence the initial error $\mathbf{e}_\mathbf{k}^\mathbf{0}$ is the negative of the exact solution on Ω_k which means

$$\mathbf{e}_k^0 = A_k^{-1}I_{k+1}^{k}A_{k+1}\mathbf{e}_{k+1}^{\frac{1}{3}} \ .$$

After coarse grid correction the error on Ω_k is

$$(Q_k)^\gamma\,\mathbf{e}_k^0$$

hence the coarse grid correction is given by:

$$\left(-I + (Q_k)^\gamma\right)\mathbf{e}_k^0$$

therefore:

$$\mathbf{e}_{k+1}^{\frac{2}{3}} = e_{k+1}^{\frac{1}{3}} + I_k^{k+1}\left(-I + (Q_k)^\gamma\right)\mathbf{e}_k^0$$

$$\mathbf{e}_{k+1}^{\frac{2}{3}} = \left\{I - I_k^{k+1}A_k^{-1}I_{k+1}^{k}A_{k+1} + I_k^{k+1}(Q_k)^\gamma A_k^{-1}I_{k+1}^{k}A_{k+1}\right\}e_{k+1}^{\frac{1}{3}} \qquad 3.6.7$$

Then:

$$\mathbf{e}_{k+1}^{1} = S_{k+1}^{\nu_2}e_{k+1}^{\frac{2}{3}} \qquad 3.6.8$$

Combining equations 3.6.6, 3.6.7,and 3.6.8 ends the proof.

Chapter four

Convergence Analysis

4.1 Introduction

4.2 Smoothing analysis

4.2.1 Smoothing property

4.2.2 Local Fourier analysis

4.3 Convergence analysis of two-grid method

4.4 Multigrid convergence

Chapter 4
Convergence Analysis

4.1 Introduction

Studying convergence of multigrid methods is not an easy task, and is still an open area of computational mathematics. The smoothing error modes, which remain after relaxation on one grid, become oscillatory on the coarse grids. Therefore, moving to coarser and coarser grid, all error components on the finest grid become oscillatory and are reduced by relaxation. For good multigrid method, the convergence factor of the multigrid method, $\|Q_k(v_1, v_2)\|$ need to be small and independent of *h*, i.e.

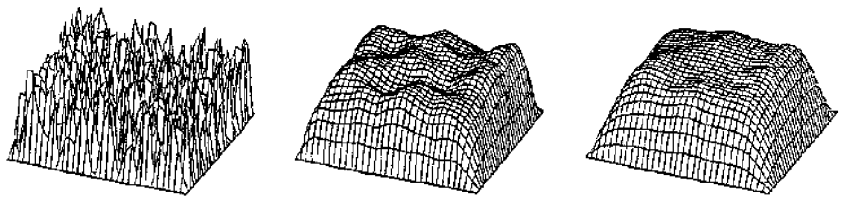
$$\|Q_k(v_1, v_2)\| \leq \text{constant} < 1$$

Where $Q_k(v_1, v_2)$ is the iteration matrix of the multigrid method. For this purpose we need the smoothing factor ρ, and two-grid convergence factor norm $\|Q_2\|$.

4.2 Smoothing Analysis

Classical iterative methods are still important but less favored, because after few iteration steps, the error of the approximations become smooth. These methods remove high frequency components (rapidly oscillating parts) leaving a smooth error, but low frequency components are reduced slowly. So that these methods are called smoothers. However, these basic

methods are known as efficient smoothers but not as efficient solvers. I mean, they are efficient in smoothing the error but not in reducing it. Figure 4.1 illustrates the error smoothing effect.



Initial errors Error after 5 iterations Error after 10 iterations

Figure 4.1[2]: Error in the Gauss-Seidel approximation of the solution of Poisson problem.

The smoothness of the error slows down the convergence of the basic iterative method.

Example 4.1

Consider:

$$u_{xx} + u_{yy} = \left(x^2 + y^2\right)e^{xy} \qquad 0 < x < 1, \;\; 0 < y < 1,$$

$$u(0, y) = 1, \; u(1, y) = e^y, \; u(x,0) = 1, \; u(x,1) = e^x$$

Table 4.1 shows the number of iterations and the approximate computer time needed by the Gauss-Siedel method with initial approximation $\mathbf{u}^0 = \mathbf{0}$ and $Tol = 10^{-5}$. these results are obtained using the mathematical software maple 12 and an intel Core 2 Duo processor. Figures 4.2, 4.3, and 4.4 shows the maximum error norm versus the number of iterations needed for various mesh sizes.

Table 4.1: Approximate computer time

size	Number of iterations	Approximate computer time
8× 8	67	0.2sec
16×16	234	3.2sec
32×32	791	60.8sec
64×64	2587	1235.8sec
128×128	8044	21860.4sec
256×256	20431	Four days

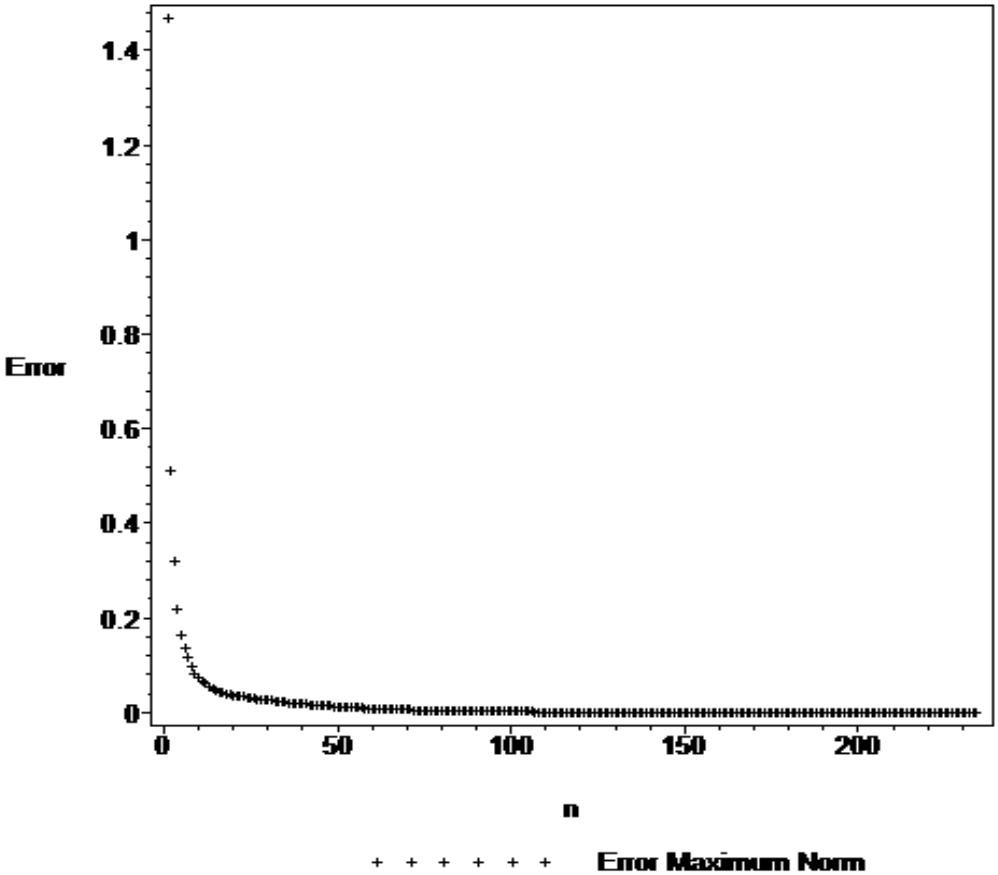


Figure 4.2: relation between maximum error norm and the number of iterations *n*, when size *16×16*

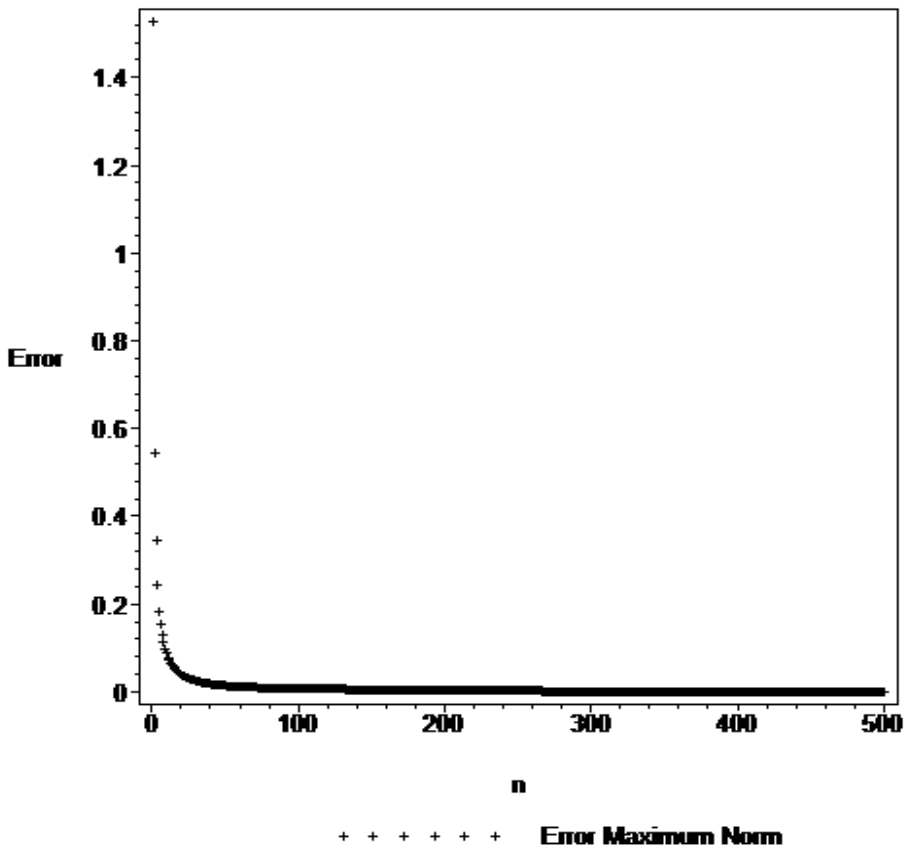


Figure 4.3: relation between maximum error norm and the number of iterations *n*, when size 32×32

