University Course

Math 228A

Numerical Solution of Differential Equations

University of California, Davis

Fall 2010

My Class Notes

Nasser M. Abbasi

Fall 2010

Contents

Chapter 1

Introduction

I took this course in Fall 2010 to learn about numerical solutions of PDE's.

course description from catalog

228A-228B-228C. Numerical Solution of Differential Equations (4-4-4) Lecture 3 hours; term paper or discussion 1 hour. Prerequisite: course 128C. Numerical solutions of initial-value, eigenvalue and boundary-value problems for ordinary differential equations. Numerical solution of parabolic and hyperbolic partial differential equations. Offered in alternate years

official class syllabus

See Professor Guy web page HTML In case the syllabus goes away, here is an archive image

Math 228A
Numerical Methods for PDEs
Fall Quarter 2010

Instructor: Professor Bob Guy
Office: MSB 2136
Email: guy@math.ucdavis.edu
Phone: 754-9201
Office Hours: Tuesday 3-4
Thursday 3-4

Textbook:

1. R. J. LeVeque. Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems. SIAM, 2007.
2. W. Briggs, V.E. Henson, and S. McCormick. A Multigrid Tutorial, Second Edition. SIAM, 2000.

Both books are published by SIAM. SIAM members receive a 30% discount on orders. Free membership in SIAM is available by joining the UCD SIAM student chapter. http://siam.math.ucdavis.edu

Webpage: http://www.math.ucdavis.edu/~guy/teaching/228a/
Homework and announcements will be posted here.

Class: Tuesday and Thursday 1:40-3:00 in Physics 140

TA: Hsiao-Chieh (Arcade) Tseng
Office: MSB 2123
Email: hctseng@math.ucdavis.edu
Office Hours: Monday 1:30-2:30
Friday 1:30-2:30

Homework
You are encouraged to talk with your classmates about homework problems. However, you must do your own write-up and write your own codes. All aspects of your write up must be clearly presented. Your writing should be clear and grammatically correct. Your codes must be thoroughly commented. All tables and figures must be appropriately labeled. You will be graded on the quality of your presentation.

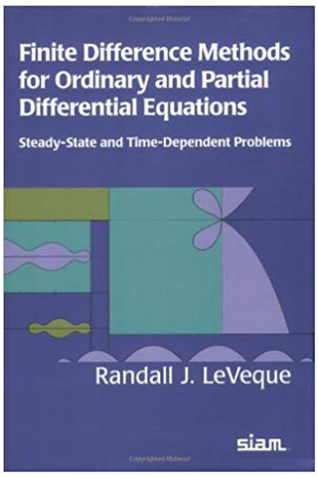
What we will cover
This coarse is part of the sequence 228A-C on numerical methods for partial differential equations. The first quarter (228A) will focus on elliptic problems. The topics we will cover this quarter are listed below.

- Introduction to Poisson equation
- Finite differences
- Convergence and accuracy
- Iterative methods
- Multigrid
- Krylov subspace methods, CG & GMRES
- FFT

Grading
Your grade will be based on your homework assignments. We will likely have 4 or 5 homework assignments during this quarter.

Programing
This class will require writing computer programs. You may use any language. If you do not have a strong preference of language, it is recommended that you use MATLAB, because it is easy to use and very powerful. All codes will be turned in and must be thoroughly commented. You can create a computer account in the math department at http://www.math.ucdavis.edu/comp/class-accts.

Text book


Finite Difference Methods
for Ordinary and Partial
Differential Equations
Steady-State and Time-Dependent Problems
Randall J. LeVeque
siam

Chapter 2

HWs

Local contents

2.1 Summary of HW's

HW	description	grade
1	Poisson pde, eigenvalue problem, refinment study 1D	5/5
2	Refinment studies, 1D, LTE, convergence	5/5
3	2D, Jacobi, SOR, Gauss-Seidel, direct solver, sparse matrices	5/5
4	Multigrid V cycle algorithm, using to solve Poisson 2D	5/5
5	Conjugate Gradient	5/5

2.2 HW 1

2.2.1 Problem description

Math 228A
Homework 1
Due Tuesday, 10/12/10

1. Let L be the linear operator $Lu = u_{xx}$, $u_x(0) = u_x(1) = 0$.
   (a) Find the eigenfunctions and corresponding eigenvalues of L.
   (b) Show that the eigenfunctions are orthogonal in the $L^2[0,1]$ inner product:
   $$\langle u, v\rangle = \int_0^1 uv\,dx.$$
   (c) It can be shown that the eigenfunctions, $\phi_j(x)$, form a complete set in $L^2[0,1]$. This means that for any $f \in L^2[0,1]$, $f(x) = \sum_j \alpha_j \phi_j(x)$. Express the solution to
   $$u_{xx} = f, \quad u_x(0) = u_x(1) = 0, \tag{1}$$
   as a series solution of the eigenfunctions.
   (d) Note that equation (1) does not have a solution for all f. Express the condition for existence of a solution in terms of the eigenfunctions of L.
2. Define the functional $F : X \to \Re$ by
   $$F(u) = \int_0^1 \frac{1}{2}(u_x)^2 + fu\,dx,$$
   where X is the space of real valued functions on $[0,1]$ that have at least one continuous derivative and are zero at $x = 0$ and $x = 1$. The Frechet derivative of F at a point u is defined to be the linear operator $F'(u)$ for which
   $$F(u+v) = F(u) + F'(u)v + R(v),$$
   where
   $$\lim_{||v||\to 0} \frac{||R(v)||}{||v||} = 0.$$
   One way to compute the derivative is
   $$F'(u)v = \lim_{\epsilon\to 0} \frac{F(u+\epsilon v) - F(u)}{\epsilon}.$$
   Note that this looks just like a directional derivative.
   (a) Compute the Frechet derivative of F.
   (b) $u \in X$ is a critical point of F if $F'(u)v = 0$ for all $v \in X$. Show that if u is a solution to the Poisson equation
   $$u_{xx} = f, \quad u(0) = u(1) = 0,$$
   then it is a critical point of F.

   Finite element methods are based on these "weak formulations" of the problem. The Ritz method is based on minimizing F and the Galerkin method is based on finding the critical points of $F'(u)$.

1

Figure 2.1: problem description

2.2.2 Problem 1

L is a second order differential operator defined by $Lu \equiv u_{xx}$ with boundary conditions on u given as $u_x(0) = u_x(1) = 0$

2.2.2.1 part a

Let $\phi(x)$ be an eigenfunction of the operator L associated with an eigenvalue λ. To obtain the eigenfunctions and eigenvalues, we solve an eigenvalue problem $L\phi = \lambda\phi$ where λ is scalar. Hence the problem is to solve the differential equation

$$\phi_{xx} - \lambda\phi = 0 \tag{1}$$

with B.C. given as $\phi'(0) = \phi'(1) = 0$. The characteristic equation is

$$r^2 - \lambda = 0$$

The roots are $r = \pm\sqrt{\lambda}$, therefore the solution to the eigenvalue problem (1) is

$$\phi(x) = c_1 e^{\sqrt{\lambda}x} + c_2 e^{-\sqrt{\lambda}x} \tag{2}$$

Where c_1,c_2 are constants.

$$\phi'(x) = c_1\sqrt{\lambda}e^{\sqrt{\lambda}x} - \sqrt{\lambda}c_2 e^{-\sqrt{\lambda}x} \tag{3}$$

First we determine the allowed values of the eigenvalues λ which satisfies the boundary conditions.

1. Assume $\lambda = 0$ The solution (2) becomes $\phi(x) = c_1 + c_2$. Hence the solution is a constant. In other words, when the eigenvalue is zero, the eigenfunction is a constant. Let us now see if this eigenfunction satisfies the B.C. Since $\phi(x)$ is constant, then $\phi'(x) = 0$, and this does satisfy the B.C. at both $x = 0$ and $x = 1$. Hence $\lambda = 0$ is an eigenvalue with a corresponding eigenfunction being a constant. We can take the constant as 1.

2. Assume $\lambda > 0$ From the first BC we have, from (3), that $\phi'(0) = 0 = c_1\sqrt{\lambda} - \sqrt{\lambda}c_2$ or
$$c_1 = c_2$$
and from the second BC we have that $\phi'(1) = 0 = c_1\sqrt{\lambda}e^{\sqrt{\lambda}} - \sqrt{\lambda}c_2e^{-\sqrt{\lambda}}$ or
$$c_1e^{\sqrt{\lambda}} - c_2e^{-\sqrt{\lambda}} = 0$$
From the above 2 equations, we find that $e^{\sqrt{\lambda}} = e^{-\sqrt{\lambda}}$ which is not possible for positive λ. Hence λ can not be positive.

3. Assume $\lambda < 0$. Let $\lambda = -\beta^2$ form some positive β. Then the solution (2) becomes
$$\phi(x) = c_1e^{i\beta x} + c_2e^{-i\beta x}$$
which can be transformed using the Euler relation to obtain
$$\begin{aligned}\phi(x) &= c_1\cos\beta x + c_2\sin\beta x \\ \phi'(x) &= -c_1\beta\sin\beta x + c_2\beta\cos\beta x \end{aligned} \tag{4}$$
Now consider the BC's. Since $\phi'(0) = 0$ we obtain $c_2 = 0$ and from $\phi'(1) = 0$ we obtain $0 = c_1\beta\sin\beta$ and hence for non trivial solution, i.e. for $c_1 \neq 0$, we must have that
$$\sin\beta = 0$$
or
$$\beta = \pm n\pi$$
but since β is positive, we consider only $\beta_n = n\pi$, where n is positive integer $n = 1, 2, 3, \cdots$

Conclusion: The eigenvalues are
$$\lambda_n = -\left(\beta_n\right)^2 = -(n\pi)^2 = \left\{0, -\pi^2, -(2\pi)^2, -(3\pi)^2, \cdots\right\}$$
And the corresponding eigenfunctions are$\phi_n(x) = \cos\beta_n x = \cos n\pi x = \{1, \cos\pi x, \cos 2\pi x, \cos 3\pi x, \cdots\}$
where $n = 0, 1, 2, \cdots$

2.2.2.2 part (b)

Given inner product defined as $\langle u, v\rangle = \int_0^1 uv\,dx$, then
$$\begin{aligned}\langle\phi_n, \phi_m\rangle &= \int_0^1 \left(\cos\beta_n x\right)\left(\cos\beta_m x\right)dx \\ &= \int_0^1 (\cos n\pi x)(\cos m\pi x)\,dx \\ &= \begin{cases} 0 & n \neq m \\ \frac{1}{2} & n = m \end{cases}\end{aligned}$$
Also, the first eigenfucntion, $\phi_0(x) = 1$ is orthogonal to all other eigenfunctions, since $\int_0^1 (\cos n\pi x)\,dx = \frac{1}{n\pi}\left[\sin n\pi x\right]_0^1 = 0$ for any integer $n > 0$.

Hence all the eigenfunctions are orthogonal to each others in $L^2[0,1]$ space.

2.2.2.3 Part (c)

Given

$$u_{xx} = f$$

$u_x(0) = u_x(1) = 0$. This is $Lu = f$. We have found the eigenfunctions $\phi(x)$ of L above. These are basis of the function space of L where f resides in. We can express f as a linear combination of the eigenfunctions of the operator L, hence we write

$$f(x) = \sum_{n=0}^{\infty} a_n \phi_n(x)$$

where $\phi_n(x)$ is the n^{th} eigenfunction of L and a_n is the corresponding coordinate (scalar).

Therefore the differential equation above can be written as

$$Lu = f(x) = \sum_{n=0} a_n \phi_n(x) \tag{1}$$

But since

$$L\phi_n = \lambda_n \phi_n$$

Then

$$L^{-1} = \frac{1}{\lambda_n}$$

Therefore, using (1), the solution is

$$u(x) = \sum_n \left(\frac{a_n}{\lambda_n} \right) \phi_n(x) \tag{2}$$

Now to find a_n, using $f(x) = \sum_n a_n \phi_n(x)$, we multiply each side by an eigenfunction, say $\phi_m(x)$ and integrate

$$\begin{aligned} \int_0^1 \phi_m(x) f(x)\, dx &= \int_0^1 \phi_m(x) \sum_n a_n \phi_n(x)(x)\, dx \\ &= \int_0^1 \sum_n a_n \phi_m(x) \phi_n(x)\, dx \\ &= \sum_n a_n \int_0^1 \phi_m(x) \phi_n(x)\, dx \end{aligned}$$

The RHS is 1/2 when $n = m$ and zero otherwise, hence the above becomes

$$\int_0^1 \phi_n(x) f(x)\, dx = \frac{a_n}{2}$$

Or

$$a_n = 2 \int_0^1 \cos(n\pi x) f(x)\, dx \tag{3}$$

Where a_n as given by (3).

If we know $f(x)$ we can determine a_n and hence the solution is now found.

2.2.2.4 part (d)

The solution found above

$$u(x) = \sum_n \left(\frac{a_n}{\lambda_n} \right) \phi_n(x)$$

Is not possible for all f. Only an f which has $a_0 = 0$ is possible. This is because $\lambda_0 = 0$, then a_0 has to be zero to obtain a solution (since L^{-1} does not exist if an eigenvalue is zero).

$a_0 = 0$ implies, by looking at (3) above, that when $n = 0$ we have

$$0 = \int_0^1 f(x)\, dx$$

So only the functions $f(x)$ which satisfy the above can be a solution to $Lu = f$ with the B.C. given.

To review: We found that $\lambda = 0$ to be a valid eigenvalue due to the B.C. being Von Neumann boundary conditions. This in resulted in a_0 having to be zero. This implied that $\int_0^1 f(x)\,dx = 0$.

Having a zero eigenvalue effectively removes one of the space dimensions that $f(x)$ can resides in.

In addition to this restriction, the function $f(x)$ is *assumed* to meet the Dirichlet conditions for Fourier series expansion, and these are

1. $f(x)$ must have a finite number of extrema in any given interval
2. $f(x)$ must have a finite number of discontinuities in any given interval
3. $f(x)$ must be absolutely integrable over a period.
4. $f(x)$ must be bounded

2.2.3 Problem 2

2.2.3.1 Part (a)

Applying the definition given

$$F'(u)\,v = \lim_{\varepsilon\to 0} \frac{F(u+\varepsilon v) - F(u)}{\varepsilon} \tag{1}$$

And using $F(u) = \int_0^1 \frac{1}{2}(u_x)^2 + f\ u\ dx$, then (1) becomes

$$F'(u)\,v = \lim_{\varepsilon\to 0} \frac{1}{\varepsilon}\left(\int_0^1 \frac{1}{2}\left[(u+\varepsilon v)_x\right]^2 + f\ \ (u+\varepsilon v)\ \ dx - \int_0^1 \frac{1}{2}(u_x)^2 + f\ u\ dx \right)$$

Simplify the above, we obtain

$$F'(u)\,v = \lim_{\varepsilon\to 0}\left(\int_0^1 \frac{\varepsilon}{2} v_x^2 dx + \int_0^1 u_x v_x dx + \int_0^1 f v dx \right)$$

Hence, as $\varepsilon \to 0$ only the first integral above vanishes (since v_x is bounded), and we have

$$F'(u)\,v = \int_0^1 u_x v_x + f v dx \tag{1A}$$

2.2.3.2 Part (b)

The solution to $u_{xx} = f(x)$ with $u(0) = u(1) = 0$ was found in class to be

$$u(x) = \sum_n \left(\frac{a_n}{\lambda_n}\right)\phi_n(x) \tag{2}$$

where

$$\phi_n(x) = \sin(n\pi x)$$

are the eigenfunctions associated with the eigenvalues $\lambda_n = -n^2\pi^2$.

Now we can use this solution in the definition of $F'(u)\,v$ found in (1A) from part (a). Substitute $u(x)$ from (2) into (1A), and also substitute $f = \sum_n a_n\phi_n(x)$ into (1A), we obtain

$$F'(u)\,v = \int_0^1 \left(\sum_n \left(\frac{a_n}{\lambda_n}\right)\phi_n(x)\right)' v' + \left(\sum_n a_n\phi_n(x)\right) v\ dx \tag{4}$$

We need to show that the above becomes zero for any $v(x) \in X$.

$$\begin{aligned} F'(u)v &= \int_0^1 \sum_n v' \left(\frac{a_n}{\lambda_n}\right) \phi_n'(x) + \sum_n v a_n \phi_n(x)\, dx \\ &= \int_0^1 \sum_n \left(v' \left(\frac{a_n}{\lambda_n}\right) \phi_n'(x) + v a_n \phi_n(x) \right) dx \\ &= \sum_n a_n \left(\int_0^1 \frac{1}{\lambda_n} v' \phi_n'(x) + v \phi_n(x)\, dx \right) \end{aligned} \tag{5}$$

Now we pay attention to the integral term above. If we can show this is zero, then we are done.

$$\begin{aligned} I &= \frac{1}{\lambda_n} \int_0^1 v' \phi_n'(x) + \int_0^1 v \phi_n(x)\, dx \\ &= I_1 + I_2 \end{aligned} \tag{6}$$

Integrate by parts I_1

$$\begin{aligned} I_1 &= \frac{1}{\lambda_n} \int_0^1 \overbrace{\phi_n'(x)\, v'\, dx}^{udv} \\ &= \frac{1}{\lambda_n} \left(\left[\phi_n'(x)\, v \right]_0^1 - \int_0^1 v(x)\, \phi_n''(x)\, dx \right) \\ &= \frac{1}{\lambda_n} \left(\overbrace{\left[v(1)\, \phi_n'(1) - v(0)\, \phi_n'(0) \right]}^{\text{zero due to boundaries on } v(x) \in X} - \int_0^1 v(x)\, \phi_n''(x)\, dx \right) \\ &= -\frac{1}{\lambda_n} \int_0^1 v(x)\, \phi_n''(x)\, dx \end{aligned}$$

But since $\phi_n(x) = \sin n\pi x$, then $\phi_n'(x) = n\pi \cos n\pi x$ and $\phi_n''(x) = -n^2\pi^2 \sin n\pi x = -n^2\pi^2\, \phi_n(x)$ then

$$I_1 = \frac{n^2\pi^2}{\lambda_n} \int_0^1 v(x)\, \phi_n(x)\, dx$$

But also $\lambda_n = -n^2\pi^2$ hence the above becomes

$$I_1 = -\int_0^1 v(x)\, \phi_n(x)\, dx$$

Therefore (6) can be written as

$$\begin{aligned} I &= I_1 + I_2 \\ &= -\int_0^1 v(x)\, \phi_n(x)\, dx + \int_0^1 v(x)\, \phi_n(x)\, dx \\ &= 0 \end{aligned}$$

Therefore, from (5), we see that

$$F'(u)v = 0$$

Hence we showed that if u is solution to $u_{xx} = f$ with $u(0) = u(1) = 0$, then $F'(u)v = 0$.

2.2.4 Problem (3)

2.2.4.1 Part (a)

Notations used: let $\tilde{f}$ to mean the approximate discrete solution at a grid point. Let f to mean the exact solution.

Using the method of undetermined coefficients, let the second derivative approximation

be

$$\tilde{f}''(x) = af\left(x - \frac{h}{2}\right) + bf(x) + cf(x+h) \tag{1}$$

Where a, b, c are constants to be found. Now using Taylor expansion, since

$$f(x+\Delta) = f(x) + \Delta f'(x) + \frac{\Delta^2}{2!}f''(x) + \frac{\Delta^3}{3!}f'''(x) + O\left(h^4\right)$$

Hence apply the above to each of the terms in the RHS of (1) and simplify

$$f\left(x - \frac{h}{2}\right) = f(x) - \frac{h}{2}f'(x) + \frac{\left(-\frac{h}{2}\right)^2}{2!}f''(x) + \frac{\left(-\frac{h}{2}\right)^3}{3!}f''(x) + \frac{\left(-\frac{h}{2}\right)^4}{4!}f^{(4)}(x) + O\left(h^5\right)$$

$$f(x) = f(x)$$

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f''(x) + \frac{h^4}{4!}f^{(4)}(x) + O\left(h^5\right)$$

Substitute the above 3 terms in (1)

$$\begin{aligned}\tilde{f}''(x) = a&\left(f(x) - \frac{h}{2}f'(x) + \frac{h^2}{8}f''(x) - \frac{h^3}{8\times 6}f'''(x) + \frac{h^4}{16\times 24}f^{(4)}(x) + O\left(h^5\right)\right)\\ &+ bf(x)\\ &+ c\left(f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(x) + O\left(h^5\right)\right)\end{aligned}$$

Collect terms

$$\begin{aligned}\tilde{f}''(x) = (a+b+c)f(x) + f'(x)h\left(-\frac{a}{2} + c\right) + f''(x)h^2\left(\frac{a}{8} + \frac{c}{2}\right) + f'''(x)h^3\left(\frac{-a}{8\times 6} + \frac{c}{6}\right)\\ + f^{(4)}h^4\left(\frac{a}{16\times 24} + \frac{c}{24}\right) + O\left(h^5\right)\end{aligned} \tag{2}$$

Hence for $\tilde{f}''(x)$ to best approximate $f''(x)$, we need

$$\begin{aligned}(a+b+c) &= 0\\ -\frac{a}{2} + c &= 0\\ h^2\left(\frac{a}{8} + \frac{c}{2}\right) &= 1\end{aligned}$$

Solving the above 3 equations we find

$$\begin{aligned}a &= \frac{8}{3h^2}\\ b &= -\frac{4}{h^2}\\ c &= \frac{4}{3h^2}\end{aligned}$$

Hence (1) becomes

$$\begin{aligned}\tilde{f}''(x) &= af\left(x - \frac{h}{2}\right) + bf(x) + cf(x+h)\\ &= \frac{8}{3h^2}f\left(x - \frac{h}{2}\right) - \frac{4}{h^2}f(x) + \frac{4}{3h^2}f(x+h)\end{aligned}$$

To examine the local truncation error, from (2), and using the solution we just found for a, b, c we find

$$\begin{aligned}\tilde{f}''(x) &= f''(x) + f'''(x)h^3\left(\frac{-\left(\frac{8}{3h^2}\right)}{8\times 6} + \frac{\left(\frac{4}{3h^2}\right)}{6}\right) + f^{(4)}h^4\left(\frac{\left(\frac{8}{3h^2}\right)}{16\times 24} + \frac{\left(\frac{4}{3h^2}\right)}{24}\right) + O\left(h^5\right)\\ &= f''(x) + f'''(x)h^3\left(\frac{1}{6h^2}\right) + f^{(4)}h^4\left(\frac{1}{16h^2}\right) + O\left(h^5\right)\\ &= f''(x) + f'''(x)\left(\frac{h}{6}\right) + f^{(4)}h^2\left(\frac{1}{16}\right) + O\left(h^5\right)\end{aligned}$$

We can truncate at either $f'''(x)$ or $f^{(4)}$. In the first case, we obtain

$$\tilde{f}''(x) = f''(x) + O(h)$$

Where $O(h) = \frac{f'''(x)}{6}h$, hence $p = 1$ in this case, and with the truncation error $\tau = \frac{f'''(x_j)}{6}h$ at each grid point.

In the second case, we obtain

$$\tilde{f}''(x) = f''(x) + \frac{f'''(x)}{6}h + O\left(h^2\right)$$

Where $O\left(h^2\right) = \frac{f^{(4)}}{16}h^2$ and $p = 2$ in this case, and with the truncation error $\tau = \frac{f^{(4)}(x_j)}{16}h^2$ at each grid point. We see that τ is smaller if we use $p = 2$ than $p = 1$.

The accuracy then depends on where we decide to truncate. For example, at $p = 1$, the error is dominated by $O(h)$, and at $p = 2$, it is $O\left(h^2\right)$.

2.2.4.2 part (b) Refinement study

Given $f(x) = \cos(2\pi x)$, first, let us find the accuracy of this scheme. The finite difference approximation formula found is

$$\tilde{f}''(x) = \frac{8}{3h^2}f\left(x - \frac{h}{2}\right) - \frac{4}{h^2}f(x) + \frac{4}{3h^2}f(x+h) \tag{1}$$

And the exact value is

$$\frac{d^2}{dx^2}\cos(2\pi x) = -4\pi^2\cos 2\pi x \tag{2}$$

To find the local error τ

$$\tau = \tilde{f}''(x) - f''(x)$$

Substitute $f(x) = \cos(2\pi x)$ in the RHS of (1) to find the approximation of the second derivative and subtract the exact result value of the second derivative from it.

Plug $f(x) = \cos(2\pi x)$ in RHS of (1) we obtain

$$\begin{aligned}\tilde{f}''(x) &= \frac{8}{3h^2}\cos\left(2\pi\left(x - \frac{h}{2}\right)\right) - \frac{4}{h^2}\cos(2\pi x) + \frac{4}{3h^2}\cos(2\pi(x+h)) \\ &= \frac{8}{3h^2}\cos(2\pi x - \pi h) - \frac{4}{h^2}\cos(2\pi x) + \frac{4}{3h^2}\cos(2\pi x + 2\pi h)\end{aligned}$$

Hence the local error τ is

$$\begin{aligned}\tau &= \tilde{f}''(x) - f''(x) \\ &= \left[\frac{8}{3h^2}\cos(2\pi x - \pi h) - \frac{4}{h^2}\cos(2\pi x) + \frac{4}{3h^2}\cos(2\pi x + 2\pi h)\right] + 4\pi^2\cos 2\pi x\end{aligned}$$

We notice that τ depends on h and x. At $x = 1$,

$$\begin{aligned}\tau &= \left[\frac{8}{3h^2}\cos(2\pi - \pi h) - \frac{4}{h^2} + \frac{4}{3h^2}\cos(2\pi + 2\pi h)\right] + 4\pi^2 \\ &= \frac{4}{3h^2}\left(\cos(2\pi h) + 2\cos(\pi h) + 3h^2\pi^2 - 3\right)\end{aligned}$$

In the following we plot local error τ as a function of h in linear scale and log scale. Here is the result.

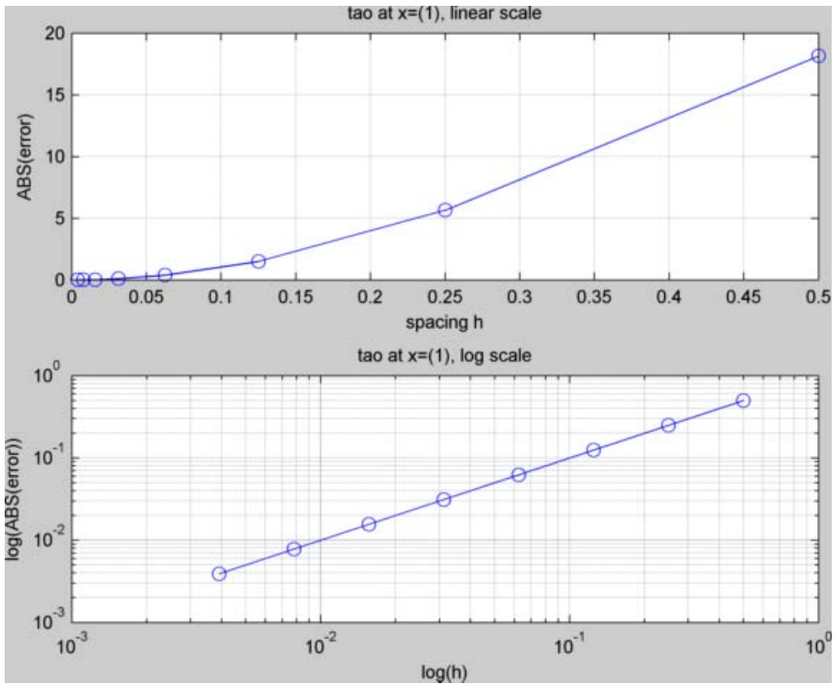


Figure 2.2: matlab HW1 partb

We notice that the log plot shows the slope $p = 2$ and not $p = 1$. This is because the $O(h)$ part turned out to be zero at $x = 1$ this is because $O(h) = \frac{f'''(x)}{6}h = \frac{(8\pi^3 \sin 2\pi x)}{6}h$ and this term is zero at $x = 1$, so the dominant error term became $O\left(h^2\right)$ which is $\frac{f^{(4)}(x_j)}{16}h^2 = \frac{16\pi^4 \cos 2\pi x}{16}h^2$ or $\pi^4 h^2$ or $O\left(h^2\right)$.

This is why we obtained $p = 2$ and not $p = 1$ at $x = 1$.

The following table show the ratio of the local error between each 2 successive halving of the spacing h. Each time h is halved, and the ratio of the error (absolute local error) is shown. We see for $x = 1$ that the ratio approaches 4. This indicates that $p = 2$.

```
EDU>> nma_HW1_partb()
h                 error             ratio
5.0000E-001      1.8145E+001      0.0000E+000
2.5000E-001      5.6483E+000      3.2125E+000
1.2500E-001      1.4936E+000      3.7816E+000
6.2500E-002      3.7872E-001      3.9439E+000
3.1250E-002      9.5014E-002      3.9859E+000
1.5625E-002      2.3775E-002      3.9965E+000
7.8125E-003      5.9449E-003      3.9991E+000
3.9063E-003      1.4863E-003      3.9998E+000
```

.

2.2.4.3 Part(c)

The refinement study in part (b) showed that the local error became smaller as h become smaller, and the error was $O\left(h^2\right)$ since $p = 2$ in the log plot.

But this is not a good test as it was done only for one point $x = 1$. We need to examine the approximation scheme at other points as well. The reason is the local error at an x location is

$$\tau = \left[\frac{8}{3h^2}\cos(2\pi x - \pi h) - \frac{4}{h^2}\cos(2\pi x) + \frac{4}{3h^2}\cos(2\pi x + 2\pi h)\right] + 4\pi^2 \cos 2\pi x$$

which can be seen to be a function of x and h. In (b) we found that at $x = 1$, $\tau = O\left(h^2\right)$ and this was because the dominant error term $O(h)$ happened to vanish at $x = 1$.

But if we examine τ at different point, say $x = 0.2$, then we will see that τ is $O(h)$ and $p = 1$.

Here is a plot of τ at $x = 0.2$ and at $x = 1$. Both showing what happens as h becomes smaller. We see that the at $x = 1$ the approximation was more accurate ($p = 2$) but at $x = 0.2$ the approximation was less accurate ($p = 1$). What we conclude from this, is that a single test is not sufficient for determine the accuracy for all points. More tests are needed at other points to have more confidence. To verify that at $x = 0.2$ we indeed have $p = 1$, we generate the error table as shown above, but for $x = 0.2$ this time.

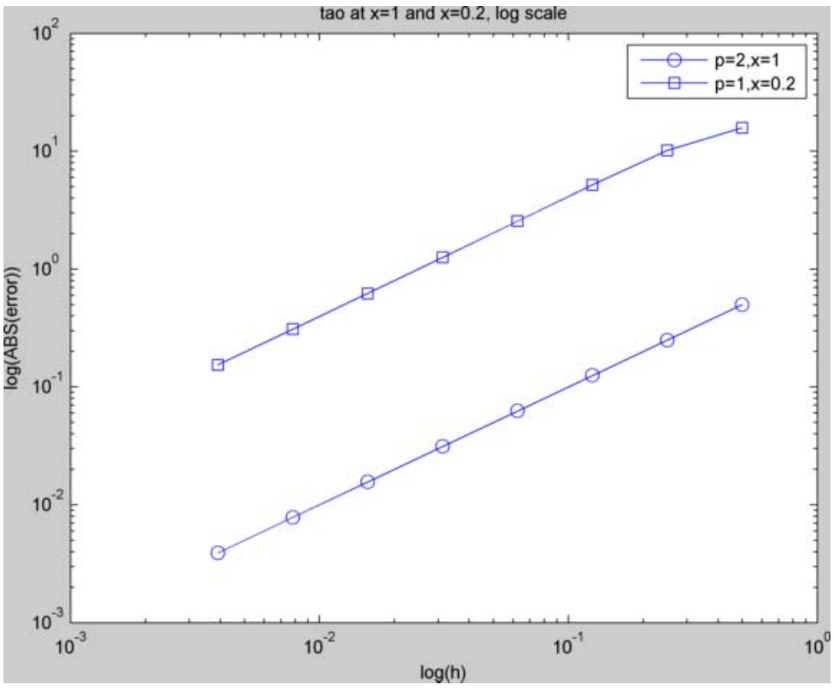


Figure 2.3: matlab HW1 partc

```
EDU>> nma_HW1_partc()
  h                  error             ratio
5.0000E-001       1.5752E+001       0.0000E+000
2.5000E-001       1.0149E+001       1.5520E+000
1.2500E-001       5.1898E+000       1.9557E+000
6.2500E-002       2.5508E+000       2.0345E+000
3.1250E-002       1.2551E+000       2.0324E+000
1.5625E-002       6.2133E-001       2.0200E+000
7.8125E-003       3.0897E-001       2.0110E+000
3.9063E-003       1.5404E-001       2.0057E+000
```

.

We see that the ratio becomes 2 this time, not 4 as we half the spacing each time. This mean $p = 1$. This means the accuracy of the formula used can depend on the location.

2.2.4.4 Part (d)

The points that we need to interpolate are $\left[\left[x - \frac{h}{2}, u\left(x - \frac{h}{2}\right)\right], [x, u(x)], [x + h, u(x + h)]\right]$ where $u = \cos(2\pi x)$

Since we require a quadratic polynomial, then we write

$$p(x) = a + bx + cx^2$$

Where $p(x)$ is the interpolant. Evaluate the above at each of the 3 points. Choose $x = 1$,

hence the points are

$$\begin{pmatrix} 1-\frac{h}{2}, u\left(1-\frac{h}{2}\right) \\ 1, u(x) \\ 1+h, u(1+h) \end{pmatrix}$$

Evaluate $p(x)$ at each of these points

$$p\left(1-\frac{h}{2}\right) = \cos\left(2\pi\left(1-\frac{h}{2}\right)\right) = a + b\left(1-\frac{h}{2}\right) + c\left(1-\frac{h}{2}\right)^2$$
$$p(1) = \cos(2\pi) = a + b + c$$
$$p(1+h) = \cos(2\pi(1+h)) = a + b(1+h) + c(1+h)^2$$

or

$$\begin{pmatrix} \left(1-\frac{h}{2}\right)^2 & \left(1-\frac{h}{2}\right) & 1 \\ 1 & 1 & 1 \\ (1+h)^2 & (1+h) & 1 \end{pmatrix} \begin{pmatrix} c \\ b \\ a \end{pmatrix} = \begin{pmatrix} \cos\left(2\pi\left(1-\frac{h}{2}\right)\right) \\ \cos(2\pi 1) \\ \cos(2\pi(1+h)) \end{pmatrix}$$
$$Av = b$$

Solving the above Vandermonde system, we obtain

$$a = \frac{1}{3h^2}\left(4(1+h)\cos(h\pi) + (h-2)(3+3h-\cos(2\pi h))\right)$$
$$b = \frac{-1}{3h^2} 4\left((h-4)\cos(\pi h) - h - 8\right)\sin^2\left(\frac{\pi h}{2}\right)$$
$$c = \frac{2}{3h^2}\left(2\cos(\pi h) - 3 + \cos(2\pi h)\right)$$

Hence

$$\begin{aligned} p(x) = &\left[\frac{1}{3h^2}\left(4(1+h)\cos(h\pi) + (h-2)(3+3h-\cos(2\pi h))\right)\right] \\ &-\left[\frac{1}{3h^2} 4\left((h-4)\cos(\pi h) - h - 8\right)\sin^2\left(\frac{\pi h}{2}\right)\right] x \\ &+\left[\frac{2}{3h^2}\left(2\cos(\pi h) - 3 + \cos(2\pi h)\right)\right] x^2 \end{aligned} \tag{1}$$

Recall, that we found, for $u = \cos(2\pi x)$, the finite difference formula was

$$\tilde{u}''(x) = \left[\frac{8}{3h^2}\cos(2\pi x - \pi h) - \frac{4}{h^2}\cos(2\pi x) + \frac{4}{3h^2}\cos(2\pi x + 2\pi h)\right] \tag{2}$$

Take the second derivative of $p(x)$ shown in (1) above

$$p''(x) = \frac{4}{3h^2}\left(2\cos(\pi h) + \cos(2\pi h) - 3\right) \tag{3}$$

But we notice that $\tilde{u}''(x)$ evaluated at $x = 1$ is

$$\tilde{u}''(1) = \frac{4}{3h^2}\left(2\cos(\pi h) + \cos(2\pi h) - 3\right)$$

which is the same as $p''(x)$.

Therefore, $p''(x)$ is the same as as the finite difference approximation evaluated at the central point of the 3 points, used to generate p.

In other words, given 3 points

$$\begin{pmatrix} x_0 - \frac{h}{2}, u(x_0) \\ x_0, u(x_0) \\ x_0 + h, u(x_0 + h) \end{pmatrix}$$

Where $u(x)$ is some function (here it was $\cos(2\pi x)$), and we generate a quadratic interpolant polynomial $p(x)$ using the above 3 points, then $p''(x)$ will given the same value as the finite difference formula evaluated at x_0.

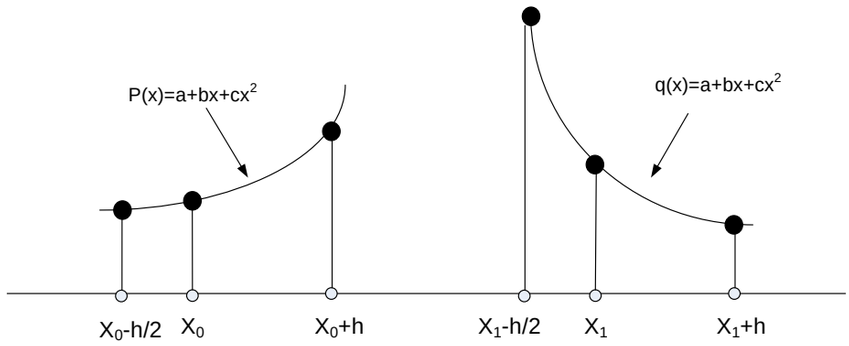
$$p''(x)\big|_{x=x_0} = \tilde{u}(x)\big|_{x=x_0}$$

For this to be valid, $p(x)$ must have been generated with the center point being x_0. If we

pick another center point x_1, and therefore have the 3 points $x_1 - h/2, x_1, x_1 + h$, and then generate a polynomial $q(x)$ as above, then we will find

$$q''(x)\big|_{x=x_1} = \tilde{u}(x)|_{x=x_1}$$

This is illustrated by the following diagram



$$p''(x)|_{x=x_0} = \tilde{u}(x)|_{x=x_0} \qquad q''(x)|_{x=x_1} = \tilde{u}(x)|_{x=x_1}$$

Figure 2.4: prob3 c

2.2.5 Appendix (Source code)

2.2.5.1 Matlab

```
%-- by Nasser M. Abbasi, Math 228A, UC Davis, Fall 2010
%-- implement part b, problem 3
function nma_HW1_partb()
%-- Generate h values to use, and define tao(h) function
N = 8;
pointAt=1;
data  = arrayfun( @(i) [1/(2^i) , local_error(1/(2^i),pointAt)] ,1:N, ...
                 'UniformOutput',false);
data = reshape(cell2mat(data),2,N)';

%-- plot the tao(h) in linear and log scale
set(0,'defaultaxesfontsize',8) ;
set(0,'defaulttextfontsize',8);

subplot(2,1,1);
plot(data(:,1),data(:,2),'-o'); grid on;
title('tao at x=(1), linear scale');
xlabel('spacing h'); ylabel('ABS(error)');

subplot(2,1,2);
loglog(data(:,1),data(:,1),'-o'); grid on;
title('tao at x=(1), log scale');
xlabel('log(h)'); ylabel('log(ABS(error))');
export_fig matlab_HW1_partb.png

%-- now generate the error table, find ratio first
error_ratio = zeros(N,1);
for i=2:N
    error_ratio(i) = data((i-1),2)/data(i,2);
end

%-- print table
fprintf('h\t\t\t\t error\t\t\t\t ratio\n');
for i=1:N
   fprintf('%6.4E\t\t%6.4E\t\t%6.4E\n',data(i,1),data(i,2),error_ratio(i));
```

```
end

end

function tao=local_error(h,x)
tao=8/(3*h^2)*cos(2*pi*(x-h/2))-4/h^2*cos(2*pi*x)+4/(3*h^2)*...
    cos(2*pi*(x+h))+(2*pi)^2*cos(2*pi*x);
end
```

```
%-- by Nasser M. Abbasi, Math 228A, UC Davis, Fall 2010
%-- implement part c, problem 3
function nma_HW1_partc()

%-- Generate h values to use, and define tao(h) function
%-- plot the tao(h) in linear and log scale
set(0,'defaultaxesfontsize',8) ;
set(0,'defaulttextfontsize',8);

%build data, x-axis is spacing h, y-axis is error
N = 8;
pointAt=1.0;
data  = arrayfun( @(i) [1/(2^i) , local_error(1/(2^i),pointAt)] ,1:N, ...
                  'UniformOutput',false);
data = reshape(cell2mat(data),2,N)';

loglog(data(:,1),data(:,1),'-o'); grid off;
title('tao at x=1 and x=0.2, log scale');
xlabel('log(h)'); ylabel('log(ABS(error))');
hold on;

pointAt=0.2;
data  = arrayfun( @(i) [1/(2^i) , local_error(1/(2^i),pointAt)] ,1:N,...
                  'UniformOutput',false);
data = reshape(cell2mat(data),2,N)';
loglog(data(:,1),data(:,2),'-s');
legend('p=2,x=1','p=1,x=0.2');

export_fig matlab_HW1_partc.png

%-- now generate the error table, find ratio first
error_ratio = zeros(N,1);
for i=2:N
    error_ratio(i) = data((i-1),2)/data(i,2);
end

%-- print table
fprintf('h\t\t\t\t error\t\t\t\t ratio\n');
for i=1:N
   fprintf('%6.4E\t\t%6.4E\t\t%6.4E\n',data(i,1),data(i,2),error_ratio(i));
end

end

function tao=local_error(h,x)
tao=8/(3*h^2)*cos(2*pi*(x-h/2))-4/h^2*cos(2*pi*x)+4/(3*h^2)*...
    cos(2*pi*(x+h))+(2*pi)^2*cos(2*pi*x);
end
```

2.2.5.2 Mathematica

HW 1, problem 3, computational part. math 228A UC davis fall 2010

Nasser M. Abbasi

This is the code used to generate the plots and tables used in HW1

define local error function

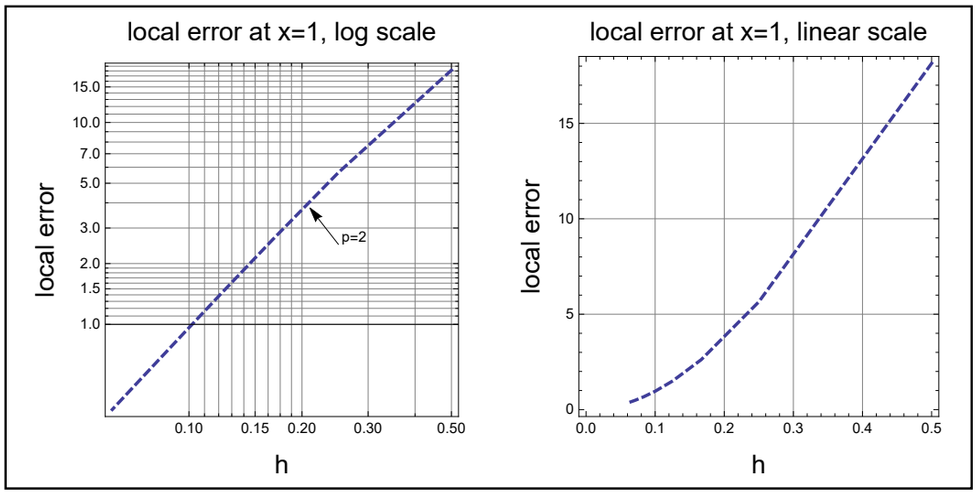
```
localError[h_, x_] :=
  Module[{}, 8/(3 h^2) Cos[2 π x - π h] - 4/h^2 Cos[2 π x] + 4/(3 h^2) Cos[2 π x + 2 π h]] + (2 π)^2 Cos[2 π x];
```

define a function to make the plots

```
makePlot[x_, s_, title_, xlabel_, ylabel_, f_] := Module[{data, n = 8},
  data = Table[{1/(2^i), Abs@localError[1/(2^i), x]}, {i, 1, n}];
  f[data, Joined → True, AxesOrigin → {0, 0},
   GridLines → Automatic, AspectRatio → 1, Frame → True, PlotRange → All,
   FrameLabel → {{ylabel, None}, {xlabel, title}}, PlotStyle → s, ImageSize → Full]
 ]
```

make plot for problem 3, part b

```
title = Style["local error at x=1, log scale", 16];
xlabel = Style["h", 16]; ylabel = Style["local error", 16];
p1 = makePlot[1, {Thick, Dashed}, title, xlabel, ylabel, ListLogLogPlot];
title = Style["local error at x=1, linear scale", 16];
p2 = makePlot[1, {Thick, Dashed}, title, xlabel, ylabel, ListPlot];
Framed[Grid[{{p1, p2}}], ImageSize → {600, 300}]
```



Generate error table, problem 3, part b

```
n = 14;
x = 1;
data = Table[{1 / (2^i), Abs@localError[1 / (2^i), x]}, {i, 1, n}];
data = Table[{data[[i, 1]], data[[i, 2]], If[i == 1, 0, data[[i - 1, 2]] / data[[i, 2]]]}, {i, 1, n}];
t = TableForm[N[data, $MachinePrecision], TableHeadings →
    {None, {"h", "local error τ", "ratio"}}, TableSpacing → {1, 6}, TableAlignments → Left];
Labeled[Framed@ScientificForm[t, {8, 6}, NumberFormat → (Row[{#1, "e", #3}] &),
   NumberPadding → {"", "0"}], Style["local error as function of h at x=1", 14], Top]
```

local error as function of h at x=1

h	local error τ	ratio
5.000000e-1	1.814508e1	0.000000e
2.500000e-1	5.648307e	3.212482e
1.250000e-1	1.493636e	3.781581e
6.250000e-2	3.787161e-1	3.943947e
3.125000e-2	9.501408e-2	3.985895e
1.562500e-2	2.377451e-2	3.996468e
7.812500e-3	5.944941e-3	3.999117e
3.906250e-3	1.486317e-3	3.999779e
1.953125e-3	3.715845e-4	3.999945e
9.765625e-4	9.289644e-5	3.999986e
4.882812e-4	2.322413e-5	3.999997e
2.441406e-4	5.806034e-6	3.999999e
1.220703e-4	1.451509e-6	4.000000e
6.103516e-5	3.628771e-7	4.000000e

Generate table for problem 3, part (c)

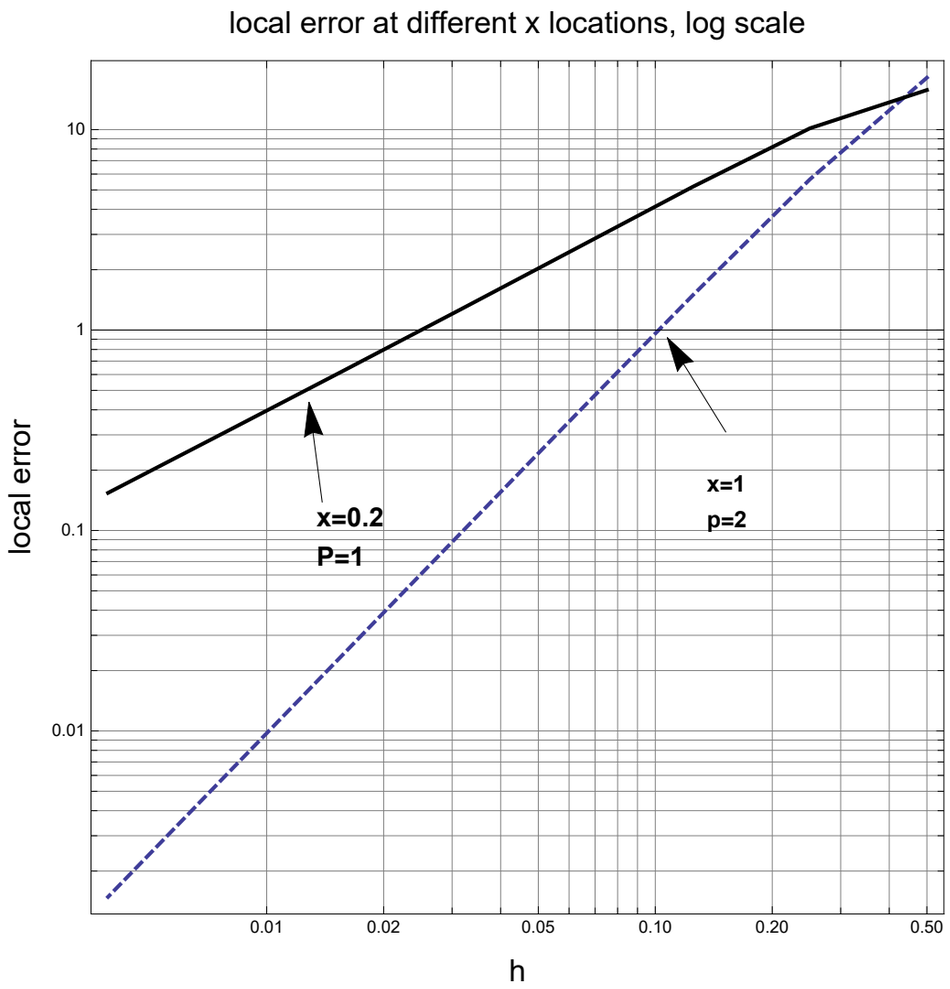
```
n = 14;
x = 0.2;
data = Table[{1 / (2^i), Abs@localError[1 / (2^i), x]}, {i, 1, n}];
data = Table[{data[[i, 1]], data[[i, 2]], If[i == 1, 0, data[[i - 1, 2]] / data[[i, 2]]]}, {i, 1, n}];
t = TableForm[N[data, $MachinePrecision], TableHeadings →
    {None, {"h", "local error τ", "ratio"}}, TableSpacing → {1, 6}, TableAlignments → Left];
Labeled[Framed@ScientificForm[t, {8, 6}, NumberFormat → (Row[{#1, "e", #3}] &),
   NumberPadding → {"", "0"}], Style["local error as function of h at x=0.2", 14], Top]
```

local error as function of h at x=0.2

h	local error τ	ratio
5.000000e-1	1.575174e1	0.000000e
2.500000e-1	1.014949e1	1.551974e
1.250000e-1	5.189762e	1.955675e
6.250000e-2	2.550829e	2.034539e
3.125000e-2	1.255100e	2.032371e
1.562500e-2	6.213251e-1	2.020037e
7.812500e-3	3.089650e-1	2.010989e
3.906250e-3	1.540406e-1	2.005737e
1.953125e-3	7.690765e-2	2.002930e
9.765625e-4	3.842539e-2	2.001480e
4.882812e-4	1.920555e-2	2.000744e
2.441406e-4	9.600990e-3	2.000372e
1.220703e-4	4.800048e-3	2.000186e
6.103516e-5	2.399851e-3	2.000144e

Generate plot for part (C)

```
title = Style["local error at different x locations, log scale", 16];
xlabel = Style["h", 16]; ylabel = Style["local error", 16];
p1 = makePlot[1, {Thick, Dashed}, title, xlabel, ylabel, ListLogLogPlot];
p2 = makePlot[0.2, {Thick, Black}, title, xlabel, ylabel, ListLogLogPlot];
Show[{p1, p2}, ImageSize → 500]
```



2.3 HW 2

2.3.1 Problem description

Math 228A
Homework 2
Due Friday, 10/22/08, 4:00

1. Use the standard 3-point discretization of the Laplacian on a regular mesh to find a numerical solution to the PDEs below. Perform a refinement study using the exact solution to compute the error that shows the rate of convergence for both the 1-norm and the max norm.

   (a) $u_{xx} = \exp(x), \quad u(0) = 0, \quad u(1) = 1$

   (b) $u_{xx} = 2\cos^2(\pi x), \quad u_x(0) = 0, \quad u_x(1) = 1$

2. Propose a discretization scheme for

   $$u_{xx} = f, \quad u_x(0) - \alpha u(0) = g, \quad u(1) = b.$$

   What is the form of the matrix and right hand side in your discrete equations? What order of accuracy do you expect?

3. As a general rule, we usually think that an $O(h^p)$ local truncation error (LTE) leads to an $O(h^p)$ error. However, in some cases the LTE can be lower order at some points without lowering the order of the error. Consider the standard second-order discretization of the Poisson equation on $[0, 1]$ with homogeneous boundary conditions. The standard discretization of this problem gives an $O(h^2)$ LTE provided the the solution is at least C^4. The LTE may be lower order because the solution is not C^4 or because we use a lower order discretization at some points.

   (a) Suppose that the LTE is $O(h^p)$ at the first grid point ($x_1 = h$). What effect does this have on the error? What is the smallest value of p that gives a second order accurate error? Hint: Use equation (2.46) from LeVeque to aid in your argument.

   (b) Suppose that the LTE is $O(h^p)$ at an interior point (i.e. a point that does not limit to the boundary as $h \to 0$). What effect does this have on the error? What is the smallest value of p that gives a second order accurate error?

   (c) Verify the results of your analysis from parts (a) and (b) using numerical tests.

Figure 2.5: problem description

2.3.2 Problem 1

Use standard 3-point discretization of the Laplacian on a regular mesh to find numerical solution to the PDE below and perform refinement study to compute the error that shows the rate of convergence for both the 1-norm and the max-norm.

2.3.2.1 part (a)

The differential equation with its boundary conditions is $u_{xx} = e^x$ with $u(0) = 0, u(1) = 1$.

Finding the analytical solution

Let D be a differential operator $D \equiv \frac{d}{dx}$. Since $(D-1)e^x = 0$, then applying $(D-1)$ to both sides of the differential equation results in

$$\left[(D-1)D^2\right](u) = 0$$

Thus the characteristic equation is $(r-1)r^2 = 0$ with roots $r_1 = 1, r_2 = 0$ and $r_3 = 0$. Therefore the complete solution is

$$u(x) = c_1 e^{r_1 x} + c_2 e^{r_2 x} + x c_3 e^{r_3 x}$$

It helps to designate in the above which part is the particular solution and which is the homogeneous

$$u(x) = \overbrace{c_1 e^x}^{u_p} + \overbrace{c_2 + c_3 x}^{y_h}$$

c_1 is found by substituting the particular solution in the original differential equation giving

$c_1 = 1$. The solution becomes[1]

$$u(x) = e^x + c_2 + xc_3$$

The remaining constants c_2 and c_3 are found by satisfying the boundary conditions on the above solution. Applying $u(0) = 0$ gives $c_2 = -1$ and applying $u(1) = 1$ gives $c_3 = 2 - e$. The solution becomes

$$u(x) = 2x - \exp(1)\, x + \exp(x) - 1$$

Scheme to use for the numerical solution

The numbering used is the one described in the class. The first grid point has an index $j = 0$, and the last grid point has an index $j = N + 1$. U is the unknown to solve for. It is the numerical solution of the differential equation at the grid points. Lower case u is the exact analytical solution found earlier.

Because this problem has Dirichlet boundary conditions, U is known at $j = 0$ and at $j = N + 1$, therefore only internal grid points are used to solve for U. These internal points are numbered $1 \cdots N$. The spacing between each grid point is $h = \frac{len}{N+1}$ where the length of the domain len is always taken to have value 1.

The physical x-coordinate of each grid point is given by $x_j = jh$. For example, when $j = 0$, the first point will have a physical x-coordinate of 0, and when $j = N + 1$, the last grid point will have a physical x-coordinate of 1.

The diagram below helps illustrate these relations and the notations used.

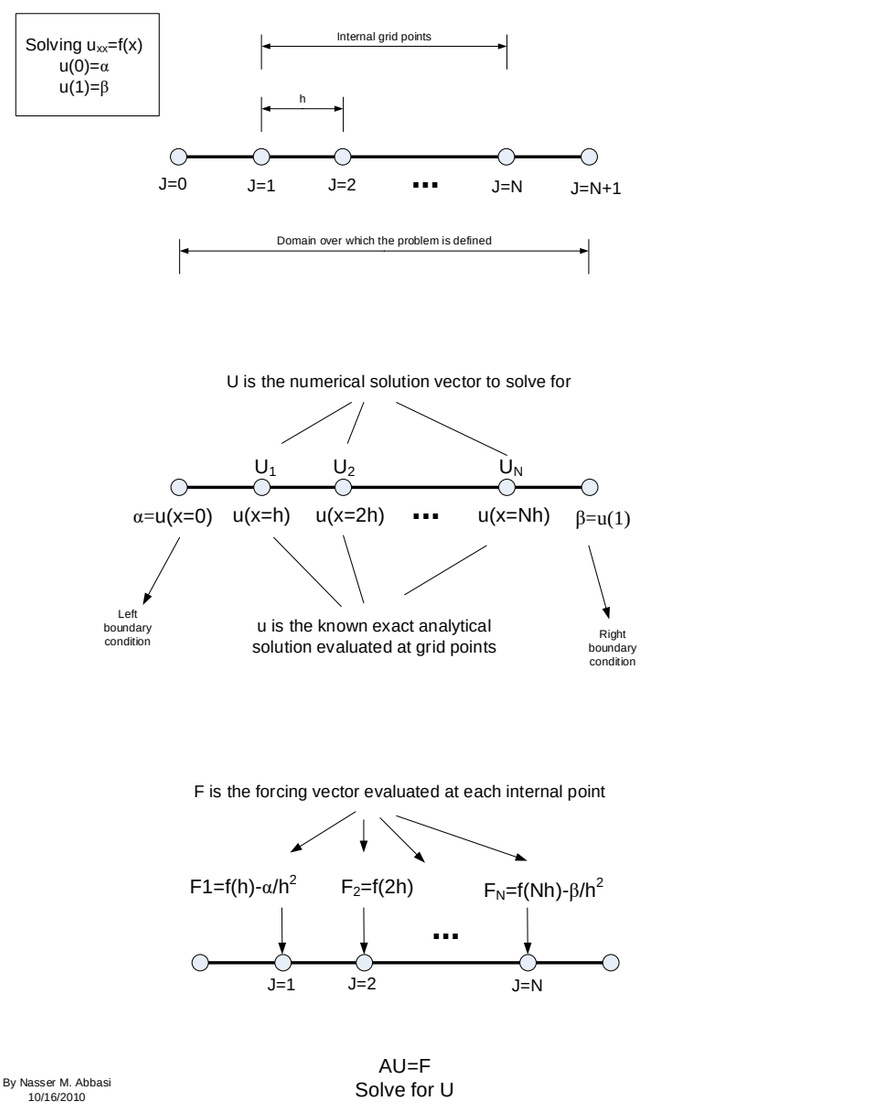


Figure 2.6: problem 1 part a scheme

[1] another way to solve this is to integrate the original differential equation twice to obtain this general solution.

The standard 3-point central difference formula $\frac{U_{j-1}-2U_j+U_{j+1}}{h^2}$ was used to approximate u_{xx} on each internal grid point. This approximation has local truncation error (LTE) of $O\left(h^2\right)$. Therefore, at each internal grid point j the equation $u_{xx} = f(x)$ is approximated by $\frac{U_{j-1}-2U_j+U_{j+1}}{h^2} = f_{x=jh}$.

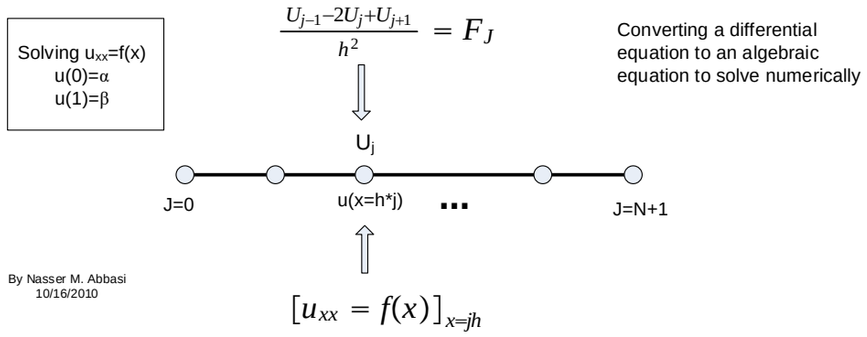


Figure 2.7: problem 1 part a scheme 2

For the special case of $j = 1$ (the first internal grid point on left side), the above formula was modified by replacing U_o by its given value (U_0 is on the boundary and its value is known). Therefore, for $j = 1$ the discrete equation becomes

$$\begin{aligned} \frac{\alpha - 2U_0 + U_1}{h^2} &= F_1 \\ \frac{-2U_0 + U_1}{h^2} &= F_1 - \frac{\alpha}{h^2} \end{aligned} \tag{2}$$

The same was done for $j = N$ (the last internal grid point on right side). The standard 3 points formula was modified by replacing U_{N+1} by its given value. Therefore, for $j = N$ the discrete equation becomes

$$\begin{aligned} \frac{U_{N-1} - 2U_N + \beta}{h^2} &= F_N \\ \frac{U_{N-1} - 2U_N}{h^2} &= F_N - \frac{\beta}{h^2} \end{aligned} \tag{3}$$

The above equations are collected and put in the form $Au = f$ resulting in

$$\frac{1}{h^2}\begin{pmatrix} -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 \end{pmatrix}\begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ \cdot \\ \cdot \\ \cdot \\ U_N \end{pmatrix} = \begin{pmatrix} f_h - \frac{\alpha}{h^2} \\ f_{2h} \\ f_{3h} \\ \cdot \\ \cdot \\ f_{(N-1)h} \\ F_{Nh} - \frac{\beta}{h^2} \end{pmatrix}$$

The above system is solved for the unknown vector U. These are the values of the numerical solution at the internal grid points. The A matrix above is symmetric and tridiagonal and of size $N \times N$ where N is the number of internal grid points.

Error norm calculation

The total error at a grid point j is given by

$$\boldsymbol{e}_j = \boldsymbol{U}_j - \boldsymbol{u}\left(x_j\right)$$

Where $\boldsymbol{U}_j$ is the numerical solution at the j^{th} grid point and $\boldsymbol{u}\left(x_j\right)$ is the exact solution sampled at the same grid point location. $\boldsymbol{e}$ is a vector of length N and represents the difference between U and u at each point.

To measure the size of the error vector $\boldsymbol{e}$, a grid norm is used in place of the standard vector norm. The following are the definitions of the norms used.

1. max-norm $\left\|\boldsymbol{e}^h\right\|_{\max} = \max_j \left|e_j\right|$

2. 1-norm $\left\|e^h\right\|_1 = h\sum_{j=1}^{N}\left|e_j\right|$

3. 2-norm $\left\|e^h\right\|_2 = \sqrt{h\sum_{j=1}^{N}\left|e_j\right|^2}$

Description of method used in Refinement study

The goal of the refinement study is to check that the scheme selected is stable. Since the scheme selected is consistent (it has an LTE of order $O\left(h^2\right)$ therefore $\|\tau\| \to 0$ as $h \to 0$), this implies that the scheme will be stable if it can be shown that the numerical solution converges to the exact solution[2].

For a stable scheme the following relation must hold

$$\|e\| \leq \left\|A^{-1}\right\| \|\tau\|$$

therefore stability can be established by verifying that error norm $\|e\|$ is also of order $O\left(h^2\right)$. This implies that $\left\|A^{-1}\right\|$ is $O(1)$. In addition, showing that $\|e\|$ is of order $O\left(h^2\right)$ implies that $\|e\| \to 0$ as $h \to 0$, which means convergence.

These are the steps carried out in the refinement study

1. The system $AU = f$ is formulated based on the scheme described above. These equations contain h (the grid space) in them as a free parameter. Initially h is given an initial starting value.

2. $A\boldsymbol{U} = \boldsymbol{f}$ is solved for $\boldsymbol{U}$.

3. The total error vector $\boldsymbol{e} = \boldsymbol{U} - \boldsymbol{u}$ is calculated and the error grid norm $\|e\|$ found using the the above definitions of norms.

4. The spacing h is divided by 2 and the above steps are repeated. The number of iterations was selected so that numerical convergence can be clearly observed. It is found that 5 or 6 iterations was sufficient to show convergence in this problem.

5. The error table and the log plots are generated. Convergence is verified by showing from the error table results that the total error norm $\|e\|$ has an order of accuracy of $O\left(h^2\right)$. This implies the numerical scheme selected is stable.

Results of the refinement study

This is a plot of $\log(h)$ against the log of the various error norms. The source code is in the appendix.

[2]Lax-Richtmyer theorem

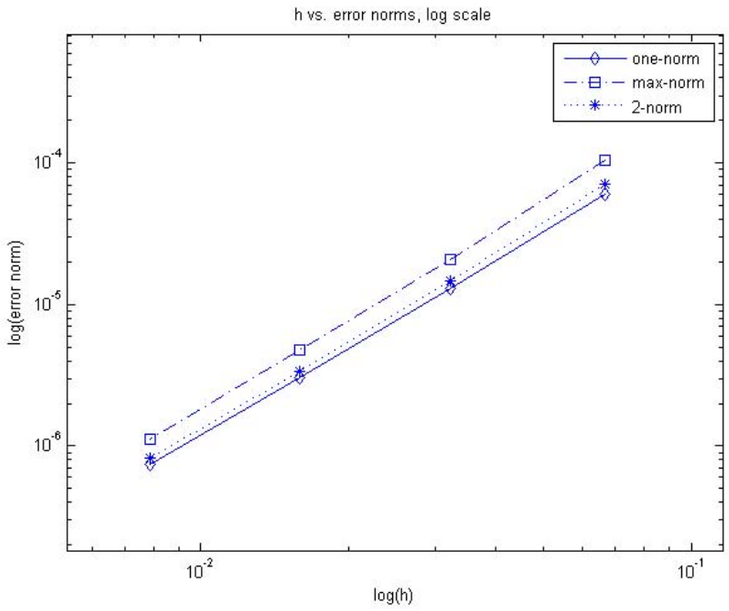


Figure 2.8: matlab HW2 part a logplot

The error table is the following

```
EDU>> nma_HW2_part_a
N           h        emax       ratio          e1       ratio          e2       ratio
16   6.6667e-002  1.0444e-004 0.0000e+000 5.9816e-005 0.0000e+000 7.0865e-005 0.0000e+000
32   3.2258e-002  2.0983e-005 4.9772e+000 1.3040e-005 4.5870e+000 1.4799e-005 4.7886e+000
64   1.5873e-002  4.7448e-006 4.4223e+000 3.0536e-006 4.2706e+000 3.4030e-006 4.3488e+000
128  7.8740e-003  1.1299e-006 4.1992e+000 7.3937e-007 4.1300e+000 8.1708e-007 4.1648e+000
256  3.9216e-003  2.7583e-007 4.0964e+000 1.8194e-007 4.0637e+000 2.0025e-007 4.0802e+000
512  1.9569e-003  6.8147e-008 4.0476e+000 4.5130e-008 4.0316e+000 4.9573e-008 4.0396e+000
1024 9.7752e-004  1.6937e-008 4.0236e+000 1.1238e-008 4.0157e+000 1.2333e-008 4.0196e+000
2048 4.8852e-004  4.2220e-009 4.0115e+000 2.8042e-009 4.0076e+000 3.0758e-009 4.0096e+000
```

.

Conclusion

The study was carried out using 3 different norms (max norm,1-norm and 2-norm). It is found that for each norm the total error ratio converged to 4, implying $p = 2$, hence the total error norm was $O\left(h^2\right)$ the same as LTE. This shows that the numerical scheme is stable and the numerical solution converges to the exact solution.

2.3.2.2 Part (b)

Solve $u_{xx} = 2\cos^2(\pi x)$ with $u_x(0) = 0, u_x(1) = 1$

2.3.2.3 Finding analytical solution

The existence of a solution is first verified. This is done in this case since for Neumann boundary conditions a solution might not even exist if the problem is not well posed. Integrating both sides of the PDE gives

$$\begin{aligned}\int_0^1 u_{xx}(x)\,dx &= \int_0^1 2\cos^2(\pi x)\,dx \\ &= u_x(1) - u_x(0) \\ &= 1\end{aligned}$$

Substituting the values of $u_x$on the boundaries, the above becomes zero, hence the problem is well posed. Now the analytical solution is found.

Integrating the differential equation once

$$u_x = x + \frac{\sin(2\pi x)}{2\pi} + c_1 \tag{1}$$

And Integrating again

$$u(x) = \frac{x^2}{2} - \frac{\cos(2\pi x)}{4\pi^2} + c_1 x + c_2 \tag{2}$$

Substituting the boundary condition $u_x(0) = 0$ in the above[3] gives $c_1 = 0$, therefore the solution is

$$u(x) = \frac{x^2}{2} - \frac{\cos(2\pi x)}{4\pi^2} + c_2 \tag{3}$$

This solution is not unique. The constant c_2 is arbitrary and an infinite number of solutions exist. A solution exist which is up to an arbitrary additive constant. To select a constant for the purpose of the numerical analysis in the refinement study, the constant is selected to give the solution zero mean. Hence

$$\int_0^1 u(x)\,dx = 0$$

$$\int_0^1 \left(\frac{x^2}{2} - \frac{\cos(2\pi x)}{4\pi^2} + c_2\right) dx = 0$$

$$\frac{1}{6} + c_2 = 0$$

Therefore $c_2 = -\frac{1}{6}$. Therefore, the exact solution used in the refinement study to compare the numerical solution against is

$$u(x) = \frac{x^2}{2} - \frac{\cos(2\pi x)}{4\pi^2} - \frac{1}{6}$$

Scheme to use for the numerical solution

Two schemes formulated, both having a local truncation error $\|\tau\|$ of order $O\left(h^2\right)$. Hence both schemes are consistent.

2.3.2.3.1 First scheme The standard 3 point centered difference formula $\frac{U_{j-1}-2U_j+U_{j+1}}{h^2}$ is used for approximating u_{xx} on internal grid points. However in this problem U_o and $U_{N+1}$are not known at the left most and the right most grid points (grid points of index 0 and $N+1$), therefore the grid points used includes these 2 points in addition to the standard internal grid points $(1 \cdots N)$ resulting in the A matrix having size $N+2$. (A will have 2 additional rows as compared to part (a)).

Now, two imaginary points are introduced into the domain, one to the left of $j = 0$, and one to the right of $j = N+1$

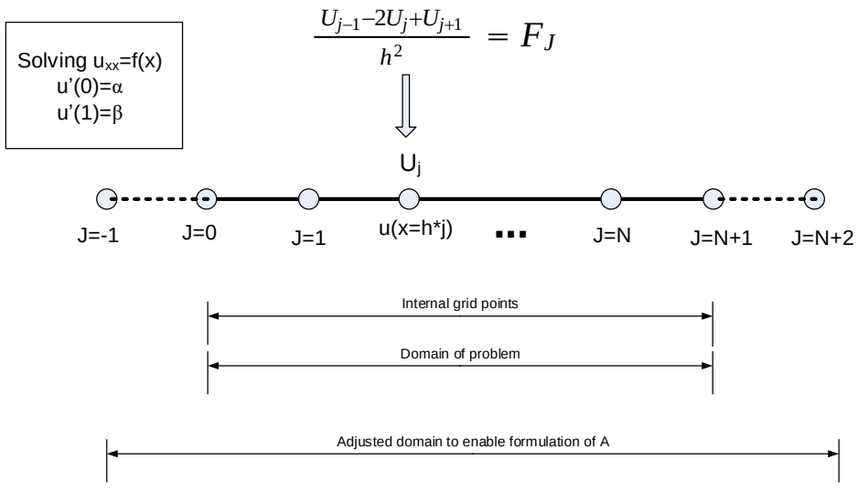


Figure 2.9: problem 1 part b scheme

For $j = 0$ the standard 3 points formulate is used to approximate u_{xx}

$$\frac{U_{-1} - 2U_0 + U_1}{h^2} = f_0 \tag{1}$$

[3]The other condition $u_x(1) = 1$ could also have been used, but the result would remain the same

And $u_x(0) = \alpha$ the 2 points central difference scheme is used

$$\frac{U_1 - U_{-1}}{2h} = \alpha \tag{2}$$

Now U_{-1} is eliminated using (1) and (2). From equation (2)

$$U_{-1} = U_1 - 2h\alpha$$

substituting the above into (1)

$$\frac{(U_1 - 2h\alpha) - 2U_0 + U_1}{h^2} = f_0$$
$$\frac{-2U_0 + 2U_1}{h^2} = f_0 + \frac{2\alpha}{h} \tag{3}$$

The above equation (3) is the discrete equation for node $j = 0$ (the first row in the A matrix). Similarly for the right-most node $j = N+1$, u_{xx} is approximated using

$$\frac{U_N - 2U_{N+1} + U_{N+2}}{h^2} = f_{N+1} \tag{4}$$

And $u_x(1) = \beta$ at node $j = N+1$ is approximated using 2 points central difference

$$\frac{U_{N+2} - U_N}{2h} = \beta \tag{5}$$

$U_{N+2}$is eliminated using (4) and (5). From equation (5)

$$U_{N+2} = U_N + 2h\beta$$

substituting the above into (4)

$$\frac{U_N - 2U_{N+1} + \left(U_N + 2h\beta\right)}{h^2} = f_{N+1}$$
$$\frac{2U_N - 2U_{N+1}}{h^2} = f_{N+1} - \frac{2\beta}{h} \tag{6}$$

The above equation (6) is the discrete equation for node $j = N+1$ (the last row in the A matrix). $Au = f$ is now setup resulting in

$$\frac{1}{h^2}\begin{pmatrix} -2 & 2 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{pmatrix}\begin{pmatrix} U_0 \\ U_1 \\ U_2 \\ \cdot \\ \cdot \\ N_{N-1} \\ U_N \\ U_{N+1} \end{pmatrix} = \begin{pmatrix} f_0 + \frac{2\alpha}{h} \\ f_1 \\ f_2 \\ \cdot \\ \cdot \\ f_N \\ f_{N+1} - \frac{2\beta}{h} \end{pmatrix}$$

The A matrix is tridiagonal, not symmetrical and singular since $null(A) = 1^T$. In some books, this matrix is called the Laplacian matrix.

2.3.2.3.2 Second scheme In this scheme, the first order derivative for the Neumann boundary condition at the left point and at the right point is approximated using

$$\alpha = \frac{1}{h}\left(\frac{3}{2}U_0 - 2U_1 + \frac{1}{2}U_2\right)$$
$$\beta = \frac{1}{h}\left(\frac{3}{2}U_{N+1} - 2U_N + \frac{1}{2}U_{N-1}\right)$$

respectively.

The derivation of the above formulas is shown in example 1.2 in the textbook. The above approximation has an LTE of $O\left(h^2\right)$. For the other internal grid points, the standard 3 points centred difference $\frac{U_{j-1} - 2U_j + U_{j+1}}{h^2}$ is used to approximate u_{xx}. Using the above approximations,

the system $AU = f$ becomes

$$\frac{1}{h^2}\begin{pmatrix} \frac{3}{2} & -2 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & -2 & \frac{3}{2} \end{pmatrix}\begin{pmatrix} U_0 \\ U_1 \\ U_2 \\ \cdot \\ \cdot \\ U_{N-1} \\ U_N \\ U_{N+1} \end{pmatrix} = \begin{pmatrix} \frac{\alpha}{h} \\ F_1 \\ F_2 \\ \cdot \\ \cdot \\ \cdot \\ F_N \\ \frac{\beta}{h} \end{pmatrix}$$

The A matrix is tridiagonal, not symmetrical and is also singular since $null(A) = 1^T$.

2.3.2.3.3 Augmenting the systems before solving The A matrix for both schemes above is singular. Both have $u = (1,1,1,\cdots,1,1)^T$ as the null vector. Using Matlab to verify, the null command can be used as follows

```
EDU>> A=nma_FDM_matrix_laplace_1D_Neumann_scheme_1(6)
A =
-2     2     0     0     0     0
1    -2     1     0     0     0
0     1    -2     1     0     0
0     0     1    -2     1     0
0     0     0     1    -2     1
0     0     0     0     2    -2
DU>>  null(A,'r')'
ans =
1     1     1     1     1     1
```

.

For the second scheme

```
EDU>> A=nma_FDM_matrix_laplace_1D_Neumann_scheme_2(6)
A =
1.5000   -2.0000    0.5000         0         0         0
1.0000   -2.0000    1.0000         0         0         0
0         1.0000   -2.0000    1.0000         0         0
0         0         1.0000   -2.0000    1.0000         0
0         0         0         1.0000   -2.0000    1.0000
0         0         0         0.5000   -2.0000    1.5000
EDU>>  null(A,'r')'
ans =
1     1     1     1     1     1
```

.

Since the matrix A is singular, before solving by Gaussian elimination it is augmented as follows

$$A \rightarrow \begin{pmatrix} A & v \\ u & 0 \end{pmatrix}$$

where v is the the null of A^* (the adjoint of A, or for a real matrix, the same as A^T) and u is the null of A.

v is found for the above 2 schemes. For the first scheme:

```
EDU>> A=nma_FDM_matrix_laplace_1D_Neumann_scheme_1(6);
EDU>> null(A','r')'
ans =
1     2     2     2     2     1
```

.

And for the second scheme

```
EDU>> A=nma_FDM_matrix_laplace_1D_Neumann_scheme_2(6);
EDU>> null(A','r')'
ans =
1.0000   -1.5000   -1.0000   -1.0000   -1.5000    1.0000
```

.

Now that v and u are found, the augmented A can be formulated.

The force vector is also augmented as follows

$$f \rightarrow \begin{pmatrix} f \\ 0 \end{pmatrix} \tag{7}$$

The augmented system is now solved for U using Gaussian elimination. In Matlab

$$U = A \backslash f$$

The resulting solution U will have an extra element at the end, which is not used in the solution. This element, called λ was verified to be equal $\frac{v \cdot f}{v \cdot v}$.

Notice that last element of f was set to 0 in (7), this is because it corresponds to having selected an exact solution with a zero mean. When the last element of f is set to 0, this corresponds to having $\sum_{i=0}^{N+1} U_i = 0$, which implies the mean of the numerical solution is zero. This follows because $null(A)$ was found to be 1^T, meaning that, from looking at $\begin{pmatrix} A & v \\ 1^T & 0 \end{pmatrix} \begin{pmatrix} U \\ \lambda \end{pmatrix} = \begin{pmatrix} f \\ 0 \end{pmatrix}$, the $dot\left(1^T, U\right) = 0$, which is the same as saying that $\sum_{i=0}^{N+1} U_i = 0$ or the mean of the numerical solution is zero.

To summarize: A constant in the analytical solution was earlier selected which resulted in the mean of the exact solution to be zero. This constant was found to be $\frac{-1}{6}$. The last entry in the augment f vector is set to be zero to cause the numerical solution to have zero mean as well.

Now that the numerical solution is found, the error vector $e = U - u$ is found and its norms are calculated to complete the refinement study. Below is the result for both schemes described above.

Refinement study results

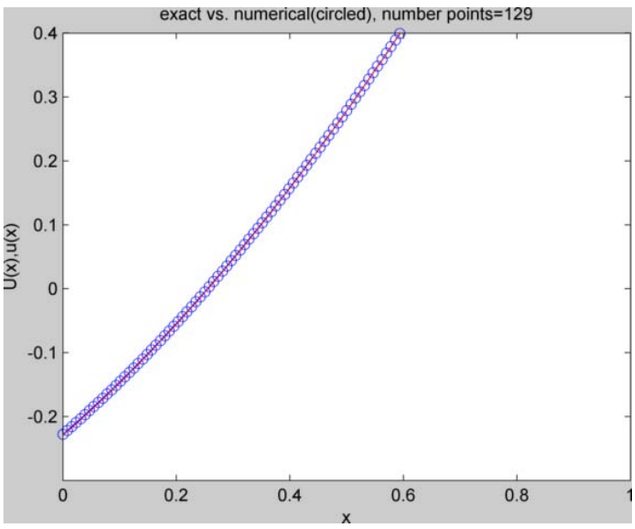


Figure 2.10: nma HW2 prob1 part b solution

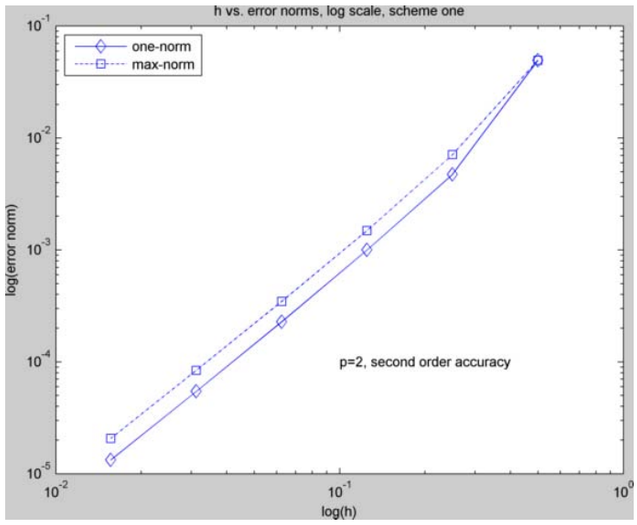


Figure 2.11: nma HW2 part b scheme one plot2

2.3.2.3.4 Scheme one

The error table is the following:

```
EDU>> nma_HW2_part_b_scheme_one
N           h       emax       ratio          e1       ratio
3    0.5000000 4.9560e-002  0.0000e+000 4.9560e-002 0.0000e+000
5    0.2500000 7.1036e-003  6.9766e+000 4.7358e-003 1.0465e+001
9    0.1250000 1.4925e-003  4.7596e+000 9.9728e-004 4.7487e+000
17   0.0625000 3.4734e-004  4.2969e+000 2.2786e-004 4.3768e+000
33   0.0312500 8.4008e-005  4.1346e+000 5.4367e-005 4.1912e+000
65   0.0156250 2.0668e-005  4.0646e+000 1.3271e-005 4.0967e+000
```

.

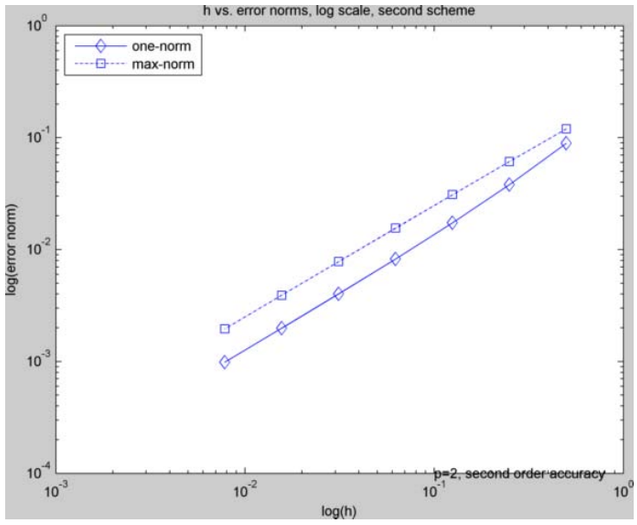


Figure 2.12: HW2 prob1 part b scheme 2

2.3.2.3.5 Scheme 2

And the error table is

```

EDU>> nma_HW2_prob1_part_b(2)
N           h        emax       ratio          e1       ratio
3    0.5000000 5.9960e-003 0.0000e+000 5.9960e-003 0.0000e+000
5    0.2500000 6.0976e-003 9.8333e-001 6.0976e-003 9.8333e-001
9    0.1250000 8.1787e-004 7.4554e+000 4.3938e-004 1.3878e+001
17   0.0625000 1.6346e-004 5.0034e+000 8.9562e-005 4.9058e+000
33   0.0312500 6.0030e-005 2.7230e+000 3.6203e-005 2.4739e+000
```

```
65    0.0156250 1.7602e-005 3.4103e+000 1.0956e-005 3.3043e+000
129   0.0078125 4.7387e-006 3.7146e+000 2.9860e-006 3.6693e+000
257   0.0039063 1.2278e-006 3.8594e+000 7.7785e-007 3.8388e+000
513   0.0019531 3.1240e-007 3.9302e+000 1.9841e-007 3.9204e+000
1025  0.0009766 7.8786e-008 3.9652e+000 5.0098e-008 3.9605e+000
```

.

Conclusion

Two different schemes were used for solving part(b). Both schemes had an LTE of $O\left(h^2\right)$, therefore they both were consistent numerical schemes. The result shows that $\|e\|$ had $O\left(h^2\right)$ implying $\left\|A^{-1}\right\|$ had order $O(1)$ and that both scheme were stable.

It is observed that the second scheme required more iterations to coverage to the exact solution compared to the first scheme. Attempt was made to find if any error was made in the derivation of the scheme or in the code, but none found. Therefore, this showed that when comparing 2 schemes with the same theoretical order of accuracy, this does not necessarily imply they will have the same exact performance on the same numerical example. Therefore, before selecting a scheme, it would be useful to always run a number of numerical tests to compare schemes against each others on different numerical test problems.