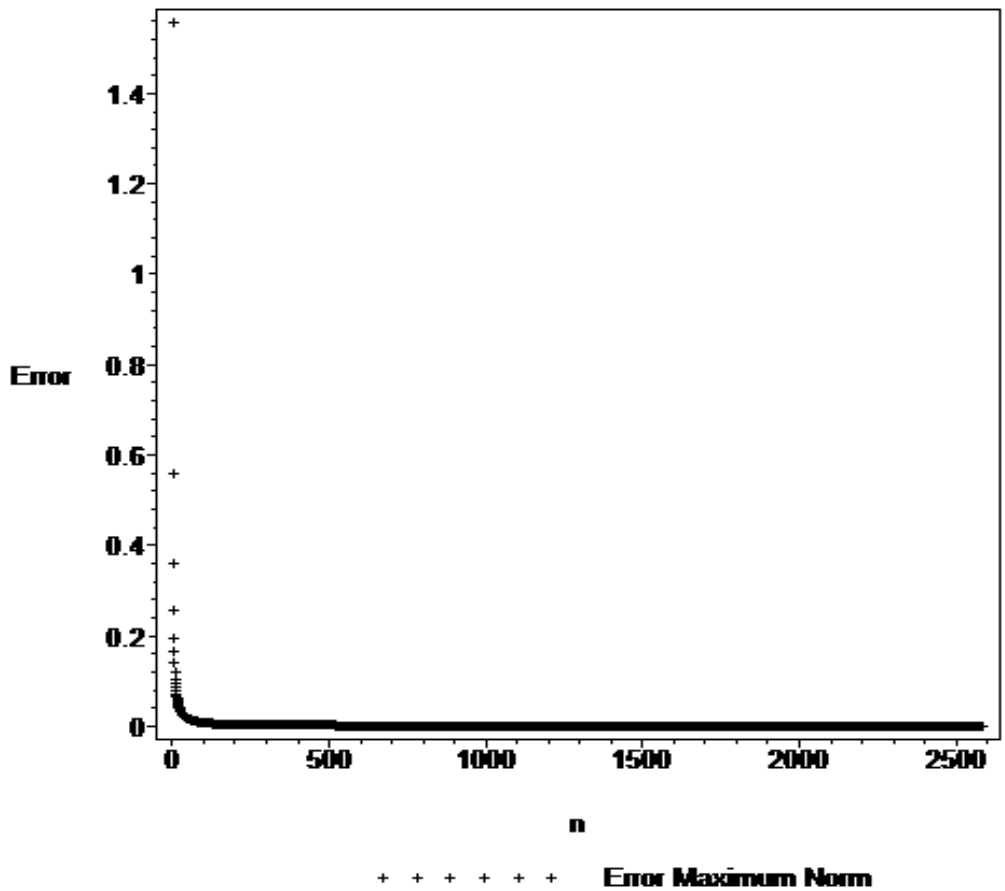


Figure 4.4: relation between maximum error norm and the number of iterations *n*, when size *128x128*

The efficiency of smoothing method can be studied by the smoothing property and by the Local Fourier Analysis (LFA).

4.2.1 Smoothing property

Discritization of the linear PDE leads to the linear system:

$$A\mathbf{u} = \mathbf{f} .$$

Using the splitting $A = M - N$, we can define the iteration method:

$$\mathbf{u}^{m+1} = T\mathbf{u}^{m} + \mathbf{L} \qquad 4.2.1$$

with $\mathbf{u}^0$ as initial approximation, *m=0,1,2,3….,* $T = M^{-1}N$ is the iteration matrix, and $\mathbf{L} = M^{-1}\mathbf{f}$. Convergence of the iteration 4.2.1 depends on the iteration matrix *T*. After *v* iterations we obtain:

$$
\begin{aligned}
\mathbf{u}^1 &= T\mathbf{u}^0 + \mathbf{L} \\
\mathbf{u}^2 &= T\mathbf{u}^1 + \mathbf{L} \\
&= T(T\mathbf{u}^0 + \mathbf{L}) + \mathbf{L} \\
&= T^2\mathbf{u}^0 + (T + I)\mathbf{L} \\
\mathbf{u}^3 &= T\mathbf{u}^2 + L \\
&= T\mathbf{u}^0 + (T^2 + T + I)\mathbf{L} \\
&. \\
&. \\
&. \\
\mathbf{u}^v &= T^v\mathbf{u}^0 + (T^{v-1} + T^{v-2} + \cdots + T + I)\mathbf{L} \\
\mathbf{u}^v &= T^v\mathbf{u}^0 + S\mathbf{f}
\end{aligned}
$$

where

$$S = \left(T^{v-1} + T^{v-2} + ... + I\right)M^{-1}$$

Let

$$\mathbf{u}^v = T\ \mathbf{u}^{v-1} + \mathbf{L}$$

and let

$$\mathbf{u} = T\mathbf{u} + \mathbf{L}$$

then we have

$$
\begin{aligned}
\mathbf{u}^v - \mathbf{u} &= T(\mathbf{u}^{v-1} - \mathbf{u}) \\
\mathbf{e}^v &= T\mathbf{e}^{v-1} \\
&= TT\mathbf{e}^{v-2} \\
&= T^2\mathbf{e}^{v-2} \\
&. \\
&= T^v\mathbf{e}^0
\end{aligned}
$$

The error $\mathbf{e}$ satisfies:

$$\mathbf{e}^v = T^v\mathbf{e}^0 \qquad 4.2.2$$

This equation gives the relation between the error before and after v-smoothing iterations, but we need to measure the smoothing behavior. For this purpose, the smoothing property will be defined. $\| \ \|$ will denote the Euclidean matrix norm.

Definition 4.2.1[1] Smoothing property

Let v be the number of iterations and h is the grid size used in iteration 4.2.1 If there exist a constant C_T and a function $\eta(v)$ such that:

$$\left\|AT^{v}\right\| \le C_T h^{-2}\eta(v) \ , \eta(v) \to 0 \ \text{ for } v \to \infty \text{ for all } h$$

Then we say that the iteration matrix T in iteration 4.2.1 has the smoothing property

Theorem4.2.1[1]

If the iteration matrix T in iteration 4.2.1 has the smoothing property, then iteration (4.2.1) is convergent.

Proof:

$$\left\|T^{v}\right\| \le \left\|A^{-1}\right\|\left\|AT^{v}\right\| \le \left\|A^{-1}\right\|C_T h^{-2}\eta(v) \ ,$$
$$\text{hence } \lim_{v\to\infty}\left\|T^{v}\right\| = 0 \text{ and so } \lim_{v\to\infty}\left\|\mathbf{e}^{v}\right\| = 0$$

We can see it is difficult to prove the smoothing property for basic iterative methods. In [3] the smoothing property is shown for the damped Jacobi iterative method. The original Jacobi iteration is:

$$\mathbf{u}^{j+1} = \mathbf{u}^{j} - D^{-1}\left(A\mathbf{u}^{j} - \mathbf{f}\right)$$

whereas, the damped Jacobi iteration is:

$$\mathbf{u}^{j+1} = \mathbf{u}^{j} - \omega D^{-1}\left(A\mathbf{u}^{j} - \mathbf{f}\right)$$

In many cases the diagonal is $D = dh^{-2}I,\ d \in \Re$. Replacing D^{-1} by $\omega h^{-2}I,\ d \in \Re$ (is suitable) we obtain:

$$\mathbf{u}^{j+1} = \mathbf{u}^{j} - \omega h^{2}\left(A\mathbf{u}^{j} - \mathbf{f}\right)$$

then the iteration matrix:

$$T = I - \omega h^{2} A$$

A possible choice of ω is $\omega = \dfrac{1}{\mathrm{C_T}}$ where $\mathrm{C_T}$ is a good bound for $h^2 A$:

$$\left\| h^{2} A \right\| \le C_T$$

where $\| \ \|$ is the spectral norm for matrices.

Two definitions are needed before discussing the smoothing property for damping Jacobi.

Definition 4.2.2 [13]:(positive semi-definite)

An $n \times n$ real symmetric matrix A is positive semi-definite if:

$$x^{T} A x \ge 0 \text{ for all } x \in \Re^{n}$$

Theorem 4.2.2: [3]

Assume that A is symmetric and positive semi-definite, then the damped Jacobi iteration satisfies the smoothing property with:

$$\eta(v) = \frac{3}{8}\left(\frac{1}{v+\frac{1}{2}}\right)$$

Proof:

The matrix $AT^{v} = A(I - \omega h^2 A)^{v}$ is symmetric, its eigenvalues μ are $\lambda(1-\omega h^2 \lambda)^{v}$, with λ eigenvalues of A.

we have:

$$\left\|AT^{v}\right\| = \sup\left\{\lambda(1-\omega h^2\lambda)^{v} \middle| : \lambda \text{ eigenvalues of } A\right\}$$

λ is nonnegative since A is positive semi-definite, and $1-\omega h^2\lambda$ is nonnegative by definition of ω. As all eigenvalues of A are in $[0, C_T h^{-2}]$, the estimate:

$$\left\|AT^{v}\right\| = \sup\left\{\lambda(1-\omega h^2\lambda)^{v} \middle| : \lambda \text{ eigenvalues of } A\right\}$$

follows.

consider $x = \omega h^2 \lambda$, x varies in [0,1] Hence we have:

$$\left\|AT^{v}\right\| = \sup\left\{C_T h^{-2} x(1-x)^{v} \ : \ 0 \le x \le 1\right\}$$

The maximum of $x(1-x)^{v}$ in [0,1] occurs when $x = \frac{1}{v+1}$. A very close upper bound for the maximum is $\eta(v) = \frac{3}{8}\left(\frac{1}{v+\frac{1}{2}}\right)$. Hence smoothing property holds.

Note that SOR should not be used as a smoothing operator. Hackbusch shows that SOR reduce the low frequencies components. But the reduction of high frequencies components usually becomes even worse.

4.2.2 Local Fourier Analysis

Local Fourier Analysis (LFA) is the most powerful tool for studying the smoothing efficiency, which was introduced by Brandt. Contributions have been made by Stüben, Trottenberg and Wesseling. Brandt have used the term local mode analysis instead of LFA, both terms denote the same approach. So LFA is used in studying the smoothing efficiency of basic iterative methods. The aim of LFA is to compute another measure of the smoothing behavior of an iterative method. This measure is called Fourier smoothing factor. The Fourier smoothing factor is very important measure for designing efficient multigrid methods. In our study we concentrate on the Fourier smoothing factor for two smoothing methods: Jacobi method and the Gauss-Seidel method. Before using this measure, we need to know more about elements of Fourier analysis.

Definition 4.2.3:

The inner product of two continuous functions f and g over a set S is defined as:

$$\langle f, g \rangle = \int_S f(x)\overline{g(x)}dx$$

where $\overline{g(x)}$ is the complex conjugate of *g(x)*.