2.3.3 Problem 2

Propose a discretization scheme for $u_{xx} = f$ with $u_x(0) - \alpha u(0) = g$ and $u(1) = b$

Answer:

This problem has Robin boundary conditions on the left side and Dirichlet boundary conditions on the right side. The condition for existence of a solution implies

$$u_x(1) - u_x(0) = \int_0^1 f(x)\,dx$$

Substituting the boundary conditions into the above results in

$$-\alpha u(0) - g = \int_0^1 f(x)\,dx$$

Hence any $f(x)$ which satisfies the above will make the problem a well posed one. In the following, 3 different schemes are presented using different methods of approximating the Robin boundary conditions. The last scheme resulted in an A matrix with the most desirable properties being tridiagonal and symmetric.

2.3.3.1 First scheme

$u_x(0)$ is approximated by 2 points centered difference by introducing an imaginary point U_{-1} resulting in

$$u_x(0) = \frac{U_{-1} - U_1}{2h}$$

Substituting the above in the Robin boundary condition gives

$$\frac{U_{-1} - U_1}{2h} - \alpha U_0 = g \tag{1}$$

$u_{xx}(0)$ is approximated using the standard 3 point formula

$$\frac{U_{-1} - 2U_0 + U_1}{h^2} = f_0 \tag{2}$$

U_{-1} is now eliminated From (1) and (2). From equation (1)

$$U_{-1} = 2h\left(g + \alpha U_0\right) + U_1$$

Substituting into (2)

$$\frac{\left[2h\left(g + \alpha U_0\right) + U_1\right] + 2U_0 + U_1}{h^2} = f_0$$

And simplifying

$$\frac{2U_0(1+\alpha h)+2U_1}{h^2} = f_0 - \frac{2g}{h}$$

This will be the equation to use at node $j = 0$ which is the first row in the A matrix.

For node $j = N$, $u_{xx}(N)$ is approximated using the standard 3 point formula

$$\frac{U_{N-1} - 2U_N + U_{N+1}}{h^2} = f_N$$

Since $U_{N+1} = b$, the above becomes

$$\frac{U_{N-1} + 2U_N}{h^2} = f_N - \frac{b}{h^2}$$

Which is the equation for node $j = N$. For all the remaining internal grid point, the standard 3 point formula is used to approximate u_{xx}. Therefore, the system which now contains $N+1$ equations is completed

$$\frac{1}{h^2}\begin{pmatrix} 2(1+\alpha h) & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 \end{pmatrix}\begin{pmatrix} U_0 \\ U_1 \\ U_2 \\ \cdot \\ \cdot \\ \cdot \\ U_{N-1} \\ U_N \end{pmatrix} = \begin{pmatrix} f_0 - \frac{2g}{h} \\ f_1 \\ f_2 \\ \cdot \\ \cdot \\ \cdot \\ f_{N-1} \\ f_N - \frac{b}{h^2} \end{pmatrix}$$

2.3.3.2 Second scheme

Approximating $u_x(0)$ by 3 points forward difference

$$u_x(0) = \frac{1}{h}\left(\frac{3}{2}U_0 - 2U_1 + \frac{1}{2}U_2\right)$$

Substituting the above in the Robin boundary condition gives

$$\frac{1}{h}\left(\frac{3}{2}U_0 - 2U_1 + \frac{1}{2}U_2\right) - \alpha U_0 = g$$

$$U_0\left(\frac{3}{2} - \alpha h\right) - 2U_1 + \frac{1}{2}U_2 = hg$$

Dividing both sides by h^2 to allow writing the matrix A with a common $\frac{1}{h^2}$ factored out

$$\frac{U_0\left(\frac{3}{2} - \alpha h\right) - 2U_1 + \frac{1}{2}U_2}{h^2} = \frac{g}{h}$$

For the right side, the same scheme is used as in the first scheme above

$$\frac{U_{N-1} + 2U_N}{h^2} = f_N - \frac{b}{h^2}$$

And the system becomes

$$\frac{1}{h^2}\begin{pmatrix} \left(\frac{3}{2} - \alpha h\right) & -2 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdot & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdot & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdot & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 \end{pmatrix}\begin{pmatrix} U_0 \\ U_1 \\ U_2 \\ \cdot \\ \cdot \\ \cdot \\ U_{N-1} \\ U_N \end{pmatrix} = \begin{pmatrix} \frac{g}{h} \\ f_1 \\ f_2 \\ \cdot \\ \cdot \\ \cdot \\ f_{N-1} \\ f_N - \frac{b}{h^2} \end{pmatrix}$$

2.3.3.3 Third scheme

Approximating $u_x(0)$ by 2 points forward difference which has an LTE of $O(h)$

$$u_x(0) = \frac{1}{h}(U_1 - U_0)$$

Substituting the above in the Robin boundary condition gives

$$\begin{aligned}\frac{1}{h}(U_1 - U_0) - \alpha U_0 &= g\\ U_1 - U_0 - \alpha h U_0 &= hg\\ U_0(-1 - \alpha h) + U_1 &= hg\end{aligned}$$

Dividing both sides by h^2 to allow writing the matrix A with a common $\frac{1}{h^2}$ factored out

$$\frac{[U_0(-1 - \alpha h) + U_1]}{h^2} = \frac{g}{h}$$

For the right side, the same scheme was used as in the first scheme above, resulting in

$$\frac{1}{h^2}\begin{pmatrix}(-1-\alpha h) & 1 & 0 & 0 & 0 & 0 & 0 & 0\\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0\\ 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0\\ 0 & 0 & 0 & 0 & \cdot & \cdot & \cdot & 0\\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1\\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2\end{pmatrix}\begin{pmatrix}U_0\\ U_1\\ U_2\\ \cdot\\ \cdot\\ \cdot\\ U_{N-1}\\ U_N\end{pmatrix} = \begin{pmatrix}\frac{g}{h}\\ f_1\\ f_2\\ \cdot\\ \cdot\\ \cdot\\ f_{N-1}\\ f_N - \frac{b}{h^2}\end{pmatrix}$$

This scheme has an LTE of $O(h)$ since an $O(h)$ approximation was used for $u_x(0)$

2.3.3.4 Order of accuracy

Each of the above 3 schemes is a consistent scheme because for the first 2 schemes the LTE is $O\left(h^2\right)$ and for the third scheme the LTE is $O(h)$. A consistent scheme is one in which $\|\tau\| \to 0$ as $h \to 0$. It expected that the final error norm $\|e\|$ will behave as $O\left(h^2\right)$ when using the first 2 scheme and as $O(h)$ when using the third scheme. This is provided A is stable. Meaning that $\left\|A^{-1}\right\|$ is of order $O(1)$.

Considering the third scheme only since it is symmetric, and using the grid 2-norm, then given that

$$\begin{aligned}\|e\|_2 &= \left\|A^{-1}\tau\right\|_2\\ &\leq \left\|A^{-1}\right\|_2 \|\tau\|_2\end{aligned}$$

And since for a symmetric matrix $\left\|A^{-1}\right\|_2 = \rho\left(A^{-1}\right) = \frac{1}{|\lambda_{\min}|}$ where $|\lambda_{\min}|$ is the smallest eigenvalue of A in absolute value, then showing that $|\lambda_{\min}| \geq 1$ would imply that $\|e\|_2 \leq \|\tau\|_2$.

There two analytical methods for finding the $\lambda_{\min}$ of A. By solving the eigenvalue problem $L\phi = \lambda\phi$ for the given boundary conditions or by using matrix algebra to solve for the eigenvalue of A directly. Attempt are made at both of these methods, but more time is needed to complete an analytical proof.

Therefore, to answer the question for this problem, a numerical method was used instead, where a refinement study was done using the third scheme found above in an attempt to show numerically that $\|e\|$ is of the same order as $\|\tau\|$ as expected. A well formed problem is constructed and used for the purpose of the numerical analysis part. The results and the conclusion follows.

2.3.3.5 Refinement study for the third scheme

The following numerical problem was select. $u_{xx} = f$ with $u_x(0) - \alpha u(0) = g$ and $u(1) = b$ with the following parameter values

$$\alpha = 1$$
$$g = 1$$
$$b = 1$$
$$f = \cos(x)$$

This problem is well posed, and has the following analytical solution

$$u(x) = \frac{1}{2}(1 + b - g - x + b\,x + g\,x + \cos(1) + x\,\cos(1) - 2\cos(x))$$

And it satisfies the necessary condition $-\alpha u(0) - g = \int_0^1 f(x)\,dx$

The following are the results of the refinement study. The source code is in the appendix.

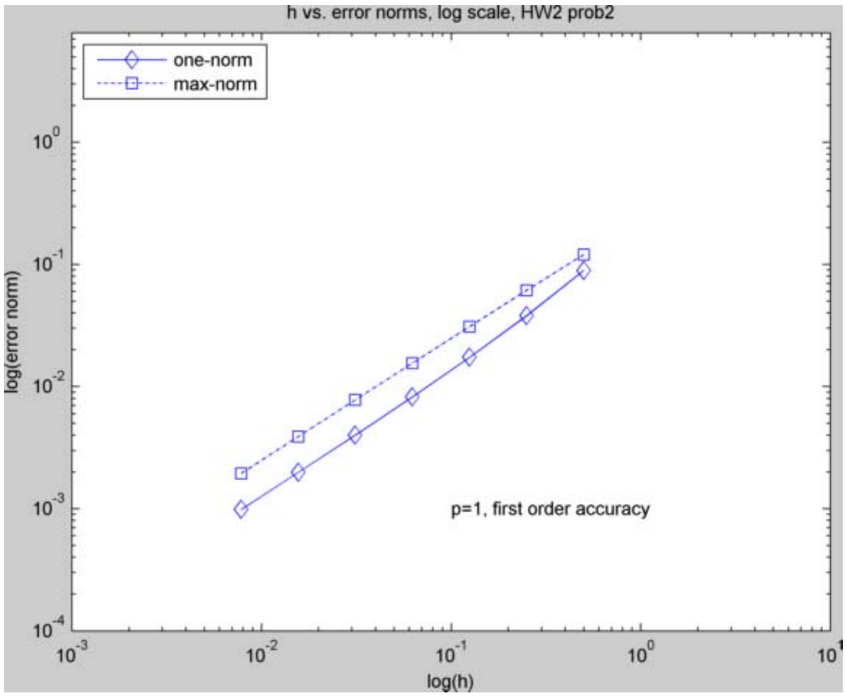


Figure 2.13: HW2 prob2

```
\begin{X301}
EDU>> close all; nma_HW2('hw2_prob2')
N           h        emax       ratio          e1         ratio
3     0.5000000  1.2015e-001  0.0000e+000  8.8980e-002  0.0000e+000
5     0.2500000  6.1299e-002  1.9601e+000  3.7962e-002  2.3439e+000
9     0.1250000  3.0950e-002  1.9806e+000  1.7318e-002  2.1921e+000
17    0.0625000  1.5550e-002  1.9904e+000  8.2379e-003  2.1023e+000
33    0.0312500  7.7938e-003  1.9952e+000  4.0129e-003  2.0529e+000
65    0.0156250  3.9016e-003  1.9976e+000  1.9798e-003  2.0269e+000
129   0.0078125  1.9520e-003  1.9988e+000  9.8324e-004  2.0136e+000
```

.

2.3.3.6 Conclusion

A scheme was implemented for Robin boundary conditions. LTE used was $O(h)$ since this specific scheme generated an A matrix with desirable form (symmetrical and tridiagonal) . The expectation of total error using this scheme was to be $O(h)$. This expectation was confirmed by doing a refinement study on the selected test problem. This result showed

that the scheme is stable and therefore we conclude that $\|A\|$ is of order $O(1)$ based on this test case.

2.3.4 Problem 3

2.3.4.1 Part (a)

From definition

$$\|e\|_\infty = -\|B\tau\|_\infty \tag{1}$$

Where $B = A^{-1}$ was generated by the use of Green function as described on page 27 of the text book. Writing the product of the matrix B with the vector $\boldsymbol{\tau}$ as

$$B\boldsymbol{\tau} = b_1\tau_1 + b_1\tau_1 + \cdots + b_N\ \tau_N$$

where $\boldsymbol{b}_j$ is the j^{th} column of matrix B, equation (1) becomes (all norms are to be taken as the max-norm, hence for clarify, the ∞ is dropped in what follows)

$$\begin{aligned}\|e\| &= -\|b_1\tau_1 + b_1\tau_1 + \cdots + b_N\tau_N\| \\ &\le -(\|b_1\tau_1\| + \|b_2\tau_2\| + \cdots + \|b_N\tau_N\|) \\ &\le -(\|b_1\|\,|\tau_1| + \|b_2\|\,|\tau_2| + \cdots + \|b_N\|\,|\tau_N|)\end{aligned} \tag{3}$$

$\|\boldsymbol{b}_j\|$ is the maximum element in the j^{th} column of the matrix B. These elements are located along the diagonal of B. Hence $\|\boldsymbol{b}_j\| = |B_{jj}|$ and (3) becomes

$$\|e\| \le -(|B_{11}|\ |\tau_1| + |B_{22}|\ |\tau_2| + \cdots + |B_{NN}|\ |\tau_N|) \tag{4}$$

From equation 2.46 in the textbook,

$$B_{ij} = \begin{cases} h(x_j - 1)x_i & i = 1, 2, \cdots j \\ h(x_i - 1)x_j & i = j, j+1, \cdots, N \end{cases} \tag{5}$$

To answer the question, assume that $|\tau_1| = O(h^p)$, and assume for the moment, for generality, that the LTE at all the other points was different, hence for other points let $|\tau_j| = O(h^q)$. Equation (4) becomes

$$\|e\| \le -O(h^p)\,|B_{11}|\ - O(h^q)\sum_{k=2}^{N}|B_{kk}| \tag{6}$$

From (5), it is found that at point $j = 1$, $|B_{11}| = h(h-1)h = h^3 - h^2 = O\left(h^2\right)$, while at the internal points, that is, at points near the middle, B_{jj} is approximated by $O(h)$, hence (6) becomes

$$\|e\| \le -O(h^p)\,O\left(h^2\right)\ - O(h^q)\sum_{k=2}^{N}O(h)$$

But $\sum_{k=2}^{N}O(h) \sim (N-1)\times O(h) = O(1)$ since $N = \frac{1}{h}$, hence the above becomes

$$\begin{aligned}\|e\| &\le -O(h^p)\,O\left(h^2\right)\ - O(h^q) \\ &= -O\left(h^{p+2}\right) - O(h^q)\end{aligned} \tag{7}$$

Therefore, if $q = p$, i.e. if the LTE was the same at each grid point, including the first, the above simplifies to

$$\begin{aligned}\|e\| &\le -O\left(h^{p+2}\right) - O(h^p) \\ &\sim O(h^p)\end{aligned}$$

Hence, having the LTE at the edge grid point be $O(h^p)$ resulted in $\|e\|_\infty$ having an order of accuracy $O(h^p)$ which is the expected. The smallest value of p that can be used for the edge point while still giving overall $\|e\| = O\left(h^2\right)$ can be found from (7)

$$\|e\| = O\left(h^2\right) = -O\left(h^{p+2}\right) - O(h^q)$$

Setting $p = 0$ and $q = 2$ results in

$$\begin{aligned}\|e\| &= -O\left(h^2\right) - O\left(h^2\right) \\ &\sim O\left(h^2\right)\end{aligned}$$

Hence $p = 0$ or $O(1)$, is possible for the LTE at the edge while arriving at an overall

$\|e\| \sim O\left(h^2\right)$.

2.3.4.2 Part (b)

From equation 2.46 in the textbook,

$$B_{ij} = \begin{cases} h(x_j - 1)x_i & i = 1, 2, \cdots j \\ h(x_i - 1)x_j & i = j, j+1, \cdots, N \end{cases} \tag{5}$$

To answer the question, assume $\left|\tau_{j=N/2}\right| = O\left(h^p\right)$, where j is the middle point, i.e. at $x_j = \frac{1}{2}$and assume for other points $\left|\tau_j\right| = O\left(h^q\right)$, hence starting from

$$\|e\| \leq -\left(|B_{11}|\ |\tau_1| + |B_{22}|\ |\tau_2| + \cdots + |B_{NN}|\ |\tau_N|\right)$$

which was derived in part(a), then the above equation becomes

$$\|e\| \leq -O\left(h^q\right) \sum_{\substack{k=1 \\ k \neq N/2}}^{N} |B_{kk}| - O\left(h^p\right) \left|B_{N/2,N/2}\right| \tag{8}$$

At the middle point from (5), $\left|B_{N/2,N/2}\right|$ can be found to be $h\left(\frac{1}{2} - 1\right)\frac{1}{2} = \frac{-1}{4}h \sim O(h)$, while for the points away from the middle, $\left|B_{jj}\right|$ is approximated by $O\left(h^2\right)$, hence (8) becomes

$$\|e\| \leq -O\left(h^q\right)\left[(N-1) \times O\left(h^2\right)\right] - O\left(h^p\right) O(h)$$

But $(N-1) \times O\left(h^2\right)$ is $O(h)$ since $N = \frac{1}{h}$, hence the above becomes

$$\begin{aligned} \|e\| &\leq -O\left(h^q\right) O(h) - O\left(h^{p+1}\right) \\ &= -O\left(h^{q+1}\right) - O\left(h^{p+1}\right) \end{aligned} \tag{9}$$

Therefore, if $q = p$, i.e. if the LTE was the same at each grid point, including the middle, then (9) becomes

$$\begin{aligned} \|e\| &= -O\left(h^{p+1}\right) - O\left(h^{p+1}\right) \\ &= O\left(h^{p+1}\right) \end{aligned}$$

Hence, having the LTE at the middle grid point be $O\left(h^p\right)$ resulted in $\|e\|_\infty$ having an order of accuracy $O\left(h^{p+1}\right)$ which is higher than the expected $O\left(h^p\right)$. The smallest value of p that can be used while still giving $\|e\| = O\left(h^2\right)$ can be found from (9)

$$\|e\| = O\left(h^2\right) = -O\left(h^{q+1}\right) - O\left(h^{p+1}\right)$$

Setting $p = 1$ and $q = 1$ results in $\|e\| \sim O\left(h^2\right)$.

Hence $p = 1$ or $O(h)$ is possible for the LTE at the middle point while still resulting in an overall $\|e\| = O\left(h^2\right)$

2.3.4.3 Part (c)

To perform the numerical tests to verify the above conclusion, a numerical problem is used with a scheme which had an LTE of $O\left(h^2\right)$. The LTE at the edge point only was then modified to test part(a) and the LTE at the middle point only was modified to test part(b).

Refinement study was done to verify that the order of accuracy remained $O\left(h^2\right)$ after each modification.

For the numerical test, problem 1, part (a) was used for this test. That problem had an LTE of $O\left(h^2\right)$ at each grid point, and the standard 3-point central difference was used for approximating u_{xx}.

To modify the LTE at a specific grid point, given that the LTE is defined as

$$\tau_j = \frac{u(j-1) - 2u\left(j\right) + u\left(j+1\right)}{h^2} - f_j$$

Where u above is the exact solution sampled at the given grid points, then to force the LTE to be $O(h)$ in the middle, the above equation was modified for $x_j = \frac{1}{2}$(middle row of

the system) to become

$$\tau_j = \frac{u(j-1) - 2u\left(j\right) + u\left(j+1\right)}{h^2} - f_j + h$$

And to force the LTE to be $O(1)$ at the first grid point, the LTE is modified for that point only (i.e. $x_j = h$, the first row of the system) to become

$$\tau_j = \frac{u(j-1) - 2u\left(j\right) + u\left(j+1\right)}{h^2} - f_j + 1$$

The refinement study which was done for problem 1, part a, is now repeated with the above modifications. The following are the results generated.

Part (A) refinement

Before making the LTE modification for the first point, the error table generated was

```
EDU>> close all; nma_HW2('hw2_prob1_parta')
N           h        emax       ratio          e1        ratio
3    0.5000000  4.3295e-003  0.0000e+000  2.1647e-003  0.0000e+000
5    0.2500000  1.0925e-003  3.9628e+000  6.8494e-004  3.1605e+000
9    0.1250000  2.7377e-004  3.9906e+000  1.8036e-004  3.7977e+000
17   0.0625000  6.8827e-005  3.9776e+000  4.5662e-005  3.9499e+000
33   0.0312500  1.7234e-005  3.9937e+000  1.1451e-005  3.9875e+000
65   0.0156250  4.3098e-006  3.9987e+000  2.8650e-006  3.9969e+000
129  0.0078125  1.0776e-006  3.9995e+000  7.1640e-007  3.9992e+000
```

.

One can see that $\|e\| = O\left(h^2\right)$ as the ratio is 4. Now the f vector was modified as described above, and here is the small code segment how this was done. Complete code listing is at the end.

```
f      = force(xcoordinates)';
f(1)   = f(1) - alpha/h^2 +1 ;
f(end) = f(end)-beta/h^2;
```

.

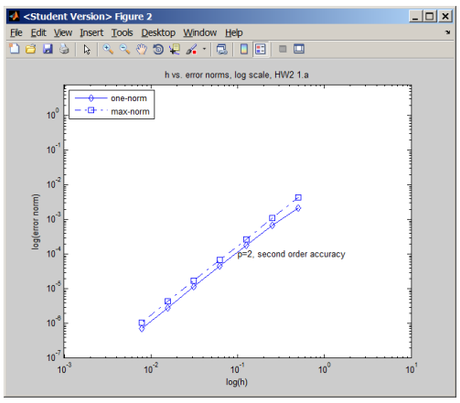
And now the program was run again

```
EDU>> close all; nma_HW2('hw2_prob3_1')
N           h        emax       ratio          e1         ratio
3    0.5000000  1.2067e-001  0.0000e+000  6.0335e-002  0.0000e+000
5    0.2500000  4.6119e-002  2.6165e+000  2.2753e-002  2.6518e+000
9    0.1250000  1.3566e-002  3.3997e+000  6.6556e-003  3.4186e+000
17   0.0625000  3.6481e-003  3.7185e+000  1.7854e-003  3.7278e+000
33   0.0312500  9.4426e-004  3.8635e+000  4.6157e-004  3.8681e+000
65   0.0156250  2.4010e-004  3.9328e+000  1.1730e-004  3.9350e+000
129  0.0078125  6.0530e-005  3.9666e+000  2.9563e-005  3.9678e+000
```

.

Examining the second table above shows that $\|e\|_\infty = O\left(h^2\right)$, since the ratio of the error norm still converged to 4, implying $p = 2$ for $\|e\|$. To compare the loglog plots, here are plots before and after the modification is made to the LTE.

LTE is O(h^2) on all points



LTE is O(h^2) on all points, except at first point where it is O(1)

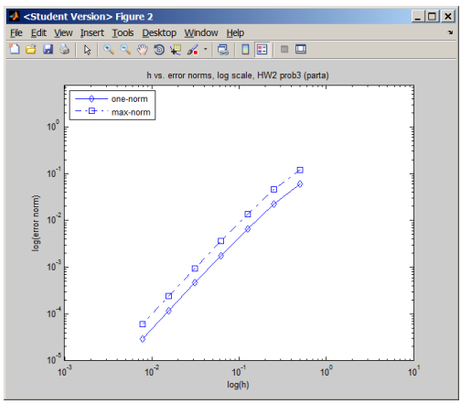


Figure 2.14: prob3 part a log

Part(b) refinement study

Before making the LTE modification for the middle point, the error table generated was

```
EDU>> close all; nma_HW2('hw2_prob1_parta')
N           h        emax       ratio          e1        ratio
3    0.5000000  4.3295e-003  0.0000e+000  2.1647e-003  0.0000e+000
5    0.2500000  1.0925e-003  3.9628e+000  6.8494e-004  3.1605e+000
9    0.1250000  2.7377e-004  3.9906e+000  1.8036e-004  3.7977e+000
17   0.0625000  6.8827e-005  3.9776e+000  4.5662e-005  3.9499e+000
33   0.0312500  1.7234e-005  3.9937e+000  1.1451e-005  3.9875e+000
65   0.0156250  4.3098e-006  3.9987e+000  2.8650e-006  3.9969e+000
129  0.0078125  1.0776e-006  3.9995e+000  7.1640e-007  3.9992e+000
```

.

One can see that $\|e\| = O\left(h^2\right)$ because the ratio is 4. Now the f vector was modified as described above, to force the middle point to have an LTE of $O(h)$, here is the small code segment showing the modification.

```
f                  = force(xcoordinates)';
f(1)               = f(1)-alpha/h^2;
middle_position    = round(length(f)/2);
f(middle_position) = f(middle_position) + h;
f(end)             = f(end)-beta/h^2;
```

.

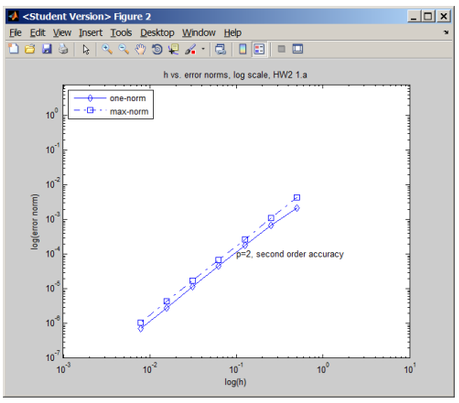
And now the program was run again

```
EDU>> close all; nma_HW2('hw2_prob3_2')
N           h        emax       ratio          e1        ratio
3    0.5000000 5.8171e-002 0.0000e+000 2.9085e-002 0.0000e+000
5    0.2500000 1.4532e-002 4.0028e+000 7.1276e-003 4.0807e+000
9    0.1250000 3.6325e-003 4.0007e+000 1.7728e-003 4.0206e+000
17   0.0625000 9.0808e-004 4.0002e+000 4.4262e-004 4.0052e+000
33   0.0312500 2.2702e-004 4.0000e+000 1.1062e-004 4.0013e+000
65   0.0156250 5.6754e-005 4.0000e+000 2.7653e-005 4.0003e+000
129  0.0078125 1.4189e-005 4.0000e+000 6.9130e-006 4.0001e+000
```

.

Examining the second table above, shows that $\|e\|_\infty = O\left(h^2\right)$, since the ratio of the error norm still converged to 4, implying $p = 2$ for $\|e\|$. To compare the loglog plots, here are plots before and after the modification is made to the LTE.

LTE is $O(h^2)$ on all points



LTE is $O(h^2)$ on all points, except at middle point where it is $O(h)$

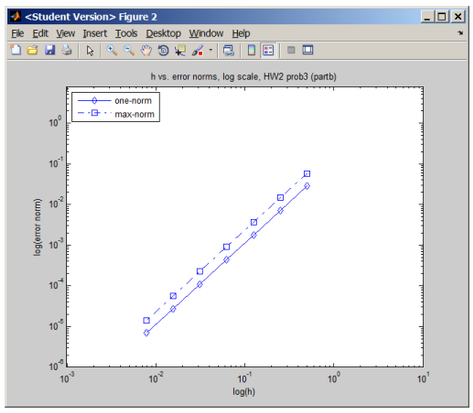


Figure 2.15: prob3 part blog

This concludes the refinement study verifying results from part(a) and part(c)

2.3.5 Source code written for this HW

```
%-----------------------------------------------------------------------
% HW 2  Math 228A, UC Davis, Fall 2010
% by Nasser M. Abbasi
%
% This .m files contains the main functions needed to solve HW2
%
% To Run:
%                From Matlab, type nma_HW2( problem_name )
%
% where problem_name is a string indicating which problem to run the
% refinment study for. The strings are
%
%         hw2_prob1_parta
%         hw2_prob1_partb_scheme_1
%         hw2_prob1_partb_scheme_2
%         hw2_prob2
%         hw2_prob3_1
%         hw2_prob3_2
%
% EXAMPLE
%
%          nma_HW2( 'hw2_prob1_parta' )
%
% make sure this m file is in your Matlab path.
%
function  nma_HW2()
close all;
problems={'hw2_prob1_parta';
    'hw2_prob1_partb_scheme_1';
    'hw2_prob1_partb_scheme_2';
    'hw2_prob2';
    'hw2_prob3_1';
    'hw2_prob3_2'};
for i=1:2 %length(problems)
    nma_HW2_sub( problems(i) )
end
fpritnf('completed...\n');

end

%-------------------------------------------------------
```

```
function  nma_HW2_sub( ID )

if nargin ~= 1
    error('one argument is required');
end

N      = 7;                              %number of iterations,
points = arrayfun( @(i) (2^i)+1 ,1:N );  % find number of grid points
h      = arrayfun( @(i) 1/(2^i) ,1:N );  % and corresponding spacings
data   = zeros(N,6);                     %allocate space for error table

%
% Main loop. For each number of points, find error norms and ratios
%
for i = 1:N

    if strcmp(ID,'hw2_prob1_partb_scheme_1')
        [e,U,u,x] = error_vector_prob1_partb_scheme1(points(i),h(i));
    elseif strcmp(ID,'hw2_prob1_partb_scheme_2')
        [e,U,u,x] = error_vector_HW2_prob1_partb_scheme2(points(i),h(i));
    elseif strcmp(ID,'hw2_prob1_parta')
        [e,U,u,x] = error_vector_HW2_prob1_parta(points(i),h(i));
    elseif strcmp(ID,'hw2_prob2')
        [e,U,u,x] = error_vector_HW2_prob2(points(i),h(i));
    elseif strcmp(ID,'hw2_prob3_1')
        [e,U,u,x] = error_vector_HW2_prob3_1(points(i),h(i));
    elseif strcmp(ID,'hw2_prob3_2')
        [e,U,u,x] = error_vector_HW2_prob3_2(points(i),h(i));
    else
        error('invalid problem name');
    end

    % the columns of data are arranged in this order
    %       npoints  h   e-max  ratio   e-1    ratio
    %         (1)   (2)    (3)   (4)    (5)     (6)

    data(i,1) = points(i);                    % number total points
    data(i,2) = h(i);
    data(i,3) = norm(e,inf);                  % e-max
    data(i,5) = h(i)*norm(e,1);               % e-1

    if i>1
        data(i,4) = data(i-1,3)/data(i,3);    %e-max ratio
        data(i,6) = data(i-1,5)/data(i,5);    %e-1 ratio
    end

    %plot exact vs. approximate solution to verification
    plot(x,U(:),'-o',x,u(:),'r');
    title(sprintf('exact vs. numerical(circled), number points=%d', ...
        points(i)));
    axis([0 1 -.3 .4]);
    xlabel('x'); ylabel('U(x),u(x)');
    drawnow;
    pause(1); % to enable animation to be seen
end

analysis_of_result(data,ID);          %generate error table/plots
end

%-----------------------------------------------------------------
% This function process the error table, formating it.
% It accept the e-norm found at each spacing, the vector with the
% corresponding h spacing value, calculates the error ratio
```

```
% and print the result to the screen
%
function analysis_of_result(data,ID)
titles ={'N','h','emax','ratio','e1','ratio'};
fms    ={'d','.7f','.4e','.4e','.4e','.4e'};

wid       = 12;
fileID    = 1;
nma_format_matrix(titles,data,wid,fms,fileID,false);

figure;
set(0,'defaultaxesfontsize',8) ;
set(0,'defaulttextfontsize',8);

loglog(data(:,2),data(:,5),'-d'); hold on;
loglog(data(:,2),data(:,3),'-.s');

xlabel('log(h)'); ylabel('log(error norm)');
legend('one-norm','max-norm','Location','NorthWest');
axis equal;
grid off;

% plot the correct title based on which problem is being solved
if strcmp(ID,'hw2_prob1_partb_scheme_1')
    text(10^-1,10^-4,'p=2, second order accuracy')
    title('h vs. error norms, log scale, HW2 1.b scheme1');
    export_fig nma_HW2_prob1_part_b_scheme_1.png
elseif strcmp(ID,'hw2_prob1_partb_scheme_2')
    text(10^-1,10^-4,'p=2, second order accuracy')
    title('h vs. error norms, log scale, HW2 1.b scheme2');
    export_fig nma_HW2_prob1_part_b_scheme_2.png
elseif strcmp(ID,'hw2_prob1_parta')
    text(10^-1,10^-4,'p=2, second order accuracy')
    title('h vs. error norms, log scale, HW2 1.a');
    export_fig nma_HW2_prob1_part_a.png
elseif strcmp(ID,'hw2_prob2')
    text(10^-1,10^-3,'p=1, first order accuracy')
    title('h vs. error norms, log scale, HW2 prob2');
    export_fig nma_HW2_prob2.png
elseif strcmp(ID,'hw2_prob3_1')
    title('h vs. error norms, log scale, HW2 prob3 (parta)');
    export_fig nma_HW2_prob3_1.png
elseif strcmp(ID,'hw2_prob3_2')
    title('h vs. error norms, log scale, HW2 prob3 (partb)');
    export_fig nma_HW2_prob3_2.png
end

end

%-----------------------------------------------------------------
% Augment the A matrix
%
function [A,f] = augment_system(A,f)
f = f(:);
v = null(A');    % Null vector of adjoint of A
u = null(A);     % Null vector of A

%lambda = dot(v,f)/dot(v,v);  % used during testing to verify it is correct

% Build the A matrix the f matrix to force zero mean for the solution
A = [A    v;
    u'   0];

```

```
f = [f ;
    0];
end

%---------------------------------------------------------
% this function builds A matrix, f vector, solves for U, the
% approximate solution, and determines the error vector e=U-u where
% u is the exact solution at the grid points
%
% For HW2, prob 1 part b, second scheme
%
function [e,U,u,xcoordinates] = ...
    error_vector_HW2_prob1_partb_scheme2(total_number_of_points,h)

    function f = force(x)
        f = 2*(cos(pi*x)).^2;
    end

    function u = exactU(x)
        % NOTE: This solution has a -1/6 as its additive constant.
        u = (x.^2)/2 - cos(2*pi*x)/(4*pi^2) - 1/6;
    end

% Boundary conditions, left and right
alpha = 0;
beta  = 1;
len   = 1;

xcoordinates = 0:h:len;                     %xcoordinates at points

%  build the A matrix
A = nma_FDM_matrix_laplace_1D_Neumann_scheme_2(total_number_of_points);
A = A/h^2;

% build the f vector, make sure to adjust the first and last entries
f      = force(xcoordinates)';
f(1)   = alpha/h;
f(end) = beta/h;

[A,f] = augment_system(A,f);
U = A\f(:);                         % solve for U, the approximate solution
%lambda = U(end);                   % save lambda for reporting
U = U(1:end-1);                     % throw away the last entry, lambda

u = exactU( xcoordinates );         % find exact solution at the grid points

% U found above has a zero mean. This was by construction. Therefore, we
% have to shift it by the current mean of the sampled version of the exact
% solution which also has a zero mean, but only in the limit. Meaning as
% the number of grid points becomes very large.
% Therfore, at each iteration the mean of u will not yet be zero due to
% having small number of samples (grid points). The current mean of u must
% be used to adjust U to make the comparison between U and u applicable.
% When the number of grid points become very large, this adjustment become
% less important, since mean(u) will start to approach zero, and adding
% a zero to U will not affect the result.

U = U + mean(u);

% Now that we have found U, the numerical solution, we find the error
% vector.

u = u(:);  U=U(:);
```

```
e = U - u;
end  % end function(error_vector_HW2_prob1_partb_scheme2)

%---------------------------------------------------------------------
% this function builds A matrix, f vector, solves for U, the approximate
% solution, and determines the error vector e=U-u where u is the exact
% solution at the grid points
%
% For HW2, prob 1 part b, first scheme
function [e,U,u,xcoordinates]=error_vector_prob1_partb_scheme1...
    (total_number_of_points,h)

    function f = force(x)
        f = cos(pi*x).^2;
        f = 2*f;
    end

    function u = exactU(x)
        u = (x.^2)/2 - cos(2*pi*x)/(4*pi^2) - 1/6;
    end

% Boundary conditions, left and right
alpha = 0;
beta  = 1;
len   = 1;

xcoordinates = 0:h:len;    %xcoordinates at points

A = nma_FDM_matrix_laplace_1D_Neumann_scheme_1(total_number_of_points);
A = A/h^2;

% build the f vector, make sure to adjust the first and last entries
f      = force(xcoordinates);
f(1)   = f(1)   + 2*alpha/h;
f(end) = f(end) - 2*beta/h;

USE_PINV = false;   % for testing to verify the augmented method against

if USE_PINV
    U   = pinv(A)*f(:);
    lambda = 0;
else
    [A,f]  = augment_system(A,f);
    U      = A\f(:);           % solve for U, the approximate solution
    lambda = U(end);           % save lambda for reporting
    U      = U(1:end-1);       % Not needed for finding error norm
end

u = exactU( xcoordinates );    % find exact solution at the grid points
U = U + mean(u);               % see note above

% Now that we have found U, the numerical solution, we find the error
% vector.

u = u(:);  U=U(:);
e = U - u;
end % end function(error_vector_prob1_partb_scheme1)

%---------------------------------------------------------------------
% this function builds A matrix, f vector, solves for U, the approximate
% solution, and determines the error vector e=U-u where u is the exact
% solution at the grid points
% HW2, prob 1, part a
```

```
%
function [e,U,u,xcoordinates]=error_vector_HW2_prob1_parta...
    (total_number_of_points,h)

    function f = force(x)
        f = exp(x);
    end

    function u = exactU(x)
        u = -1 + exp(x) + 2.*x - exp(1).*x;
    end

% Boundary conditions, left and right
alpha = 0;
beta  = 1;
len   = 1;

number_internal_points = total_number_of_points-2;
xcoordinates           = h:h:len-h;       %xcoordinates at internal points

A = nma_FDM_matrix_laplace_1D_dirichlet(number_internal_points );
A = A/h^2;

% build the f vector, make sure to adjust the first and last entries
f      = force(xcoordinates)';
f(1)   = f(1)-alpha/h^2;
f(end) = f(end)-beta/h^2;

U = A\f;                          % solve for U, the approximate solution
u = exactU( xcoordinates )';      % find exact solution at the grid points
e = U - u;                        % find the total error vector e

end

%-------------------------------------------------------------
% this function builds A matrix, f vector, solves for U, the approximate
% solution, and determines the error vector e=U-u where u is the exact
% solution at the grid points
% HW2 prob2
%
function [e,U,u,xcoordinates] = error_vector_HW2_prob2...
    (total_number_of_points,h)

    function f = force(x)
        f = cos(x);
    end

    function u = exactU(x,b,g)
        u = (1/2)*(1+b-g-x+b*x+g*x+cos(1)+cos(1)*x-2*cos(x));
    end

% Boundary conditions, left and right
alpha = 1;
b     = 1;
g     = 1;
len   = 1;
xcoordinates = 0:h:len-h;                      %xcoordinates at points

%  build the A matrix
A = nma_FDM_matrix_laplace_1D_robin(total_number_of_points-1,h,alpha);
A = A/h^2;

% build the f vector, make sure to adjust the first and last entries
```

```
f      = force(xcoordinates)';
f(1)   = g/h;
f(end) = f(end) - b/h^2;

U = A\f(:);                       % solve for U, the approximate solution
u = exactU( xcoordinates,b,g );  % find exact solution at the grid points
u = u(:);  U=U(:);
e = U - u;
end

%-----------------------------------------------------------------------
% this function builds A matrix, f vector, solves for U, the approximate
% solution, and determines the error vector e=U-u where u is the exact
% solution at the grid points
% HW2, prob 1, part a
%
function [e,U,u,xcoordinates]=error_vector_HW2_prob3_1...
    (total_number_of_points,h)

    function f = force(x)
        f = exp(x);
    end

    function u = exactU(x)
        u = -1 + exp(x) + 2.*x - exp(1).*x;
    end

% Boundary conditions, left and right
alpha = 0;
beta  = 1;
len   = 1;

number_internal_points = total_number_of_points-2;
xcoordinates           = h:h:len-h;       %xcoordinates at internal points

A = nma_FDM_matrix_laplace_1D_dirichlet(number_internal_points );
A = A/h^2;

% build the f vector, make sure to adjust the first and last entries
f      = force(xcoordinates)';
f(1)   = f(1) - alpha/h^2 +1 ;      % make LTE O(1) instead of O(h^2)
f(end) = f(end)-beta/h^2;

U = A\f;                           % solve for U, the approximate solution
u = exactU( xcoordinates )';       % find exact solution at the grid points
e = U - u;                         % find the total error vector e

end

%-----------------------------------------------------------------------
% this function builds A matrix, f vector, solves for U, the approximate
% solution, and determines the error vector e=U-u where u is the exact
% solution at the grid points
% HW2, prob 1, part a
%
function [e,U,u,xcoordinates]=error_vector_HW2_prob3_2...
    (total_number_of_points,h)

    function f = force(x)
        f = exp(x);
    end

    function u = exactU(x)
```

```
        u = -1 + exp(x) + 2.*x - exp(1).*x;
    end

% Boundary conditions, left and right
alpha = 0;
beta  = 1;
len   = 1;

number_internal_points = total_number_of_points-2;
xcoordinates           = h:h:len-h;       %xcoordinates at internal points

A = nma_FDM_matrix_laplace_1D_dirichlet(number_internal_points );
A = A/h^2;

% build the f vector, make sure to adjust the first and last entries
f                  = force(xcoordinates)';
f(1)               = f(1)-alpha/h^2;
middle_position    = round(length(f)/2);
f(middle_position) = f(middle_position) + h; %Change LTE from O(h^2)to O(h)
f(end)             = f(end)-beta/h^2;

U = A\f;                          % solve for U, the approximate solution
u = exactU( xcoordinates )';      % find exact solution at the grid points
e = U - u;                        % find the total error vector e

end
```

```
function A = nma_FDM_matrix_laplace_1D_dirichlet(N )
%%
% nma_FDM_matrix_laplace_1D_dirichlet(N)
%
% returns the A matrix, which is the system finite difference matrix for
% numerical solution of 1-D laplace equation Uxx = f, with dirichlet
% boundary conditions on both sides of the element. Based on 3 point
% scheme    U'' =  U(j-1)-2*U(j)+U(j+1)
% notice that spacing h between the nodes is not used. The caller must
% divide by h^2 this A matrix abone return.
%
% INPUT:  N, the number of nodes
% OUTPUT: A, the matrix , see below
%
% EXAMPLE:
% A = nma_FDM_matrix_laplace_1D_dirichlet(6)
%
%     -2     1     0     0     0     0
%      1    -2     1     0     0     0
%      0     1    -2     1     0     0
%      0     0     1    -2     1     0
%      0     0     0     1    -2     1
%      0     0     0     0     1    -2
%
% A = A/h^2;  % h is space between points
% U = A\f;  % solve for U, where Uxx=f, need to set f as function before.
%

A = zeros(N);

for i=1:N
    for j=1:N
        if(i==j)
            A(i,j) = -2;
        else
            if( i==j+1 || i==j-1 )
                A(i,j) = 1;
```

```
            end
        end
    end
end

end
```

```
function  A = nma_FDM_matrix_laplace_1D_Neumann_scheme_1(N )
%-----------------------------------------------------------------
% nma_FDM_matrix_laplace_1D_Neumann_scheme_1(N )
%
% returns the A matrix, which is the system finite difference matrix for
% numerical solution of 1-D laplace equation Uxx = f, with nuemman
% boundary conditions on both sides of the element. Based on 3 point
% scheme    U'' =  U(j-1)-2*U(j)+U(j+1)  for middle points and based on
% scheme  ((-2*U(0)+2*U(1))/(h^2))=f(0)+((2*a)/h) for the left most point,
% where a = U'(0), and h is the spacing. and for for the right most point
% ((2*U(N)-2*U(N+1))/(h^2)) = f(N+1)-((2*b)/h) where b=U'(1)
%
% notice that spacing h between the nodes is not used. The caller must
% divide by h^2 this A matrix abone return.
%
% INPUT:  N, the number of nodes
% OUTPUT: A, the matrix , see below
%
% EXAMPLE:
% A = nma_FDM_matrix_laplace_1D_Neumann_scheme_1(5)
%     -2     2     0     0     0
%      1    -2     1     0     0
%      0     1    -2     1     0
%      0     0     1    -2     1
%      0     0     0     2    -2
%
%
% A = A/h^2;  % h is space between points
% U = A\f;  % solve for U, where Uxx=f, need to set f as function before.
%
% copyright: Nasser M. Abbasi
% 10/18/2010

A = zeros(N);
for i=1:N
    for j=1:N
        if(i==j)
            A(i,j) = -2;
        else
            if( i==j+1 || i==j-1 )
                A(i,j) = 1;
            end
        end
    end
end

%fix A above for Von Neumann B.C. only 1st and last rows need fixed
A(1,2)       = 2;
A(end,end-1) = 2;
end
```

```
function A = nma_FDM_matrix_laplace_1D_Neumann_scheme_2(N)
%-----------------------------------------------------------------
% A = nma_FDM_matrix_laplace_1D_Neumann_scheme_2(N)
%
% returns the A matrix, which is the system finite difference matrix for
% numerical solution of 1-D laplace equation Uxx = f, with nuemman
```

```
% boundary conditions on both sides of the element. Based on 3 point
% scheme    U'' =  U(j-1)-2*U(j)+U(j+1)  for middle points and the left
% and right most points, the following scheme is used
%    In this scheme, the Neumann boundary condition is approximated using
%
%       a =(1/h)((3/2)U(0)-2*U(1)+(1/2)U(2))
%       b =(1/h)((3/2)U(N+1)-2*U(N)+(1/2)U(N+1))
%
% where a=U'(0) and b=U'(1)
%
% notice that spacing h between the nodes is not used. The caller must
% divide by h^2 this A matrix abone return.
%
% INPUT:  N, the number of nodes
% OUTPUT: A, the matrix , see below
%
% EXAMPLE:
% A = nma_FDM_matrix_laplace_1D_Neumann_scheme_2(5)
%
%     1.5000   -2.0000    0.5000         0         0
%     1.0000   -2.0000    1.0000         0         0
%          0    1.0000   -2.0000    1.0000         0
%          0         0    1.0000   -2.0000    1.0000
%          0         0    0.5000   -2.0000    1.5000
%
%
% A = A/h^2;  % h is space between points
% U = A\f;  % solve for U, where Uxx=f, need to set f as function before.
%
% copyright: Nasser M. Abbasi
% 10/18/2010

A = zeros(N);
for i=1:N
    for j=1:N
        % as before, part(a), Dirichlet  B.C.
        if(i==j)
            A(i,j) = -2;
        else
            if( i==j+1 || i==j-1 )
                A(i,j) = 1;
            end
        end
    end
end

%fix A above for Von Neumann B.C. only 1st and last rows need fixed
A(1,1) = 3/2;
A(1,2) = -2;
A(1,3) = 1/2;

A(end,end)   = (3/2);
A(end,end-1) =  -2;
A(end,end-2) = (1/2);

end
```

```
function A = nma_FDM_matrix_laplace_1D_robin(N,h,alpha)
%-------------------------------------------------------------------
% A = nma_FDM_matrix_laplace_1D_robin(N,h)
%
% returns the A matrix, which is the system finite difference matrix for
% numerical solution of 1-D laplace equation Uxx = f, with nuemman
% boundary conditions on both sides of the element. Based on 3 point
```

```
%
% INPUT:  N, the number of nodes
%         h, spacing between nodes
%
% OUTPUT: A, the matrix , see below
%
%
% A = A/h^2;  % h is space between points
% U = A\f;  % solve for U, where Uxx=f, need to set f as function before.
%
% copyright: Nasser M. Abbasi
% 10/18/2010

A = zeros(N);
for i=1:N
    for j=1:N
        % as before, part(a), Dirichlet  B.C.
        if(i==j)
            A(i,j) = -2;
        else
            if( i==j+1 || i==j-1 )
                A(i,j) = 1;
            end
        end
    end
end

%fix A above
A(1,1) = -1-alpha*h;

end
```

```
function [hd,bdy] =  nma_format_matrix(headings,data,wid,fms,fid,flag)
%   nma_format_matrix()
%
%   prints matrix of numerical data with headings in formatted way
%
%   INPUT:
%   headings: a cell array of strings for the headings of each column
%   data: a matrix, contains the numerical data
%   wid: how wide a field to use (see below for example)
%   fms: the formating cell string array to use for the numerical data
%        each string corresponds to formatting a field in the numberical
%        data
%   fid: the file id to print to. Use 1 for stdout
%   flag: if 0, then will not print anything, just format and return
%         result
%
%   Examples calling
%
%   titles={'number of projects','sales','profit'};
%   data=[2 rand(1) rand(1);
%         3 rand(1) rand(1)];
%
%   Suppose we want to format each numerical number above as %16.5E, then
%   in this case 10 will be the field width, and '.5E' is what to use for
%   the fms argument
%
%   wid    = 16;
%   fms    = {'d','.4E','.5E'};
%   fileID = 1;
%   nma_format_matrix(titles,data,wid,fms,fileID,false);
%
%   version 1.0  10/15/2010
```

```
%   All blame to Nasser M. Abbasi
%   Thanks for help from matlab newsgroup: Ross W, Bruno Luong
%


%
% do some basic checking on input
%
if nargin ~= 6
    error 'number of arguments must be 6'
end

if ~iscell(headings)
    error 'headings must be cell array'
end

if ~iscell(fms)
    error 'fms must be cell array'
end

if isempty(headings)
    error 'headings can not be empty';
end

if isempty(data)
    error 'data can not be empty';
end

if isempty(fms)
    error 'fms can not be empty';
end

[nRowH,nColH] = size(headings);
if(nRowH>1)
    error 'headings can not have more than one row';
end

[nRowFms,nColFms] = size(fms);
if(nRowFms>1)
    error 'fms can not have more than one row';
end

if(nColH ~= nColFms)
    error 'headings must have same number of columns as fms';
end

[~,nCol] = size(data);
if(nColH ~= nCol)
    error 'data must have same number of columns as headings';
end
%
% end basic checking on input, now do the formating and printing
%

fmt = arrayfun(@(x) ['%',num2str(wid),'s'],1:nCol,'UniformOutput',false);
if flag
   fprintf(fid,[fmt{:} '\n'],headings{:});
end

hd=sprintf([fmt{:} '\n'],headings{:});

fmt = arrayfun(@(x) ['%',num2str(wid),fms{x}],1:nCol,'UniformOutput',false);
if flag
   fprintf(fid,[fmt{:} '\n'],data');
```

```
96 end
97 bdy=sprintf([fmt{:} '\n'],data');
98
99 end
```

2.4 HW 3

2.4.1 residual error animation

Animation of the residual error R as it changes during iterative solution. Tolerance used for these is h^2, stopping criteria used is relative error < tolerance Only SOR was done.

2.4.1.1 SOR with $h = 2^{-7}$

These animations are large in size. (Click on any to see in actual size, will open in new window)

2.4.1.2 SOR with $h = 2^{-5}$

These animations are smaller in size. (Click on any to see in actual size, will open in new window)

2.4.2 Animation for density plot animations of solvers for problem 1

Animation plots below show solver as it updates each grid point by point in its main loop

The above shows each of the solvers (Jacobi, Gauss-Seidel, SOR) in the process of updating the grid during one iteration of the main loop.

Notice the how GS and SOR solvers update the solution immediately (left to right, down to top numbering is used), while Jacobi solver updates the solution only at the end each iterative step.

This one below was done for $h = 2^{-3}$. Clicking on it shows animation.

2.4.3 Animations of iterative solution

Animations of iterative solution (Click on image to see animation), Stopping criteria used is relative residual method, tolerance is h^2

2.4.4 Problem 1

Problem statement

1. Use Jacobi, Gauss-Seidel, and SOR (with optimal ω) to solve

$$\Delta u = -\exp\left(-(x-0.25)^2 - (y-0.6)^2\right)$$

on the unit square $(0,1) \times (0,1)$ with homogeneous Dirichlet boundary conditions. Find the solution for mesh spacings of $h = 2^{-5}$, 2^{-6}, and 2^{-7}. What tolerance did you use? What stopping criteria did you use? What value of ω did you use? Report the number of iterations it took to reach convergence for each method for each mesh.

Figure 2.16: Problem 1

Answer

Using the method of splitting, the iteration matrix T is found, for each method, for solving $Au = f$.

Let $A = M - N$, then $Au = f$ becomes

$$\begin{aligned} (M-N)u &= f \\ Mu &= Nu + f \\ u^{[k+1]} &= M^{-1}Nu^{[k]} + M^{-1}f \end{aligned} \tag{1}$$

2.4.4.1 Finding iterative matrix for the different solvers

The Jacobi method

For the Jacobi method $M = D$ and $N = L + U$, where D is the diagonal of A, L is the negative of the strictly lower triangle matrix of A and U is negative of the strictly upper triangle matrix of A. Hence(1) becomes

$$u^{[k+1]} = D^{-1}(L + U)u^{[k]} + D^{-1}f$$
$$u^{[k+1]} = Tu^{[k]} + C$$

Where T is called the Jacobi iteration matrix. Since $A = D - L - U$, hence $L + U = D - A$, therefore the iteration matrix T can be written as

$$T = D^{-1}(D - A)$$
$$= I - D^{-1}A$$

or

$$T = (I - D^{-1}A)$$
$$C = D^{-1}f$$

The Gauss-Seidel method

For Gauss-Seidel, $M = D - L$ and $N = U$, hence (1) becomes

$$u^{[k+1]} = (D - L)^{-1} U\, u^{[k]} + (D - L)^{-1} f$$

or

$$u^{[k+1]} = Tu^{[k]} + C$$

where

$$T = (D - L)^{-1} U$$
$$C = (D - L)^{-1} f$$

The SOR method

For SOR, $M = \frac{1}{\omega}(D - \omega L)$ and $N = \frac{1}{\omega}((1 - \omega) D + \omega U)$. Hence (1) becomes

$$\begin{aligned} u^{[k+1]} &= \left[\frac{1}{\omega}(D - \omega L)\right]^{-1} \left[\frac{1}{\omega}((1 - \omega) D + \omega U)\right] u^{[k]} + \left[\frac{1}{\omega}(D - \omega L)\right]^{-1} f \\ &= (D - \omega L)^{-1} ((1 - \omega) D + \omega U)\, u^{[k]} + \omega (D - \omega L)^{-1} f \\ &= Tu^{[k]} + C \end{aligned}$$

where

$$T = (D - \omega L)^{-1} ((1 - \omega) D + \omega U)$$
$$C = \omega (D - \omega L)^{-1}$$

Summary of iterative matrices used

This table summarizes the expression for the iterative matrix T and for the matrix C in the equation $u^{[k+1]} = Tu^{[k]} + C$ for the different methods used.

method	T	C
Jacobi	$(I - D^{-1}A)$	$D^{-1}f$
GS	$(D - L)^{-1} U$	$(D - L)^{-1} f$
SOR	$(D - \omega L)^{-1} ((1 - \omega) D + \omega U)$	$\omega (D - \omega L)^{-1}$

The discretized algebraic equation resulting from approximating $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f\left(x, y\right)$ with Dirichlet boundary conditions is based on the use of the standard 5 point Laplacian

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{1}{h^2}\left(U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{i,j}\right)$$

which has local truncation error $O\left(h^2\right)$.

The notation used above is based on the following grid formating

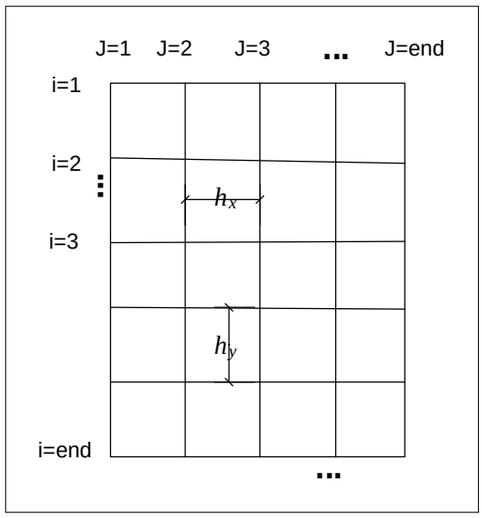


Figure 2.17: Grid notation

The derivation of the above formula is as follows: Consider a grid where the mesh spacing in the x-direction is h_x and in the y-direction is h_y. Then $\frac{\partial^2 u}{\partial x^2}$ is approximated, at position (i,j) by $\frac{U_{i-1,j}-2U_{i,j}+U_{i+1,j}}{h_x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ is approximated, at position (i,j), by $\frac{U_{i,j-1}-2U_{i,j}+U_{i,j+1}}{h_y^2}$ therefore $\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2} \approx \frac{U_{i-1,j}-2U_{i,j}+U_{i+1,j}}{h_x^2}+\frac{U_{i,j-1}-2U_{i,j}+U_{i,j+1}}{h_y^2}$, and since $\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=f_{i,j}$ at that position, this results in

$$\frac{U_{i-1,j}-2U_{i,j}+U_{i+1,j}}{h_x^2}+\frac{U_{i,j-1}-2U_{i,j}+U_{i,j+1}}{h_y^2}=f_{i,j}$$

Solving for $U_{i,j}$ from the above gives

$$\begin{aligned}U_{i,j} &= \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{i-1,j}h_y^2+U_{i+1,j}h_y^2+U_{i,j-1}h_x^2+U_{i,j+1}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j}\\ &= \frac{1}{2\left(h_x^2+h_y^2\right)}\left(\left(U_{i-1,j}+U_{i+1,j}\right)h_y^2+\left(U_{i,j-1}+U_{i,j+1}\right)h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j}\end{aligned}$$

The following table shows the formula for updating $U_{i,j}$ using each of the 3 methods for the 2D case, under the assumption of uniform mesh spacing, i.e. when $h_x=h_y$ which simplifies the above formula as given below

method	Formula for updating $U_{i,j}$, uniform mesh
Jacobi	$U_{i,j}^{[k+1]}=\frac{1}{4}\left(U_{i-1,j}^{[k]}+U_{i+1,j}^{[k]}+U_{i,j-1}^{[k]}+U_{i,j+1}^{[k]}-h^2f_{i,j}\right)$
GS	$U_{i,j}^{[k+1]}=\frac{1}{4}\left(U_{i-1,j}^{[k+1]}+U_{i+1,j}^{[k]}+U_{i,j-1}^{[k+1]}+U_{i,j+1}^{[k]}-h^2f_{i,j}\right)$
SOR	$U_{i,j}^{[k+1]}=\frac{\omega}{4}\left(U_{i-1,j}^{[k+1]}+U_{i+1,j}^{[k]}+U_{i,j-1}^{[k+1]}+U_{i,j+1}^{[k]}-h^2f_{i,j}\right)+(1-\omega)U_{i,j}^{[k]}$

note that for SOR, the general formula, using nonuniform mesh can be derived as follows

$$\begin{aligned}U_{(gs)i,j}^{[k+1]} &= \frac{1}{4}\left(U_{i-1,j}^{[k+1]}+U_{i+1,j}^{[k]}+U_{i,j-1}^{[k+1]}+U_{i,j+1}^{[k]}-h^2f_{i,j}\right)\\ U_{(sor)i,j}^{[k+1]} &= U_{i,j}^{[k]}+\omega\left(U_{(gs)i,j}^{[k+1]}-U_{i,j}^{[k]}\right)\end{aligned}$$

The second formula above can be simplified by substituing the first equation into it to yield

$$\begin{aligned}U_{(sor)i,j}^{[k+1]} &= U_{i,j}^{[k]}+\omega\left(\frac{1}{4}\left(U_{i-1,j}^{[k+1]}+U_{i+1,j}^{[k]}+U_{i,j-1}^{[k+1]}+U_{i,j+1}^{[k]}-h^2f_{i,j}\right)-U_{i,j}^{[k]}\right)\\ &= \frac{\omega}{4}\left(U_{i-1,j}^{[k+1]}+U_{i+1,j}^{[k]}+U_{i,j-1}^{[k+1]}+U_{i,j+1}^{[k]}-h^2f_{i,j}\right)+(1-\omega)U_{i,j}^{[k]}\end{aligned}$$

Which is what shown in the table above. Using the same procedure, but for the general

case, results in

$$U^{[k+1]}_{(gs)i,j} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U^{[k+1]}_{i-1,j}h_y^2 + U^{[k]}_{i+1,j}h_y^2 + U^{[k+1]}_{i,j-1}h_x^2 + U^{[k]}_{i,j+1}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j}$$
$$U^{[k+1]}_{(sor)i,j} = U^{[k]}_{i,j} + \omega\left(U^{[k+1]}_{(gs)i,j} - U^{[k]}_{i,j}\right)$$

Hence, again, the second equation above becomes

$$\begin{aligned} U^{[k+1]}_{(sor)i,j} &= U^{[k]}_{i,j} + \omega\left(\frac{1}{2\left(h_x^2+h_y^2\right)}\left(U^{[k+1]}_{i-1,j}h_y^2 + U^{[k]}_{i+1,j}h_y^2 + U^{[k+1]}_{i,j-1}h_x^2 + U^{[k]}_{i,j+1}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j} - U^{[k]}_{i,j}\right) \\ &= \omega\left[\frac{1}{2\left(h_x^2+h_y^2\right)}\left(U^{[k+1]}_{i-1,j}h_y^2 + U^{[k]}_{i+1,j}h_y^2 + U^{[k+1]}_{i,j-1}h_x^2 + U^{[k]}_{i,j+1}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j}\right] + (1-\omega)\,U^{[k]}_{i,j} \end{aligned}$$

Hence, for non-uniforma mesh the above update table becomes

method	Formula for updating $U_{i,j}$, non-uniform mesh
Jacobi	$U^{[k+1]}_{i,j} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(\left(U_{i-1,j}+U_{i+1,j}\right)h_y^2 + \left(U_{i,j-1}+U_{i,j+1}\right)h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j}$
GS	$U^{[k+1]}_{i,j} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U^{[k+1]}_{i-1,j}h_y^2 + U^{[k]}_{i+1,j}h_y^2 + U^{[k+1]}_{i,j-1}h_x^2 + U^{[k]}_{i,j+1}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j}$
SOR	$U^{[k+1]}_{i,j} = \omega\left[\frac{1}{2\left(h_x^2+h_y^2\right)}\left(U^{[k+1]}_{i-1,j}h_y^2 + U^{[k]}_{i+1,j}h_y^2 + U^{[k+1]}_{i,j-1}h_x^2 + U^{[k]}_{i,j+1}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j}\right] + (1-\omega)\,U^{[k]}_{i,j}$

The residual formula is $R^{[k]} = f - Au^{[k]}$, which can be written using the above notations as

$$R_{i,j} = f_{i,j} - \left(\frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{h_x^2} + \frac{U_{i,j-1} - 2U_{i,j} + U_{i,j+1}}{h_y^2}\right)$$

and for a uniform mesh the above becomes

$$R_{i,j} = f_{i,j} - \frac{1}{h^2}\left(U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{i,j}\right)$$

2.4.4.2 Tolerance used

The tolerance ϵ used is based on the global error in the numerical solution. For this problem $\|e\| = Ch^2$ where C is a constant C. Two different values for the constant were tried in the implementation: 0.1 and 1.

2.4.4.3 stopping criteria

The stopping criteria used was based on the use of the relative residual. Given

$$R^{[k]} = f - Au^{[k]}$$

The iterative process was stopped when the following condition was satisfied

$$\frac{\left\|R\left[^{k}\right]\right\|}{\|f\|} < \epsilon$$

Other possible stopping criterion are (but were not used) are

1. Absolute error: convergence achieved when $\left\|u^{[k+1]} - u^{[k]}\right\| < \epsilon$
2. Relative error: convergence achieved when $\frac{\left\|u^{[k+1]} - u^{[k]}\right\|}{\left\|u^{[k]}\right\|} < \epsilon$

The above two criterion are not used as they do not perform as well for Jacobi and Gauss-Seidel.

2.4.4.4 ω used for SOR

The ω_{opt} used is based on the relation to the spectral radius of the *Jacobi* iteration matrix. This relation was derived in class

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho^2_{Jacobian}}}$$

where for the $2D$ Poisson problem $\rho_{Jacobian}$ is given as $\cos(\pi h)$ where h is the grid spacing. Using this in the above and approximating for small h results in

$$\omega_{opt} \approx 2(1 - \pi h)$$

For the given h values in the problem, the following values were used for ω_{opt}

$h_x = h_y = h$	ω_{opt}
2^{-3}	1.214 6
2^{-4}	1.607 3
2^{-5}	1.803 7
2^{-6}	1.901 8
2^{-7}	1.950 9

Notice that the above approximation formula is valid for small h and will not result in good convergence for SOR if used for large h.

Update 12/29/2010: After more analysis, the following formula is found to produce better results

$$t = cos(\pi\ h_x) + cos(\pi h_y)$$
$$poly(x) = x^2 t^2 - 16x + 16$$

Solve the above polynomial for x and then take the smaller root. This will be ω_{opt}, Using this results in

$h_x = h_y = h$	ω_{opt}
2^{-3}	1.4464
2^{-4}	1.6763
2^{-5}	1.821
2^{-6}	1.9064
2^{-7}	1.9509

2.4.4.5 Algorithm details

In this section, some of the details of the implementation are described for reference. The Matrix A is not used directly in the iterative method, but its structure is briefly described.

In the discussion below, updating the grid is described for the Jacobi method, showing how the new values of the dependent variable u are calculated.

Numbering system and grid updating

The numbering system used for the grid is the one described in class. The indices for the unknown $u_{i,j}$ are numbered row wise, left to right, bottom to top. This follows the standard Cartesian coordinates system. The reason for this, is to allow the use of the standard formula for the 5 point Laplacian. The following diagram illustrate the 5 point Laplacian borrowed from the text book, figure 5, page 61 "*Finite Difference Methods for Ordinary and Partial Differential Equations*" by Randall J. LeVeque

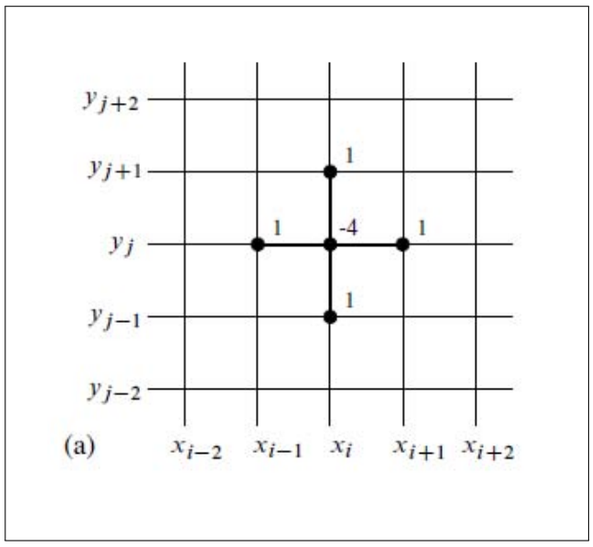


Figure 2.18: 5-point stencile for Laplacian at x(i,j)

Lower case n is used to indicate the number of unknowns along one dimension, and upper case N is used to indicate the total number of unknowns.

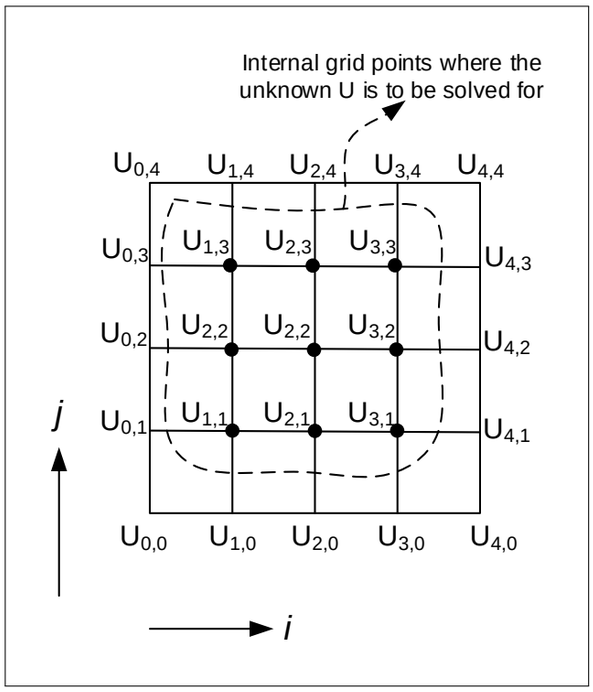


Figure 2.19: stencil move

In the diagram above, $n = 3$ is the number of unknowns on each one row or one column, and since there are 3 internal rows, there will be 9 unknowns, all are located on internal grid points. There are a total of 25 grid points, 16 of which are on the boundaries and 9 internal.

To update the grid, the 5 point Laplacian is centered at each internal grid point and then moved left to right, bottom to top, each time an updated value of the unknown is generated and stored in an auxiliary grid.

In the Jacobian method, the new value at the location $x_{i,j}$ is not used in the calculation to update its neighbors when the stencil is moved, but

Only when the whole grid has been swept by the Laplacian will the grid be updated by copying the content of the auxiliary grid into it.

In the Gauss-Seidel and SOR, this not the case. Once a grid point have been updated, its new value is used immediately in the update of its neighbor as the Laplacian sweeps the grid. No auxiliary grid is required. In other words, the updates happens 'in place'.

Continuing this example, the following diagram shows how each grid point is updated (In a parallel computation, these operations can all be done at once for the Jacobi solver, but not for the Gauss-Seidel or the SOR solver, unless different numbering scheme is used such as black-red numbering).

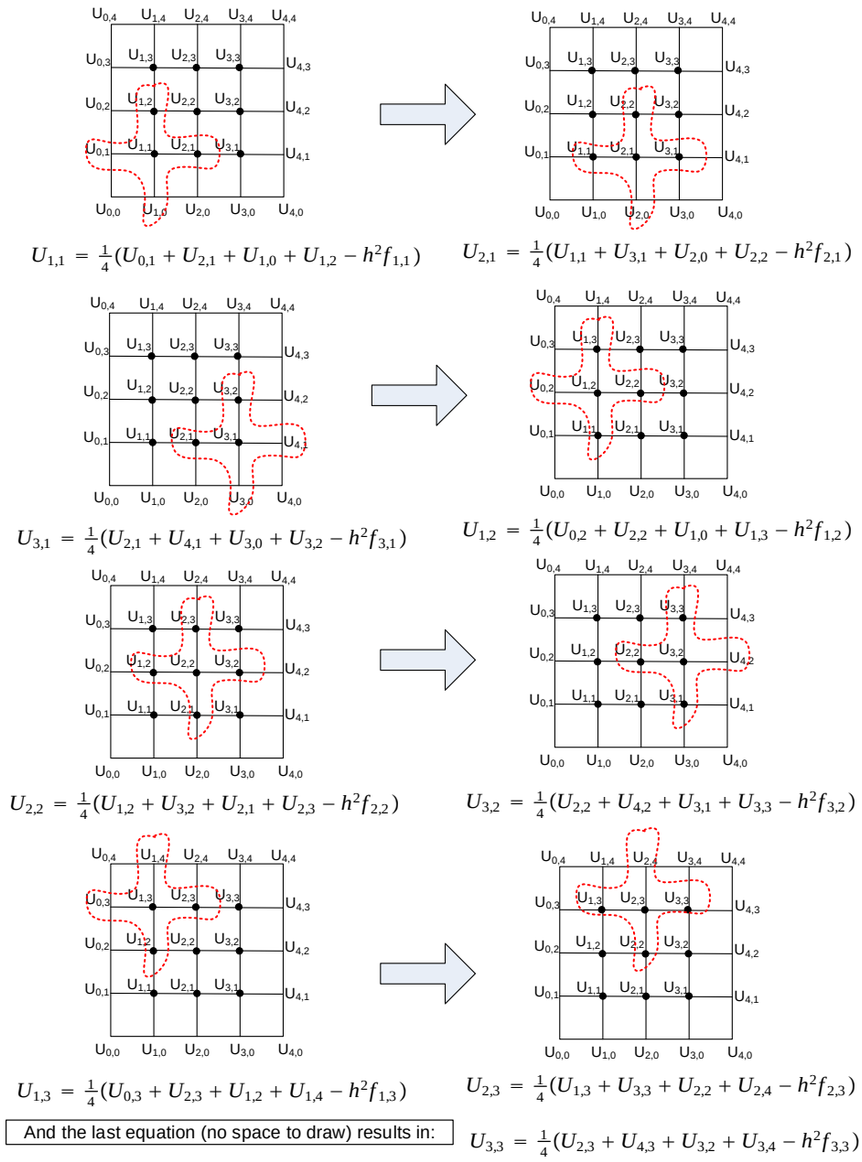


Figure 2.20: stencil move 2

Now that the grid has been updated once, the process is repeated again. This process is continued until convergence is achieved.

2.4.4.6 Structure of $Au = f$

The structure of $Au = f$ system matrix is now described. As an example, for number of unknowns = 9 the following characteristics of the matrix A can be seen

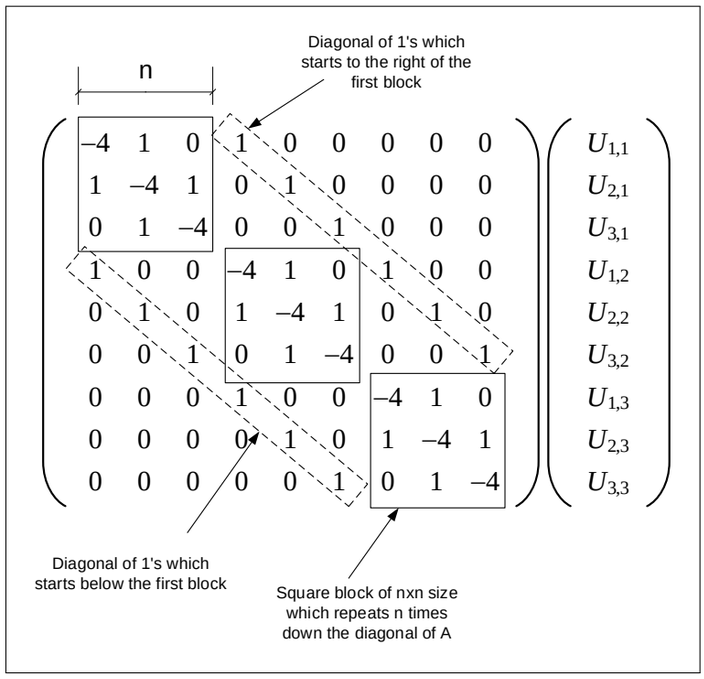


Figure 2.21: A structure

Structure of the A matrix for elliptic 2D PDE with Dirchillet boundary conditions for non-uniform mesh

This section was added at a later time here for completion, and is not required for the HW. Below the A matrix form is derived for the case for non-uniform mesh (this means h_x is not neccessarily the same as h_y) and also, the number of grid points in the x-direction is not the same as the number of grid points in the y-direction.

To ease the derivation, we will use a 5×5 grid as an example, hence this will generate 9 equations as there are 9 unknowns. From this derivation we will be able to see the form of the A matrix.

In addition, in this derivation, we will use i to represent row number, and j to represent column number, as this more closely matches the convention.

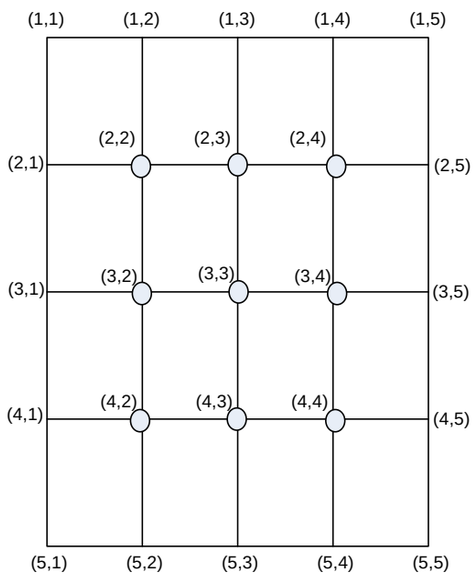


Figure 2.22: new grid

The unknowns are shown above in the circles. From earlier, we found that the discretization

for the elliptic PDE at a node is

$$U_{i,j} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{i,j-1}h_y^2 + U_{i,j+1}h_y^2 + U_{i-1,j}h_x^2 + U_{i+1,j}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,j}$$

We will now write the 9 equations for each node, starting with $(2,2)$ to node $(4,4)$

$$U_{2,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,1}h_y^2 + U_{2,3}h_y^2 + U_{1,2}h_x^2 + U_{3,2}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,2}$$

$$U_{2,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,2}h_y^2 + U_{2,4}h_y^2 + U_{1,3}h_x^2 + U_{3,3}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,3}$$

$$U_{2,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,3}h_y^2 + U_{2,5}h_y^2 + U_{1,4}h_x^2 + U_{3,4}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,4}$$

$$U_{3,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,1}h_y^2 + U_{3,3}h_y^2 + U_{2,2}h_x^2 + U_{4,2}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,2}$$

$$U_{3,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,2}h_y^2 + U_{3,4}h_y^2 + U_{2,3}h_x^2 + U_{4,3}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,3}$$

$$U_{3,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,3}h_y^2 + U_{3,5}h_y^2 + U_{2,4}h_x^2 + U_{4,4}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,4}$$

$$U_{4,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,1}h_y^2 + U_{4,3}h_y^2 + U_{3,2}h_x^2 + U_{5,2}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,2}$$

$$U_{4,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,2}h_y^2 + U_{4,4}h_y^2 + U_{3,3}h_x^2 + U_{5,3}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,3}$$

$$U_{4,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,3}h_y^2 + U_{4,5}h_y^2 + U_{3,4}h_x^2 + U_{5,4}h_x^2\right) - \frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,4}$$

Now, moving the knowns to the right, results in

$$\begin{aligned}
2\left(h_x^2+h_y^2\right)U_{2,2} - U_{2,3}h_y^2 - U_{3,2}h_x^2 &= -h_x^2h_y^2f_{2,2} + U_{2,1}h_y^2 + U_{1,2}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,3} - U_{2,2}h_y^2 - U_{2,4}h_y^2 - U_{3,3}h_x^2 &= -h_x^2h_y^2f_{2,3} + U_{1,3}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,4} - U_{2,3}h_y^2 - U_{3,4}h_x^2 &= -h_x^2h_y^2f_{2,4} + U_{2,5}h_y^2 + U_{1,4}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{3,2} - U_{3,3}h_y^2 - U_{2,2}h_x^2 - U_{4,2}h_x^2 &= -h_x^2h_y^2f_{3,2} + U_{3,1}h_y^2\\
2\left(h_x^2+h_y^2\right)U_{3,3} - U_{3,2}h_y^2 - U_{3,4}h_y^2 - U_{2,3}h_x^2 - U_{4,3}h_x^2 &= -h_x^2h_y^2f_{3,3}\\
2\left(h_x^2+h_y^2\right)U_{3,4} - U_{3,3}h_y^2 - U_{2,4}h_x^2 - U_{4,4}h_x^2 &= -h_x^2h_y^2f_{3,4} + U_{3,5}h_y^2\\
2\left(h_x^2+h_y^2\right)U_{4,2} - U_{4,3}h_y^2 - U_{3,2}h_x^2 &= -h_x^2h_y^2f_{4,2} + U_{4,1}h_y^2 + U_{5,2}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,3} - U_{4,2}h_y^2 - U_{4,4}h_y^2 - U_{3,3}h_x^2 &= -h_x^2h_y^2f_{4,3} + U_{5,3}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,4} - U_{4,3}h_y^2 - U_{3,4}h_x^2 &= -h_x^2h_y^2f_{4,4} + U_{4,5}h_y^2 + U_{5,4}h_x^2
\end{aligned}$$

Now, we can write $Ax = b$ as, letting $\beta = 2\left(h_x^2+h_y^2\right)$

$$\begin{pmatrix}
\beta & -h_y^2 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0\\
-h_y^2 & \beta & -h_y^2 & 0 & -h_x^2 & 0 & 0 & 0 & 0\\
0 & -h_y^2 & \beta & 0 & 0 & -h_x^2 & 0 & 0 & 0\\
-h_x^2 & 0 & 0 & \beta & -h_y^2 & 0 & -h_x^2 & 0 & 0\\
0 & -h_x^2 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & -h_x^2 & 0\\
0 & 0 & -h_x^2 & 0 & -h_y^2 & \beta & 0 & 0 & -h_x^2\\
0 & 0 & 0 & -h_x^2 & 0 & 0 & \beta & -h_y^2 & 0\\
0 & 0 & 0 & 0 & -h_y^2 & 0 & -h_y^2 & \beta & -h_y^2\\
0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & -h_y^2 & \beta
\end{pmatrix}
\begin{pmatrix}
U_{22}\\U_{23}\\U_{24}\\U_{32}\\U_{33}\\U_{34}\\U_{42}\\U_{43}\\U_{44}
\end{pmatrix} =
\begin{pmatrix}
-h_x^2h_y^2f_{2,2} + U_{2,1}h_y^2 + U_{1,2}h_x^2\\
-h_x^2h_y^2f_{2,3} + U_{1,3}h_x^2\\
-h_x^2h_y^2f_{2,4} + U_{2,5}h_y^2 + U_{1,4}h_x^2\\
-h_x^2h_y^2f_{3,2} + U_{3,1}h_y^2\\
-h_x^2h_y^2f_{3,3}\\
-h_x^2h_y^2f_{3,4} + U_{3,5}h_y^2\\
-h_x^2h_y^2f_{4,2} + U_{4,1}h_y^2 + U_{5,2}h_x^2\\
-h_x^2h_y^2f_{4,3} + U_{5,3}h_x^2\\
-h_x^2h_y^2f_{4,4} + U_{4,5}h_y^2 + U_{5,4}h_x^2
\end{pmatrix}$$

Now we will do another case $n_x \neq n_y$

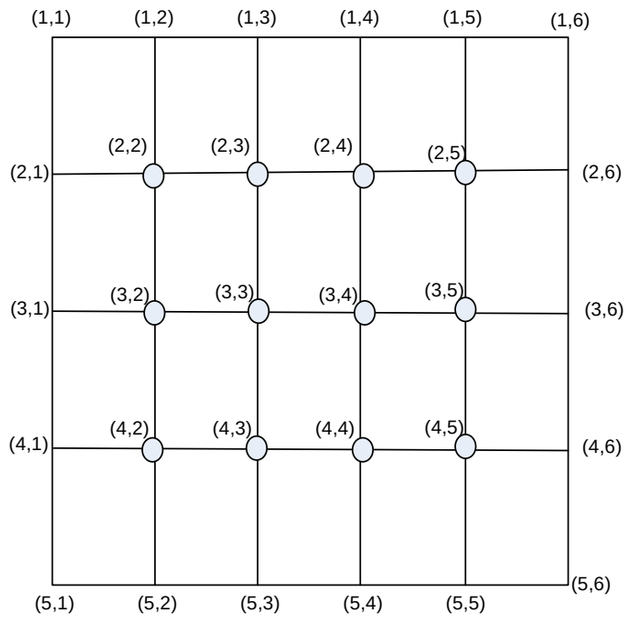


Figure 2.23: new grid 2

The unknowns are shown above in the circles. From earlier, we found that the discretization for the elliptic PDE at a node is

$$U_{i,j} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{i,j-1}h_y^2 + U_{i,j+1}h_y^2 + U_{i-1,j}h_x^2 + U_{i+1,j}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{i,j}$$

We will now write the 12 equations for each node, starting with $(2,2)$ to node $(4,5)$

$$U_{2,2} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{2,1}h_y^2 + U_{2,3}h_y^2 + U_{1,2}h_x^2 + U_{3,2}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{2,2}$$

$$U_{2,3} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{2,2}h_y^2 + U_{2,4}h_y^2 + U_{1,3}h_x^2 + U_{3,3}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{2,3}$$

$$U_{2,4} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{2,3}h_y^2 + U_{2,5}h_y^2 + U_{1,4}h_x^2 + U_{3,4}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{2,4}$$

$$U_{2,5} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{2,4}h_y^2 + U_{2,6}h_y^2 + U_{1,5}h_x^2 + U_{3,5}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{2,5}$$

$$U_{3,2} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{3,1}h_y^2 + U_{3,3}h_y^2 + U_{2,2}h_x^2 + U_{4,2}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{3,2}$$

$$U_{3,3} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{3,2}h_y^2 + U_{3,4}h_y^2 + U_{2,3}h_x^2 + U_{4,3}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{3,3}$$

$$U_{3,4} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{3,3}h_y^2 + U_{3,5}h_y^2 + U_{2,4}h_x^2 + U_{4,4}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{3,4}$$

$$U_{3,5} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{3,4}h_y^2 + U_{3,6}h_y^2 + U_{2,5}h_x^2 + U_{4,5}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{3,4}$$

$$U_{4,2} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{4,1}h_y^2 + U_{4,3}h_y^2 + U_{3,2}h_x^2 + U_{5,2}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{4,2}$$

$$U_{4,3} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{4,2}h_y^2 + U_{4,4}h_y^2 + U_{3,3}h_x^2 + U_{5,3}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{4,3}$$

$$U_{4,4} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{4,3}h_y^2 + U_{4,5}h_y^2 + U_{3,4}h_x^2 + U_{5,4}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{4,4}$$

$$U_{4,5} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{4,4}h_y^2 + U_{4,6}h_y^2 + U_{3,5}h_x^2 + U_{5,5}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{4,5}$$

Now, moving the knowns to the right, results in

$$\begin{aligned}
2\left(h_x^2+h_y^2\right)U_{2,2}-U_{2,3}h_y^2-U_{3,2}h_x^2 &= -h_x^2h_y^2f_{2,2}+U_{2,1}h_y^2+U_{1,2}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,3}-U_{2,2}h_y^2-U_{2,4}h_y^2-U_{3,3}h_x^2 &= -h_x^2h_y^2f_{2,3}+U_{1,3}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,4}-U_{2,3}h_y^2-U_{2,5}h_y^2-U_{3,4}h_x^2 &= -h_x^2h_y^2f_{2,4}+U_{1,4}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,5}-U_{2,4}h_y^2+U_{3,5}h_x^2 &= -h_x^2h_y^2f_{2,5}+U_{2,6}h_y^2+U_{1,5}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{3,2}-U_{3,3}h_y^2-U_{2,2}h_x^2-U_{4,2}h_x^2 &= -h_x^2h_y^2f_{3,2}+U_{3,1}h_y^2\\
2\left(h_x^2+h_y^2\right)U_{3,3}-U_{3,2}h_y^2-U_{3,4}h_y^2-U_{2,3}h_x^2-U_{4,3}h_x^2 &= -h_x^2h_y^2f_{3,3}\\
2\left(h_x^2+h_y^2\right)U_{3,4}-U_{3,3}h_y^2-U_{3,5}h_y^2-U_{2,4}h_x^2-U_{4,4}h_x^2 &= -h_x^2h_y^2f_{3,4}\\
2\left(h_x^2+h_y^2\right)U_{3,5}-U_{3,4}h_y^2+U_{2,5}h_x^2+U_{4,5}h_x^2 &= -h_x^2h_y^2f_{3,4}+U_{3,6}h_y^2\\
2\left(h_x^2+h_y^2\right)U_{4,2}-U_{4,3}h_y^2-U_{3,2}h_x^2 &= -h_x^2h_y^2f_{4,2}+U_{4,1}h_y^2+U_{5,2}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,3}-U_{4,2}h_y^2-U_{4,4}h_y^2-U_{3,3}h_x^2 &= -h_x^2h_y^2f_{4,3}+U_{5,3}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,4}-U_{4,5}h_y^2-U_{4,3}h_y^2-U_{3,4}h_x^2 &= -h_x^2h_y^2f_{4,4}+U_{5,4}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,5}-U_{4,4}h_y^2-U_{3,5}h_x^2 &= -h_x^2h_y^2f_{4,5}+U_{4,6}h_y^2+U_{5,5}h_x^2
\end{aligned}$$

Now, we can write $Ax=b$ as, letting $\beta=2\left(h_x^2+h_y^2\right)$

$$\begin{pmatrix}
\beta & -h_y^2 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
-h_y^2 & \beta & -h_y^2 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & -h_y^2 & \beta & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0\\
-h_x^2 & 0 & 0 & 0 & \beta & -h_y^2 & 0 & 0 & -h_x^2 & 0 & 0 & 0\\
0 & -h_x^2 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & -h_x^2 & 0 & 0\\
0 & 0 & -h_x^2 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & -h_x^2 & 0\\
0 & 0 & 0 & -h_x^2 & 0 & 0 & -h_y^2 & \beta & 0 & 0 & 0 & -h_x^2\\
0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & \beta & -h_y^2 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & -h_y^2 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & -h_y^2 & \beta & -h_y^2\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & -h_y^2 & \beta
\end{pmatrix}
\begin{pmatrix} U_{22}\\ U_{23}\\ U_{24}\\ U_{25}\\ U_{32}\\ U_{33}\\ U_{34}\\ U_{35}\\ U_{42}\\ U_{43}\\ U_{44}\\ U_{45}\end{pmatrix}
=
\begin{pmatrix}
-h_x^2h_y^2f_{2,2}+U_{2,1}h_y^2+U_{1,2}h_x^2\\
-h_x^2h_y^2f_{2,3}+U_{1,3}h_x^2\\
-h_x^2h_y^2f_{2,4}+U_{2,5}h_y^2+U_{1,4}h_x^2\\
-h_x^2h_y^2f_{2,5}+U_{2,6}h_y^2+U_{1,5}h_x^2\\
-h_x^2h_y^2f_{3,2}+U_{3,1}h_y^2\\
-h_x^2h_y^2f_{3,3}\\
-h_x^2h_y^2f_{3,4}+U_{3,5}h_y^2\\
-h_x^2h_y^2f_{3,4}+U_{3,6}h_y^2\\
-h_x^2h_y^2f_{4,2}+U_{4,1}h_y^2+U_{5,2}h_x^2\\
-h_x^2h_y^2f_{4,3}+U_{5,3}h_x^2\\
-h_x^2h_y^2f_{4,4}+U_{4,5}h_y^2+U_{5,4}h_x^2\\
-h_x^2h_y^2f_{4,5}+U_{4,6}h_y^2+U_{5,5}h_x^2
\end{pmatrix}$$

To create the above matrix, as sparse matrix, call the following Matlab code

```
A=lap2d(4,3,hx,hy);
%Where in the above, 4 is nx, which is the number of nodes in the x-
    direction.
%These are internal nodes. ny is number of nodes in the y-direction.
%hx is the spacing between nodes in the x-direction.
%hy is spacing between nodes in the y-direction

function L2 = lap2d(nx,ny,hx,hy)
Lx=lap1dy(nx,hx,hy);  %makes the MAIN block, only one block
Ly=lap1dx(ny,hx,hy);  %makes the off block, only one block

Ix=speye(nx);
Iy=speye(ny);

Lm=kron(Iy,Lx); %does the central diagonal
Lo=kron(Ly,Ix); %does the off diagonal
L2=Lm+Lo;

%------------------------
function L=lap1dy(n,hx,hy)
e=ones(n,1);
T1=2*(hx^2+hy^2);
B=[-e*hy^2 (1/2)*T1*e -e*hy^2];
L=spdiags(B,[-1 0 1],n,n);

%-------------------------------------
%
```

```
%
function L=lap1dx(n,hx,hy)
e=ones(n,1);
T1=2*(hx^2+hy^2);
B=[-e*hx^2 (1/2)*T1*e -e*hx^2];
L=spdiags(B,[-1 0 1],n,n);
```

.