For discrete functions f and g the inner product is defined as:

$$\langle f, g \rangle = \sum_{S} f(x)\overline{g(x)}$$

Definition 4.2.4:

Two functions are orthogonal on a set S if :

$$\langle f, g \rangle = 0$$

a set of functions $\{f_i\}_{i=1}^{n}$ is orthogonal set if :

$$\langle f_i, f_j \rangle = 0 \quad \text{when } i \neq j$$

Lemma 4.2.1 [1] Orthogonality in one dimension.

Let $I = \{0,1,2,....n-1\}$. and $\psi_j(\theta_k) = e^{ij\theta_k}$, where $\theta_k = \dfrac{2\pi k}{n}$, $j \in I, i = \sqrt{-1}$.

Then: $\sum_{j=0}^{n-1} \Psi_j(\theta_k)\Psi_j(-\theta_l) = n\delta_{kl}$, with δ_{kl} the Kronecker delta.

Proof:

If $k = l$, then $\sum_{j=0}^{n-1} \psi_j(\theta_k)\psi_j(-\theta_k) = \sum_{j=0}^{n-1} e^{ij\theta_k} e^{-ij\theta_k} = n$.

But if $k \neq l$, then $\sum_{j=0}^{n-1} \psi_j(\theta_k)\psi_j(-\theta_l) = \sum_{j=0}^{n-1} e^{ij(\theta_k - \theta_l)}$

which is a geometric series, so it is equal to:

$$\frac{1 - e^{in(\theta_k - \theta_l)}}{1 - e^{i(\theta_k - \theta_l)}} = \frac{1 - e^{2\pi i(k-l)}}{1 - e^{\frac{2\pi i}{n}(k-l)}} = 0.$$

Theorem 4.2.3 [1] Discrete Fourier Transform in one dimension.

Every discrete function $u : I \to \Re$, can be written as:

$$u_j = \sum_{k=-m}^{m+p} c_k \psi_j(\theta_k). \qquad 4.2.3$$

where $I = \{1,2,...,n-1\}, \psi_j(\theta_k) = e^{ij\theta_k}$, and $\theta_k = \frac{2\pi k}{n}, j \in I.$

.

For n odd, $p = 0$ and $m = \frac{n-1}{2}$. and for n even, $p = 1$ and $m = \frac{n}{2} - 1$, and

$$c_k = \frac{1}{n} \sum_{j=0}^{n-1} u_j \psi_j(-\theta_k) \qquad 4.2.4$$

The functions $\psi_j(\theta)$ are called Fourier modes or Fourier components.

Proof:

If we choose c_k as in equation 4.2.4, then:

$$\begin{aligned}\sum_{k=-m}^{m+p} c_k \psi_j(\theta_k) &= \frac{1}{n}\sum_{k=-m}^{m+p}\sum_{l=0}^{n-1} u_l \psi_l(-\theta_k)\psi_j(\theta_k) \\ &= \frac{1}{n}\sum_{l=0}^{n-1} u_l \sum_{k=0}^{n-1} e^{\frac{i2\pi(k-m)(j-l)}{n}} \\ &= \frac{1}{n}\sum_{l=0}^{n-1} u_l e^{\frac{i2\pi m(l-j)}{n}} \sum_{k=0}^{n-1} \psi_k(\theta_j)\psi_k(-\theta_l) \\ &= u_j\end{aligned}$$

Conversely, assume that equation 4.2.3 holds. We want to show (4.2.4) as follows:

$$\begin{aligned}\frac{1}{n}\sum_{j=0}^{n-1} u_j \psi_j(-\theta_k) &= \frac{1}{n}\sum_{l=-m}^{m+p}\sum_{l=0}^{n-1} c_l \psi_j(-\theta_k)\psi_j(\theta_l) \\ &= \frac{1}{n}\sum_{l=-m}^{m+p} c_l \sum_{j=0}^{n-1} \psi_j(-\theta_k)\psi_j(\theta_l). \\ &= \sum_{l=-m}^{m+p} c_l \delta_{kl} = c_{k.}\end{aligned}$$

In two dimensions:

Let $I = \{j : j = (j_1, j_2), j_1 = 0,1,\ldots,n_1 - 1, j_2 = 0,1,\ldots,n_2 - 1\}$

and let $\Theta = \left\{\theta = (\theta_1, \theta_2) : \theta_1 = \frac{2\pi k_1}{n_1}, \theta_2 = \frac{2\pi k_2}{n_2}\right\}$

where $k_1 = -m_1,\ldots,m_1 + p_1$

and $k_2 = -m_2,\ldots,m_2 + p_2$

$m_1 = \frac{(n_1 - 1)}{2}$ for n_1 odd and $p_1 = 1, m_1 = \frac{n_1}{2} - 1$ for n_1 even.and $p_1 = 0,$

$m_2 = \frac{(n_2 - 1)}{2}$ for $n_2$odd and $p_2 = 1, m_2 = \frac{n_2}{2} - 1$ for n_2 even.$p_2 = 0$

The following lemma shows that the set:

$$\psi = \{\psi_j(\theta) : j \in I \text{ and } \theta \in \Theta\}$$

is orthogonal.

Lemma 4.2.2[1] Orthogonality in two dimension.

If we define $\psi_j(\theta) = e^{ij\theta}$, with $j \in I$, $\theta \in \Theta$,

then $\sum_{j=1}^{n-1} \psi_j(\theta)\psi_j(-\upsilon) = \begin{cases} n_1 n_2 & \text{, if } \upsilon = \theta \\ 0 & \text{, if } \upsilon \neq \theta \end{cases}$,

where $\theta, \upsilon \in \Theta$.

Proof:

From the previous lemma,

$$\begin{aligned}\sum_{j=1}^{n-1} \psi_j(\theta)\psi_j(-\upsilon) &= \sum_{j=1}^{n-1} e^{ij(\theta-v)} \\ &= \sum_{j_1=1}^{n_1-1} e^{ij_1(\theta_1-v_1)} \sum_{j_2=1}^{n_2-1} e^{ij_2(\theta_2-v_2)} \\ &= \begin{cases} n_1 n_2 & \text{, if } \upsilon = \theta \\ 0 & \text{, if } \upsilon \neq \theta \end{cases}\end{aligned}$$

Theorem 4.2.4[1] Discrete Fourier transform in two dimension.

Let $I = \{j = (j_1, j_2), \text{where } j_1 = 1,2,...,n_1 - 1 \text{ and } j_2 = 1,2,...,n_2 - 1\}$, then every $u : I \to \Re$ can be written as:

$$u_j = \sum_{\theta \in \Theta} c_\theta \psi_j(\theta),$$

with

$$c_\theta = \frac{1}{n_1 n_2} \sum_{j \in I} u_j \psi_j(-\theta).$$

where Θ is defined as in lemma 4.2.2.

Proof: generalization of theorem 4.2.1.

Let:

$$A\mathbf{u} = \mathbf{f}$$

Using a classical iterative method gives:

$$\mathbf{u}^{m+1} = S\mathbf{u}^{m} + M^{-1}\mathbf{f}, \text{ where } A = M - N, \text{and } \mathrm{S} = \mathrm{M}^{-1}N \qquad 4.2.5$$

After few v-iterations, the error become smooth, so that these iterative methods are called smoothing methods. The relation between the error before and after v-iterations is given by:

$$\mathbf{e}^{v} = S^{v}\mathbf{e}^{0}$$

Definition 4.2.4[17]:

A set $\{\psi_j\}$ of functions is complete if and only if any function in Euclidean space can be written as a linear combination of functions from the set $\{\psi_j\}$.

Assume that the operator S has a complete set of eigenfunctions or local modes $\psi(\theta)$, $\theta \in \Theta$, where Θ is some discrete index set.

Hence,

$$S^{v}\psi(\theta) = \lambda^{v}(\theta)\psi(\theta) \qquad 4.2.5$$

where $\lambda(\theta), \theta \in \Theta$, are the eigenvalues of the operator S, and $\psi(\theta)$ is an eigenfunction of the operator S. In this case $\psi(\theta)$ is called local mode.

We can write the error before v-smoothing steps as:

$$\mathbf{e}^0 = \sum_{k=1}^{n-1} c_\theta^0 \psi(\theta), \text{ where } \theta = \frac{2k\pi}{n}, k = 1,2,...,n-1 \qquad 4.2.6$$

and the error after v-smoothing steps as:

$$\mathbf{e}^v = \sum_{k=1}^{n-1} c_\theta^v \psi(\theta), \text{ where } \theta = \frac{2k\pi}{n}, k = 1,2,...,n-1 \qquad 4.2.7$$

The relation between c_θ^0 and c_θ^v is important. It gives the effect of v-smoothing steps on the error. From equations 4.2.5, 4.2.6, and 4.2.7 we get:

$$c_\theta^v = \lambda^v(\theta) c_\theta^0 \qquad 4.2.8$$

The eigenvalue $\lambda(\theta)$ is called the amplification factor of the local mode $\psi(\theta)$.

For the smoothing factor we need to distinguish between high and low frequency components.

Definition 4.2.5: High and low frequencies [1]

Consider the set $\Theta = \left\{\theta : \theta = \frac{\pi k}{n}, k = 1,2,.....,n-1\right\}$. We say that $\psi(\theta)$ is a high frequency component (rough) if and only if

$$\theta \in \Theta^{high} = \Theta \bigcap \left[\frac{\pi}{2}, \pi\right],$$

and is a low frequency (smooth) if and only if

$$\theta \in \Theta^{low} = \Theta / \Theta^{high} \ .$$

So that the error grid function can be presented as:

$$\mathbf{e}^0 = \sum_{\theta\in\Theta} c_\theta^0 \psi(\theta) = \sum_{\theta\in\Theta^{high}} c_\theta^0 \psi(\theta) + \sum_{\theta\in\Theta^{low}} c_\theta^0 \psi(\theta)$$

Definition 4.2.6 [1]: Fourier smoothing factor

The Fourier smoothing factor ρ of the smoothing method in equation 4.2.5 is defined by:

$$\rho = \sup\left\{ |\lambda(\theta)| : \theta \in \Theta^{high} \right\}.$$

Hence, after v-smoothing iterations the amplitude of the high frequency components of the error in equation 4.2.8 are multiplied by a factor ρ^v or smaller.

Examining the quality of smoothing method, we need to determine the Fourier smoothing factor ρ. To do this, we have to solve the eigenvalue problem:

$$S\psi(\theta) = \lambda(\theta)\psi(\theta), \text{ where } S = M^{-1}N$$

which means

$$N\psi(\theta) = \lambda(\theta) M\psi(\theta).$$

This relation can be written by stencil notation as:

$$\sum_{j\in Z} {}^{n}{}_{j}\psi_{x+jh}(\theta) = \lambda(\theta) \sum_{j\in Z} {}^{m}{}_{j}\psi_{x+jh}(\theta) \qquad 4.2.9$$

Local Fourier analysis can be simplified by assuming that the coefficients in the partial differential equation to be solved are constant.