A matrix format for sparse storage for elliptic 2D with non-homegenous Nuemman boundary conditions

This section was added at later time, and not required for this HW.

To be able to forumlate the A matrix as sparse matrix, we need to find what the form will be in the case when one or more of the boundary conditions has Nuemman coditions on it. We will start as above, and start with assuming Nuemman conditions now exist on the left edge and find out what the form of the A matrix will turn out to be.

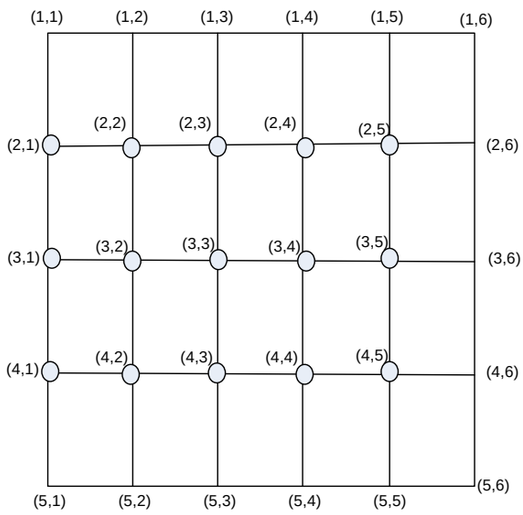


Figure 2.24: new grid 2 nuemman on left

The unknowns are shown above in the circles. From earlier, we found that the discretization for the elliptic PDE at an internal node is (note that in the above, i is the row number, and j is column number)

$$U_{i,j} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(U_{i,j-1}h_y^2 + U_{i,j+1}h_y^2 + U_{i-1,j}h_x^2 + U_{i+1,j}h_x^2\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{i,j}$$

And for Nuemman, non-homegenouse boundary conditions, the left edge unknowns are given by

$$U_{i,1} = \frac{1}{2\left(h_x^2 + h_y^2\right)}\left(2U_{i,2}\ h_y^2 + \left(U_{i-1,1} + U_{i+1,1}\right)h_x^2 - h_y^2 h_x g_{i,1}\right) - \frac{h_x^2 h_y^2}{2\left(h_x^2 + h_y^2\right)} f_{i,1}$$

We will now write the 15 equations for each node, starting with $(2,1)$ to node $(4,5)$

$$U_{2,1} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(2U_{2,2}\ h_y^2+\left(U_{1,1}+U_{3,1}\right)h_x^2-h_y^2h_xg_{2,1}\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,1}$$

$$U_{2,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,1}h_y^2+U_{2,3}h_y^2+U_{1,2}h_x^2+U_{3,2}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,2}$$

$$U_{2,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,2}h_y^2+U_{2,4}h_y^2+U_{1,3}h_x^2+U_{3,3}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,3}$$

$$U_{2,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,3}h_y^2+U_{2,5}h_y^2+U_{1,4}h_x^2+U_{3,4}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,4}$$

$$U_{2,5} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,4}h_y^2+U_{2,6}h_y^2+U_{1,5}h_x^2+U_{3,5}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,5}$$

$$U_{3,1} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(2U_{3,2}\ h_y^2+\left(U_{2,1}+U_{4,1}\right)h_x^2-h_y^2h_xg_{3,1}\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,1}$$

$$U_{3,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,1}h_y^2+U_{3,3}h_y^2+U_{2,2}h_x^2+U_{4,2}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,2}$$

$$U_{3,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,2}h_y^2+U_{3,4}h_y^2+U_{2,3}h_x^2+U_{4,3}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,3}$$

$$U_{3,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,3}h_y^2+U_{3,5}h_y^2+U_{2,4}h_x^2+U_{4,4}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,4}$$

$$U_{3,5} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,4}h_y^2+U_{3,6}h_y^2+U_{2,5}h_x^2+U_{4,5}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,4}$$

$$U_{4,1} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(2U_{4,2}\ h_y^2+\left(U_{3,1}+U_{5,1}\right)h_x^2-h_y^2h_xg_{4,1}\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,1}$$

$$U_{4,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,1}h_y^2+U_{4,3}h_y^2+U_{3,2}h_x^2+U_{5,2}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,2}$$

$$U_{4,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,2}h_y^2+U_{4,4}h_y^2+U_{3,3}h_x^2+U_{5,3}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,3}$$

$$U_{4,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,3}h_y^2+U_{4,5}h_y^2+U_{3,4}h_x^2+U_{5,4}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,4}$$

$$U_{4,5} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,4}h_y^2+U_{4,6}h_y^2+U_{3,5}h_x^2+U_{5,5}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,5}$$

Now, moving the knowns to the right, results in

$$\begin{aligned}
2\left(h_x^2+h_y^2\right)U_{2,1}-2U_{2,2}\ h_y^2-U_{3,1}h_x^2 &= U_{1,1}h_x^2-h_y^2h_xg_{2,1}-h_x^2h_y^2f_{2,1}\\
2\left(h_x^2+h_y^2\right)U_{2,2}-U_{2,3}h_y^2-U_{3,2}h_x^2 &= -h_x^2h_y^2f_{2,2}+U_{2,1}h_y^2+U_{1,2}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,3}-U_{2,2}h_y^2-U_{2,4}h_y^2-U_{3,3}h_x^2 &= -h_x^2h_y^2f_{2,3}+U_{1,3}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,4}-U_{2,3}h_y^2-U_{2,5}h_y^2-U_{3,4}h_x^2 &= -h_x^2h_y^2f_{2,4}+U_{1,4}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{2,5}-U_{2,4}h_y^2+U_{3,5}h_x^2 &= -h_x^2h_y^2f_{2,5}+U_{2,6}h_y^2+U_{1,5}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{3,1}-2U_{3,2}\ h_y^2-U_{4,1}h_x^2-U_{2,1}h_x^2 &= -h_y^2h_xg_{3,1}-h_x^2h_y^2f_{3,1}\\
2\left(h_x^2+h_y^2\right)U_{3,2}-U_{3,3}h_y^2-U_{2,2}h_x^2-U_{4,2}h_x^2 &= -h_x^2h_y^2f_{3,2}+U_{3,1}h_y^2\\
2\left(h_x^2+h_y^2\right)U_{3,3}-U_{3,2}h_y^2-U_{3,4}h_y^2-U_{2,3}h_x^2-U_{4,3}h_x^2 &= -h_x^2h_y^2f_{3,3}\\
2\left(h_x^2+h_y^2\right)U_{3,4}-U_{3,3}h_y^2-U_{3,5}h_y^2-U_{2,4}h_x^2-U_{4,4}h_x^2 &= -h_x^2h_y^2f_{3,4}\\
2\left(h_x^2+h_y^2\right)U_{3,5}-U_{3,4}h_y^2+U_{2,5}h_x^2+U_{4,5}h_x^2 &= -h_x^2h_y^2f_{3,4}+U_{3,6}h_y^2\\
2\left(h_x^2+h_y^2\right)U_{4,1}-2U_{4,2}\ h_y^2-U_{3,1}h_x^2 &= U_{5,1}h_x^2-h_y^2h_xg_{4,1}-h_x^2h_y^2f_{4,1}\\
2\left(h_x^2+h_y^2\right)U_{4,2}-U_{4,3}h_y^2-U_{3,2}h_x^2 &= -h_x^2h_y^2f_{4,2}+U_{4,1}h_y^2+U_{5,2}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,3}-U_{4,2}h_y^2-U_{4,4}h_y^2-U_{3,3}h_x^2 &= -h_x^2h_y^2f_{4,3}+U_{5,3}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,4}-U_{4,5}h_y^2-U_{4,3}h_y^2-U_{3,4}h_x^2 &= -h_x^2h_y^2f_{4,4}+U_{5,4}h_x^2\\
2\left(h_x^2+h_y^2\right)U_{4,5}-U_{4,4}h_y^2-U_{3,5}h_x^2 &= -h_x^2h_y^2f_{4,5}+U_{4,6}h_y^2+U_{5,5}h_x^2
\end{aligned}$$

Now, we can write $Ax=b$ as, letting $\beta=2\left(h_x^2+h_y^2\right)$

$$\begin{pmatrix}
\beta & -2h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & -h_y^2 & \beta & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0\\
-h_x^2 & 0 & 0 & 0 & 0 & \beta & -2h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0\\
0 & -h_x^2 & 0 & 0 & 0 & 0 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0\\
0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0\\
0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2\\
0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & \beta & -2h_y^2 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & \beta & -h_y^2 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -h_y^2 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & -h_y^2\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta
\end{pmatrix}
\begin{pmatrix}
U_{21}\\ U_{22}\\ U_{23}\\ U_{24}\\ U_{25}\\ U_{31}\\ U_{32}\\ U_{33}\\ U_{34}\\ U_{35}\\ U_{41}\\ U_{42}\\ U_{43}\\ U_{44}\\ U_{45}
\end{pmatrix}=
\begin{pmatrix}
U_{1,1}h_x^2-h_y^2h_xg_{2,1}-h_x^2h_y^2f_{2,1}\\
-h_x^2h_y^2f_{2,2}+U_{2,1}h_y^2+U_{1,2}h_x^2\\
-h_x^2h_y^2f_{2,3}+U_{1,3}h_x^2\\
-h_x^2h_y^2f_{2,4}+U_{2,5}h_y^2+U_{1,4}h_x^2\\
-h_x^2h_y^2f_{2,5}+U_{2,6}h_y^2+U_{1,5}h_x^2\\
-h_y^2h_xg_{3,1}-h_x^2h_y^2f_{3,1}\\
-h_x^2h_y^2f_{3,2}+U_{3,1}h_y^2\\
-h_x^2h_y^2f_{3,3}\\
-h_x^2h_y^2f_{3,4}+U_{3,5}h_y^2\\
-h_x^2h_y^2f_{3,4}+U_{3,6}h_y^2\\
U_{5,1}h_x^2-h_y^2h_xg_{4,1}-h_x^2h_y^2f_{4,1}\\
-h_x^2h_y^2f_{4,2}+U_{4,1}h_y^2+U_{5,2}h_x^2\\
-h_x^2h_y^2f_{4,3}+U_{5,3}h_x^2\\
-h_x^2h_y^2f_{4,4}+U_{4,5}h_y^2+U_{5,4}h_x^2\\
-h_x^2h_y^2f_{4,5}+U_{4,6}h_y^2+U_{5,5}h_x^2
\end{pmatrix}$$

The above is the matrix for the case of non-homogeneous Neumann boundary conditions on the left edge.

We will now do the same edge, but with homogeneous boundary conditions to see the difference. Recall, that when the edge is insulated, then

$$U_{i,1}=\frac{h_xh_y}{\left(h_x^2+h_y^2\right)}\left(\frac{h_x}{2h_y}\left(U_{i-1,1}+U_{i+1,1}\right)-\frac{h_y}{h_x}U_{i,2}\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{i,1}$$

Using the above, we write the 15 equations starting with $(2,1)$ to node $(4,5)$

$$U_{2,1} = \frac{h_x h_y}{\left(h_x^2+h_y^2\right)}\left(\frac{h_x}{2h_y}\left(U_{1,1}+U_{3,1}\right)-\frac{h_y}{h_x}U_{2,2}\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,1}$$

$$U_{2,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,1}h_y^2+U_{2,3}h_y^2+U_{1,2}h_x^2+U_{3,2}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,2}$$

$$U_{2,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,2}h_y^2+U_{2,4}h_y^2+U_{1,3}h_x^2+U_{3,3}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,3}$$

$$U_{2,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,3}h_y^2+U_{2,5}h_y^2+U_{1,4}h_x^2+U_{3,4}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,4}$$

$$U_{2,5} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{2,4}h_y^2+U_{2,6}h_y^2+U_{1,5}h_x^2+U_{3,5}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{2,5}$$

$$U_{3,1} = \frac{h_x h_y}{\left(h_x^2+h_y^2\right)}\left(\frac{h_x}{2h_y}\left(U_{2,1}+U_{4,1}\right)-\frac{h_y}{h_x}U_{3,2}\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,1}$$

$$U_{3,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,1}h_y^2+U_{3,3}h_y^2+U_{2,2}h_x^2+U_{4,2}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,2}$$

$$U_{3,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,2}h_y^2+U_{3,4}h_y^2+U_{2,3}h_x^2+U_{4,3}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,3}$$

$$U_{3,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,3}h_y^2+U_{3,5}h_y^2+U_{2,4}h_x^2+U_{4,4}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,4}$$

$$U_{3,5} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{3,4}h_y^2+U_{3,6}h_y^2+U_{2,5}h_x^2+U_{4,5}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{3,4}$$

$$U_{4,1} = \frac{h_x h_y}{\left(h_x^2+h_y^2\right)}\left(\frac{h_x}{2h_y}\left(U_{3,1}+U_{5,1}\right)-\frac{h_y}{h_x}U_{4,2}\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,1}$$

$$U_{4,2} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,1}h_y^2+U_{4,3}h_y^2+U_{3,2}h_x^2+U_{5,2}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,2}$$

$$U_{4,3} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,2}h_y^2+U_{4,4}h_y^2+U_{3,3}h_x^2+U_{5,3}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,3}$$

$$U_{4,4} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,3}h_y^2+U_{4,5}h_y^2+U_{3,4}h_x^2+U_{5,4}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,4}$$

$$U_{4,5} = \frac{1}{2\left(h_x^2+h_y^2\right)}\left(U_{4,4}h_y^2+U_{4,6}h_y^2+U_{3,5}h_x^2+U_{5,5}h_x^2\right)-\frac{h_x^2h_y^2}{2\left(h_x^2+h_y^2\right)}f_{4,5}$$

Now, moving the knowns to the right, results in

$$\begin{aligned}
\frac{\left(h_x^2+h_y^2\right)}{h_x h_y}U_{2,1} - \frac{h_x}{2h_y}U_{3,1} + \frac{h_y}{h_x}U_{2,2} &= \frac{h_x}{2h_y}U_{1,1} - h_x h_y f_{2,1} \\
2\left(h_x^2+h_y^2\right)U_{2,2} - U_{2,3}h_y^2 - U_{3,2}h_x^2 &= -h_x^2h_y^2f_{2,2} + U_{2,1}h_y^2 + U_{1,2}h_x^2 \\
2\left(h_x^2+h_y^2\right)U_{2,3} - U_{2,2}h_y^2 - U_{2,4}h_y^2 - U_{3,3}h_x^2 &= -h_x^2h_y^2f_{2,3} + U_{1,3}h_x^2 \\
2\left(h_x^2+h_y^2\right)U_{2,4} - U_{2,3}h_y^2 - U_{2,5}h_y^2 - U_{3,4}h_x^2 &= -h_x^2h_y^2f_{2,4} + U_{1,4}h_x^2 \\
2\left(h_x^2+h_y^2\right)U_{2,5} - U_{2,4}h_y^2 + U_{3,5}h_x^2 &= -h_x^2h_y^2f_{2,5} + U_{2,6}h_y^2 + U_{1,5}h_x^2 \\
\frac{\left(h_x^2+h_y^2\right)}{h_x h_y}U_{3,1} - \frac{h_x}{2h_y}U_{2,1} - \frac{h_x}{2h_y}U_{4,1} + \frac{h_y}{h_x}U_{3,2} &= -h_x h_y f_{3,1} \\
2\left(h_x^2+h_y^2\right)U_{3,2} - U_{3,3}h_y^2 - U_{2,2}h_x^2 - U_{4,2}h_x^2 &= -h_x^2h_y^2f_{3,2} + U_{3,1}h_y^2 \\
2\left(h_x^2+h_y^2\right)U_{3,3} - U_{3,2}h_y^2 - U_{3,4}h_y^2 - U_{2,3}h_x^2 - U_{4,3}h_x^2 &= -h_x^2h_y^2f_{3,3} \\
2\left(h_x^2+h_y^2\right)U_{3,4} - U_{3,3}h_y^2 - U_{3,5}h_y^2 - U_{2,4}h_x^2 - U_{4,4}h_x^2 &= -h_x^2h_y^2f_{3,4} \\
2\left(h_x^2+h_y^2\right)U_{3,5} - U_{3,4}h_y^2 + U_{2,5}h_x^2 + U_{4,5}h_x^2 &= -h_x^2h_y^2f_{3,4} + U_{3,6}h_y^2 \\
\frac{\left(h_x^2+h_y^2\right)}{h_x h_y}U_{4,1} - \frac{h_x}{2h_y}U_{3,1} + \frac{h_y}{h_x}U_{4,2} &= \frac{h_x}{2h_y}U_{5,1} - h_x h_y f_{4,1} \\
2\left(h_x^2+h_y^2\right)U_{4,2} - U_{4,3}h_y^2 - U_{3,2}h_x^2 &= -h_x^2h_y^2f_{4,2} + U_{4,1}h_y^2 + U_{5,2}h_x^2 \\
2\left(h_x^2+h_y^2\right)U_{4,3} - U_{4,2}h_y^2 - U_{4,4}h_y^2 - U_{3,3}h_x^2 &= -h_x^2h_y^2f_{4,3} + U_{5,3}h_x^2 \\
2\left(h_x^2+h_y^2\right)U_{4,4} - U_{4,5}h_y^2 - U_{4,3}h_y^2 - U_{3,4}h_x^2 &= -h_x^2h_y^2f_{4,4} + U_{5,4}h_x^2 \\
2\left(h_x^2+h_y^2\right)U_{4,5} - U_{4,4}h_y^2 - U_{3,5}h_x^2 &= -h_x^2h_y^2f_{4,5} + U_{4,6}h_y^2 + U_{5,5}h_x^2
\end{aligned}$$

Now, we can write $Ax = b$ as, letting $\beta = 2\left(h_x^2+h_y^2\right)$ and $\gamma = \frac{\left(h_x^2+h_y^2\right)}{h_x h_y}$

$$\begin{pmatrix}
\gamma & \frac{h_y}{h_x} & 0 & 0 & 0 & -\frac{h_x}{2h_y} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -h_y^2 & \beta & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & 0 & 0 \\
-\frac{h_x}{2h_y} & 0 & 0 & 0 & 0 & \gamma & \frac{h_y}{h_x} & 0 & 0 & 0 & -\frac{h_x}{2h_y} & 0 & 0 & 0 & 0 \\
0 & -h_x^2 & 0 & 0 & 0 & 0 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 \\
0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 & 0 \\
0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 & 0 & 0 & -h_x^2 & 0 \\
0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & 0 & 0 & 0 & 0 & -h_x^2 \\
0 & 0 & 0 & 0 & 0 & -\frac{h_x}{2h_y} & 0 & 0 & 0 & 0 & \gamma & \frac{h_y}{h_x} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & \beta & -h_y^2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -h_y^2 & 0 & 0 & 0 & -h_y^2 & \beta & -h_y^2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta & -h_y^2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -h_x^2 & 0 & 0 & 0 & -h_y^2 & \beta
\end{pmatrix}
\begin{pmatrix} U_{21} \\ U_{22} \\ U_{23} \\ U_{24} \\ U_{25} \\ U_{31} \\ U_{32} \\ U_{33} \\ U_{34} \\ U_{35} \\ U_{41} \\ U_{42} \\ U_{43} \\ U_{44} \\ U_{45} \end{pmatrix} =
\begin{pmatrix}
\frac{h_x}{2h_y}U_{1,1} - h_x h_y f_{2,1} \\
-h_x^2h_y^2f_{2,2} + U_{2,1}h_y^2 + U_{1,2}h_x^2 \\
-h_x^2h_y^2f_{2,3} + U_{1,3}h_x^2 \\
-h_x^2h_y^2f_{2,4} + U_{2,5}h_y^2 + U_{1,4}h_x^2 \\
-h_x^2h_y^2f_{2,5} + U_{2,6}h_y^2 + U_{1,5}h_x^2 \\
-h_x h_y f_{3,1} \\
-h_x^2h_y^2f_{3,2} + U_{3,1}h_y^2 \\
-h_x^2h_y^2f_{3,3} \\
-h_x^2h_y^2f_{3,4} + U_{3,5}h_y^2 \\
-h_x^2h_y^2f_{3,4} + U_{3,6}h_y^2 \\
\frac{h_x}{2h_y}U_{5,1} - h_x h_y f_{4,1} \\
-h_x^2h_y^2f_{4,2} + U_{4,1}h_y^2 + U_{5,2}h_x^2 \\
-h_x^2h_y^2f_{4,3} + U_{5,3}h_x^2 \\
-h_x^2h_y^2f_{4,4} + U_{4,5}h_y^2 + U_{5,4}h_x^2 \\
-h_x^2h_y^2f_{4,5} + U_{4,6}h_y^2 + U_{5,5}h_x^2
\end{pmatrix}$$

Storage requirements for the different solvers

The total number of grid points along the x or the y direction is given by $\frac{length}{h} + 1$, where length is always 1, and hence n, the number of unknowns in one dimension is 2 less than the above number (since U is known at the boundaries).

It is important to note that the matrix A is not used explicitly to solve for the unknowns in the iterative schemes. Storage is needed only for the internal grid points, which is the number of the unknowns n^2. An auxiliary grid is used to hold the updated values in the Jacobian method. In addition, an auxiliary grid is required for $f_{i,j}$, the force function. Hence, in total $3n^2$ storage is needed.

Comparing this to the storage needed in the case of direct solver, where the storage for A alone is n^4. This shows the main advantage of the iterative methods compared to the direct method when A is a dense matrix. (Use of sparse matrix become necessary if direct solver is to be used for large n problems).

The following table summarizes the above discussion for the h values given in this problem. In this calculations, double precision (8 bytes) per grid point is assumed. For single precision half of this storage would be needed.

h	n	number of unknowns n^2	storage	size of A n^4	storage for dense A
2^{-5}	30	900	30 (k Bytes)	810,000	6.4 MBytes
2^{-6}	62	3844	0.1 (MB)	14,776,336	118 MBytes
2^{-7}	126	15 876	0.5 (MB)	252,047,376	2 GB

Loop algorithm for each method

This is a short description of the algorithm used by each method. In the following $u_{i,j}^{new}$ represents the new value (residing on the auxiliary grid) of the unknown, and $u_{i,j}^{current}$ is the current value. Initially $u^{current}$ is set to zero at each internal grid point. (any other initial value could also have been used).

Jacobi method algorithm

$k := 0$ (*counter*)
$\epsilon := Ch^2$ (*tolerance*)
$u^{current} := u^{new} := 0$ (*initialize storage*)
$f := f(x,y)$ (*initialize f on grid points*)
CONVERGED := *false*
WHILE NOT CONVERGED
  LOOP i
    LOOP j
      $u_{i,j}^{new} := \frac{1}{4}\left(u_{i-1,j}^{current} + u_{i+1,j}^{current} + u_{i,j-1}^{current} + u_{i,j+1}^{current} - h^2 f_{i,j}\right)$
      $R_{i,j} = f_{i,j} - \frac{1}{h^2}\left(u_{i-1,j}^{current} + u_{i+1,j}^{current} + u_{i,j-1}^{current} + u_{i,j+1}^{current} - 4u_{i,j}^{current}\right)$ (*residual*)
    END LOOP j
  END LOOP i
  $u^{current} := u^{new}$ (*update*)
  $k := k+1$
  IF $\left(\frac{\|R\|}{\|f\|} < \epsilon\right)$ *THEN* (*Norms are grid norms. see code*)
    CONVERGED := *true*
  END IF
END WHILE

Gauss-Seidel method

$k := 0$ (*counter*)
$\epsilon := Ch^2$ (*tolerance*)
$u^{current} := u^{new} := 0$ (*initialize storage*)
$f := f(x,y)$ (*initialize f on grid points*)
$CONVERGED := false$
$WHILE\ NOT\ CONVERGED$
    $LOOP\ i$
        $LOOP\ j$
            $u_{ij}^{new} := \frac{1}{4}\left(u_{i-1,j}^{new} + u_{i+1,j}^{current} + u_{i,j-1}^{new} + u_{i,j+1}^{current} - h^2 f_{i,j}\right)$
            $R_{i,j} = f_{i,j} - \frac{1}{h^2}\left(u_{i-1,j}^{current} + u_{i+1,j}^{current} + u_{i,j-1}^{current} + u_{i,j+1}^{current} - 4u_{i,j}^{current}\right)$ (*residual*)
        $END\ LOOP\ j$
    $END\ LOOP\ i$
    $u^{current} := u^{new}$ (*update*)
    $k := k+1$
    $IF\ \left(\frac{\|R\|}{\|f\|} < \epsilon\right)\ THEN$ (*Norms are grid norms. see code*)
        $CONVERGED := true$
    $END\ IF$
$END\ WHILE$

SOR method

$k := 0$ (*counter*)
$\epsilon := Ch^2$ (*tolerance*)
$u^{current} := u^{new} := 0$ (*initialize storage*)
$f := f(x,y)$ (*initialize f on grid points*)
$CONVERGED := false$
$WHILE\ NOT\ CONVERGED$
    $LOOP\ i$
        $LOOP\ j$
            $u_{ij}^{new} := \frac{\omega}{4}\left(u_{i-1,j}^{new} + u_{i+1,j}^{current} + u_{i,j-1}^{new} + u_{i,j+1}^{current} - h^2 f_{i,j}\right) + (1-\omega)\, u_{i,j}^{current}$
            $R_{i,j} = f_{i,j} - \frac{1}{h^2}\left(u_{i-1,j}^{current} + u_{i+1,j}^{current} + u_{i,j-1}^{current} + u_{i,j+1}^{current} - 4u_{i,j}^{current}\right)$ (*residual*)
        $END\ LOOP\ j$
    $END\ LOOP\ i$
    $u^{current} := u^{new}$ (*update*)
    $k := k+1$
    $IF\ \left(\frac{\|R\|}{\|f\|} < \epsilon\right)\ THEN$ (*Norms are grid norms. see code*)
        $CONVERGED := true$
    $END\ IF$
$END\ WHILE$

Notice that when $\omega = 1$, SOR becomes the same as *Gauss – Seidel*. When $\omega > 1$ the method is called overrelaxed and when $\omega < 1$ it is called underrelaxed.

2.4.4.7 Result of computation

The above 3 algorithms where implemented and ran for the 3 values of h given. The following tables summarizes the results obtained. The source code is in the appendix. This is a diagram illustrating the final converged solution from one of the runs.

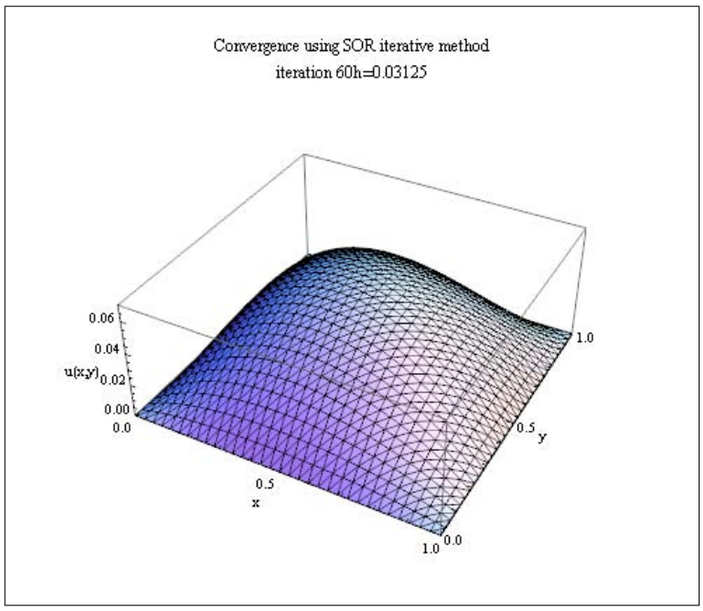


Figure 2.25: final convergedsolution

Number of steps to reach convergence

These table show the number of iterations to converge. The first was based on using $\varepsilon = 0.1h^2$ for tolerance and the second used $\varepsilon = h^2$

Number of iteration to reach convergence using tolerance $\varepsilon = 0.1h^2$

method	$h = 2^{-3}$ $n = 7$	$h = 2^{-4}$ $n = 15$	$h = 2^{-5}$ $n = 31$	$h = 2^{-6}$ $n = 63$	$h = 2^{-7}$ $n = 127$
Jacobi	82	400	1886	8689	note[4]
Gauss-Seidel	43	202	944	3446	19674
SOR	13	26	53	106	215

Number of iteration to reach convergence using tolerance $\varepsilon = h^2$

method	$h = 2^{-3}$ $n = 7$	$h = 2^{-4}$ $n = 15$	$h = 2^{-5}$ $n = 31$	$h = 2^{-6}$ $n = 63$	$h = 2^{-7}$ $n = 127$
Jacobi	52	281	1409	6779	31702
Gauss – Seidel	28	142	706	3391	15852
SOR	10	20	40	80	159

Convergence plots for each method

These error plots where generated only for tolerance $\varepsilon = 1 \times h^2$. They show how the log of the norm of the relative residual changes as the number of iterations changed until convergence is achieved.

In these plots, the *yaxis* is $\log\left(\frac{\|R^{[k]}\|}{\|f\|}\right)$ or $\log\left(\frac{\|f-Au^{[k]}\|}{\|f\|}\right)$, and the *xaxis* is the k value (the iteration number).

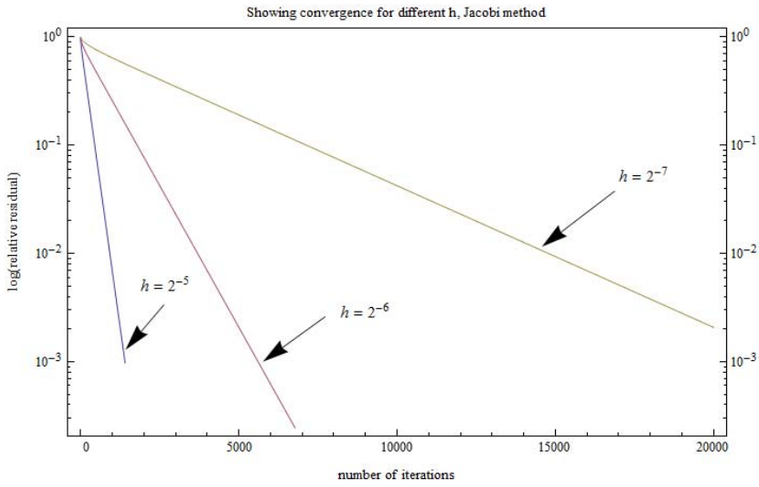


Figure 2.26: error plot Jacobi

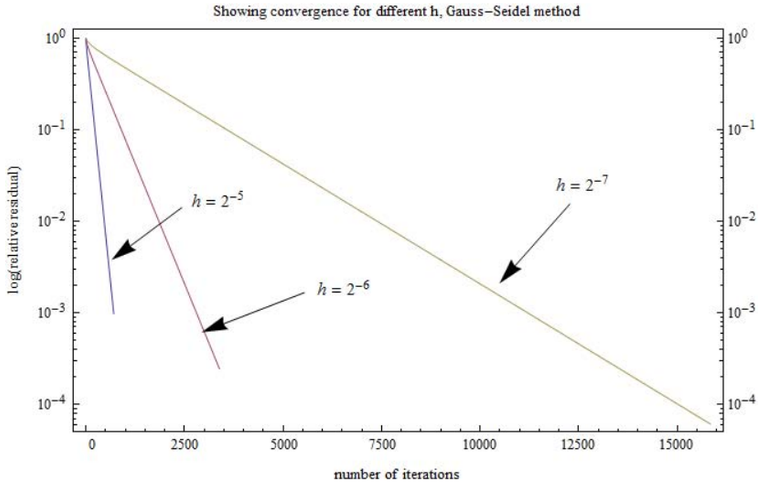


Figure 2.27: error plot GS

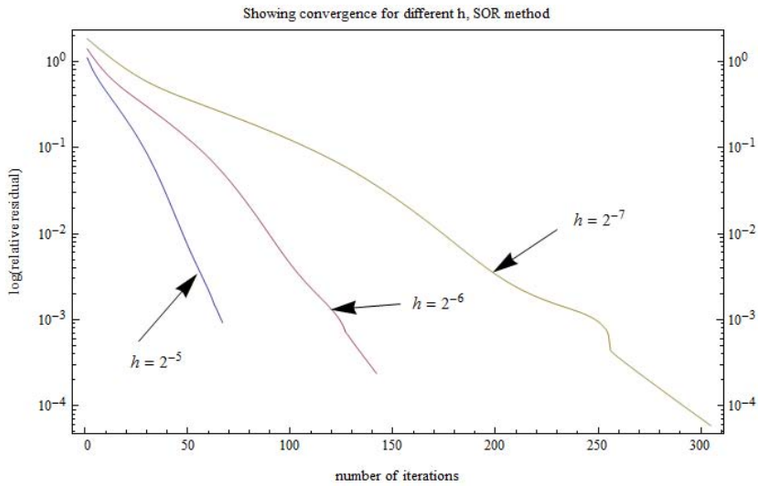


Figure 2.28: error plot SOR

Plots for comparing convergence of the 3 methods for $\epsilon = 1 \times h^2$ for different h values

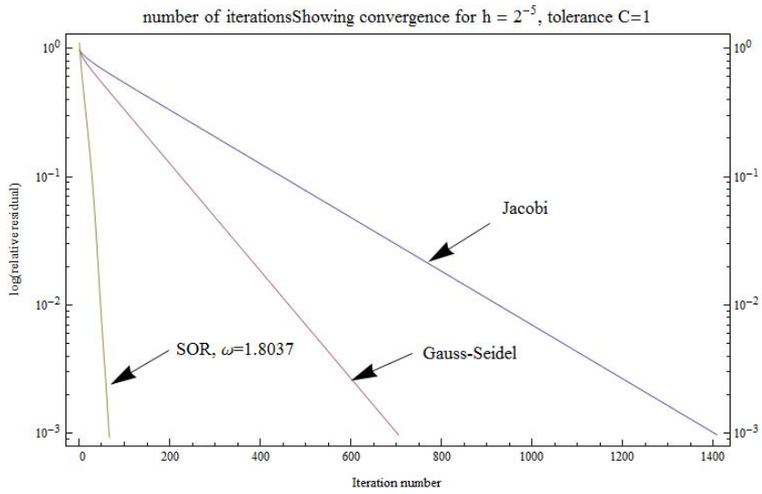


Figure 2.29: prob1 compare 3h 25

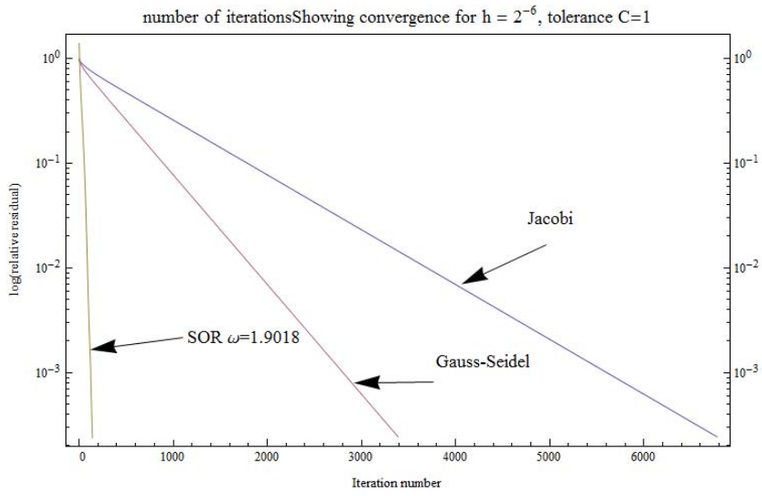


Figure 2.30: prob1 compare 3h 26

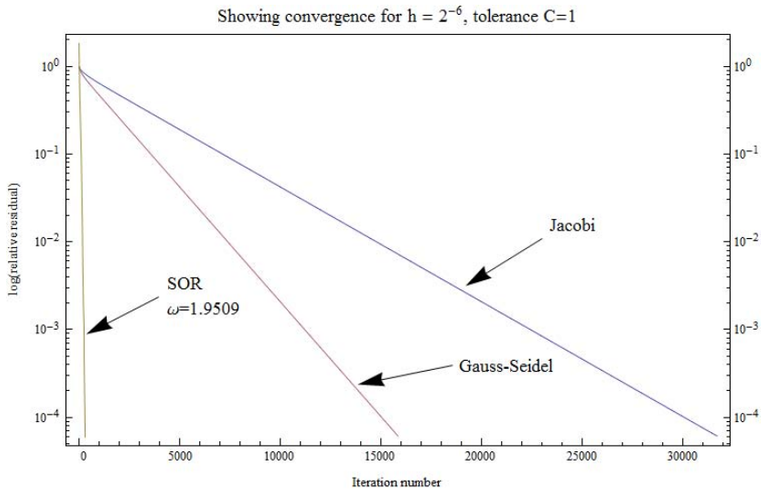


Figure 2.31: prob1 compare 3h 27

2.4.4.8 Conclusions and summary

1. SOR is the fastest iterative solver of the three solvers.
2. SOR method required calculation of an optimal ω to use. For this problem, this calculation was not difficult. In other problems it can be difficult to determine before hand the optimal ω. Some SOR methods use an adaptive ω where ω is readjusted as the solution progresses.

3. The use of relative residual to determine the condition of convergence required applying the matrix A without actually storing A.
4. Jacobi method required an additional auxiliary grid storage, hence its memory requirement was twice as much as Gauss-Seidel or SOR.
5. Jacobi method was the simplest to implement, but it was the slowest to converge.
6. Jacobi method is more suitable for use in parallel processing, where each grid point can be updated independent of the grid point next to it. This is not possible with Gauss-Seidel nor SOR due to the dependency of updates on its immediate grid points. However, if a red-black numbering is used, then it would be possible to implement these methods in parallel in 2 stages.
7. All methods are guaranteed to converge eventually, as the spectral radius of the iterative matrix for each method is less than one.

2.4.5 Problem 2

2. When solving parabolic equations numerically, one frequently needs to solve an equation of the form
$$u - \delta \Delta u = f,$$
where $\delta > 0$. The analysis and numerical methods we have discussed for the Poisson equation can be applied to the above equation. Suppose we are solving the above equation on the unit square with Dirichlet boundary conditions. Use the standard five point stencil for the discrete Laplacian.
   (a) Analytically compute the eigenvalues of the Jacobi iteration matrix, and show that the Jacobi iteration converges.
   (b) If $h = 10^{-2}$ and $\delta = 10^{-3}$, how many iterations of SOR would it take to reduce the error by a factor of 10^{-6}? How many iterations would it take for the Poisson equation? Use that the spectral radius of SOR is
$$\rho_{\text{sor}} = \omega_{\text{opt}} - 1,$$
where
$$\omega_{\text{opt}} = \frac{2}{1 + \sqrt{1 - \rho_J^2}},$$
and where ρ_J is the spectral radius of Jacobi.

Figure 2.32: Problem 2

2.4.5.1 Part(a)

Given
$$u - \delta \Delta u = f$$
And using standard 5 point Laplacian for the approximation of Δ, the above can be written as
$$u - \delta A u = f \tag{1}$$
Where A is the Jacobian matrix for 2D
$$\frac{1}{h^2}\begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & \ddots & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix}$$

Hence (1) becomes

$$(I - \delta A)\, u = f \tag{2}$$

Let

$$B = (I - \delta A) \tag{3}$$

Then (2) becomes

$$Bu = f \tag{3A}$$

To obtain the iterative matrix for the above system, the method of matrix splitting is used. Let $B = M - N$. Equation (3A) becomes

$$\begin{aligned}(M - N)\, u &= f \\ Mu &= Nu + f\end{aligned}$$

M is selected so that it is invertible and such that M^{-1} is easy to compute, the iterative equation results

$$u^{[k+1]} = \left(M^{-1}N\right) u^{[k]} + M^{-1}f$$

Where iterative matrix T_j is

$$T_j = \left(M^{-1}N\right) \tag{3B}$$

For convergence it is required that the spectral radius of T_j be less than one. $\rho\left(T_j\right)$ is the largest eigenvalue of T_j in absolute terms. The largest eigenvalue of T_j is now found as follows.

For the Jacobi method let $M = D$, and $N = L + U$, where D is the diagonal of B, L is the negative of the strictly lower triangle matrix of B and U is negative of the strictly upper triangle matrix of B. (3B) becomes

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

But $B = D - L - U$, hence $L + U = D - B$ and the above becomes

$$\begin{aligned}T_j &= D^{-1}\,(D - B) \\ &= I - D^{-1}B\end{aligned} \tag{4}$$

Now the spectral radius of T_j is determined. First D^{-1} is found. But first B needs to be determined. From (3)

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

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

Therefore, D the diagonal matrix of B is easily found

$$D = \begin{pmatrix} 1+\frac{4\delta}{h^2} & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1+\frac{4\delta}{h^2} & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1+\frac{4\delta}{h^2} & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1+\frac{4\delta}{h^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1+\frac{4\delta}{h^2} \end{pmatrix}$$

Now that D is known, the eigenvalue μ_{kl} of the iteration matrix T_j shown in (4) can be written down as

$$\begin{aligned} T_j &= I - D^{-1}B \\ &\Rightarrow \\ \mu_{kl} &= 1 - \left(1+\frac{4\delta}{h^2}\right)^{-1} \lambda_{kl} \end{aligned} \tag{5}$$

where λ_{kl} is the eigenvalues of B. But from (3), $B = (I - \delta A)$, hence (5) becomes

$$\mu_{kl} = 1 - \left(1+\frac{4\delta}{h^2}\right)^{-1} (1 - \delta\nu_{kl}) \tag{6}$$

Where ν_{kl} is the eigenvalue of A, the standard Jacobi A matrix for 2D, with eigenvalues given in textbook (page 63, equation 3.15)

$$\nu_{kl} = \frac{2}{h^2}\left(\cos(k\pi h) + \cos(l\pi h) - 2\right)$$

Using this in (6) results in

$$\begin{aligned} \mu_{kl} &= 1 - \left(1+\frac{4\delta}{h^2}\right)^{-1}\left(1 - \frac{2\delta}{h^2}\cos(k\pi h) - \frac{2\delta}{h^2}\cos(l\pi h) + \frac{4\delta}{h^2}\right) \\ &= 1 - \left(\frac{h^2}{h^2+4\delta}\right)\left(1 - \frac{2\delta}{h^2}\cos(k\pi h) - \frac{2\delta}{h^2}\cos(l\pi h) + \frac{4\delta}{h^2}\right) \\ &= 1 - \left(\frac{h^2}{h^2+4\delta}\right) - \left(\frac{4\delta}{h^2+4\delta}\right) + \frac{2\delta}{h^2}\left(\frac{h^2}{h^2+4\delta}\right)\{\cos(k\pi h) + \cos(l\pi h)\} \\ &= \left(\frac{\overbrace{h^2+4\delta-h^2-4\delta}^{=0}}{h^2+4\delta}\right) + \left(\frac{2\delta}{h^2+4\delta}\right)\{\cos(k\pi h) + \cos(l\pi h)\} \\ &= \left(\frac{2\delta}{h^2+4\delta}\right)\{\cos(k\pi h) + \cos(l\pi h)\} \end{aligned}$$

The largest value of the above occurs when $\cos(k\pi h) + \cos(l\pi h)$ is maximum, which is 2. Therefore

$$\rho\left(T_j\right) = \left(\frac{4\delta}{h^2+4\delta}\right)$$

Which is less than one for any $\delta > 0$.

Hence it is now shown that Jacobi iteration converges for this system.

2.4.5.2 Part (b)

Reducing the error by factor 10^{-6} implies

$$\left\|e^k\right\| < 10^{-6}\left\|e^0\right\| \tag{1}$$

but by definition

$$\left\|e^k\right\| = \rho_{sor}^k \left\|e^0\right\|$$

Hence (1) becomes

$$\rho_{sor}^k < 10^{-6}$$

Where the solution for k at equality is found (rounded to the largest integer if needed). Hence the above becomes, after taking logarithms of both sides

$$\begin{aligned} k \log \rho_{sor} &= -6 \\ k &= \frac{-6}{\log \rho_{sor}} \end{aligned} \tag{2}$$

But

$$\rho_{sor} = \omega_{opt} - 1$$

Where

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho_j^2}}$$

And $\rho_j = \left(\frac{4\delta}{h^2+4\delta}\right)$ from part(a), hence the above becomes

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \left(\frac{4\delta}{h^2+4\delta}\right)^2}}$$

Hence

$$\rho_{sor} = \frac{2}{1 + \sqrt{1 - \left(\frac{4\delta}{h^2+4\delta}\right)^2}} - 1$$

Substituting the numerical values $h = 10^{-2}, \delta = 10^{-3}$ in the above results in

$$\begin{aligned} \rho_{sor} &= \frac{2}{1 + \sqrt{1 - \left(\frac{4\left(10^{-3}\right)}{\left(10^{-2}\right)^2+4\left(10^{-3}\right)}\right)^2}} - 1 \\ &= 0.64 \end{aligned}$$

Therefore, from (2)

$$\begin{aligned} k &= \frac{-6}{\log(0.64)} \\ &= 30.95 \end{aligned}$$

rounding up gives

$$k = 31$$

2.4.6 Problem 3

3. In this problem we compare the speed of SOR to a direct solve using Gaussian elimination. At the end of this assignment is MATLAB code to form the matrix for the 2D discrete Laplacian. The code for the 3D matrix is similar. Note that with 1 GB of memory, you can handle grids up to about 1000×1000 in 2D and $40 \times 40 \times 40$ in 3D with a direct solve. The range of grids you will explore depends on the amount of memory you have.

   (a) Solve the PDE from problem 1 using a direct solve. Put timing commands in your code and report the time to solve for a range of mesh spacings. Use SOR to solve on the same meshes and report the time and number of iterations. Comment on your results.

   (b) Repeat the previous part in three spatial dimensions for a range of mesh spacings. Change the right side of the equation to be a three dimensional Gaussian. Comment on your results.

Figure 2.33: Problem 3

2.4.6.1 Part(a)

To solve the problem using direct solver, the matrix A is constructed (sparse matrix), and the vector f is evaluated using the given function $f\left(x, y\right)$, this results in an $Au = f$ system, which is then solved using a direct solver.

Recall from problem (1) that the A matrix has the following form (as an example, for 3×3 internal grid, or for $h=0.2$)

$$\begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix} \begin{pmatrix} U_{1,1} \\ U_{2,1} \\ U_{3,1} \\ U_{1,2} \\ U_{2,2} \\ U_{3,2} \\ U_{1,3} \\ U_{2,3} \\ U_{3,3} \end{pmatrix} = h^2 \begin{pmatrix} f_{1,1} \\ f_{2,1} \\ f_{3,1} \\ f_{1,2} \\ f_{2,2} \\ f_{3,2} \\ f_{1,3} \\ f_{2,3} \\ f_{3,3} \end{pmatrix}$$

The matrix A is set up, as sparse for the following set of values

h	internal grid size ($n=\frac{1}{h}-1$)
2^{-5}	31×31
2^{-6}	63×63
2^{-7}	127×127
2^{-8}	255×255
2^{-9}	511×511
2^{-10}	1023×1023

A function is written to evaluate $f\left(x,y\right)$ at each of the internal grid points and reshaped to be column vector in the order shown above. Then the direct solver is called.

Next, the SOR solver is used for each of the above spacings. First ω was calculated for each h using $\omega_{opt}\approx2\left(1-\pi h\right)$ resulting in

h	ω_{opt}
2^{-5}	1.803 7
2^{-6}	1.901 8
2^{-7}	1.950 9
2^{-8}	1.975 5
2^{-9}	1.987 7
2^{-10}	1.993 9

Then the SOR solver which was written in problem (1) was called for each of the above cases. The next section shows the results obtained. The source code is in the appendix.

2.4.6.2 Result of computation

The following is an image of f(x,y) on the grid

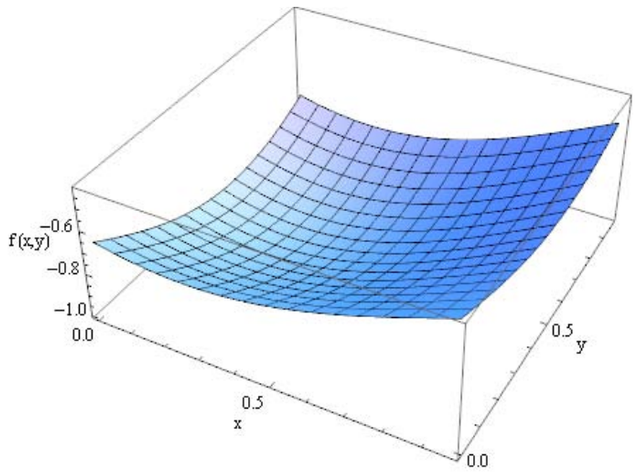


Figure 2.34: image of $f(x,y)$

And the solution obtained by direct solver on 2D is the following (implemented in Matlab and in Mathematica)

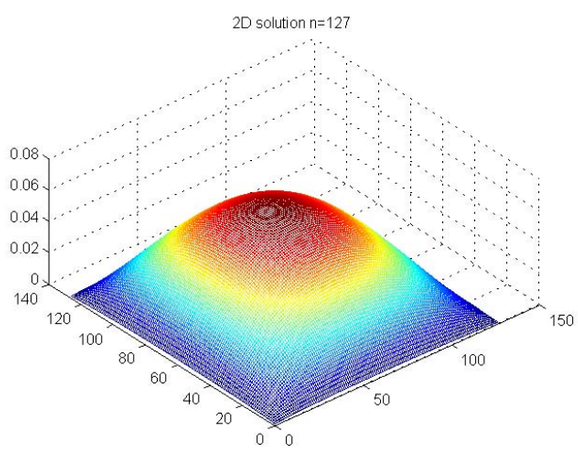


Figure 2.35: Solution by direct solver, Matlab

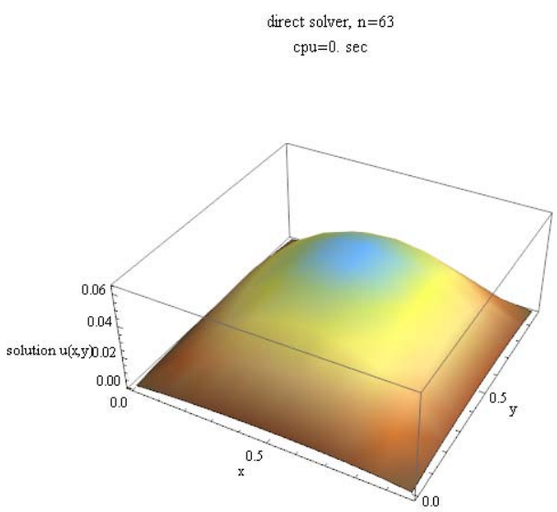


Figure 2.36: Solution by direct solver using Mathematica

The CPU results below are in seconds. The function cputime() was used to obtain cpu time used in Matlab. For SOR, the cpu time was that of the whole iterative loop until convergence

was achieved. In Mathematica, the command Timing[] which measures the CPU time was used. These are the results obtained using Matlab 2010a and using Mathematica 7.0[5]

In this table, the grid size n represents the number of internal grid points in one dimension. For example, for $n = 31$, the grid size will be 31×31. The number of non zero elements shown in the table relates to storage used by the sparse matrix and was obtained in Matab by calling nnz(A).

h	n	N number of unknowns	number non zero elements	Direct Solver CPU MATLAB	Direct Solver CPU Mathematica	SOR Solver CPU	k SOR number of iterations
2^{-5}	31	961	4,681	0.015	0.016	0	68
2^{-6}	63	3,969	19,593	0.125	0.016	0.094	143
2^{-7}	127	16,129	80,137	0.250	0.063	0.6	306
2^{-8}	255	65,025	324,105	1.544	0.344	5.2	661
2^{-9}	511	261,121	1,303,561	5.538	1.90	48.9	1427
2^{-10}	1023	1,046,529	5,228,553	27.113	14.57	532	3088

These 2 plots illustrate the CPU difference, done on a normal scale and on log scale. (using Matlab results only).

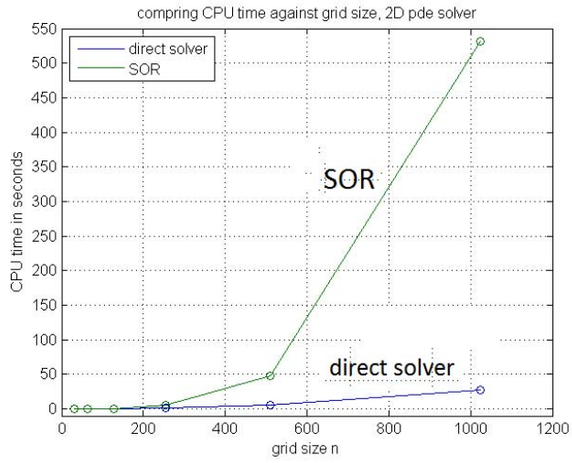


Figure 2.37: prob3 part a compare CPU normal scale

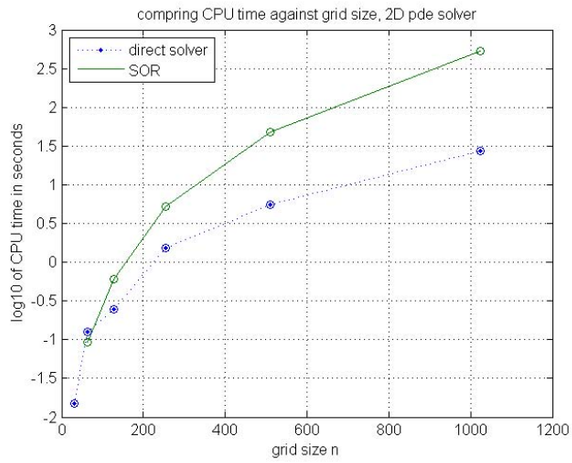


Figure 2.38: plot prob3 part a compare CPU

Comments on results obtained

[5]Matlab 2010a, on windows 7, 64 bit OS, intel i7 930, with 8 GB installed physical RAM.

CPU performance for SOR is given by

$$work = number\ iterations \times work\ per\ iteration$$

The number of iterations depends on the constant used for tolerance. Let k be the number of iterations, and let the tolerance be Ch^2 where h is the spacings. Hence

$$k = \frac{\log\left(Ch^2\right)}{\log \rho_{sor}} = \frac{\log C + 2\log h}{\log(1-2\pi h)} \approx \frac{\log C + 2\log h}{-2\pi h} \approx O\left(h^{-1}\log h\right)$$

But $h = O\left(\frac{1}{n}\right)$ where n is the number of grid points in one dimension. Therefore

$$k = O(n \log n)$$

And since there are n^2 unknowns, the work per iteration is $O\left(n^2\right)$, hence for SOR performance, work becomes

$$CPU_{sor} = O\left(n^3 \log n\right)$$

Expressing this in terms of the unknowns $N = n^2$ gives

$$CPU_{sor} = O\left(N^{\frac{3}{2}} \log N\right)$$

For direct solver, the work is proprprtional to (Nb) where b is the banwidth (when using nested dissection method)[6]. The bandwidth is n, hence for direct solver on 2D using sparse matrices, the performance is n^3

$$CPU_{direct} = O\left(N^{\frac{3}{2}}\right)$$

In summary

method	CPU (in terms of number of unknowns N)	CPU in terms of n
SOR 2D	$O\left(N^{\frac{3}{2}} \log N\right)$	$O\left(n^3 \log n\right)$
direct solver 2D	$O\left(N^{\frac{3}{2}}\right)$	$O\left(n^3\right)$

For small number of unknowns, SOR was very competitive with direct solver but when the number of unknowns became larger than about $N \approx 100$, the direct solver is faster as the effect of the $\log n$ factor starts to take effect on the performance of SOR. The results shown in the plots above confirmed this performance analysis.

2.4.6.3 Part(b)

The goal is to solve

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = -\exp(-(x-0.25)^2 - (y-0.6)^2 - z^2)$$

On the unit cube. Referring to the following diagram made to help in setting up the 3D scheme to approximate the above PDE

[6] See textbook, page 68.

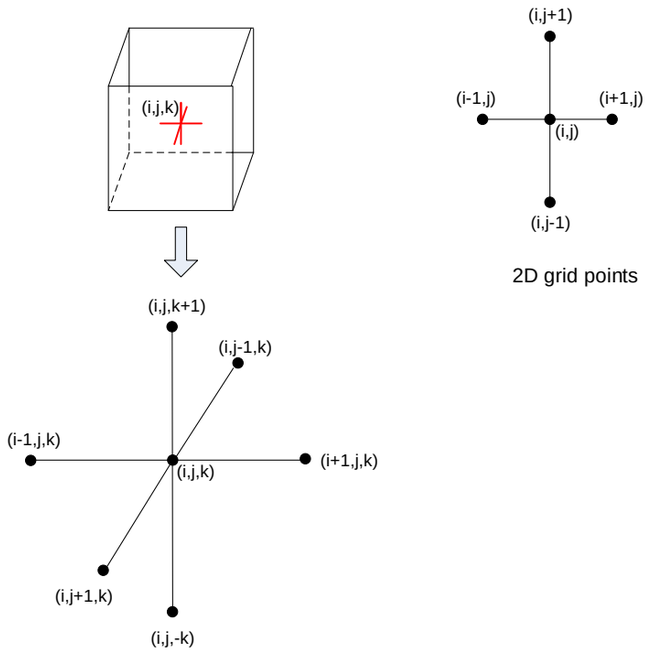


Figure 2.39: 3D axis

The discrete approximation to the PDE can be written as

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} = \frac{1}{h^2}\left(U_{i-1,j,k} + U_{i+1,j,k} + U_{i,j-1},k + U_{i,j+1,k} + U_{i,k,k-1} + U_{i,j,k+1} - 6U_{i,j,k}\right)$$

Hence the SOR scheme becomes

$$U_{i,j,k}^{[k+1]} = \frac{\omega}{6}\left(U_{i-1,j,k}^{[k+1]} + U_{i+1,j,k}^{[k]} + U_{i,j-1,k}^{[k+1]} + U_{i,j+1,k}^{[k]} + U_{i,j,k-1}^{[k+1]} + U_{i,j,k+1}^{[k]} - h^2 f_{i,j}\right) + (1-\omega)\, U_{i,j,k}^{[k]}$$

For the direct solver, the A matrix needs to be formulated. From

$$\frac{1}{h^2}\left(U_{i-1,j,k} + U_{i+1,j,k} + U_{i,j-1},k + U_{i,j+1,k} + U_{i,k,k-1} + U_{i,j,k+1} - 6U_{i,j,k}\right) = f_{i,j,k}$$

And solving for $U_{i,j,k}$ results in

$$U_{i,j,k} = \frac{1}{6}\left(U_{i-1,j,k} + U_{i+1,j,k} + U_{i,j-1},k + U_{i,j+1,k} + U_{i,k,k-1} + U_{i,j,k+1} - h^2 f_{i,j,k}\right)$$

To help make the A matrix, using an example with $n = 2$, the following diagram is made with the standard numbering on each node

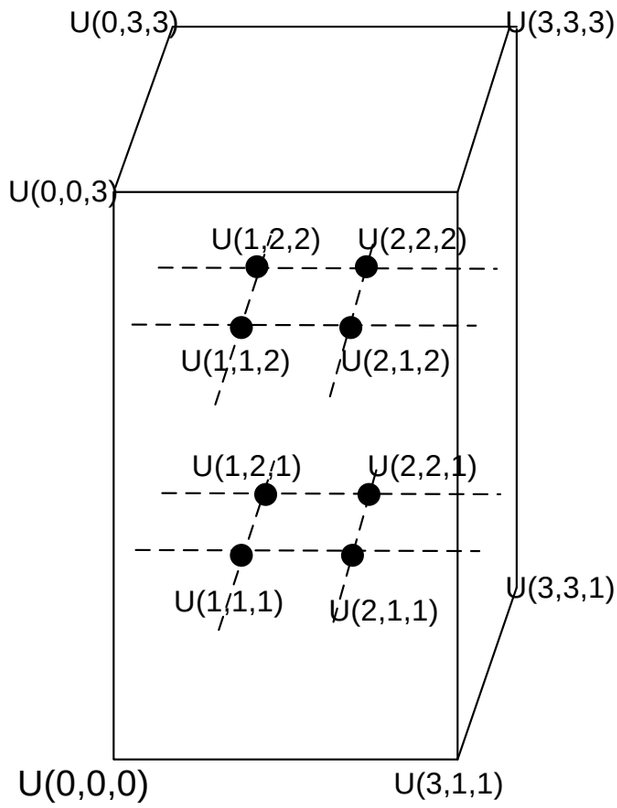


Figure 2.40: 3d grid example

By traversing the grid, left to right, then inwards into the paper, then upwards, the following A matrix results

$$\begin{pmatrix}
-6 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & -6 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & -6 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & -6 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & -6 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & -6 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & -6 & 1 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & -6
\end{pmatrix}
\begin{pmatrix} U_{1,1,1} \\ U_{2,1,1} \\ U_{1,2,1} \\ U_{2,2,1} \\ U_{1,1,2} \\ U_{2,1,2} \\ U_{1,2,2} \\ U_{2,2,2} \end{pmatrix}
= h^3
\begin{pmatrix} f_{1,1,1} \\ f_{2,1,1} \\ f_{1,2,1} \\ f_{2,2,1} \\ f_{1,1,2} \\ f_{2,1,2} \\ f_{1,2,2} \\ f_{2,2,2} \end{pmatrix}$$

Figure 2.41: repating A sructure n2

One can see the recursive pattern involved in these A matrices. Each A matrix contains inside it a block on its diagonal which repeats n times. Each block in turn, contain inside it, on its diagonal, smaller block, which also repeats n times.

It is easier to see the pattern of building A by using numbers for the grid points, and label them in the same order as they would be visited, this allowed one to see the connection between each grid point to the other much easier. For example, for $n = 2$,

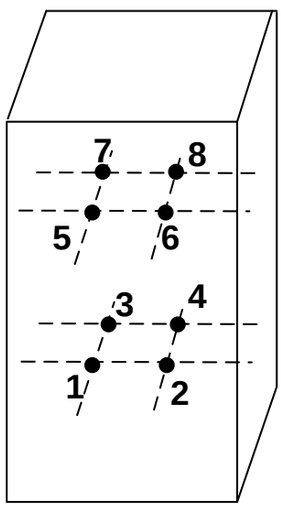


Figure 2.42: 3d grid n2 numbers

One can see now more easily the connections. grid point 1 has connection to only $2,3,5$ points. This means when looking at the A matrix, there will be a 1 in the first row, at columns $2,3,5$. Similarly, point 2 has connections only to $1,4,6$, which means in the second row, there will be a 1 at columns $1,4,6$. Extending the number of points to $n = 3$ to better see the pattern of A results in

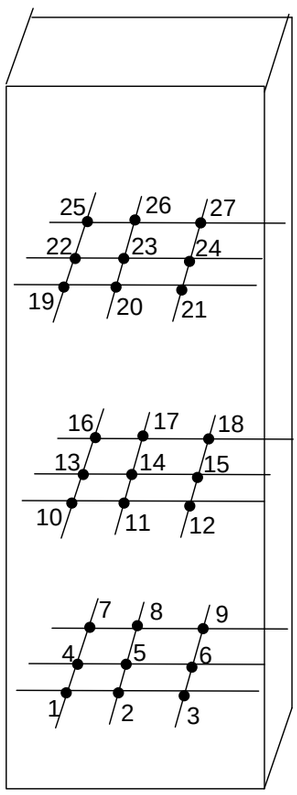


Figure 2.43: 3d grid n3 numbers

From the above, one can see clearly that, for example, point 1 is connected only to $2,4,10$ and 2 is connected to $1,3,5,11$ and so on. The above shows that each point will have a connection to a point which is numbered n^2 higher than the grid point itself. n^2 is the size of the grid in each surface. Hence, the general A matrix, for the above example, can now be written as

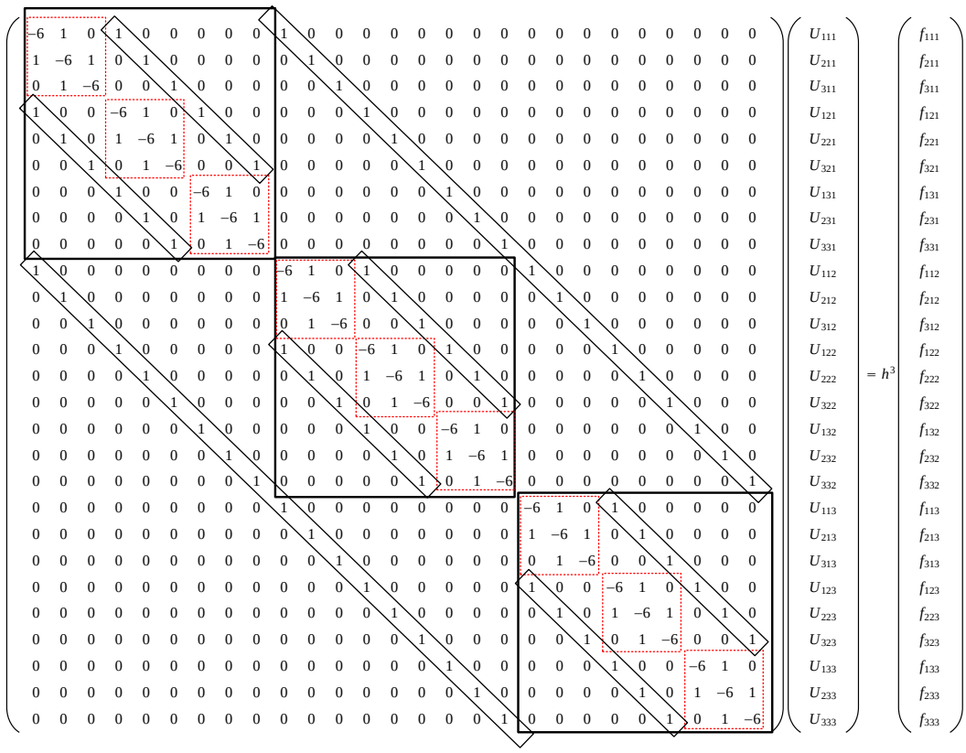


Figure 2.44: A sructure 3D

One can see the recursive structure again. There are $n = 3$ main repeating blocks on the diagonal, and each one of them in turn has $n = 3$ repeating blocks on its own diagonal. Here $n = 3$, the number of grid points along one dimension.

Now that the A structure is understood, the Matlab code for filling the sparse matrix is modified for the 3D case as follows

```
function L3 = lap3d(n)

L2=lap2d(n,n);
e=ones(n^3,1);
L=spdiags([e e],[-n^2 n^2],n^3,n^3);
Iz=speye(n);

L3=kron(Iz,L2)+L;
end

%-------
function L2 = lap2d(nx,ny)

Lx=lap1d(nx);
Ly=lap1d(ny);

Ix=speye(nx);
Iy=speye(ny);

L2=kron(Iy,Lx)+kron(Ly,Ix);
end

function L=lap1d(n)
e=ones(n,1);
L=spdiags([e -3*e e],[-1 0 1],n,n);
end
```

.

To test, for example, for $n = 2$

```
EDU>> full(lap3d(2))
ans =
    -6     1     1     0     1     0     0     0
     1    -6     0     1     0     1     0     0
     1     0    -6     1     0     0     1     0
     0     1     1    -6     0     0     0     1
     1     0     0     0    -6     1     1     0
     0     1     0     0     1    -6     0     1
     0     0     1     0     1     0    -6     1
     0     0     0     1     0     1     1    -6
```

.

Using the above function, the solution was found using direct solver.

Result of computation

The results for the 3D solver are as follows. In this table, n represents the number of grid points in one dimension. Hence $n = 10$ represents a 3D space of [10,10,10] points. The number of non zero elements in the table relates to the sparse matrix used for the direct solver and was obtained using Matab call nnz(A).

h	n	N total number unknowns (n^3)	make sparse CPU time	A\f CPU time	Total Direct Solver CPU	Total SOR Solver CPU	SOR number iterations
0.090909	10	1,000	0.047	0	0.047	0	23
0.047619	20	8,000	0.062	0.44	0.502	0.078	44
0.032258	30	27,000	0.016	3.90	3.90	0.405	65
0.027778	35	42,875	0.359	8.70	8.80	0.75	77
0.024390	40	64,000	0.328	21.20	21.50	1.29	88
0.021739	45	91,125	0.296	39.80	40.00	2.11	100
0.019608	50	125,000	0.624	84.20	84.80	3.24	112
0.017857	55	166,375	0.421	157.30	157.70	4.9	125
0.016393	60	216,000	0.889	244.10	244.20	7.17	138

For the direct solver, Matlab ran out of memory at $n = 65$ as shown below

```

EDU>> nma_HW3_problem_3_partB_direct_solver
.....

*******************************
grid is 3D [60,60,60]
h=0.016393
cpu time for making sparse matrix=1.060807 seconds
dimensions of A (sparse matrix) is [216000,216000]
nnz(A)= 1490400
cpu time for direct solver=240.693943 seconds
*******************************
grid is 3D [65,65,65]
h=0.015152
cpu time for making sparse matrix=0.826805 seconds
dimensions of A (sparse matrix) is [274625,274625]
nnz(A)= 1897025
??? Error using ==> mldivide
Out of memory. Type HELP MEMORY for your options.

Error in ==> nma_HW3_problem_3_partB_direct_solver at 54
    u = A\f;
```

.

This plot illustrates the CPU difference table on a log scale

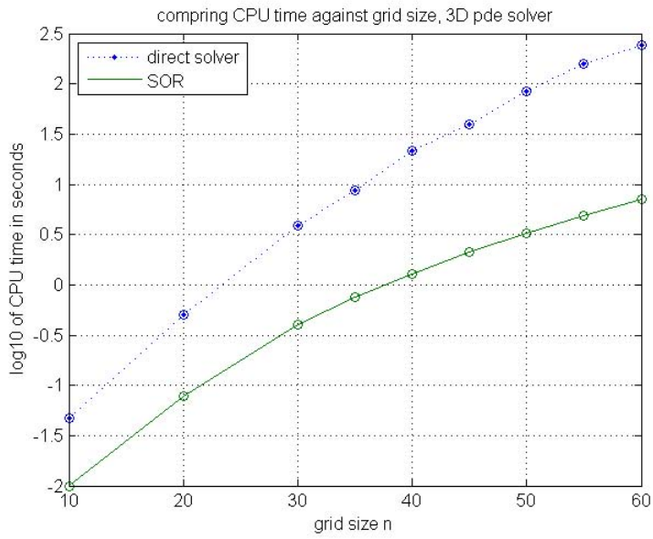


Figure 2.45: prob3 part B compare CPU log scale

Comments on results obtained

CPU performance for SOR is given by

$$\text{work} = \text{number iterations} \times \text{work per iteration}$$

The number of iterations depends on the constant used for tolerance. Let k be the number of iterations, and let the tolerance be Ch^2 where h is the spacings. Hence

$$k = \frac{\log\left(Ch^2\right)}{\log \rho_{sor}} = \frac{\log C + 2\log h}{\log\left(1 - 2\pi h\right)} \approx \frac{\log C + 2\log h}{-2\pi h} \approx O\left(h^{-1}\log h\right)$$

But $h = O\left(\frac{1}{n}\right)$ where n is the number of grid points in one dimension. Hence

$$k = O\left(n \log n\right)$$

And since there are n^3 unknowns (compared to n^2 in 2D), then work per iteration is $O\left(n^3\right)$, hence for SOR performance becomes

$$CPU_{sor} = O\left(n^4 \log n\right)$$

Expressing this in terms of $N = n^3$ as the number of unknowns, gives

$$CPU_{sor} = O\left(N^{\frac{4}{3}} \log N\right)$$

For direct solver, the work is proprprtional to (Nb) where b is the banwidth (when using nested dissection method)[7]. The bandwidth is n^2 in this case and not n as was with 2D. Hence the total cost is

$$\begin{aligned} CPU_{direct} &= O\left(N \times n\right) \\ &= O\left(n^5\right) \\ &= O\left(N^{\frac{5}{3}}\right) \end{aligned}$$

Hence

method	CPU (in terms of N)	CPU in terms of n
SOR 3D	$O\left(N^{\frac{4}{3}} \log N\right)$	$O\left(n^4 \log n\right)$
direct solver 3D	$O\left(N^{\frac{5}{3}}\right)$	$O\left(n^5\right)$

[7]See textbook, page 68.

The above shows that SOR is faster than direct solver performance. The results shown in the plots above confirmed this analytical performance prediction showing SOR to be faster. To verify the above, a plot was made using the above complexity cost measure to determine if the resulting shape matches the one obtained from the actual runs above. The following plot shows the complexity cost made on sequence of values that represent n

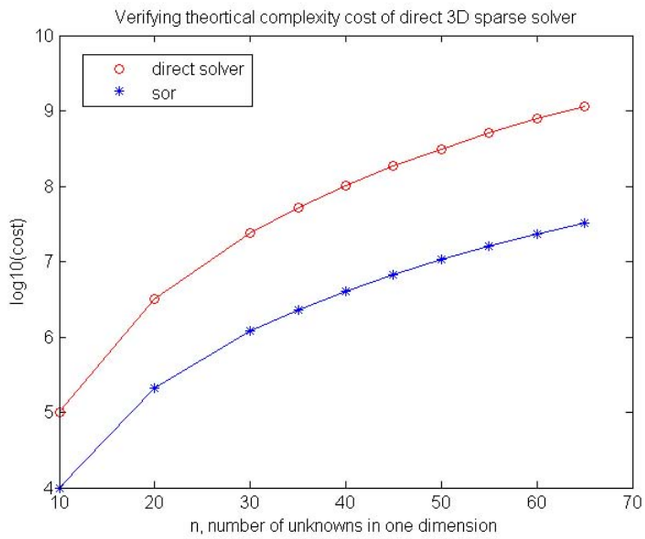


Figure 2.46: verify cost of 3D

The following matlab code was in part used to generate the above

```
n=[10 20 30 35 40 45 50 55 60 65];
plot(n,log10(n.^5),'ro',n,log10(n.^4.*log10(n)),'*');
```

It can be seen that the cost curves matches those produced with the actual runs, but for a scaling factor as can be expected.

Therefore one can conclude that in 3D SOR is faster than direct solver. This result was surprising as the expectation was that the direct solver will be faster than SOR in 3D as it was in 2D. Attempts were made to find any errors in the code that can explain this, and none were found.

2.4.7 Problem 4

3. In this problem we compare the speed of SOR to a direct solve using Gaussian elimination. At the end of this assignment is MATLAB code to form the matrix for the 2D discrete Laplacian. The code for the 3D matrix is similar. Note that with 1 GB of memory, you can handle grids up to about 1000×1000 in 2D and $40 \times 40 \times 40$ in 3D with a direct solve. The range of grids you will explore depends on the amount of memory you have.

   (a) Solve the PDE from problem 1 using a direct solve. Put timing commands in your code and report the time to solve for a range of mesh spacings. Use SOR to solve on the same meshes and report the time and number of iterations. Comment on your results.

   (b) Repeat the previous part in three spatial dimensions for a range of mesh spacings. Change the right side of the equation to be a three dimensional Gaussian. Comment on your results.

Figure 2.47: Problem 3

Periodic boundary conditions mean that the solution must be such that $u'(0) = u'(1)$ and $u(0) = u(1)$. As an example, the following is a solution to $u''(x) = f(x)$ with Periodic boundary conditions just for illustration

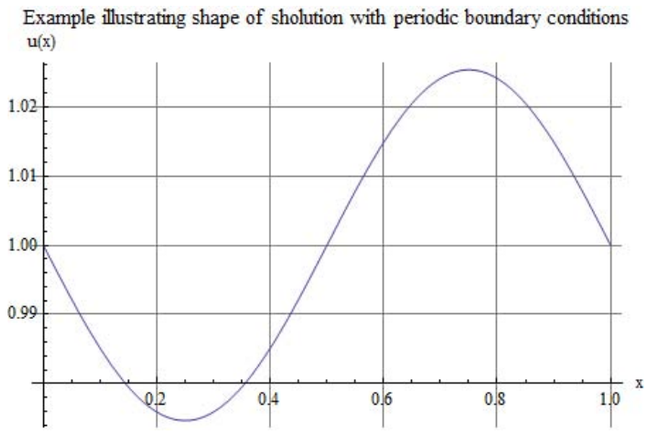


Figure 2.48: example periodic BC

2.4.7.1 part(a)

Using the standard numbering system

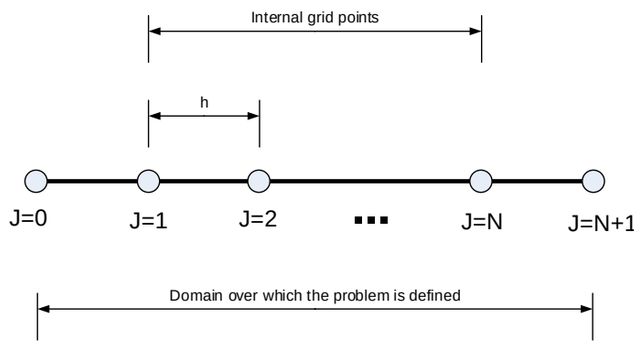


Figure 2.49: problem 4 part a scheme

In the above diagram, u_0 represents u at $x = 0$ and u_{N+1} represents u at $x = 1$. The 3 point discrete Laplacian for 1-D at x_0 is given by

$$u_0'' = \frac{u_{-1} - 2u_0 + u_1}{h^2} \tag{1}$$

where x_{-1} is an imaginary grid point to the left of x_0 in the diagram above.

Expanding u_{-1} about u_0 by Taylor results in $u_{-1} = u_0 - hu_0'$, hence

$$u_0' = \frac{u_0 - u_{-1}}{h} \tag{2}$$

Similarly, by Taylor expansion of u_N about u_{N+1} results in

$$u_N = u_{N+1} - hu_{N+1}'$$

Hence

$$u_{N+1}' = \frac{u_{N+1} - u_N}{h} \tag{3}$$

But $u_0' = u_{N+1}'$ from boundary conditions, hence (2)=(3) which results in

$$\frac{u_0 - u_{-1}}{h} = \frac{u_{N+1} - u_N}{h}$$

Solving now for u_{-1} from the above gives

$$u_{-1} = u_0 + u_N - u_{N+1}$$

But $u_{N+1} = u_0$, also from the boundary conditions, hence the above results in

$$u_{-1} = u_N$$

Use the above value of u_{-1} in (1) gives

$$u_0'' = \frac{u_N - 2u_0 + u_1}{h^2}$$

Similarly the derivation for u''_{N+1} results in

$$u''_{N+1} = \frac{u_N - 2u_{N+1} + u_1}{h^2}$$

For every other internal grid point $i = 1 \cdots N$ the standard 3 point central difference is used

$$u''_i = \frac{1}{h^2}(u_{i-1} - 2u_i + u_{i+1})$$

Therefore, the following set of equations are obtained

$$\begin{aligned}
\tfrac{1}{h^2}(u_N - 2u_0 + u_1) &= f_0 && i = 0 \\
\tfrac{1}{h^2}(u_{i-1} - 2u_i + u_{i+1}) &= f_i && i = 1 \cdots N \\
\tfrac{1}{h^2}(u_N - 2u_{N+1} + u_1) &= f_{N+1} && i = N+1
\end{aligned}$$

And the system can now be put in the form $Au = f$ resulting in

$$\frac{1}{h^2}\begin{pmatrix}
-2 & 1 & 0 & 0 & \cdots & 1 & 0 \\
1 & -2 & 1 & 0 & \cdots & 0 & 0 \\
0 & 1 & -2 & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & -2 & 1 & 0 & \vdots \\
0 & 0 & 0 & \cdots & \ddots & \cdots & 0 \\
0 & 0 & 0 & \cdots & 1 & -2 & 1 \\
0 & 1 & 0 & \cdots & 0 & 1 & -2
\end{pmatrix}\begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_N \\ u_{N+1} \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_N \\ f_{N+1} \end{pmatrix} \tag{4}$$

The above A matrix is singular since $A\boldsymbol{b} = 0$ for $\boldsymbol{b}$ the vector 1^T. Hence the null space of A contains a vector other than the 0 vector meaning that A is singular.