If $\sum_{j\in Z} n_j e^{(x+j)i\theta} = e^{ix\theta}\sum_{j\in N} n_j e^{ij\theta}$ then $\psi_x(\theta) = e^{ix\theta}$ satisfies 4.2.9 with:

$$\lambda(\theta) = \frac{\sum_{j\in Z} n_j e^{ij\theta}}{\sum_{j\in Z} m_j e^{ij\theta}}$$

Example 4.1:

For Laplace's equation:

$$-u_{xx} - u_{yy} = 0$$

The correspondence splitting gives:

$$M = \begin{bmatrix} & 0 & \\ -1 & 4 & 0 \\ & -1 & \end{bmatrix} \qquad N = \begin{bmatrix} & 1 & \\ 0 & 0 & 1 \\ & 0 & \end{bmatrix}$$

and

$$\lambda(\theta) = \frac{e^{i\theta_1} + e^{i\theta_2}}{-e^{-i\theta_1} + 4 - e^{-i\theta_2}}$$

the Fourier smoothing factor $\rho = \frac{1}{2}$

Finally, Table 4.2 shows the smoothing factors of Jacobi, damped Jacobi, and Gauss-Seidel methods. It shows that Gauss-Siedel method as the best smoother for the multigrid method.

Table 4.2[2]: Smoothing factors

Iterative Method	**Smoothing Factor**	**Smoothing**
Jacobi	1	No
Damped Jacobi (w=0.5)	0.75	unsatisfactory
Damped Jacobi (w=0.8)	0.6	acceptable
GS	0.5	good

4.3 Convergence analysis of two-grid method.

The purpose of two-grid analysis is to show that the rate of convergence of two-grid method is independent of the grid size *h*. In the first part of this section, we will show how local mode analysis can be used to derive bounds for $\|Q_2\|$ quantitatively, which means that we are interested in h-independent real bounds for $\|Q_2\|$. In the second part, we are interested in qualitative consideration that will help to make the requirements to be satisfied by the smoother and transfer operators.

To simplify the analysis of the convergence of the two grid method, we omit the boundary conditions and study all operators on an infinite grid i.e. Instead of $\Omega_h = \left\{ jh : j \in 0,1,....,\frac{1}{n} \right\}$. And the iteration matrix for two grid methods become:

$$Q_2 = S_h^{v_2} \left[I - I_H^h A_H^{-1} I_h^H A_h \right] S_h^{v_1} \qquad 4.3.1$$

on infinite grid

Where:

$S_h^{v_2}$: is the iteration matrix of the smoothing method.
I : extended unit matrix.
I_H^h : extended prolongation operator.
I_h^H : extend restriction operator.
A_h : exteneded discrete operator on the fine grid.
A_H : exteneded discrete operator on the coarse grid.

In one dimension multigrid methods can be analyzed easier.

Studying convergence of two-grid method with qualitative consideration depends on $\|Q_2\|$, where the norm used is the Euclidean norm. For simplicity, we assume that $v_2 = 0$ i.e.

$$Q_2 = \left[I - I_H^h A_H^{-1} I_h^H A_h\right] S_h^{v_1} \qquad 4.3.2$$

So we can write

$$Q_2 = \left(A_h^{-1} - I_H^h A_H^{-1} I_h^H\right)\left(A_h S_h^{v_1}\right) \qquad 4.3.3$$

So that

$$\|Q_2\| \le \left\|A_h^{-1} - I_H^h A_H^{-1} I_h^H\right\| \left\|A_h S_h^{v_1}\right\| \qquad 4.3.4$$

We see that $\|Q_2\|$ depends on $\left\|A_h^{-1} - I_H^h A_H^{-1} I_h^H\right\|$ and $\left\|A_h S_h^{v_1}\right\|$. For these two factors, we need the following definitions.

Definition 4.3.1[1] smoothing property

S has the smoothing property if there exist a constant C_s and a function $\eta(v_1)$ independent of h such that:

$$\left\|A_h S_h^{v_1}\right\| \le C_s h^{-2} \eta(v_1) \text{ where } \eta(v_1) \to 0 \text{ for } v_1 \to \infty \qquad 4.3.5$$

Definition 4.3.2[1] Approximation property

The approximation property holds if there exists a constant C_A independent of h such that:

$$\left\|A_h^{-1} - I_H^h A_H^{-1} I_h^H\right\| \le C_A h^2 \qquad 4.3.6$$

If these two properties hold, then it is easy to talk about the h-independent convergence of two-grid method.

Theorem 4.3.1: h-independent two-grid rate of convergence

Let the smoothing property and approximation property hold then there exists a number $\bar{v}$ independent of h such that:

$$\|Q_2\| \le C_s C_A \eta(v_1) < 1 \ , \forall v_1 \ge \bar{v} \qquad 4.3.7$$

Proof:

$$\|Q_2\| \le \left\| A_h^{-1} - I_H^h A_H^{-1} I_h^H \right\| \left\| A_h S_h^{v_2} \right\|$$

Based on the previous results, we will study the convergence of multigrid method.

4.4 Multigrid convergence

Convergence analysis of the two-grid method, can be generalized to a multigrid method. In this section, we assume that $A_l u_l = f_l$ is the linear system obtained from discetization of a PDE on Ω_l

Definition 4.4.1[1] smoothing property

The smoothing iteration matrix S_k has the smoothing property if there exist a constant C_s and a function $\eta(v_1)$ independent of h_k such that:

$$\left\| A_k S_k^{v_1} \right\| \le C_s h_k^{-2} \eta(v_1), \quad \eta(v_1) \to 0 \text{ for } v_1 \to \infty$$

Definition 4.4.2[1] approximation property

The approximation property holds if there exists a constant C_A independent of h_k such that:

$$\left\| A_k^{-1} - I_{k-1}^k A_{k-1}^{-1} I_k^{k-1} \right\| \le C_A h_k^2$$

Lemma 4.4.1[1]

Let the smoothing property hold, and assume that there exists a constant C_p independent of k such that:

$$\left\| I_{k-1}^k \mathbf{u}_{k-1} \right\| \ge C_p^{-1} \left\| \mathbf{u}_{k-1} \right\|, \ \forall \mathbf{u}_{k-1} \qquad 4.4.1$$

then:

$$\left\| A_{k-1}^{-1} I_k^{k-1} A_k S_k^{v_2} \right\| \le C_p \left(1 + \left\| Q_k(v_1, 0) \right\| \right)$$

Proof:

It has been shown that if S_k has smoothing property, then the smoothing method is convergent. Hence we can choose v_2 such that $\left\| S_k^{v_1} \right\| < 1$

From equation 4.4.1 we get:

$$\begin{aligned}
\left\| A_{k-1}^{-1} I_k^{k-1} A_k S_k^{v_1} \right\| &\le C_p \left\| I_{k-1}^k A_{k-1}^{-1} I_k^{k-1} A_k S_k^{v_1} \right\| \\
&= C_p \left\| S_k^{v_1} - \left(A_k^{-1} - I_{k-1}^k A_{k-1}^{-1} I_k^{k-1} \right) A_k S_k^{v_1} \right\| \\
&= C_p \left\| S_k^{v_1} - Q_k(v_1, 0) \right\| \\
&< C_p \left(1 + \left\| Q_k(v_1, 0) \right\| \right)
\end{aligned}$$

The following inequality is necessary for the next theorem

$$\zeta_1 \le \zeta,\ \zeta_k \le \zeta + C\zeta_{k-1}^{\gamma},\ \ k \ge 2 \qquad 4.4.2$$

Lemma 4.4.2[1]

Assume $\gamma C > 1$. if $\gamma \ge 2,\ \zeta \le \tilde{\zeta} = \frac{\gamma - 1}{\gamma}(\gamma C)^{\frac{-1}{\gamma-1}}$ then the solution

of inequality 4.4.2 is bounded by $\zeta_k \le z < 1$ where z is related to ζ

by:

$$\zeta = z - Cz^{\gamma} \qquad (*)$$

and z satisfies:

$$z \le \frac{\gamma}{\gamma - 1}\zeta$$

Proof:

We have $\zeta_k \le z_k$, with z_k defined by:

$$z_1 = \zeta \ \text{ and } \ z_k = \zeta + Cz_{k-1}^{\gamma}$$

Since $\{z^k\}$ is monotonically increasing, we have $z_k < z$, with z the smallest solution of consider $f(z) = z - Cz^{\gamma}$. The maximum of $f(z)$ is reached in $z = z^* = (\gamma C)^{\frac{-1}{\gamma-1}} < 1$ and $f(z^*) = \tilde{\zeta}$. For $\zeta \le \tilde{\zeta}$ equation (*) has a solution $z \le z^* < 1$.

We have:

$$\zeta = z - Cz^{\gamma} \ge z - \frac{1}{\gamma}z = \frac{\gamma - 1}{\gamma}z$$

Then:

$$z \le \frac{\gamma}{\gamma - 1}\zeta \ .$$

Theorem 4.4.1[1] rate of convergence of multigrid method

Let the smoothing property and approximation property hold assume $\gamma \ge 2$

let

$$\left\| I_{k-1}^{k}\mathbf{u}_{k-1} \right\| \ge C_p^{-1}\left\| \mathbf{u}_{k-1} \right\|, \ \forall \mathbf{u}_{k-1}$$

and

$$\left\| I_{k-1}^{k}\mathbf{u}_{k-1} \right\| \ge c_p\left\| \mathbf{u}_{k-1} \right\|, \ \forall \mathbf{u}_{k-1}$$

C_p^{-1} and c_p independent of *k*. let $\tilde{\zeta} \in (0,1)$ be given. Then there is a number $\tilde{v}$ independent of *k* such that the iteration matrix $Q^k(v_1,0)$ satisfies:

$$\left\| Q_k(v_1,0) \right\| \le \tilde{\zeta} < 1 \quad \text{if} \ \ v_1 \ge \tilde{v}$$

Proof:

In chapter three, the iteration matrix of multigrid method was found as:

$$Q_k(v_1,0) = \tilde{Q}_k(v_1,0) + \left(S^k\right)^{v_2} I_{k-1}^{k}\left(Q_{k-1}\right)^{\gamma}\left(A_{k-1}\right)^{-1} I_k^{k-1} A_k \left(S_k\right)^{v_1}$$

With

$$\tilde{Q}_k(v_1,0) = \left(S_k\right)^{v_2}\left\{ I - I_{k-1}^{k}\left(A_{k-1}\right)^{-1} I_k^{k-1} A_k \right\}\left(S_k\right)^{v_1}$$

Then we have

$$\left\| \tilde{Q}_k(v_1,0) \right\| \le C_S C_A \eta(v_1) \quad .$$

choosing a number $\zeta \in (0, \tilde{\zeta})$ with $\tilde{\zeta} = \frac{\gamma - 1}{\gamma}(\gamma C)^{\frac{-1}{\gamma - 1}}$ and a number $\tilde{v}$ such that

$C_S C_A \eta(v_1) < \zeta$, $v_1 \geq \tilde{v}$ and that:

$\zeta_k \leq \zeta + c_p \zeta_{k-1}^{\gamma} C_p (1 + \zeta) \leq \zeta + C \zeta_{k-1}^{\gamma}$, with $C = 2C_p c_p$ and

$\zeta_k = \left\| Q^k (v_1, 0) \right\|$, then it follows that

$$\left\| Q^k (v_1, 0) \right\| = \zeta_k \leq \frac{\gamma}{\gamma - 1} \zeta < 1 \qquad k = 2,3,.....K$$

If necessary v_1 is increased such that: $\zeta \leq \frac{\gamma - 1}{\gamma} \tilde{\zeta}$

4.5 Computational results

In this section, we introduce some numerical results obtained by several researchers. Table 4.3 shows number of iterations and times for the defect reduction of factor 10^{-12} for different cycles and different restriction operator. It is obvious that $V(2,1)$ with HW is the most efficient.