To determine the dimension of the null space, the rank of A is determined. Removing the last column and the last row of A results in an $n-1$ by $n-1$ matrix

$$A_{n-1} = \begin{pmatrix}
-2 & 1 & 0 & 0 & \cdots & 1 \\
1 & -2 & 1 & 0 & \cdots & 0 \\
0 & 1 & \ddots & 1 & 0 & \cdots \\
0 & 0 & 1 & -2 & 1 & 0 \\
0 & 0 & 0 & 1 & -2 & 1 \\
0 & 0 & 0 & \cdots & 1 & -2
\end{pmatrix}$$

The square matrix inside of A_{n-1} that extends from the first row to the one row before the last row is of size $n-2$

$$A_{n-2} = \begin{pmatrix}
-2 & 1 & 0 & 0 & \cdots \\
1 & -2 & 1 & 0 & \cdots \\
0 & 1 & \ddots & 1 & 0 \\
0 & 0 & 1 & -2 & 1 \\
0 & 0 & 0 & 1 & -2
\end{pmatrix}$$

And this matrix is a full rank as it is the A matrix used for 1-D with Dirichlet boundary conditions and this matrix is known to be invertible (same one used in HW2).

Therefore, the rank of A can not be less than $n-2$ where n is the size of A.

In other words, the size of the null space of A can at most be 2. To determine if the size of the null space of A can be just one, the matrix A_{n-1}shown above has to be invertible as well.

One way to show that A_{n-1} is invertible, is to show that the last column of A_{n-1} is linearly independent to any of the remaining columns of A_{n-1}.

The last column of A_{n-1} is $c_{n-1}\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 1 \\ -2 \end{pmatrix}$ and this column is linearly independent with the first

column of A_{n-1} which is $c_1 = \begin{pmatrix} -2 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$ since $a \times c_{n-1} + b \times c_1 = 0$ only when a, b are both zero. The same can be shown with all the other columns of A_{n-1}, they are all linearly independent to the last columns of c_{n-1}. Since all the other $n-2$ columns of A_{n-1} are linearly independent with each others (they make up the Dirichlet matrix known to be invertible) then A_{n-1} is invertible. This shows that the rank of A is $n-1$, hence the null space of A has dimension 1 only. In other words, only the and only the vector 1^T in the null of A. (Since A is symmetric, the null space of the adjoint of A is the same).

2.4.7.2 part (b)

In terms of looking at the conditions of solvability, the continuous case is considered and then the conditions are translated to the discrete case.

The pde $u''(x) = f(x)$ with periodic boundary conditions has an eigenvalue which is zero (the boundary conditions $u'(1) = u'(0)$ results in this), hence

$$0 = \int_0^1 f(x)\, dx$$

Is the solvability condition which results from the $u'(1) = u'(0)$ boundary conditions. (same argument that was carried out in part (d), problem 1, HW1 which had Neumann boundary conditions is used). Now, what solvability conditions does $u(0) = u(1)$ add to this if any?

Since

$$u''(x) = f(x)$$

Then integrating once gives

$$u'(1) - u'(0) = \int_0^1 f(x)\, dx + C_1$$

But since $u'(1) = u'(0)$, then the above implies that

$$0 = C_1$$

And integrating twice the PDE results in

$$u(1) - u(0) = C_2$$

But $u(1) - u(0) = 0$, hence $C_2 = 0$. So the only solvability condition is based on the fact that an eigenvalue is zero, which implies

$$\int_0^1 f(x)\, dx = 0$$

This is the same as was the case with Neumann boundary conditions. In the discrete case, this implies that solvability condition becomes the discrete integration (Riemman sum)

$$h \sum_{j=0}^{j=N+1} f\left(jh\right) = 0$$

For 2D, by extension of the above, there will be 2 eigenvalues of zero values, hence the discrete solvability condition becomes

$$h^2 * \sum_{i=0}^{i=N+1} \left(\sum_{j=0}^{j=N+1} f\left(ih, jh\right) \right) = 0$$

2.4.7.3 Part(c)

Since this is an iff problem, then the following needs to be shown

1. If v is in the null space of A then v is an eigenvector of T with eigenvalue 1
2. If v is an eigenvector of T with eigenvalue 1 then v is in the null space of A

Solving part(1)

Since v is in null space of A, then by definition

$$Av = 0$$

But $A = M - N$, hence the above becomes

$$(M - N)v = 0$$
$$Mv - Nv = 0$$

Since M is invertible by definition, then M^{-1} exists. Premultiply both sides by M^{-1}

$$M^{-1}Mv - M^{-1}Nv = M^{-1}0$$

But $M^{-1}0 = 0$ then the above becomes

$$Iv - M^{-1}Nv = 0$$
$$M^{-1}Nv = v$$
$$Tv = v$$

Therefore v is an eigenvector of T with an eigenvalue of 1.

Solving part(2)

Since v is an eigenvector of T with eigenvalue 1 then

$$Tv = \lambda v$$

With $\lambda = 1$, and since $T = M^{-1}N$, then the above becomes

$$M^{-1}Nv = v$$

Multiply both sides by M

$$Nv = Mv$$

Therefore

$$Mv - Nv = 0$$
$$(M - N)v = 0$$

Hence

$$Av = 0$$

Therefore v is in the null space of A.

2.4.8 References

1. Applied Mathematica by David Logan, chapter 8

2.4.9 Source code

```
function nma_build_HW3()

list = dir('*.m');

if isempty(list)
    fprintf('no matlab files found\n');
    return
end

for i=1:length(list)
    name=list(i).name;
    fprintf('processing %s\n',name)
    p0 = fdep(list(i).name,'-q');
    [pathstr, name_of_matlab_function, ext] = fileparts(name);

    %make a zip file of the m file and any of its dependency
    p1=dir([name_of_matlab_function '.fig']);
    if length(p1)==1
        files_to_zip =[p1(1).name;p0.fun];
    else
```

```
        files_to_zip =p0.fun;
    end

    zip([name_of_matlab_function '.zip'],files_to_zip)

end

end
```

```
function nma_cpu_plot_2D()
%
% to plot O(n^4) vs. O(n^3 log(n))
% used to verify the cost complexity for problem 3, part (b), HW3
% Math 228A
% Nasser M. Abbasi
%

close all;
n=10:1:40;
plot(n,n.^2,'r',n,n.^3.*log(n));
legend('direct solver','sor')

figure
close all;
n=10:1:4000;
plot(n,n.^(5/2),'r',n,n.^(3/2).*log(n));
legend('direct solver','sor')

figure
close all;
n=[10 20 30 35 40 45 50 55 60 65];
plot(n,log10(n.^5),'ro',n,log10(n.^4.*log10(n)),'*');
legend('direct solver','sor')
hold on
n=[10 20 30 35 40 45 50 55 60 65];
plot(n,log10(n.^5),'r-',n,log10(n.^4.*log10(n)),'-');
title('Verifying theortical complexity cost of direct 3D sparse solver');
xlabel('n, number of unknowns in one dimension');
ylabel('log10(cost)');

end
```

```
function  nma_HW3_prob3_parta_SOR()
% file name: nma_HW3_prob3_parta_SOR.m
%
% This solves the SOR for 2D for HW3, problem 3, parta
% Math 228A, Fall 2010, UC Davis
% Nasser M. Abbasi
%

DOPLOTS = true; %set to false if do not want to see plots

% setup the spacings needed for the problem
%spacings = [2^-5 2^-6 2^-7 2^-8 2^-9 2^-10];
spacings = [2^-5 2^-6 2^-7 ];
omega    = arrayfun(@(i) 2*(1-pi*spacings(i)),1:length(spacings));

for m = 1:length(spacings)

    h = spacings(m);

    % evaluate f(x,y) on grid
```

```
    [X,Y] = meshgrid(0:h:1, 0:h:1);
    f     = -exp(-(X-0.25).^2-(Y-0.6).^2);
    nPoints = size(f,1);
    fv    = reshape(f,nPoints^2,1); % use grid vector norm
    normf = sqrt(h) * norm(fv,2);

    w = omega(m);

    fprintf('********************\n');
    fprintf('gridsize=[%d,%d]\n',nPoints-2,nPoints-2);

    % initialize space (grid) for residual calculation and for solution
    resid = zeros(nPoints,nPoints);
    u     = zeros(nPoints,nPoints);
    unew  = u;

    done      = false; %flag set to true in loop below when it converges
    tolerance = h^2;   % set tolerance
    k         = 1;     % initialize iteration counter

    t         = cputime;

    while not(done)
        for i = 2 : nPoints-1
            for j = 2 : nPoints-1

                resid(i,j)= f(i,j) - 1/h^2 * ( u(i-1,j) + u(i+1,j) + ...
                    u(i,j-1) + u(i,j+1) - 4*u(i,j) );

                unew(i,j) = w/4* ( unew(i-1,j) + u(i+1,j) + unew(i,j-1)+...
                    u(i,j+1) - h^2*f(i,j)) + (1-w)*u(i,j);
            end
        end

        u      = unew;
        residv = reshape(resid,nPoints^2,1); % use grid vector norm
        normResidue = sqrt(h) * norm(residv,2);

        if ( normResidue / normf) <tolerance
            done = true;
        else
            k = k+1;
        end

        if DOPLOTS
            subplot(1,2,1);
            mesh(X ,Y ,resid );

            subplot(1,2,2);
            mesh(X ,Y ,u );

            drawnow;
        end

    end


    fprintf('cpu time for SOR=%f seconds\n',cputime-t);
    fprintf('number of iterations = %d\n',k);

end

end
```

```
function nma_HW3_prob3_partb_3d_SOR()

% file name: nma_HW3_prob3_partb_3d_SOR.m
%
% This does the SOR solver for 3D for HW3, problem 3, partB
% Math 228A, Fall 2010, UC Davis
% SOR 3D solver
%
% Nasser M. Abbasi
%

DOPLOTS = true; %set to false if do not want to see plots

% setup the spacings needed for the problem
%gridsize = [10 20 30 35 40 45 50 55 60 65];
gridsize = [10 20 ];
spacings = arrayfun(@(i) 1/(gridsize(i)+1),1:length(gridsize));
omega    = arrayfun(@(i) 2*(1-pi*spacings(i)) ,1:length(spacings));

fprintf('----- Starting 3D SOR solver ---------\n');

for m = 1:length(spacings)

    h = spacings(m);
    nInternalGridPoints = gridsize(m);
    nPoints = nInternalGridPoints+2;

    fprintf('*******************************\n');
    fprintf('grid is 3D [%d,%d,%d]\n',nInternalGridPoints, ...
        nInternalGridPoints,nInternalGridPoints);

    fprintf('h=%f\n',h);

    [X,Y,Z] = meshgrid(0:h:1, 0:h:1,0:h:1);

    % initialize space (grid) for residual calculation and for solution
    u       = 0.*X+0.*Y+0.*Z;
    unew    = u;
    resid   = u;

    % evaluate f(x,y,z) on grid
    f       = -exp(-(X-0.25).^2-(Y-0.6).^2 - Z.^2);
    normf   = sqrt(h)* norm( reshape(f(2:end-1,2:end-1,2:end-1),...
        nInternalGridPoints^3,1),2);

    w         = omega(m); %optimal w for SOR
    done      = false; %flag set to true in loop below when it converges
    tolerance = 0.1*h^2;   % set tolerance
    k         = 1;     % initialize iteration counter

    t         = cputime;

    while not(done)
        for i = 2 : nPoints-1
            for j = 2 : nPoints-1
                for z = 2 : nPoints-1
                    resid(i,j,z)= f(i,j,z) - 1/h^2 * ( u(i-1,j,z) + ...
                        u(i+1,j,z) + u(i,j-1,z) + u(i,j+1,z) - ...
                        6*u(i,j,z) + u(i,j,z-1) + u(i,j,z+1));

                    unew(i,j,z) = w/6* ( unew(i-1,j,z) + u(i+1,j,z) + ...
                        unew(i,j-1,z) + u(i,j+1,z) + unew(i,j,z-1) + ...
                        u(i,j,z+1)- h^2*f(i,j,z)) + (1-w)*u(i,j,z);
```

```
                end
            end
        end

        u = unew;
        if DOPLOTS
            subplot(1,2,1);
            mesh(X(:,:,nPoints-1),Y(:,:,nPoints-1),resid(:,:,nPoints-1));

            subplot(1,2,2);
            mesh(X(:,:,nPoints-1),Y(:,:,nPoints-1),u(:,:,nPoints-1));

            drawnow;
        end

        % can't do norm on 3D, change to vector
        residv = reshape(resid(2:end-1,2:end-1,2:end-1),...
            nInternalGridPoints^3,1);
        normResidue = sqrt(h) * norm(residv,2);

        if (normResidue/normf) < tolerance
            done = true;
        else
            k = k+1;
        end

    end

    fprintf('cpu time for 3D SOR solver =%f seconds\n',cputime-t);
    fprintf('number of iterations = %d\n',k);
end

end
```

```
function nma_HW3_problem_3_part_A_graph_plot()
% This   used to generate plot to compare CPU time of SOR
% and direct solver for 2D problem
% file name nma_HW3_problem_3_part_A_graph_plot.m

close all;
x=[31 63 127 255 511 1023];
directCPU=[0.015 .125 .250 1.544 5.538 27.113];
sorCPU=[0 0.094 0.6 5.2 48 532];

%plot(x,log10(directCPU),':.',x,log10(sorCPU));
plot(x,directCPU,'-',x,sorCPU);

title('compring CPU time against grid size, 2D pde solver');
xlabel('grid size n');
%ylabel('log10 of CPU time in seconds');
ylabel('CPU time in seconds');
hold on;
legend('direct solver','SOR','Location','NorthWest');

grid on
%plot(x,log10(directCPU),'o',x,log10(sorCPU),'o');
plot(x,directCPU,'o',x,sorCPU,'o');
ylim([-10 550]);

end
```

```
function nma_HW3_problem_3_part_B_graph_plot()
```

```
% This  used to generate plot to compare CPU time of SOR
% and direct solver for 3D problem
% file name nma_HW3_problem_3_part_B_graph_plot.m
% Nasser M. Abbasi
% 11/8/2010

x=[10 20 30 35 40 45 50 55 60];
directCPU=[0.047 0.5 3.9 8.8 21.50 40 84 157.7 244.4];
sorCPU=[0.01 0.078 0.405 0.75 1.29 2.11  3.24  4.9  7.17];

plot(x,log10(directCPU),':.',x,log10(sorCPU));
%plot(x,directCPU,'-',x,sorCPU);

title('compring CPU time against grid size, 3D pde solver');
xlabel('grid size n');
ylabel('log10 of CPU time in seconds');
%ylabel('CPU time in seconds');
hold on;
legend('direct solver','SOR','Location','NorthWest');

grid on
plot(x,log10(directCPU),'o',x,log10(sorCPU),'o');
%plot(x,directCPU,'o',x,sorCPU,'o');

end
```

```
function nma_HW3_problem_3_partA()
%
%  name: nma_HW3_problem_3_partA.m
%  purpose: direct solver for HW3, problem 3, part A. 2D
%  UC Davis math 228A
%
%  description of algorithm:
%    This script when called, will find the solution to the problem
%    Au=f as described in the HW, by using sparse matrix and direct
%    solver.  The script will solve the problem for the following h
%    spacings: 2^-5 2^-6 2^-7 2^-8 2^-9 2^-10
%
%    It will find the cpu time used and print to the screen the result
%    for each grid space.
%
% external functions called:
%    This scripts makes calls to nma_lap2d() to geberate
%    the sparse matrices.
%
% date written: 11/5/2010
% by: Nasser M. Abbasi
%

close all; clear all;

DOPLOTS  = true; %set to false if do not want to see plots
%spacings = [2^-5 2^-6 2^-7 2^-8 2^-9 2^-10];
spacings = [2^-5 2^-6 2^-7 ];

gridSize = arrayfun(@(i) 1/spacings(i)-1,1:length(spacings));

for i = 1:length(spacings)

    h = spacings(i);
    n = gridSize(i);

    fprintf('n=%d\n',n);
```

```
38
39      % evaluate f(x,y) on grid and convert to vector
40      [X,Y] = meshgrid(h:h:1-h, h:h:1-h);
41      f     = -exp(-(X-0.25).^2-(Y-0.6).^2);
42      f     = reshape(f,n^2,1);
43
44      t = cputime;
45      A = nma_lap2d(n,n)./h^2;  %make the A matrix
46      fprintf('cpu time for making sparse matrix=%f seconds\n',cputime-t);
47      fprintf('nonzero elements=%d\n',nnz(A));
48
49
50      t = cputime;
51      u = A\f;
52      fprintf('cpu time for direct solver=%f seconds\n',cputime-t);
53
54      % plot solution if needed
55      if DOPLOTS
56          u=reshape(u,n,n);
57          mesh(u);
58          title(sprintf('2D solution n=%d',n));
59          drawnow;
60      end
61
62  end
63
64  end
```

```
1   function mma_HW3_problem_3_partB_direct_solver()
2   %
3   %  script file, name: mma_HW3_problem_3_partB_direct_solver.m
4   %
5   %  purpose: direct solver for HW3, problem 3, part B (3D).
6   %  UC Davis math 228A
7   %
8   %  description of algorithm:
9   %    This script when called, will find the solution to the problem
10  %    Au=f as described in the HW, by using sparse matrix and direct
11  %    solver.  The script will solve the problem for the following h
12  %    spacings: 2^-3, 2^-4, 2^-5 or grid size n=7,15,31
13  %
14  %    It will find the cpu time used and print to the screen the result
15  %    for each grid space.
16  %
17  % external functions called:
18  %    This makes calls to nma_lap3d() to generate
19  %    the sparse matrices.
20  %
21  % date written: 11/5/2010
22  % by: Nasser M. Abbasi
23  %
24  close all; clear all;
25  %gridsize = [10 20 30 35 40 45 50 55 60 65];
26  gridsize = [10 20 30];
27  spacings = arrayfun(@(i) 1/(gridsize(i)+1),1:length(gridsize));
28
29  fprintf('----- Starting 3D direct solver ---------\n');
30
31  DOPLOTS=true;  %set to false if do not see plot of solution
32
33  for i = 1:length(spacings)
34
35      h = spacings(i);
```

```
    n = gridsize(i);

    fprintf('******************************\n');
    fprintf('grid is 3D [%d,%d,%d]\n',n,n,n);
    fprintf('h=%f\n',h);

    % evaluate f(x,y,z) on grid and convert to vector
    [X,Y,Z] = meshgrid(h:h:1-h, h:h:1-h,h:h:1-h);
    f     = -exp(-(X-0.25).^2-(Y-0.6).^2 - Z.^2);
    f     = reshape(f,n^3,1);

    t = cputime;
    A = nma_lap3d(n)./h^2; % make the A matrix, sparse
    fprintf('cpu time for making sparse matrix=%f seconds\n',cputime-t);
    [nRow,nCol]=size(A);
    fprintf('dimensions of A (sparse matrix) is [%d,%d]\n',nRow,nCol);
    fprintf('nnz(A)= %d\n',nnz(A));

    t = cputime;
    u = A\f;
    fprintf('cpu time for direct solver=%f seconds\n',cputime-t);

    % plot solution if needed
    if DOPLOTS
        u=reshape(u,n,n,n);
        mesh(u(:,:,n-1));
        title(sprintf('3D solution, top surface only, n=%d',n));
        hold on;
    end

end

end
```

```
function nma_nnz_estimate()
% to estimate nnz() as function of n for 2D sparse matrix
% Nasser M. Abbasi
% HW3 math 228A

clear all;
close all;
n=[3 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 200 300 400 500 600];
y=arrayfun(@(i) nnz(lap2d(n(i),n(i))),1:numel(n));
plot(n,y,'r');
hold on;
plot(n,n.^2.25);
title('estimating nnz order for 2D sparse matrix');
xlabel('n, number of grid points in one dimension');
ylabel('nnz, number of non-zero elements');
legend('Matlab nnz()','n^2.2')

end
```

```
function nnz_estimate_3D()
% file name  nnz_estimate_3D.m
% to estimate nnz() as function of n for 3D sparse matrix
% Nasser M. Abbasi
% HW3, Math 228A

clear all;
close all;
n=[3 10 20 30 40 50 60 70 80 100 150 200];
```

```
y=arrayfun(@(i) nnz(lap3d(n(i))),1:numel(n));
plot(n,y,'r');
hold on;
plot(n,n.^3.35);
title('estimating nnz order for 3D sparse matrix');
xlabel('n, number of grid points in one dimension');
ylabel('nnz, number of non-zero elements');
legend('Matlab nnz()','n^3.35')

end
```

2.5 HW 4

2.5.1 Problem 1

1. Write a multigrid V-cycle code to solve the Poisson equation in two dimensions on the unit square with Dirichlet boundary conditions. Use full weighting for restriction, bilinear interpolation for prolongation, and red-black Gauss-Seidel for smoothing.

   Note: If you cannot get a V-cycle code working, write a simpler code such as a 2-grid code. You can also experiment in one dimension (do not use GSRB in 1D). You may turn in one of these simplified codes for reduced credit. You should state what your code does, and use your code for the second problem of this assignment.

Figure 2.50: Problem 1

The multigrid V cycle algorithm was implemented in Matlab 2010a. The documented source code is included in the appendix of this problem.

For relaxation, Gauss-Seidel with red-black (GSRB) ordering was used as the default. A relax() function was written to implement this method, in addition, this function can also implement relaxation using these solvers: Jacobi, Gauss-Seidel Lex, and SOR. Selecting the relaxation method is done via an argument option. These different methods are implemented in the relax() function for future numerical experimentation. GSRB is the one used for all the solutions below as required by the problem statement and is the default method. GSRB is known to have good smoothing rates and is suitable for parallelism as well.

For mapping from the fine mesh to coarse mesh, the full weighting method is used.

For mapping from coarse mesh to fine mesh, bilinear interpolation is used.

Additional auxiliary functions are written for performing the following: finding the norm (mesh norm), finding the residue and validating dimensions of the input.

The following diagram illustrates the call flow chart for a program making a call to the V cycle algorithm, such as the program written to solve problem 2 and 3. It shows the Matlab functions used, and the interface between them. This flow diagram also shows a full multigrid solver (FMG) function, which was implemented on top of the V cycle algorithm, but was not used to generate the results in problem 2 as the problem asked to use V cycle algorithm only. FMG cycle algorithm was implemented for future reference and to study its effect on reducing number of iterations. A note on this is can be found the end of this assignment.

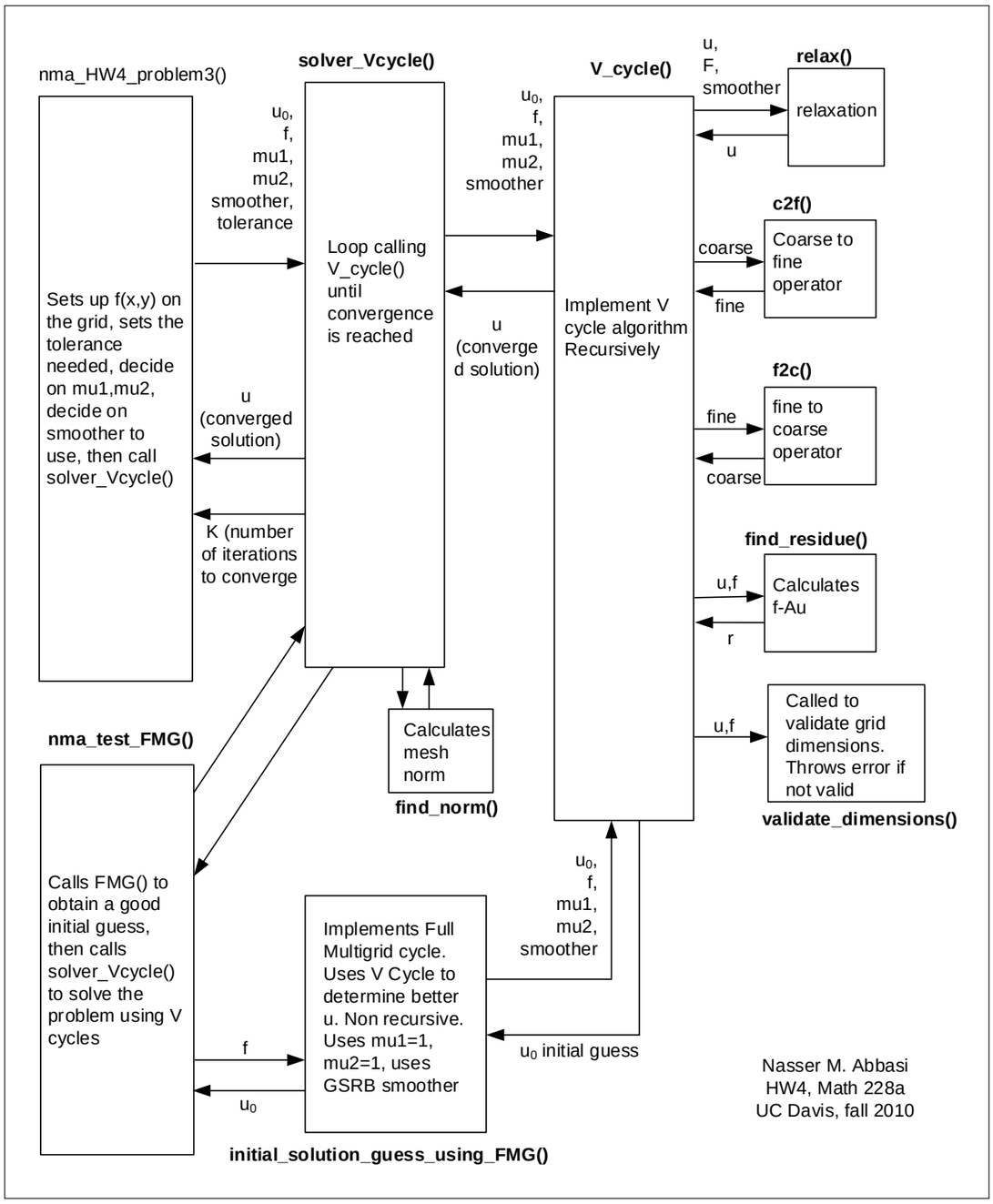


Figure 2.51: algorithm flow chart

2.5.1.1 Restriction and prolongation operators

The restriction operator I_h^{2h} (fine to coarse mesh) mapping uses full weighting, while the prolongation operator I_{2h}^{h} (coarse to fine mesh) uses bilinear interpolation.

For illustration, the following diagram shows applying these operators for the 1D case for a mesh of 9 points. The edge points are boundary points and in this problem (Dirichlet homogeneous boundary conditions), these will always be zero.

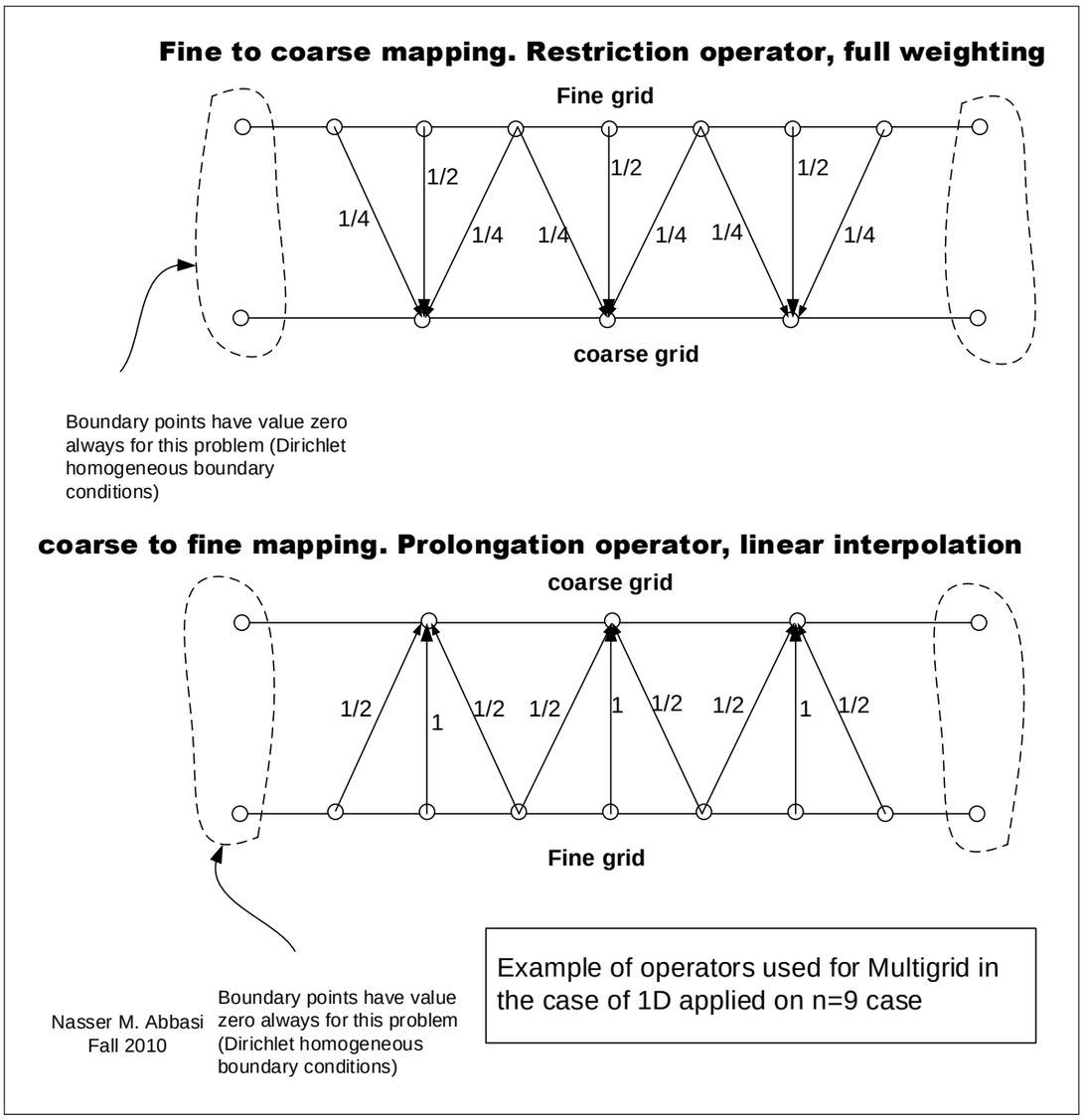


Figure 2.52: operator diagrams

2.5.1.2 V cycle algorithm

The multigrid V cycle algorithm is recursive in nature. The following is description of the algorithm

```
VCYCLE algorithm
----------------
input:  u, f, mu1, mu2
output: u (more accurate u)

Let n be the number of grid points of u in one dimension

IF n = 3 THEN
   find u by direct solution of 3x3 grid
ELSE
   apply mu1 smoothing on u
   residue = find residue (f-Au)
   residue = apply fine-to-coarse mapping on residue
   correction = CALL VCYCLE(ZERO,residue,mu1,mu2)
   correction = apply coarse-to-fine mapping on correction
   u = u + correction
   apply mu2 smoothing on u
END IF

RETURN u
```

.

Problem 2 below also has a diagram which helps understand this algorithm more. The Matlab function shown below implements the above algorithm.

2.5.1.3 Multigrid V cycle function (V_cycle.m)

This function implements one multigrid V cycle. It is recursive function

2.5.1.4 Solver using V cycle (solver_Vcycle.m)

This function is an interface to V cycle algorithm to use for solving the 2D Poisson problem. It uses V_cycle.m in a loop until convergence is reached.

2.5.1.5 Relaxation or smoother function (relax.m)

This function implements Gauss-Seidel red-black solver. It is a little longer than needed as it also implements other solvers as was mentioned in the introduction. These are added for future numerical experimentation. The

algorithm is straight forward. If the sum of the row and column index adds to an even value, then the grid point is considered a red grid point, else it is black. The red grid points are smoothed first, then the black grid points are smoothed.

2.5.1.6 Find residual function (find_residue.m)

This function is called from a number of locations to obtain the residue mesh. The residue is defined as

$$r_{i,j} = f_{i,j} - \frac{1}{h^2}\left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}\right)$$

2.5.1.7 Find norm function (find_norm.m)

This function is called from number of locations to obtain the norm of the mesh or any 2D matrix.

2.5.1.8 Validate boundary conditions (check_all_zero_boundaries.m)

An auxiliary function used by a number of functions to validate that input has consistent boundaries for this problem.

2.5.1.9 Coarse to fine prolongation operator 2D (c2f.m)

This function is the prolongation bilinear interpolation which implements coarse to fine grid mapping on 2D grid.

2.5.1.10 Fine to coarse full weight restriction operator 2D (f2c.m)

This function is the full weight restriction operator which implements the fine grid to coarse grid mapping on 2D grid.

2.5.1.11 Validate u and f have consistent dimensions(validate_dimensions.m)

An auxiliary function used by number of other function to validate that input dimensions are consistent.

2.5.1.12 Validate grid for consistent dimensions (validate_dimensions_1.m)

An auxiliary function used by number of other function to validate that a grid dimensions are consistent.

2.5.1.13 FMG solver (initial_solution_guess_using_FMG.m)

Implements a full multigrid cycle using V cycle algorithm as building block. Used to compare effect on solution only.

2.5.1.14 Restriction operator for 1D (f2c_1D.m)

This function is the full weight restriction operator which implements the fine grid to coarse grid mapping on 1D grid

2.5.1.15 Prolongation operator for 1D (c2f_1D.m)

This function is the prolongation bilinear interpolation which implements coarse to fine grid mapping on 1D grid

2.5.2 Problem 2

2. Numerically estimate the average convergence factor,

$$\left(\frac{\|e^{(k)}\|_\infty}{\|e^{(0)}\|_\infty}\right)^{1/k},$$

for different numbers of presmoothing steps, ν_1, and postsmoothing steps, ν_2, for $\nu = \nu_1 + \nu_2 \leq 4$. Be sure to use a small value of k because convergence may be reached very quickly. What test problem did you use? Report the results in a table, and discuss which choices of ν_1 and ν_2 give the most efficient solver.

Figure 2.53: Problem 2

The test problem used is

$$\nabla u = 0$$

with zero boundary conditions on the unit square. This has a known solution which is zero.

Initial guess is a random solution generated using matlab rand(). Hence $\left\|e^{(0)}\right\|$ has the same norm as the initial guess.

The V Cycle function written for problem 1 was used to generate the average convergence factor. For each combination of ν_1, ν_2, a table was generated which contained the following columns:

1. Cycle number.
2. The norm of the residue $\left\|r^{(k)}\right\| = \left\|f - Au^{(k)}\right\|$ after each cycle.
3. Ratio of the current residue norm to the previous residue norm $\frac{\left\|r^{(k+1)}\right\|}{\left\|r^{(k)}\right\|}$.
4. Error norm $\left\|e^{(k)}\right\| = \left\|u - u^{(k)}\right\| = \left\|u^{(k)}\right\|$ (since exact solution is zero).
5. The ratio of the current error norm to the previous error norm $\frac{\left\|e^{(k+1)}\right\|}{\left\|e^{(k)}\right\|}$.
6. The average convergence factor up to each cycle $\left(\frac{\left\|e^{(k)}\right\|_\infty}{\left\|e^{(0)}\right\|_\infty}\right)^{\left(\frac{1}{k}\right)}$.

The problem asked to generate result for $v \leq 4$. This solution extended this to $v \leq 8$ to use the results for future study if needed. The summary table below shows the final result for $k = 15$. Each individual table generated for each combination of ν_1, ν_2 is listed in the appendix of this problem. The function HW4_problem2() was used to generate these tables and to calculate the work done for each solver combination of ν_1, ν_2.

2.5.2.1 Average convergence factor and work unit estimates

To determine the most efficient solver, the amount of work by each solver that achieves the same convergence is determined. The solver with the least amount of work is deemed the most efficient. The total amount of work for convergence is defined as

$$\text{WORK=number of Iterations for convergence } \times \text{work per iteration} \tag{1}$$

An iteration is one full V cycle. Each V cycle contains a number of levels. The same number of levels exist on the left side of the V cycle as on the right side of the V cycle. On the left side of the V cycle, work at each level consist of the following items

1. Work to perform ν_1 number of pre-smooth operations.
2. Work needed to map to the next lower level of the grid.
3. Work needed to compute the residue.

The following diagram helps to illustrates this.

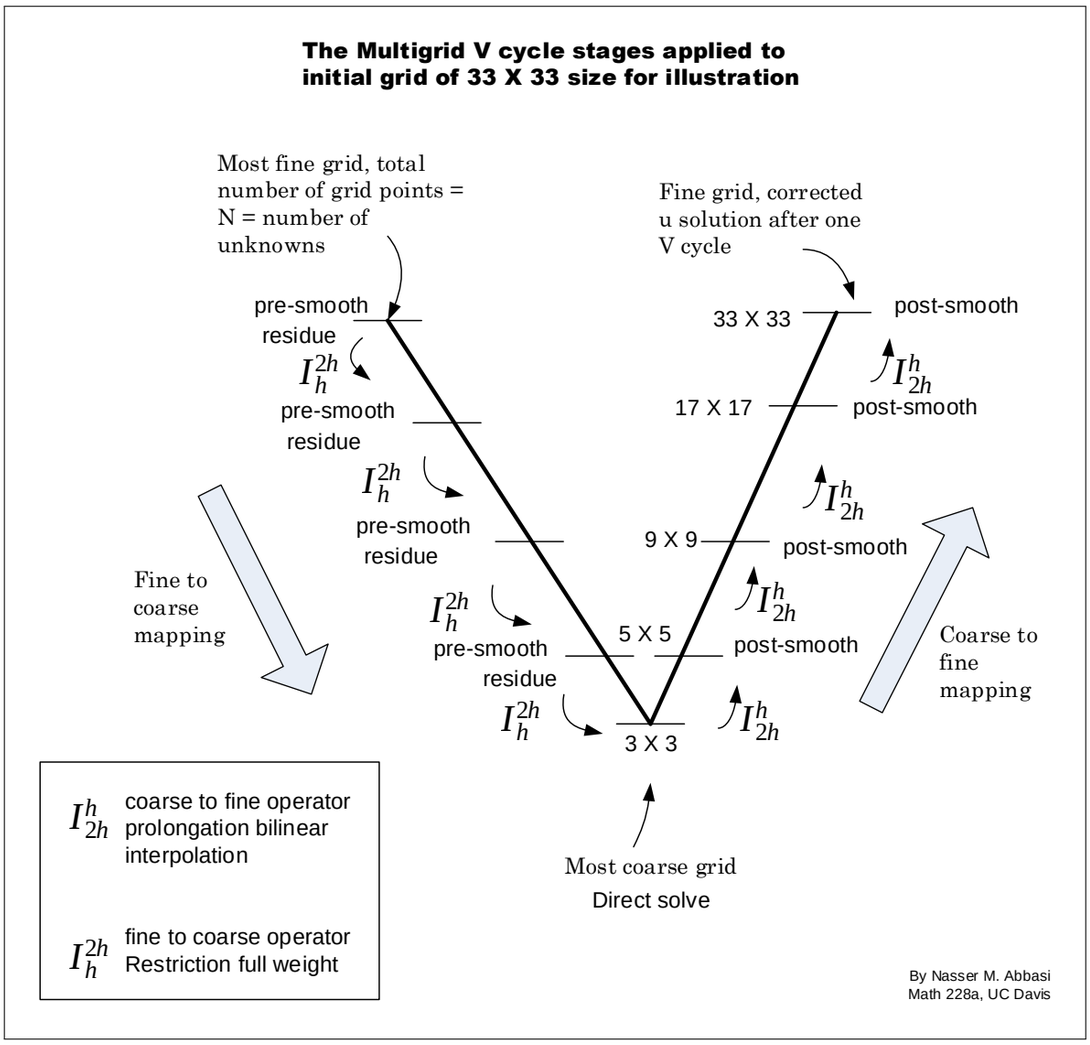


Figure 2.54: problem 2 v cycle shape

On the right side of the V cycle, work at each level consist of the following items

1. Work to perform ν_2 number of post-smooth operations.
2. Work needed to map to the next lower level of the grid.

At each level, the work is proportional to the size of the grid at that level. For smoothing, it is estimated that 7 flops are needed to obtain an average of each grid point. (5 additions, one multiplication, one division) based on the use of the following formula

$$u_{i,j} = \frac{1}{4}\left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - h^2 f\right)$$

Therefore work needed for smoothing is $7\times(v_1 + v_2)\times N$ where N is the total number of grid points at that level. (Boundary grid points are not involved in this work, but for simplicity of analysis, the total number of grid points N is used).

Work needed for finding the residual is also about $7N$. Work needed for mapping to the next grid level is about $6N$.

On the right branch of the cycle no residual calculation is required. To simplify the analysis, it is assumed that the same work is performed at each level on both sides of the V cycle.

Therefore, Letting N be the number of grid points at the most fine level (the number of unknowns), the total work per cycle is found to be

$$\begin{aligned}\textit{work per V cycle} &= (7(v_1 + v_2) + 13)N + (7(v_1 + v_2) + 13)\frac{N}{4} + (7(v_1 + v_2) + 13)\frac{N}{16} + \cdots \\ &= (7(v_1 + v_2) + 13)N\left[1 + \frac{1}{4} + \frac{1}{4^2} + \cdots + \frac{1}{4^{L-1}}\right]\end{aligned}$$

Where L is the number of levels. In the limit as L becomes very large, this becomes a geometric series whose sum is $\frac{(7(v_1+v_2)+13)N}{1-r}$ where $r = \frac{1}{4}$

Therefore, total amount of work from (1) becomes

$$W = M \times \frac{4}{3}\left(7\left(v_1 + v_2\right) + 13\right) N$$

Where M is number of iterations. Using the same tolerance ε for all solvers, $M = \frac{\log(\varepsilon)}{\log(\rho)}$. *Using the average convergence rate found as an estimate for the spectral radius* ρ, W can now be found as a function of N

$$W = \left(\frac{\log\left(\varepsilon\right)}{\log\left(\rho\right)}\right)\left(\frac{4}{3}\left(7\left(v_1 + v_2\right) + 13\right) N\right)$$

For the purpose of comparing the different solvers, the value of ε used is not important as long as it is the same value in all cases. Hence, for numerical computation, let $\varepsilon = 10^{-6}$ and the above becomes

$$W = \left(\frac{-6}{\log\left(\rho\right)}\right)\left(\frac{4}{3}\left(7\left(v_1 + v_2\right) + 13\right) N\right)$$

The program written for this problem uses the above equation to calculate the work done for each solver. The result is shown below. This table shows the work done by each solver for convergence based on the same tolerance. As mentioned above, changing the tolerance value will not change the result, as the effect will be to scale all result by the same amount.