Table 4.3[2]: *V* and *W* cycles

Cycle	FW		HW	
	iterations	Time (msec)	iterations	Time (msec)
$V(0,1)$	26	1290	167	7310
$V(1,1)$	12	759	13	740
$V(2,1)$	10	759	9	629
$V(2,2)$	9	799	8	669
$W(0,1)$	20	2269	34	3780
$W(1,1)$	10	1379	10	1379
$W(2,1)$	9	1450	9	1479
$W(2,2)$	8	1469	8	1460

Table 4.4 shows the infinite norm $\|u - u_h\|_\infty$ of the error for the FMG and *V*-cycles using different grids. It is clear that the FMG produces the least error.

Table 4.4[2]: Infinite error norm

Grid	FMG	$V(0,1)$	$V(1,1)$
32×32	0.31E-5	0.26E-4	0.47E-5
64×64	0.77E-6	0.83E-5	0.12E-5
128×128	0.19E-6	0.27E-5	0.31E-6
256×256	0.48E-7	0.87E-6	0.78E-7

Table 4.5 shows the convergence factor obtained with damped Jacobi and FW for Poisson problem for different sweeps of presmoothing. The convergence factor when $\omega = \frac{4}{5}$ better than the convergence factor when $\omega = \frac{1}{2}$.

Table 4.5[2]: Convergence factor

	$v = 1$	$v = 2$	$v = 3$	$v = 4$
$\omega = \frac{4}{5}$	0.6	0.36	0.216	0.137
$\omega = \frac{1}{2}$	0.75	0.563	0.422	0.316

Table 4.6 shows that the computer time is proportional to *N* where *N* is the number of grid points in each dimension. In other words, the computer time is of order *N*. this means that FMG is of order *N*.

Table 4.6[22]: FMG with GS as smoother

grid	error	CPU time	Ratio
512×512	0.00767841645	36s	
1024×1024	0.00381202826	149s	4.1388889
2048×2048	0.00190166438	598s	4.1342282

4.6 Conclusion

Basic iterative methods such as the Jacobi, Gauss-Seidel , and the SOR methods are used to solve the linear system obtained from the discretization

of the PDE problem. For small linear systems, these methods are efficient but not for large systems. Jacobi and Gauss-Seidel methods (not the SOR) are efficient as smoothers. This means they are efficient in smoothing the error but not efficient in reducing it. Multigrid methods accelerate basic iterative methods by making use of different grids and the smoothing property of some classical methods. Computational results from different sources, shows that multigrid methods are efficient in reducing smooth errors by using coarser grids. The rate of convergence of these methods is independent of the mesh size, a property that makes multigrid methods superior to classical iterative methods.

The following table shows the order of different classical method, as well as, the order of multigrid methods which is linear in N, where N is the number of unknowns and ε is a given stopping criterion (tolerance).

Table 4.7[2]: Number of operations for different solvers for Poisson problem in 2D

Method	Number of operations
Gaussian elimination	$O(N^2)$
Jacobi iteration	$O(N^2 \log \varepsilon)$
Gauss-Seidel iteration	$O(N^2 \log \varepsilon)$
SOR	$O(N^{\frac{3}{2}} \log \varepsilon)$
Multigrid (iterative)	$O(N \log \varepsilon)$
Multigrid (FMG)	$O(N)$

References

1. Wesseling P. **An introduction to Multigrid Methods**. New York: John Wiley & sons, 1992. 284pp.
2. Trottenberg U., Oosterlee C.W. and Schuller A., **Multigrid**. Academic Press, 2001. 631pp
3. Hackbusch W. and Trottenberg U., **Multigrid Methods**. Springer-Verlag, Berlin, 1982. 630pp
4. Briggs W.L., Henson V.E and McCormick S.F., **A Multigrid Tutorial**. 2nd edition. SIAM (Society for Industrial and Applied Mathematics), 2000. 193pp.
5. Walter A. Strauss, **Partial Differential Equations**. John Wiley & sons, 1992. 425pp.
6. Kincaid D., Cheney W., **Numerical Analysis**. Mathematics of Scientific Computing, American Mathematical Society, 1991. 690pp.
7. Wienands R., Joppich W., **Practical Fourier Analysis for Multigrid Methods.** Chapman & Hall / CRC press. 2005. 212pp.
8. Saad Y. **Iterative methods for Sparse Linear Systems**. SIAM (Society for Industrial and Applied Mathematics). 2000. 447pp.
9. Hackbusch W. and Trottenberg U., **Multigrid Methods II**. Springer-Verlag, Berlin, 1986. 335pp.
10. Phillips G.M.M., Taylor P.J., **Theory and Applications of Numerical Analysis**. 2nd edition, Elsevier. 1996. 447pp.
11. Demmel J. **Applied Numerical Linear Algebra**, SIAM (Society for Industrial and Applied Mathematics). 1997. 411pp.
12. Stoer J., Bulisch R., **Introduction to Numerical Analysis.** 2nd edition, VI. New York, 1993. 660pp.

13. Kress R., **Numerical Analysis**. Springer-Verlag. New York, 1998. 326pp.
14. McCormick F.S., **Multigrid Methods**. Marcel Dekker Inc., New York. 1988. 282pp.
15. Rude U., **Fully Adaptive Multigrid Methods**. SIAM (Society for Industrial and Applied Mathematics). 1993. 115pp.
16. Burden R.L., Faires J.D., **Numerical Analysis**. 5th edition. PWS, Boston. 1993. 767pp.
17. Schatzman M., **Numerical Analysis: a Mathematical Introduction**. Oxford University press 2002. 469pp.
18. Stuben K., **Multigrid Tutorial**. SCAI. 28pp.
19. Erkoc S. **Fundamentals of Quantum Mechanics.** CRC press 2006.40pp.
20. Bramble J. H. **Multigrid Methods**. John Wiley & sons, 1993. 178pp.
21. Dick E., Riemslaagh K., Vierendeels J. **Multigrid Methods VI**. Springer-Verlag, Berlin, 2000.292pp.
22. Shi Z., Xue-jun X.U., Huang Y. **Economical cascadic multigrid method.** China press springer. 2007. 16pp.

Appendix

Matlab Code for multigrid methods:

```
%MGLab V0.00beta   Interactive Multigrid Package

% James Bordner and Faisal Saied
% Department of Computer Science
% University of Illinois at Urbana-Champaign
% 10 April 1995

  include_flags
  include_globals
  include_figs
  demo_globals

% Initialize parameter defaults

  set_defaults;

% == MAIN MENU ======================================

bgc = [0.9 0.9 1.0];

main_fig = figure('Position', main_position,...
  'Name', 'MGLab',...
  'NumberTitle', 'off', ...
  'Color','black');

% == MGLab Menu Item ================================

f_mglab=menu_header(main_fig,'MGLab','on','on','w');
   menu_item(f_mglab,'Run', 'off','on',bgc,'[sol1,resids1,its1]=run;');
   menu_item(f_mglab,'Show Params','off','on',bgc,'show_params;');
   menu_item(f_mglab,'Version Info','off','on',bgc,'version_info;');
   menu_item(f_mglab,'Reset','off','on',bgc,'set_defaults;');
   menu_item(f_mglab,'Restart','off','on',bgc,'close(main_fig);            close;
MGLab');
   menu_item(f_mglab,'Quit','off','on',bgc,'close(main_fig); close');
```

```
% == Problem Menu Item ==================================

f_problem=menu_header(main_fig,'Problem','on','on','w');

  menu_item(f_problem,'Poisson','on','on',bgc,...
    'problem_flag = POISSON;generate_matrix=1;');
  f_problem_1 = menu_item(f_problem,'Helmholtz', 'off','on',bgc,...
    'problem_flag = HELMHOLTZ;generate_matrix=1;prob_args(1) = -
10;');
    f_problem_11=menu_header(f_problem_1,'k = ','on','on','w');
      menu_item(f_problem_11,'-10','off','on',bgc,...
        'prob_args(1)=-10;');
      menu_item(f_problem_11,'-5','off','on',bgc,...
        'prob_args(1)=-5;');
      menu_item(f_problem_11,'-1','off','on',bgc,...
        'prob_args(1)=-1;');
      menu_item(f_problem_11,'0','off','on',bgc,...
        'prob_args(1)=0;');
      menu_item(f_problem_11,'1','off','on',bgc,...
        'prob_args(1)=1;');
      menu_item(f_problem_11,'5','off','on',bgc,...
        'prob_args(1)=5;');
      menu_item(f_problem_11,'10','off','on',bgc,...
        'prob_args(1)=10;');
      menu_item(f_problem_11,'10+ i','off','on',bgc,...
        'prob_args(1)=10+sqrt(-1);');
  f_problem_2 = menu_item(f_problem,'Convection-Diffusion',
'off','on',bgc,...
    'problem_flag=CONVECT_DIFFUSE;generate_matrix=1;');
    f_problem_21=menu_header(f_problem_2,'Lambda = ','on','on','w');
      menu_item(f_problem_21,'0','off','on',bgc,...
        'prob_args(1)=0;');
      menu_item(f_problem_21,'10','off','on',bgc,...
        'prob_args(1)=10;');
      menu_item(f_problem_21,'100','off','on',bgc,...
        'prob_args(1)=100;');
      menu_item(f_problem_21,'1000','off','on',bgc,...
        'prob_args(1)=1000;');
    f_problem_22=menu_header(f_problem_2,'Sigma = ','on','on','w');
      menu_item(f_problem_22,'0','on','on',bgc,...
        'prob_args(2)=0;');
```

```
      menu_item(f_problem_22,'5','off','on',bgc,...
         'prob_args(2)=5;');
      menu_item(f_problem_22,'10','off','on',bgc,...
         'prob_args(2)=10;');
      menu_item(f_problem_22,'20','off','on',bgc,...
         'prob_args(2)=20;');
      menu_item(f_problem_22,'50','off','on',bgc,...
         'prob_args(2)=50;');
      menu_item(f_problem_22,'100','off','on',bgc,...
         'prob_args(2)=100;');
      menu_item(f_problem_22,'-50','off','on',bgc,...
         'prob_args(2)=-50;');
      menu_item(f_problem_22,'-100','off','on',bgc,...
         'prob_args(2)=-100;');
  f_problem_3=menu_item(f_problem,'Cut Square', 'off','on',bgc,...
    'problem_flag   =   CUT_SQUARE;generate_matrix=1;prob_args(1)   =
10;');
    f_problem_31=menu_header(f_problem_3,'Alpha = ','on','on','w');
      menu_item(f_problem_31,'0.001','off','on',bgc,...
         'prob_args(1)=0.001;');
      menu_item(f_problem_31,'0.01','off','on',bgc,...
         'prob_args(1)=0.01;');
      menu_item(f_problem_31,'0.1','off','on',bgc,...
         'prob_args(1)=0.1;');
      menu_item(f_problem_31,'1','off','on',bgc,...
         'prob_args(1)=1;');
      menu_item(f_problem_31,'10','off','on',bgc,...
         'prob_args(1)=10;');
      menu_item(f_problem_31,'100','off','on',bgc,...
         'prob_args(1)=100;');
      menu_item(f_problem_31,'1000','off','on',bgc,...
         'prob_args(1)=1000;');
  menu_item(f_problem,'Poisson-Boltzmann', 'off','off',bgc,...
    'problem_flag=POISSON_BOLTZMAN;generate_matrix=1;');
  f_problem_4=menu_header(f_problem,'Problem Size','off','on','w');
     menu_item(f_problem_4,'  7   ','off','on',bgc,...
          [['nx1=7;ny1=7;generate_matrix=1;generate_rhs=1;'];...
           ['coarse_level=min([coarse_level max_level(nx1)]);']']);
     menu_item(f_problem_4,' 15   ','off','on',bgc,...
          [['nx1=15;ny1=15;generate_matrix=1;generate_rhs=1;'];...
           ['coarse_level=min([coarse_level max_level(nx1)]);']']);
```

```
    menu_item(f_problem_4,' 31   ','off','on',bgc,...
          [['nx1=31;ny1=31;generate_matrix=1;generate_rhs=1;'];...
           ['coarse_level=min([coarse_level max_level(nx1)]);']']);
    menu_item(f_problem_4,' 63   ','off','on',bgc,...
          [['nx1=63;ny1=63;generate_matrix=1;generate_rhs=1;'];...
           ['coarse_level=min([coarse_level max_level(nx1)]);']']);
    menu_item(f_problem_4,'127   ','off','on',bgc,...
          [['nx1=127;ny1=127;generate_matrix=1;generate_rhs=1;'];...
           ['coarse_level=min([coarse_level max_level(nx1)]);']']);
    menu_item(f_problem_4,'255   ','off','on',bgc,...
          [['nx1=255;ny1=255;generate_matrix=1;generate_rhs=1;'];...
           ['coarse_level=min([coarse_level max_level(nx1)]);']']);

% == Solver Menu Item ==================================

f_solver=menu_header(main_fig,'Solver','on','on','w');

  menu_item(f_solver,'V-Cycle','off','on',bgc,...
     'solver_flag = VMG;');
  menu_item(f_solver,'PCG','off','on',bgc,...
     'solver_flag = PCG;');
  menu_item(f_solver,'BiCG-STAB','off','on',bgc,...
     'solver_flag = BICG_STAB;');
  menu_item(f_solver,'CGS','off','on',bgc,...
     'solver_flag = CGS;');
  menu_item(f_solver,'TFQMR','off','off',bgc,...
     'solver_flag = TFQMR;');

  f_solver_1=menu_item(f_solver,'GMRES(k)','off','on',bgc,...
     'solver_flag = GMRES;');
    f_solver_11=menu_header(f_solver_1,'k = ','on','on','w');
      menu_item(f_solver_11,'1','off','on',bgc,'restart=1;');
      menu_item(f_solver_11,'5','off','on',bgc,'restart=5;');
      menu_item(f_solver_11,'10','off','on',bgc,'restart=10;');
      menu_item(f_solver_11,'15','off','on',bgc,'restart=15;');
      menu_item(f_solver_11,'20','off','on',bgc,'restart=20;');

  f_solver_2 = menu_item(f_solver,'SOR','off','on',bgc,...
     'solver_flag = SOR;');
    f_solver_21=menu_header(f_solver_2,'omega = ','on','on','w');
      menu_item(f_solver_21,'1','off','on',bgc,'SOR_omega=1;');
```

```
        menu_item(f_solver_21,'1.1','off','on',bgc,'SOR_omega=1.1;');
        menu_item(f_solver_21,'1.2','off','on',bgc,'SOR_omega=1.2;');
        menu_item(f_solver_21,'1.3','off','on',bgc,'SOR_omega=1.3;');
        menu_item(f_solver_21,'1.4','off','on',bgc,'SOR_omega=1.4;');
        menu_item(f_solver_21,'1.5','off','on',bgc,'SOR_omega=1.5;');
        menu_item(f_solver_21,'1.6','off','on',bgc,'SOR_omega=1.6;');
        menu_item(f_solver_21,'1.7','off','on',bgc,'SOR_omega=1.7;');
        menu_item(f_solver_21,'1.8','off','on',bgc,'SOR_omega=1.8;');
        menu_item(f_solver_21,'1.9','off','on',bgc,'SOR_omega=1.9;');

    menu_item(f_solver,'Full-Multigrid','on','on',bgc,...
      'solver_flag = FMG;');

    f_solver_precon=menu_header(f_solver,'Preconditioner','on','on','w');
        menu_item(f_solver_precon,'V-Cycle','off','on',bgc,...
            'precon_flag = MG_CYCLE;');
        menu_item(f_solver_precon,'Jacobi','off','on',bgc,...
            'precon_flag = JACOBI;');
        menu_item(f_solver_precon,'Block-Jacobi','off','off',bgc,...
            'precon_flag = BLOCK_JACOBI;');
        menu_item(f_solver_precon,'Gauss-Seidel','off','on',bgc,...
            'precon_flag = GAUSS_SEIDEL;');
        menu_item(f_solver_precon,'ILU','off','off',bgc,...
            'precon_flag = ILU');
        menu_item(f_solver_precon,'SSOR','off','off',bgc,...
            'precon_flag = SSOR');
        menu_item(f_solver_precon,'None','off','on',bgc,...
            'precon_flag = NONE;');

    f_solver_stop=menu_header(f_solver,'Stopping Criteria','off','on','w');
      f_stop_1=menu_header(f_solver_stop,'Residual
Tolerance','on','off','w');
        menu_item(f_stop_1,'None','off','on',bgc,'rtol=0;');
        menu_item(f_stop_1,'1e-1','off','on',bgc,'rtol=1e-1;');
        menu_item(f_stop_1,'1e-2','off','on',bgc,'rtol=1e-2;');
        menu_item(f_stop_1,'1e-3','off','on',bgc,'rtol=1e-3;');
        menu_item(f_stop_1,'1e-4','off','on',bgc,'rtol=1e-4;');
        menu_item(f_stop_1,'1e-5','off','on',bgc,'rtol=1e-5;');
        menu_item(f_stop_1,'1e-6','off','on',bgc,'rtol=1e-6;');
        menu_item(f_stop_1,'1e-7','off','on',bgc,'rtol=1e-7;');
        menu_item(f_stop_1,'1e-8','off','on',bgc,'rtol=1e-8;');
```

```
  menu_item(f_stop_1,'1e-9','off','on',bgc,'rtol=1e-9;');
  menu_item(f_stop_1,'1e-10','off','on',bgc,'rtol=1e-10;');
  menu_item(f_stop_1,'1e-12','off','on',bgc,'rtol=1e-12;');
  menu_item(f_stop_1,'1e-14','off','on',bgc,'rtol=1e-14;');
  menu_item(f_stop_1,'1e-16','off','on',bgc,'rtol=1e-16;');
f_stop_2=menu_header(f_solver_stop,'(Precon)Residual Tolerance',...
    'off','on','w');
  menu_item(f_stop_2,'None','off','on',bgc,'prtol=0;');
  menu_item(f_stop_2,'1e-1','off','on',bgc,'prtol=1e-1;');
  menu_item(f_stop_2,'1e-2','off','on',bgc,'prtol=1e-2;');
  menu_item(f_stop_2,'1e-3','off','on',bgc,'prtol=1e-3;');
  menu_item(f_stop_2,'1e-4','off','on',bgc,'prtol=1e-4;');
  menu_item(f_stop_2,'1e-5','off','on',bgc,'prtol=1e-5;');
  menu_item(f_stop_2,'1e-6','off','on',bgc,'prtol=1e-6;');
  menu_item(f_stop_2,'1e-7','off','on',bgc,'prtol=1e-7;');
  menu_item(f_stop_2,'1e-8','off','on',bgc,'prtol=1e-8;');
  menu_item(f_stop_2,'1e-9','off','on',bgc,'prtol=1e-9;');
  menu_item(f_stop_2,'1e-10','off','on',bgc,'prtol=1e-10;');
  menu_item(f_stop_2,'1e-12','off','on',bgc,'prtol=1e-12;');
  menu_item(f_stop_2,'1e-14','off','on',bgc,'prtol=1e-14;');
  menu_item(f_stop_2,'1e-16','off','on',bgc,'prtol=1e-16;');

f_stop_3=menu_header(f_solver_stop,'Iteration Limit','off','on','w');
  menu_item(f_stop_3,' None','off','on',bgc,'max_it=0;');
  menu_item(f_stop_3,'    1','off','on',bgc,'max_it=1;');
  menu_item(f_stop_3,'    2','off','on',bgc,'max_it=2;');
  menu_item(f_stop_3,'    3','off','on',bgc,'max_it=3;');
  menu_item(f_stop_3,'    5','off','on',bgc,'max_it=5;');
  menu_item(f_stop_3,'   10','off','on',bgc,'max_it=10;');
  menu_item(f_stop_3,'   20','off','on',bgc,'max_it=20;');
  menu_item(f_stop_3,'   30','off','on',bgc,'max_it=30;');
  menu_item(f_stop_3,'   50','off','on',bgc,'max_it=50;');
  menu_item(f_stop_3,'  100','off','on',bgc,'max_it=100;');
  menu_item(f_stop_3,'  200','off','on',bgc,'max_it=200;');
  menu_item(f_stop_3,'  300','off','on',bgc,'max_it=300;');
  menu_item(f_stop_3,'  500','off','on',bgc,'max_it=500;');
  menu_item(f_stop_3,' 1000','off','on',bgc,'max_it=1000;');
f_stop_4=menu_header(f_solver_stop,'Time Limit','off','off','w');
  menu_item(f_stop_4,'None','off','on',bgc,'max_time=0;');
  menu_item(f_stop_4,'1 sec','off','on',bgc,'max_time=1;');
  menu_item(f_stop_4,'5 sec','off','on',bgc,'max_time=5;');
```

```
      menu_item(f_stop_4,'10 sec','off','on',bgc,'max_time=10;');
      menu_item(f_stop_4,'30 sec','off','on',bgc,'max_time=30;');
      menu_item(f_stop_4,'1 min','off','on',bgc,'max_time=1*60;');
      menu_item(f_stop_4,'5 min','off','on',bgc,'max_time=5*60;');
      menu_item(f_stop_4,'10 min','off','on',bgc,'max_time=10*60;');
      menu_item(f_stop_4,'30 min','off','on',bgc,'max_time=30*60;');
      menu_item(f_stop_4,'1 hour','off','on',bgc,'max_time=60*60;');
    f_stop_5=menu_header(f_solver_stop,'MFlop Limit','off','off','w');
      menu_item(f_stop_5,'None','off','on',bgc,'max_mflop=0;');
      menu_item(f_stop_5,'   1','off','on',bgc,'max_mflop=1;');
      menu_item(f_stop_5,'   5','off','on',bgc,'max_mflop=5;');
      menu_item(f_stop_5,'  10','off','on',bgc,'max_mflop=10;');
      menu_item(f_stop_5,'  20','off','on',bgc,'max_mflop=20;');
      menu_item(f_stop_5,'  50','off','on',bgc,'max_mflop=50;');
      menu_item(f_stop_5,' 100','off','on',bgc,'max_mflop=100;');

% == MG Parameters ==================================

  f_solver_mg=menu_header(main_fig,'MG-Parameters','on','on','w');
    f_mg_1 = menu_header(f_solver_mg,'Number of Levels','on','on','w');
      menu_item(f_mg_1,'1','off','on',bgc,...
        'coarse_level=min([1,max_level(nx1)]); generate_matrix=1;');
      menu_item(f_mg_1,'2','off','on',bgc,...
        'coarse_level=min([2,max_level(nx1)]); generate_matrix=1;');
      menu_item(f_mg_1,'3','off','on',bgc,...
        'coarse_level=min([3,max_level(nx1)]); generate_matrix=1;');
      menu_item(f_mg_1,'4','off','on',bgc,...
        'coarse_level=min([4,max_level(nx1)]); generate_matrix=1;');
      menu_item(f_mg_1,'5','off','on',bgc,...
        'coarse_level=min([5,max_level(nx1)]); generate_matrix=1;');
    f_mg_2=menu_header(f_solver_mg,'Smoother','off','on','w');
      f_mg_21=menu_item(f_mg_2,'Weighted Jacobi','on','on',bgc,...
          'smooth_flag=WEIGHTED_JACOBI;');
       f_mg_211=menu_header(f_mg_21,'Weight = ','on','on','w');
          menu_item(f_mg_211,'1.00','off','on',bgc,'wt=1.0;');
          menu_item(f_mg_211,'0.95','off','on',bgc,'wt=0.95;');
          menu_item(f_mg_211,'0.90','off','on',bgc,'wt=0.90;');
          menu_item(f_mg_211,'0.85','off','on',bgc,'wt=0.85;');
          menu_item(f_mg_211,'0.80','off','on',bgc,'wt=0.80;');
      menu_item(f_mg_2, 'Gauss-Seidel','off','on',bgc,...
          'smooth_flag=GAUSS_SEIDEL;');
```

```
  menu_item(f_mg_2, 'Red/Black Gauss-Seidel','off','off',bgc,...
      'smooth_flag=RB_GAUSS_SEIDEL;');
  f_mg_22=menu_header(f_mg_2,'Pre-smoothings','on','on','w');
    menu_item(f_mg_22,'0','off','on','w','nu1=0;');
    menu_item(f_mg_22,'1','off','on','w','nu1=1;');
    menu_item(f_mg_22,'2','off','on','w','nu1=2;');
    menu_item(f_mg_22,'3','off','on','w','nu1=3;');
    menu_item(f_mg_22,'4','off','on','w','nu1=4;');
    menu_item(f_mg_22,'5','off','on','w','nu1=5;');
  f_mg_23=menu_header(f_mg_2,'Post-smoothings','off','on','w');
    menu_item(f_mg_23,'0','off','on','w','nu2=0;');
    menu_item(f_mg_23,'1','off','on','w','nu2=1;');
    menu_item(f_mg_23,'2','off','on','w','nu2=2;');
    menu_item(f_mg_23,'3','off','on','w','nu2=3;');
    menu_item(f_mg_23,'4','off','on','w','nu2=4;');
    menu_item(f_mg_23,'5','off','on','w','nu2=5;');

f_mg_3=menu_header(f_solver_mg,'Restriction','off','on','w');
  menu_item(f_mg_3, 'Injection','off','on',bgc,...
    'restrict_flag=INJECTION;');
  menu_item(f_mg_3, 'Half Weighting','off','on',bgc,...
    'restrict_flag=HALF_WEIGHTING;');
  menu_item(f_mg_3, 'Full Weighting','off','on',bgc,...
    'restrict_flag=FULL_WEIGHTING;');
  menu_item(f_mg_3, 'Bilinear Adjoint','off','off',bgc,...
    'restrict_flag=BILINEAR_ADJOINT;');

f_mg_4=menu_header(f_solver_mg,'Prolongation','off','on','w');
  menu_item(f_mg_4, 'Linear','off','on',bgc,...
    'interp_flag=LINEAR;');
  menu_item(f_mg_4, 'Cubic','off','on',bgc,...
    'interp_flag=CUBIC;');
  menu_item(f_mg_4, 'Operator-based','off','off',bgc,...
    'interp_flag=OPERATOR_BASED;');
  menu_item(f_mg_4, 'Explicit/Bilinear','off','off',bgc,...
    'interp_flag=EXPLICIT_BILINEAR;');

f_mg_5=menu_header(f_solver_mg,'Coarse-grid Solver','off','on','w');
  menu_item(f_mg_5,'Sparse GE','off','on',bgc,...
    'coarse_solver_flag=DIRECT;');
  menu_item(f_mg_5,'Smoother','off','on',bgc,...
```

```
        'coarse_solver_flag=SMOOTHER;');
      menu_item(f_mg_5,'PCG','off','off',bgc,...
        'coarse_solver_flag = PCG;');
      menu_item(f_mg_5,'BiCG-STAB','off','off',bgc,...
        'coarse_solver_flag = BICG_STAB;');
      f_mg_51=menu_item(f_mg_5,'GMRES(k)','off','off',bgc,...
         'coarse_solver_flag = GMRES;');
        f_mg_511=menu_header(f_mg_51,'k = ','on','on','w');
          menu_item(f_mg_511,'1','off','on',bgc,'restart=1;');
          menu_item(f_mg_511,'5','off','on',bgc,'restart=5;');
          menu_item(f_mg_511,'10','off','on',bgc,'restart=10;');
          menu_item(f_mg_511,'15','off','on',bgc,'restart=15;');
          menu_item(f_mg_511,'20','off','on',bgc,'restart=20;');

    f_mg_6=menu_header(f_solver_mg,'Coarse-grid
Operator','off','on','w');
      menu_item(f_mg_6,'Standard 5pt','off','on',bgc,...
        'coarsening_flag=STANDARD;');
      menu_item(f_mg_6,'Galerkin coarsening','off','off',bgc,...
        'coarsening_flag=GALERKIN;');
      menu_item(f_mg_6,'Coeff. Averaging','off','off',bgc,...
        'coarsening_flag = AVERAGING;');

    f_mg_7=menu_header(f_solver_mg,'MG Cycle','off','on','w');
      menu_item(f_mg_7,'V-Cycle','off','on',bgc,...
        'cycle_flag=V_CYCLE;');
      menu_item(f_mg_7,'W-Cycle','off','on',bgc,...
        'cycle_flag=W_CYCLE;');
      menu_item(f_mg_7,'Half V-Cycle','off','off',bgc,...
        'cycle_flag=HALF_V_CYCLE;');

% == Results Menu Item ==================================

f_results=menu_header(main_fig,'Visualize','on','on','w');

  menu_item(f_results,'Convergence History','off','on',bgc,...
    ' subplot(1,1,1);semilogy(its1,resids1,"r-",its1,resids1,"wo")');
  menu_item(f_results,'Computed Solution (surf)','off','on',bgc,...
    ' subplot(1,1,1);surf(reshape(sol1,nx1,ny1));shading interp;');
  menu_item(f_results,'Computed Solution (pcolor)','off','on',bgc,...
    ' subplot(1,1,1);pcolor(reshape(sol1,nx1,ny1));shading interp;');
```

```
  f_results_1=menu_header(f_results,'X-Axis','off','on','w');
     menu_item(f_results_1,'Iterations','off','on',bgc,...
       'x_axis_flag=ITERATIONS;');
     menu_item(f_results_1,'Time','off','off',bgc,...
       'x_axis_flag=TIME;');
     menu_item(f_results_1,'MFlops','off','off',bgc,...
       'x_axis_flag=MFLOPS;');
  f_results_2=menu_header(f_results,'Y-Axis','off','on','w');
     menu_item(f_results_2,'Residual','off','on',bgc,...
       'y_axis_flag=ITERATIONS;');
     menu_item(f_results_2,'Precon. Residual','off','off',bgc,...
       'y_axis_flag=RESIDUAL;');
     menu_item(f_results_2,'MFlops','off','off',bgc,...
       'y_axis_flag=PRECON_RESIDUAL;');

f_demos=menu_header(main_fig,'Demos','on','on','w');
  menu_item(f_demos,'Smoothers','off','on',bgc,'demo1;');
  menu_item(f_demos,'Fourier analysis','off','on',bgc,'demo2;');
  menu_item(f_demos,'Truncation error','off','on',bgc,'demo3;');
%MG_CYCLE Multigrid cycle algorithm
%
%       U_OUT = MG_CYCLE(LEVEL, B, U_IN) uses the multigrid cycle
defined
%      by the global variable "cycle_flag" to recursively solve the linear
%      system AX=B at the given level.  If the optional starting value U_IN
%      is not passed then U_IN is set to 0's.
%
%      Accesses global variables in "include_flags"

% James Bordner and Faisal Saied
% Department of Computer Science
% University of Illinois at Urbana-Champaign
% 10 April 1995

function u_out = mg_cycle(level, b, u_in)

include_flags

% Use the zero vector for u_in as the default

if nargin == 2,
```

```
  u_in = zeros(size(b));
end

if (cycle_flag == V_CYCLE)
   u_out = vmg_cycle(level, b, u_in);
elseif (cycle_flag == W_CYCLE)
   u_out = wmg_cycle(level, b, u_in);
elseif (cycle_flag == HALF_V_CYCLE)
   u_out = halfvmg_cycle(level, b, u_in);
end
%RESIDUAL Compute the residual at the given level.
%
%      R = RESIDUAL(LEVEL, B, U) returns the residual R of the system
%      AU=B at the given grid level.
%
%      Accesses global variables in "include_globals"

% James Bordner and Faisal Saied
% Department of Computer Science
% University of Illinois at Urbana-Champaign
% 10 April 1995

function   r = residual(level, b, u)

include_globals

eval(['r = b - A', num2str(level), ' * u;']);
%RESTRICT Transfer residual from the current grid to the next coarser
grid.
%
%            RHS_C = RESTRICT(LEVEL,R) uses the restriction scheme
defined by
%      "restrict_flag" to transfer the vector R on the current level LEVEL
%      to the vector RHS_C on the next coarser level LEVEL+1.
%
%      Accesses global variables in "include_flags"
%      Accesses global variables in "include_globals"

% James Bordner and Faisal Saied
% Department of Computer Science
% University of Illinois at Urbana-Champaign
```

```
% 10 April 1995

function rhs_c = restrict(level,r)
include_globals
extract_globals
include_flags

%  2-D RESTRICTIONS

   nx0_f = nx_f+2;
   ny0_f = ny_f+2;
   N0_f = nx0_f*ny0_f;
   dx=1;
   dy=nx0_f;

%  Generate r0 by padding r with boundary elements (0's)

   r0 = zeros(N0_f,1);
   for iy=1:ny_f
   for ix=1:nx_f
      r0(nx0_f+1 + ix + nx0_f*(iy-1)) = r(ix+nx_f*(iy-1));
   end
   end

%  Generate indicies of corresponding coarse vector elements in fine vector

   I = zeros(N_c,1);
   for iy=1:ny_c
   for ix=1:nx_c
      I(ix + nx_c*(iy-1)) = 2*ix + 2*iy*nx0_f + 1;
   end
   end

if restrict_flag == INJECTION

   rhs_c = r0(I);

elseif restrict_flag == HALF_WEIGHTING

   rhs_c = .5*r0(I) + ...
         .125*(r0(I+dx) + r0(I-dx) + r0(I+dy) + r0(I-dy));
```

```
elseif restrict_flag == FULL_WEIGHTING
  rhs_c = .25*r0(I) + ...
        .125*(r0(I+dx) + r0(I-dx) + r0(I+dy) + r0(I-dy)) + ...
        .0625*(r0(I+dx+dy) + r0(I-dx+dy) + r0(I+dx-dy) + r0(I-dx-dy));

elseif restrict_flag == BILINEAR_ADJOINT

  eval(['PROLONG = ARRAY',num2str(level), ';']);
  rhs_c = PROLONG' * r;

end

rhs_c = 4*rhs_c;

%SMOOTH Smooth a vector.
%
%         U_OUT = SMOOTH(LEVEL, B, U, FLAG) applies a smoother
defined by the
%       global flag "smooth_flag" and the system AU=B to the vector U on
the
%      given grid level.  FLAG is set to 'pre', 'post', or 'coarse' and
%      defines the number of smoothings applied.
%
%      Accesses global variables in "include_globals"
%      Accesses global variables in "include_flags"

% James Bordner and Faisal Saied
% Department of Computer Science
% University of Illinois at Urbana-Champaign
% 10 April 1995

function   u_out = smooth(level, b, u, flag)

include_globals
include_flags

if strcmp(flag, 'pre') == 1
  nu = nu1;
elseif  strcmp(flag, 'post') == 1
  nu = nu2;
```

```
elseif strcmp(flag, 'coarse') == 1
  nu = 30;
end

eval(['A = A',num2str(level),';']);

if smooth_flag == WEIGHTED_JACOBI

  D = wt * (1./spdiags(A,[0]));
  for i = 1:nu
     u = u + D.*(b - A*u);
  end

elseif smooth_flag == GAUSS_SEIDEL

  L = tril(A);
  for i = 1:nu
     u = u + L\(b - A*u);
  end

elseif smooth_flag == RB_GAUSS_SEIDEL

  eval(['N = N',num2str(level),';']);
  red = [1:2:N]; black = [2:2:N];
  D = 1./spdiags(A,[0]);

  for i = 1:nu
    u(red)   = (b(red) - A(red,black) * u(black)) .* D(red);
    u(black) = (b(black) - A(black,red) * u(red)) .* D(black);
  end

end

u_out = u;

%SOLVE  Solve a linear system.
%
%[X,RESIDS,ITS]=
SOLVE(A,B,X0,RTOL,PRTOL,MAX_IT,MAX_TIME,MAX_MFLOP,...
%       RESTART) applies a solver defined by "solver_flag", with the given
%       tolerances and limits, to a linear system AX=B.  The solution X,
```

```
%       residual history RESIDS, and iterations ITS are returned.
%       Accesses global variables in "include_flags"

% James Bordner and Faisal Saied
% Department of Computer Science
% University of Illinois at Urbana-Champaign
% 10 April 1995

function [x,resids,its] = solve(A,b,x0,...
   rtol,prtol,max_it,max_time,max_mflop,restart)

include_flags

disp (sprintf('Running...\n'));
if solver_flag == VMG
   [x,resids,its] = vmg (A,b,x0,rtol,prtol,max_it,max_time,max_mflop);
elseif solver_flag == FMG
   [x,resids,its] = fmg (A,b);
elseif solver_flag == PCG
   [x,resids,its] = pcg (A,b,x0,rtol,prtol,max_it,max_time,max_mflop);
elseif solver_flag == BICG_STAB
   [x,resids,its]                    =                    pbicgstab
(A,b,x0,rtol,prtol,max_it,max_time,max_mflop);
elseif solver_flag == CGS
   [x,resids,its] = pcgs (A,b,x0,rtol,prtol,max_it,max_time,max_mflop);
elseif solver_flag == GMRES
   [x,resids,its]                    =                    pgmres
(A,b,x0,rtol,prtol,max_it,max_time,max_mflop,restart);
elseif solver_flag == SOR
   [x,resids,its] = sor (A,b,x0,rtol,prtol,max_it,max_time,max_mflop);
end

fprintf('Relative residual = %g \n', norm(b-A*x))
disp (sprintf('Done.\n'));
%VMG_CYCLE V-Cycle algorithm.
% B,
%          U_OUT = VMG_CYCLE(LEVEL,U_IN) uses the V-cycle to
recursively
%       solve the linear system AX=B at the given level.  If the optional
%       starting value U_IN is not passed then U_IN is set to 0's.
%
```

```
% James Bordner and Faisal Saied
% Department of Computer Science
% University of Illinois at Urbana-Champaign
% 10 April 1995

function u_out = vmg_cycle(level, b, u_in)

% Use the zero vector for u_in as the default

if nargin == 2,
   u_in = zeros(size(b));
end

if level == coarsest(level)
   u_out   = coarse_grid_solve(level, b);
else
   u       = smooth(level, b, u_in, 'pre');
   r       = residual(level, b, u);
   b_c     = restrict(level, r);
   u_c     = vmg_cycle(level+1, b_c);
   correct = interpolate(level, u_c);
   u       = u + correct;
   u_out   = smooth(level, b, u, 'post');
end
%WMG_CYCLE W-Cycle algorithm.
%
%         U_OUT = WMG_CYCLE(LEVEL, B, U_IN) uses the W-cycle to
recursively
%       solve the linear system AX=B at the given level.  If the optional
%       starting value U_IN is not passed then U_IN is set to 0's.
%

% James Bordner and Faisal Saied
% Department of Computer Science
% University of Illinois at Urbana-Champaign
% 10 April 1995

function u_out = wmg_cycle(level, b, u_in)
% Use the zero vector for u_in as the default

if nargin == 2,
```

```
   u_in = zeros(size(b));
end

if level == coarsest(level)
   u_out   = coarse_grid_solve(level, b);
else
   u       = smooth(level, b, u_in, 'pre');
   r       = residual(level, b, u);
   b_c     = restrict(level, r);
   u_c     = wmg_cycle(level+1, b_c);
   if (level < coarsest(level)),
      u_c     = wmg_cycle(level+1, b_c, u_c);
   end
   correct = interpolate(level, u_c);
   u       = u + correct;
   u_out   = smooth(level, b, u, 'post');
end
```

جامعة النجاح الوطنية
كلية الدراسات العليا

طرق متعددة الشبكات للمعادلات التفاضلية الجزئية الناقصة

إعداد
رانية طالب محمد ونان

إشراف
د. أنور صالح

قدمتْ هذه الأطروحة استكمالا لمتطلبات درجة الماجستير في الرياضيات بكلية الدراسات العليا في جامعة النجاح الوطنية في نابلس. فلسطين.

2010

طرق متعددة الشبكات للمعادلات التفاضلية الجزئية الناقصة

إعداد

رانية طالب محمد ونان

إشراف

د. أنور صالح

الملخص

المعادلات التفاضلية الجزئية تظهر في الأنظمة الرياضية التي تصف الظواهر الطبيعية. طرق مختلفة يمكن استعمالها لحل مثل هذه المعادلات. في هذه الاطروحة ستتم مراجعة عامة للطرق التقليدية وكذلك الطرق المتعددة الشبكات الأحدث . الطرق التقليدية المستخدمة هي طريقة جاكوبي وطريقة جاوس–سايدل و طريقة SOR . طريقة جاكوبي وطريقة جاوس–سايدل تعتبر جيدة في تنعيم الخطأ ولكن ليس في تصغيره، صفة التنعيم حفزت العمل على الطرق متعددة الشبكات.

معادلة بواسون في البعدين الاول والثاني استخدمت كنموذج لهذه الدراسة. هذه الدراسة بينت ان سرعة التقارب لهذه الطرق لا تعتمد على البعد بين النقاط. هذه الخاصية جعلت الطرق متعددة الشبكات مسَرع جيد للطرق التقليدية.