2.5.2.2 Result

$v = (v_1, v_2)$	$v_1 + v_2$	$CF = \left(\frac{\left\|e^{(k)}\right\|_\infty}{\left\|e^{(0)}\right\|_\infty}\right)^{\left(\frac{1}{k}\right)}$	work $\frac{-6}{\log(\rho)} \times \frac{4}{3}\left(7\left(v_1 + v_2\right) + 13\right) N$
(0,1)	1	0.374588	213 N
(0,2)	2	0.202747	177 N
(0,3)	3	0.132526	174.9 N
(0,4)	4	0.098857	182.7 N
(1,0)	1	0.323688	182.4 N
(1,1)	2	0.116811	129.4 N
(1,2)	3	0.079578	138.4 N
(1,3)	4	0.060307	150.5 N
(1,4)	5	0.049019	164.1 N
(2,0)	2	0.171973	158.5 N
(2,1)	3	0.079845	138.6 N
(2,2)	4	0.060424	150.6 N
(2,3)	5	0.049068	164.2 N
(2,4)	6	0.041573	178.4 N
(3,0)	3	0.117023	163.6 N
(3,1)	4	0.060444	150.6 N
(3,2)	5	0.049072	164.2 N
(3,3)	6	0.041575	178.4 N
(3,4)	7	0.036128	192.6 N
(4,0)	4	0.088624	174.6 N
(4,1)	5	0.049075	164.2 N
(4,2)	6	0.041575	178.4 N
(4,3)	7	0.036128	192.6 N
(4,4)	8	0.031908	206.7 N

2.5.2.3 Conclusion

From the above result, The least work was for the (1,1) solver, followed by (1,2) which had about the same as the (2,1) solver. This result shows that using $v = 2$ or $v = 3$ *is the most*

efficient solver in terms of least work required.

Notice that in full multigrid, the combination which makes up the value v is important (While for the case of the 2 level multigrid, this is not the case). For example, as shown in the above table, work for solver $v = (1,2)$ was 138 N while work for solver $v = (3,0)$ was 163 N even though they both add to same total number of smooth operations $v = 3$.

2.5.2.4 Appendix. Tables for each combination of $\nu = (\nu_1, \nu_2)$

The following tables are the result of running problem 2 program on the test problem. The fields for each table are described above. The last row in each table contain the result for $k = 15$. The value of the average convergence factor used is that for $k = 15$ under the column heading convergence factor. This below is link to the text file containing the tables as they are printed by the matlab function.

Result for $v_1 = 0, v_2 = 0$

cycle	|residue|	ratio	|error|	ratio	convergence factor
1	4.197646e+004	0.000000	3.791167e+000	0.000000	0.836431
2	4.197672e+004	1.000006	3.618870e+000	0.954553	0.893542
3	4.197682e+004	1.000002	3.571746e+000	0.986978	0.923661
4	4.197685e+004	1.000001	3.557261e+000	0.995944	0.941224
5	4.197686e+004	1.000000	3.552573e+000	0.998682	0.952445
6	4.197687e+004	1.000000	3.551027e+000	0.999565	0.960141
7	4.197687e+004	1.000000	3.550514e+000	0.999856	0.965717
8	4.197687e+004	1.000000	3.550344e+000	0.999952	0.969931
9	4.197687e+004	1.000000	3.550287e+000	0.999984	0.973225
10	4.197687e+004	1.000000	3.550268e+000	0.999995	0.975870
11	4.197687e+004	1.000000	3.550262e+000	0.999998	0.978039
12	4.197687e+004	1.000000	3.550260e+000	0.999999	0.979850
13	4.197687e+004	1.000000	3.550259e+000	1.000000	0.981386
14	4.197687e+004	1.000000	3.550259e+000	1.000000	0.982704
15	4.197687e+004	1.000000	3.550259e+000	1.000000	0.983848

.

Result for $v_1 = 0, v_2 = 1$

cycle	|residue|	ratio	|error|	ratio	convergence factor
1	6.512884e+003	0.000000	1.841085e+000	0.000000	0.406191
2	1.096932e+003	0.168425	7.221287e-001	0.392230	0.399150
3	2.518487e+002	0.229594	2.950874e-001	0.408636	0.402287
4	6.501181e+001	0.258138	1.172566e-001	0.397362	0.401050
5	1.807445e+001	0.278018	4.592625e-002	0.391673	0.399157
6	5.295406e+000	0.292977	1.774863e-002	0.386459	0.397012
7	1.610302e+000	0.304094	6.756385e-003	0.380671	0.394635
8	5.023475e-001	0.311959	2.532700e-003	0.374860	0.392107
9	1.594271e-001	0.317364	9.349623e-004	0.369156	0.389488
10	5.118833e-002	0.321077	3.401773e-004	0.363841	0.386844
11	1.656749e-002	0.323658	1.221213e-004	0.358993	0.384226
12	5.392540e-003	0.325489	4.331077e-005	0.354654	0.381670
13	1.762359e-003	0.326814	1.519371e-005	0.350807	0.379202
14	5.776815e-004	0.327789	5.278592e-006	0.347420	0.376839
15	1.897765e-004	0.328514	1.818208e-006	0.344449	0.374588

.

Result for $v_1 = 0, v_2 = 2$

cycle	|residue|	ratio	|error|	ratio	convergence factor
1	2.332824e+003	0.000000	1.130306e+000	0.000000	0.249375
2	1.658927e+002	0.071112	2.573654e-001	0.227695	0.238289

```
 3   1.740638e+001      0.104926   5.746902e-002        0.223297     0.233183
 4   2.213859e+000      0.127187   1.255196e-002        0.218413     0.229399
 5   3.107344e-001      0.140359   2.675256e-003        0.213135     0.226050
 6   4.627116e-002      0.148909   5.553596e-004        0.207591     0.222863
 7   7.199976e-003      0.155604   1.123881e-004        0.202370     0.219813
 8   1.162230e-003      0.161421   2.224051e-005        0.197890     0.216945
 9   1.934709e-004      0.166465   4.319318e-006        0.194210     0.214293
10   3.300557e-005      0.170597   8.260186e-007        0.191238     0.211868
11   5.734479e-006      0.173743   1.559942e-007        0.188851     0.209664
12   1.009112e-006      0.175973   2.916007e-008        0.186930     0.207668
13   1.790692e-007      0.177452   5.405721e-009        0.185381     0.205863
14   3.194089e-008      0.178372   9.953326e-010        0.184126     0.204228
15   5.714226e-009      0.178900   1.822506e-010        0.183105     0.202747
```

.

Result for $v_1 = 0, v_2 = 3$

```
cycle  |residue|          ratio     |error|              ratio   convergence
   factor
 1   1.279733e+003      0.000000   8.246208e-001        0.000000     0.181933
 2   6.128118e+001      0.047886   1.239874e-001        0.150357     0.165393
 3   4.197852e+000      0.068502   1.807404e-002        0.145773     0.158576
 4   3.457379e-001      0.082361   2.513173e-003        0.139049     0.153451
 5   3.161771e-002      0.091450   3.385354e-004        0.134704     0.149504
 6   3.097433e-003      0.097965   4.446410e-005        0.131343     0.146311
 7   3.196538e-004      0.103200   5.723743e-006        0.128727     0.143659
 8   3.438009e-005      0.107554   7.254505e-007        0.126744     0.141427
 9   3.818351e-006      0.111063   9.087146e-008        0.125262     0.139533
10   4.343261e-007      0.113747   1.128253e-008        0.124159     0.137913
11   5.025553e-008      0.115709   1.391556e-009        0.123337     0.136520
12   5.884932e-009      0.117100   1.707747e-010        0.122722     0.135313
13   6.948378e-010      0.118071   2.087896e-011        0.122260     0.134261
14   8.250871e-011      0.118745   2.545414e-012        0.121913     0.133339
15   9.836463e-012      0.119217   3.096536e-013        0.121652     0.132526
```

.

Result for $v_1 = 0, v_2 = 4$

```
cycle  |residue|          ratio     |error|              ratio    convergence
   factor
 1   8.351773e+002      0.000000   6.609090e-001        0.000000     0.145814
 2   3.043485e+001      0.036441   7.256809e-002        0.109800     0.126532
 3   1.541666e+000      0.050655   7.850138e-003        0.108176     0.120091
 4   9.413409e-002      0.061060   8.032536e-004        0.102324     0.115379
 5   6.459266e-003      0.068618   7.934903e-005        0.098785     0.111851
 6   4.798607e-004      0.074290   7.650179e-006        0.096412     0.109116
 7   3.773372e-005      0.078635   7.251982e-007        0.094795     0.106945
 8   3.090474e-006      0.081902   6.795204e-008        0.093701     0.105192
 9   2.605052e-007      0.084293   6.317243e-009        0.092966     0.103758
10   2.240698e-008      0.086014   5.841772e-010        0.092473     0.102570
11   1.954999e-009      0.087250   5.382824e-011        0.092144     0.101575
12   1.723247e-010      0.088146   4.948089e-012        0.091924     0.100734
13   1.530346e-011      0.088806   4.541247e-013        0.091778     0.100015
14   1.366630e-012      0.089302   4.163526e-014        0.091682     0.099395
15   1.225631e-013      0.089683   3.814695e-015        0.091622     0.098857
```

.

Result for $v_1 = 1, v_2 = 0$

```
cycle  |residue|          ratio     |error|              ratio     convergence
   factor
 1   6.179909e+003      0.000000   1.360828e+000        0.000000     0.300234
```

```
 2    1.275576e+003      0.206407   4.199343e-001      0.308587      0.304382
 3    3.536744e+002      0.277266   1.341956e-001      0.319563      0.309361
 4    1.100024e+002      0.311027   4.345678e-002      0.323832      0.312917
 5    3.621869e+001      0.329254   1.417374e-002      0.326157      0.315521
 6    1.220133e+001      0.336879   4.640907e-003      0.327430      0.317475
 7    4.148812e+000      0.340030   1.522676e-003      0.328099      0.318972
 8    1.413642e+000      0.340734   5.000095e-004      0.328375      0.320132
 9    4.811293e-001      0.340347   1.642075e-004      0.328409      0.321041
10    1.633133e-001      0.339437   5.390811e-005      0.328293      0.321759
11    5.524973e-002      0.338305   1.768674e-005      0.328091      0.322330
12    1.862458e-002      0.337098   5.798496e-006      0.327844      0.322786
13    6.255900e-003      0.335895   1.899475e-006      0.327581      0.323152
14    2.094091e-003      0.334739   6.217315e-007      0.327318      0.323448
15    6.986989e-004      0.333653   2.033475e-007      0.327067      0.323688
```

.

Result for $v_1 = 1, v_2 = 1$

```
    cycle      |residue|       ratio         |error|       ratio   convergence
    factor
       1   1.030835e+003    0.000000   4.727264e-001    0.000000    0.103251
       2   5.883607e+001    0.057076   5.305423e-002    0.112230    0.107647
       3   3.801623e+000    0.064614   6.112766e-003    0.115217    0.110114
       4   2.784357e-001    0.073241   7.123738e-004    0.116539    0.111686
       5   2.222565e-002    0.079823   8.361417e-005    0.117374    0.112801
       6   1.895919e-003    0.085303   9.862335e-006    0.117951    0.113643
       7   1.697849e-004    0.089553   1.167283e-006    0.118358    0.114305
       8   1.581453e-005    0.093144   1.384918e-007    0.118645    0.114839
       9   1.525515e-006    0.096463   1.645918e-008    0.118846    0.115277
      10   1.520761e-007    0.099688   1.958413e-009    0.118986    0.115643
      11   1.563620e-008    0.102818   2.332146e-010    0.119083    0.115952
      12   1.653635e-009    0.105757   2.778772e-011    0.119151    0.116215
      13   1.792425e-010    0.108393   3.312230e-012    0.119198    0.116442
      14   1.983378e-011    0.110653   3.949174e-013    0.119230    0.116639
      15   2.231664e-012    0.112518   4.709496e-014    0.119253    0.116811
```

.

Result for $v_1 = 1, v_2 = 2$

```
    cycle     |residue|       ratio         |error|       ratio   convergence
    factor
       1   3.491916e+002    0.000000   3.054693e-001    0.000000    0.066754
       2   1.021899e+001    0.029265   2.334872e-002    0.076436    0.071431
       3   4.299404e-001    0.042073   1.841565e-003    0.078872    0.073830
       4   2.178631e-002    0.050673   1.473129e-004    0.079993    0.075325
       5   1.245425e-003    0.057165   1.187207e-005    0.080591    0.076350
       6   7.801009e-005    0.062637   9.606537e-007    0.080917    0.077093
       7   5.235377e-006    0.067112   7.790676e-008    0.081098    0.077652
       8   3.692823e-007    0.070536   6.325950e-009    0.081199    0.078087
       9   2.697616e-008    0.073050   5.140286e-010    0.081257    0.078433
      10   2.019954e-009    0.074879   4.178622e-011    0.081292    0.078714
      11   1.539723e-010    0.076226   3.397763e-012    0.081313    0.078947
      12   1.189244e-011    0.077238   2.763303e-013    0.081327    0.079143
      13   9.277782e-013    0.078014   2.247589e-014    0.081337    0.079309
      14   7.294259e-014    0.078621   1.828290e-015    0.081345    0.079453
      15   5.769829e-015    0.079101   1.487324e-016    0.081351    0.079578
```

.

Result for $v_1 = 1, v_2 = 3$

```
    cycle  |residue|          ratio      |error|          ratio   convergence
    factor
```

```
 1    1.748348e+002    0.000000    2.273544e-001    0.000000    0.049683
 2    4.020464e+000    0.022996    1.322844e-002    0.058184    0.053766
 3    1.341199e-001    0.033359    7.939589e-004    0.060019    0.055774
 4    5.362431e-003    0.039982    4.829726e-005    0.060831    0.056998
 5    2.429919e-004    0.045314    2.956914e-006    0.061223    0.057819
 6    1.205611e-005    0.049615    1.816104e-007    0.061419    0.058404
 7    6.374272e-007    0.052872    1.117251e-008    0.061519    0.058839
 8    3.520088e-008    0.055223    6.879119e-010    0.061572    0.059174
 9    2.002565e-009    0.056890    4.237580e-011    0.061601    0.059439
10    1.162928e-010    0.058072    2.611069e-012    0.061617    0.059653
11    6.852108e-012    0.058921    1.609119e-013    0.061627    0.059830
12    4.079843e-013    0.059541    9.917484e-015    0.061633    0.059978
13    2.447990e-014    0.060002    6.112860e-016    0.061637    0.060104
14    1.477344e-015    0.060349    3.767977e-017    0.061640    0.060213
15    8.954816e-017    0.060614    2.322670e-018    0.061642    0.060307
```

.

Result for $v_1 = 1, v_2 = 4$

```
cycle  |residue|          ratio        |error|          ratio   convergence
  factor
 1    1.042637e+002    0.000000    1.827858e-001    0.000000    0.039944
 2    2.005234e+000    0.019232    8.672196e-003    0.047445    0.043533
 3    5.546631e-002    0.027661    4.241753e-004    0.048912    0.045257
 4    1.849251e-003    0.033340    2.101081e-005    0.049533    0.046290
 5    6.998200e-005    0.037843    1.046739e-006    0.049819    0.046975
 6    2.890955e-006    0.041310    5.229125e-008    0.049956    0.047459
 7    1.267590e-007    0.043847    2.615867e-009    0.050025    0.047818
 8    5.783896e-009    0.045629    1.309520e-010    0.050061    0.048092
 9    2.709911e-010    0.046853    6.558096e-012    0.050080    0.048309
10    1.292323e-011    0.047689    3.285046e-013    0.050091    0.048485
11    6.237388e-013    0.048265    1.645757e-014    0.050098    0.048629
12    3.035654e-014    0.048669    8.245760e-016    0.050103    0.048750
13    1.486172e-015    0.048957    4.131659e-017    0.050106    0.048853
14    7.307172e-017    0.049168    2.070333e-018    0.050109    0.048942
15    3.604223e-018    0.049324    1.037465e-019    0.050111    0.049019
```

.

Result for $v_1 = 2, v_2 = 0$

```
cycle     |residue|          ratio       |error|          ratio    convergence
  factor
 1    2.655117e+003    0.000000    7.196022e-001    0.000000    0.158763
 2    2.807305e+002    0.105732    1.214403e-001    0.168760    0.163685
 3    4.389822e+001    0.156371    2.105498e-002    0.173377    0.166854
 4    7.595636e+000    0.173028    3.666160e-003    0.174123    0.168643
 5    1.345020e+000    0.177078    6.386485e-004    0.174201    0.169740
 6    2.391057e-001    0.177771    1.111764e-004    0.174081    0.170456
 7    4.245722e-002    0.177567    1.933186e-005    0.173885    0.170941
 8    7.518789e-003    0.177091    3.357127e-006    0.173658    0.171279
 9    1.327337e-003    0.176536    5.821965e-007    0.173421    0.171515
10    2.335780e-004    0.175975    1.008285e-007    0.173186    0.171682
11    4.097873e-005    0.175439    1.743931e-008    0.172960    0.171798
12    7.168928e-006    0.174943    3.012563e-009    0.172746    0.171876
13    1.250905e-006    0.174490    5.198010e-010    0.172544    0.171928
14    2.177582e-007    0.174081    8.959148e-011    0.172357    0.171958
15    3.782731e-008    0.173712    1.542621e-011    0.172184    0.171973
```

.

Result for $v_1 = 2, v_2 = 1$

cycle	|residue|	ratio	|error|	ratio	convergence factor
1	3.587387e+002	0.000000	3.008484e-001	0.000000	0.065744
2	1.067855e+001	0.029767	2.317076e-002	0.077018	0.071158
3	4.487287e-001	0.042021	1.842948e-003	0.079538	0.073848
4	2.256832e-002	0.050294	1.485356e-004	0.080597	0.075480
5	1.286179e-003	0.056990	1.204521e-005	0.081093	0.076571
6	8.087232e-005	0.062878	9.796869e-007	0.081334	0.077345
7	5.482293e-006	0.067789	7.979984e-008	0.081454	0.077919
8	3.922682e-007	0.071552	6.504963e-009	0.081516	0.078360
9	2.913427e-008	0.074271	5.304691e-010	0.081548	0.078708
10	2.219978e-009	0.076198	4.326818e-011	0.081566	0.078989
11	1.722107e-010	0.077573	3.529630e-012	0.081576	0.079221
12	1.353118e-011	0.078573	2.879518e-013	0.081581	0.079415
13	1.073247e-012	0.079317	2.349245e-014	0.081585	0.079580
14	8.572977e-014	0.079879	1.916672e-015	0.081587	0.079721
15	6.884988e-015	0.080310	1.563774e-016	0.081588	0.079845

.

Result for $v_1 = 2, v_2 = 2$

cycle	|residue|	ratio	|error|	ratio	convergence factor
1	1.448199e+002	0.000000	2.243955e-001	0.000000	0.049037
2	3.594191e+000	0.024818	1.315787e-002	0.058637	0.053622
3	1.246015e-001	0.034667	7.947099e-004	0.060398	0.055792
4	5.109282e-003	0.041005	4.857254e-005	0.061120	0.057079
5	2.356797e-004	0.046128	2.984395e-006	0.061442	0.057926
6	1.184050e-005	0.050240	1.838170e-007	0.061593	0.058521
7	6.316880e-007	0.053350	1.133526e-008	0.061666	0.058961
8	3.512166e-008	0.055600	6.994189e-010	0.061703	0.059297
9	2.008926e-009	0.057199	4.316969e-011	0.061722	0.059561
10	1.171965e-010	0.058338	2.664984e-012	0.061733	0.059775
11	6.933122e-012	0.059158	1.645328e-013	0.061739	0.059951
12	4.143073e-013	0.059758	1.015864e-014	0.061742	0.060098
13	2.494234e-014	0.060203	6.272422e-016	0.061745	0.060223
14	1.509931e-015	0.060537	3.872984e-017	0.061746	0.060331
15	9.179027e-017	0.060791	2.391465e-018	0.061747	0.060424

.

Result for $v_1 = 2, v_2 = 3$

cycle	|residue|	ratio	|error|	ratio	convergence factor
1	8.776592e+001	0.000000	1.811025e-001	0.000000	0.039576
2	1.834493e+000	0.020902	8.646274e-003	0.047742	0.043468
3	5.258689e-002	0.028666	4.246673e-004	0.049116	0.045274
4	1.790526e-003	0.034049	2.109371e-005	0.049671	0.046336
5	6.865074e-005	0.038341	1.052908e-006	0.049916	0.047030
6	2.859888e-006	0.041659	5.267602e-008	0.050029	0.047517
7	1.260987e-007	0.044092	2.638218e-009	0.050084	0.047876
8	5.776016e-009	0.045806	1.322046e-010	0.050111	0.048150
9	2.713848e-010	0.046985	6.626843e-012	0.050126	0.048365
10	1.297016e-011	0.047793	3.322268e-013	0.050133	0.048539
11	6.271114e-013	0.048350	1.665719e-014	0.050138	0.048683
12	3.056636e-014	0.048742	8.352047e-016	0.050141	0.048802
13	1.498397e-015	0.049021	4.187935e-017	0.050143	0.048904
14	7.375846e-017	0.049225	2.099994e-018	0.050144	0.048992
15	3.641929e-018	0.049376	1.053040e-019	0.050145	0.049068

.

Result for $v_1 = 2, v_2 = 4$

cycle	|residue|	ratio	|error|	ratio	convergence factor
1	5.968518e+001	0.000000	1.526966e-001	0.000000	0.033368
2	1.071291e+000	0.017949	6.187596e-003	0.040522	0.036772
3	2.628731e-002	0.024538	2.577593e-004	0.041657	0.038333
4	7.693324e-004	0.029266	1.085425e-005	0.042110	0.039244
5	2.541368e-005	0.033033	4.592064e-007	0.042307	0.039838
6	9.120864e-007	0.035890	1.946889e-008	0.042397	0.040254
7	3.457788e-008	0.037911	8.262608e-010	0.042440	0.040559
8	1.357937e-009	0.039272	3.508443e-011	0.042462	0.040792
9	5.454790e-011	0.040170	1.490141e-012	0.042473	0.040976
10	2.223580e-012	0.040764	6.330013e-014	0.042479	0.041124
11	9.153192e-014	0.041164	2.689178e-015	0.042483	0.041245
12	3.793199e-015	0.041441	1.142507e-016	0.042485	0.041347
13	1.579443e-016	0.041639	4.854164e-018	0.042487	0.041434
14	6.599536e-018	0.041784	2.062445e-019	0.042488	0.041508
15	2.764795e-019	0.041894	8.763145e-021	0.042489	0.041573

.

Result for $v_1 = 3, v_2 = 0$

cycle	|residue|	ratio	|error|	ratio	convergence factor
1	1.731895e+003	0.000000	4.772061e-001	0.000000	0.105284
2	1.332957e+002	0.076965	5.436461e-002	0.113923	0.109518
3	1.472030e+001	0.110433	6.363908e-003	0.117060	0.111977
4	1.734653e+000	0.117841	7.494297e-004	0.117762	0.113396
5	2.065908e-001	0.119096	8.847251e-005	0.118053	0.114312
6	2.463413e-002	0.119241	1.045852e-005	0.118212	0.114953
7	2.936045e-003	0.119186	1.237354e-006	0.118311	0.115427
8	3.497108e-004	0.119109	1.464715e-007	0.118375	0.115791
9	4.163098e-005	0.119044	1.734462e-008	0.118416	0.116080
10	4.953753e-006	0.118992	2.054331e-009	0.118442	0.116314
11	5.892482e-007	0.118950	2.433477e-010	0.118456	0.116507
12	7.006984e-008	0.118914	2.882724e-011	0.118461	0.116669
13	8.330007e-009	0.118881	3.414861e-012	0.118460	0.116806
14	9.900258e-010	0.118851	4.044995e-013	0.118453	0.116923
15	1.176348e-010	0.118820	4.790965e-014	0.118442	0.117023

.

Result for $v_1 = 3, v_2 = 1$

cycle	|residue|	ratio	|error|	ratio	convergence factor
1	1.806744e+002	0.000000	2.248328e-001	0.000000	0.049132
2	4.176399e+000	0.023116	1.317724e-002	0.058609	0.053662
3	1.375470e-001	0.032934	7.958156e-004	0.060393	0.055818
4	5.462156e-003	0.039711	4.864574e-005	0.061127	0.057100
5	2.477092e-004	0.045350	2.989565e-006	0.061456	0.057946
6	1.236926e-005	0.049935	1.841867e-007	0.061610	0.058541
7	6.603866e-007	0.053389	1.136149e-008	0.061685	0.058980
8	3.687804e-008	0.055843	7.012560e-010	0.061722	0.059316
9	2.122171e-009	0.057546	4.329681e-011	0.061742	0.059581
10	1.246363e-010	0.058731	2.673686e-012	0.061752	0.059795
11	7.424379e-012	0.059568	1.651230e-013	0.061759	0.059970
12	4.467406e-013	0.060172	1.019837e-014	0.061762	0.060118
13	2.707929e-014	0.060615	6.298990e-016	0.061765	0.060243
14	1.650363e-015	0.060946	3.890651e-017	0.061766	0.060350
15	1.009940e-016	0.061195	2.403156e-018	0.061767	0.060444

.

Result for $v_1 = 3, v_2 = 2$

cycle	|residue|	ratio	|error|	ratio	convergence factor
1	8.757448e+001	0.000000	1.810759e-001	0.000000	0.039570
2	1.833553e+000	0.020937	8.646696e-003	0.047752	0.043469
3	5.258255e-002	0.028678	4.247632e-004	0.049124	0.045278
4	1.791120e-003	0.034063	2.110165e-005	0.049679	0.046340
5	6.870264e-005	0.038357	1.053438e-006	0.049922	0.047035
6	2.863317e-006	0.041677	5.270844e-008	0.050035	0.047522
7	1.263069e-007	0.044112	2.640110e-009	0.050089	0.047881
8	5.788133e-009	0.045826	1.323120e-010	0.050116	0.048155
9	2.720719e-010	0.047005	6.632824e-012	0.050130	0.048370
10	1.300844e-011	0.047812	3.325556e-013	0.050138	0.048544
11	6.292148e-013	0.048370	1.667508e-014	0.050142	0.048687
12	3.068072e-014	0.048760	8.361702e-016	0.050145	0.048807
13	1.504560e-015	0.049039	4.193112e-017	0.050147	0.048909
14	7.408807e-017	0.049242	2.102755e-018	0.050148	0.048996
15	3.659443e-018	0.049393	1.054505e-019	0.050149	0.049072

.

Result for $v_1 = 3, v_2 = 3$

cycle	|residue|	ratio	|error|	ratio	convergence factor
1	5.945561e+001	0.000000	1.526825e-001	0.000000	0.033365
2	1.070297e+000	0.018002	6.187752e-003	0.040527	0.036772
3	2.627477e-002	0.024549	2.577899e-004	0.041661	0.038335
4	7.691734e-004	0.029274	1.085633e-005	0.042113	0.039246
5	2.541274e-005	0.033039	4.593210e-007	0.042309	0.039840
6	9.121526e-007	0.035894	1.947469e-008	0.042399	0.040256
7	3.458310e-008	0.037914	8.265419e-010	0.042442	0.040561
8	1.358230e-009	0.039274	3.509771e-011	0.042463	0.040794
9	5.456298e-011	0.040172	1.490758e-012	0.042475	0.040978
10	2.224324e-012	0.040766	6.332842e-014	0.042481	0.041125
11	9.156769e-014	0.041167	2.690464e-015	0.042484	0.041247
12	3.794885e-015	0.041443	1.143087e-016	0.042487	0.041349
13	1.580225e-016	0.041641	4.856765e-018	0.042488	0.041436
14	6.603117e-018	0.041786	2.063606e-019	0.042489	0.041510
15	2.766417e-019	0.041896	8.768302e-021	0.042490	0.041575

.

Result for $v_1 = 3, v_2 = 4$

cycle	|residue|	ratio	|error|	ratio	convergence factor
1	4.335915e+001	0.000000	1.321583e-001	0.000000	0.028880
2	6.849016e-001	0.015796	4.660361e-003	0.035263	0.031913
3	1.472864e-002	0.021505	1.688654e-004	0.036234	0.033293
4	3.797836e-004	0.025785	6.183105e-006	0.036616	0.034094
5	1.109514e-005	0.029214	2.274057e-007	0.036779	0.034615
6	3.520256e-007	0.031728	8.380416e-009	0.036852	0.034978
7	1.176545e-008	0.033422	3.091300e-010	0.036887	0.035245
8	4.060755e-010	0.034514	1.140828e-011	0.036904	0.035448
9	1.429858e-011	0.035212	4.211199e-013	0.036914	0.035608
10	5.099387e-013	0.035664	1.554713e-014	0.036919	0.035737
11	1.833989e-014	0.035965	5.740235e-016	0.036922	0.035843
12	6.634068e-016	0.036173	2.119492e-017	0.036923	0.035932
13	2.409612e-017	0.036322	7.826178e-019	0.036925	0.036007
14	8.778704e-019	0.036432	2.889877e-020	0.036926	0.036072
15	3.205646e-020	0.036516	1.067132e-021	0.036927	0.036128

.

Result for $v_1 = 4, v_2 = 0$

```
cycle  |residue|          ratio       |error|          ratio    convergence
   factor
 1   1.305638e+003       0.000000   3.571047e-001      0.000000    0.078787
 2   7.990465e+001       0.061200   3.078639e-002      0.086211    0.082415
 3   6.837976e+000       0.085577   2.728251e-003      0.088619    0.084433
 4   6.150176e-001       0.089941   2.433551e-004      0.089198    0.085600
 5   5.555443e-002       0.090330   2.176670e-005      0.089444    0.086355
 6   5.009766e-003       0.090178   1.949873e-006      0.089581    0.086885
 7   4.509998e-004       0.090024   1.748412e-007      0.089668    0.087277
 8   4.055972e-005       0.089933   1.568822e-008      0.089728    0.087580
 9   3.645908e-006       0.089890   1.408358e-009      0.089772    0.087821
10   3.276772e-007       0.089875   1.264748e-010      0.089803    0.088017
11   2.945016e-008       0.089876   1.136069e-011      0.089826    0.088180
12   2.647044e-009       0.089882   1.020666e-012      0.089842    0.088317
13   2.379448e-010       0.089891   9.171024e-014      0.089853    0.088434
14   2.139097e-011       0.089899   8.241165e-015      0.089861    0.088536
15   1.923168e-012       0.089906   7.405975e-016      0.089866    0.088624
```

.

Result for $v_1 = 4, v_2 = 1$

```
cycle  |residue|          ratio       |error|          ratio    convergence
   factor
 1   1.093773e+002       0.000000   1.812869e-001      0.000000    0.039616
 2   2.098783e+000       0.019188   8.652527e-003      0.047728    0.043484
 3   5.704834e-002       0.027182   4.249640e-004      0.049114    0.045285
 4   1.882553e-003       0.032999   2.111018e-005      0.049675    0.046345
 5   7.106054e-005       0.037747   1.053863e-006      0.049922    0.047039
 6   2.943509e-006       0.041423   5.273130e-008      0.050036    0.047526
 7   1.297051e-007       0.044065   2.641368e-009      0.050091    0.047884
 8   5.950199e-009       0.045875   1.323812e-010      0.050118    0.048158
 9   2.802074e-010       0.047092   6.636618e-012      0.050133    0.048373
10   1.342545e-011       0.047913   3.327617e-013      0.050140    0.048547
11   6.507751e-013       0.048473   1.668619e-014      0.050145    0.048690
12   3.179931e-014       0.048864   8.367653e-016      0.050147    0.048810
13   1.562661e-015       0.049141   4.196279e-017      0.050149    0.048912
14   7.710595e-017       0.049343   2.104432e-018      0.050150    0.048999
15   3.816096e-018       0.049492   1.055388e-019      0.050151    0.049075
```

.

Result for $v_1 = 4, v_2 = 2$

```
cycle  |residue|          ratio       |error|          ratio    convergence
   factor
 1   5.956504e+001       0.000000   1.526861e-001      0.000000    0.033366
 2   1.070895e+000       0.017979   6.187900e-003      0.040527    0.036773
 3   2.628769e-002       0.024547   2.577967e-004      0.041661    0.038335
 4   7.695512e-004       0.029274   1.085667e-005      0.042113    0.039246
 5   2.542668e-005       0.033041   4.593380e-007      0.042309    0.039841
 6   9.127487e-007       0.035897   1.947552e-008      0.042399    0.040256
 7   3.460995e-008       0.037918   8.265816e-010      0.042442    0.040561
 8   1.359449e-009       0.039279   3.509957e-011      0.042464    0.040794
 9   5.461792e-011       0.040177   1.490844e-012      0.042475    0.040978
10   2.226781e-012       0.040770   6.333235e-014      0.042481    0.041126
11   9.167659e-014       0.041170   2.690642e-015      0.042484    0.041247
12   3.799678e-015       0.041447   1.143166e-016      0.042487    0.041349
13   1.582320e-016       0.041644   4.857121e-018      0.042488    0.041436
14   6.612229e-018       0.041788   2.063764e-019      0.042489    0.041510
15   2.770358e-019       0.041897   8.769001e-021      0.042490    0.041575
```

.

Result for $v_1 = 4, v_2 = 3$

cycle	|residue|	ratio	|error|	ratio	convergence factor
1	4.335295e+001	0.000000	1.321581e-001	0.000000	0.028880
2	6.848903e-001	0.015798	4.660374e-003	0.035264	0.031913
3	1.472868e-002	0.021505	1.688664e-004	0.036235	0.033293
4	3.797890e-004	0.025786	6.183161e-006	0.036616	0.034094
5	1.109540e-005	0.029215	2.274083e-007	0.036779	0.034615
6	3.520365e-007	0.031728	8.380531e-009	0.036852	0.034978
7	1.176590e-008	0.033422	3.091348e-010	0.036887	0.035245
8	4.060938e-010	0.034514	1.140848e-011	0.036905	0.035448
9	1.429932e-011	0.035212	4.211278e-013	0.036914	0.035608
10	5.099674e-013	0.035664	1.554744e-014	0.036919	0.035737
11	1.834100e-014	0.035965	5.740358e-016	0.036922	0.035843
12	6.634495e-016	0.036173	2.119540e-017	0.036923	0.035932
13	2.409774e-017	0.036322	7.826365e-019	0.036925	0.036007
14	8.779315e-019	0.036432	2.889949e-020	0.036926	0.036072
15	3.205874e-020	0.036516	1.067159e-021	0.036927	0.036128

.

Result for $v_1 = 4, v_2 = 4$

cycle	|residue|	ratio	|error|	ratio	convergence factor
1	3.318582e+001	0.000000	1.163430e-001	0.000000	0.025424
2	4.677808e-001	0.014096	3.627130e-003	0.031176	0.028154
3	8.965233e-003	0.019165	1.161573e-004	0.032025	0.029389
4	2.073972e-004	0.023134	3.758056e-006	0.032353	0.030104
5	5.448802e-006	0.026272	1.221050e-007	0.032492	0.030567
6	1.551551e-007	0.028475	3.974903e-009	0.032553	0.030889
7	4.638434e-009	0.029895	1.295102e-010	0.032582	0.031125
8	1.427642e-010	0.030779	4.221525e-012	0.032596	0.031306
9	4.472442e-012	0.031327	1.376361e-013	0.032603	0.031447
10	1.416727e-013	0.031677	4.487966e-015	0.032607	0.031561
11	4.520389e-015	0.031907	1.463521e-016	0.032610	0.031655
12	1.449490e-016	0.032066	4.772759e-018	0.032611	0.031734
13	4.664276e-018	0.032179	1.556520e-019	0.032613	0.031801
14	1.504811e-019	0.032262	5.076339e-021	0.032613	0.031858
15	4.864498e-021	0.032326	1.655598e-022	0.032614	0.031908

.

2.5.3 Problem 3

3. Use your V-cycle code to solve

$$\Delta u = -\exp\left(-(x-0.25)^2 - (y-0.6)^2\right)$$

on the unit square $(0,1)\times(0,1)$ with homogeneous Dirichlet boundary conditions using a grid spacing of 2^{-7}. How many steps of pre and postsmoothing did you use? What tolerance did you use? How many cycles did it take to converge? Compare the amount of work needed to reach convergence with your solvers from Homework 3 taking into account how much work is involved in a V-cycle.

Figure 2.55: problem 3

This problem was solved using multigrid V cycle method. The following is the solution found

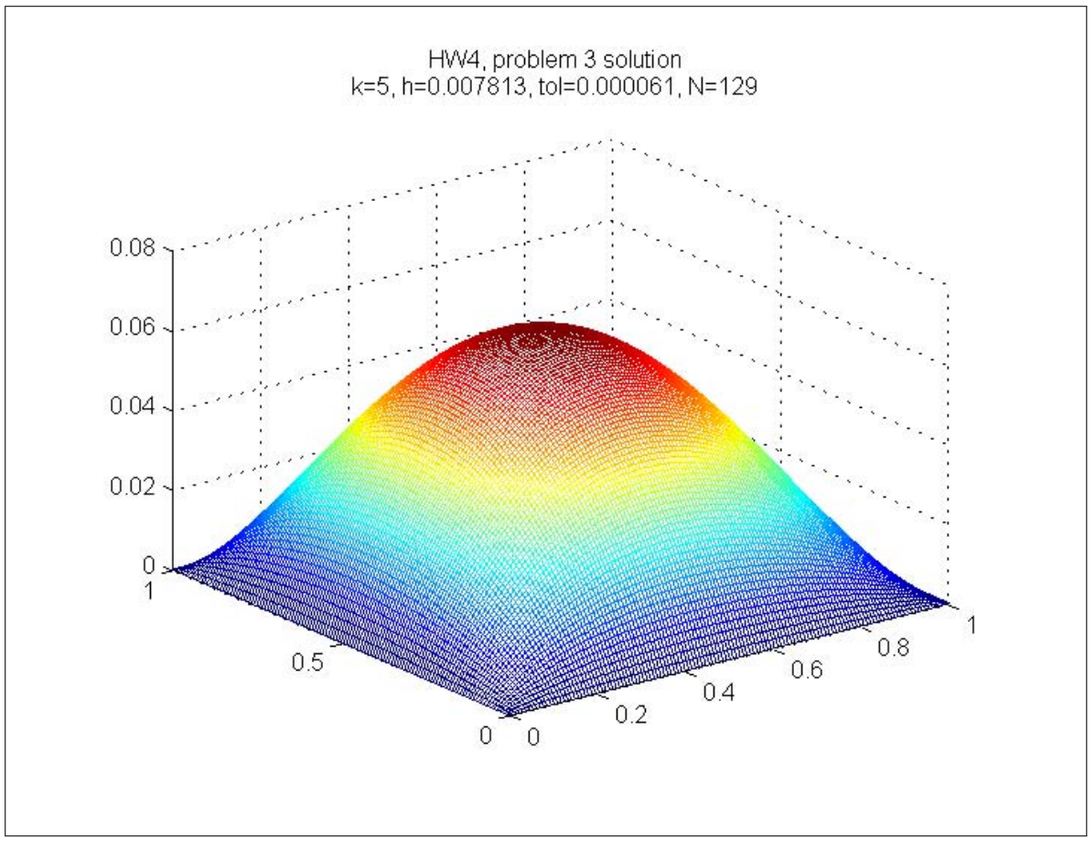


Figure 2.56: problem 3 solutions found earlier

Tolerance used is $h^2 = 0.000061$, the number of grid points along one dimension is $n = 129$, the spacing is 2^{-7}.

V cycle method converged in 5 iterations. The number of pre-smooth is 1 and the number of post smooth is 1. These are selected since in problem 2 it was found they lead to the most efficient solver.

Hence, the amount of work done

$$W = \left(\frac{\log(\varepsilon)}{\log(\rho)}\right)\left(\frac{3}{4}(7(v_1 + v_2) + 13)N\right)$$

From the table in problem 2, $\rho = 0.116811$ for the solver $(1,1)$, and given that $N = (n-2)^2 = 127^2 = 16129$, hence the above becomes

$$\begin{aligned} W &= \left(\frac{\log_{10}(0.000061)}{\log_{10}(0.116811)}\right)\left(\frac{3}{4}(7(1+1)+13)16129\right) \\ &= 1.4762 \times 10^6 \text{ operations} \end{aligned}$$

The above is compared with the solvers used in HW3, for same tolerance as above and same h, from HW3, the results were the following

method	Number of iterations
Jacobi	31702
Gauss-Seidel	15852
SOR	306

To compare work between all methods, it is required to find the work per iteration for the Jacobi, GS and SOR.

Work per iteration in these methods required only one smooth operation, and one calculation for the residue. No mapping between different grid sizes was needed. Hence, assuming about 13 flops to calculate the averaging and residue per one grid point, work per iteration for the above solver becomes

$$\textit{Work Per iteration} = (6N + 7N) = 13N$$

where N is the total number of grid points which is 16129 in this example.

Therefore, total work can be found for all the methods, including the multigrid solver. The following table summarizes the result

method	Number of iterations	W (flops)
Jacobi	31702	$31702 \times 13 \times 16129 = 6.6472 \times 10^9$
Gauss-Seidel	15852	$15852 \times 13 \times 16129 = 3.3238 \times 10^9$
SOR	306	$306 \times 13 \times 16129 = 6.4161 \times 10^7$
Multigrid V Cycle	5	1.4762×10^6

Using the multigrid as a base measure, and normalizing other solvers relative to it, the above becomes

method	work
Jacobi	$\frac{6.6472 \times 10^9}{1.4762 \times 10^6} = 4502.9$
Gauss-Seidel	$\frac{3.3238 \times 10^9}{1.4762 \times 10^6} = 2251.6$
SOR	$\frac{6.4161 \times 10^7}{1.4762 \times 10^6} = 43.464$
Multigrid V Cycle	1

The above shows clearly that Multigrid is the most efficient solver. SOR required about 44 times as much work, GS over 2251 more work, and Jacobi about 4500 more work.

2.5.4 Note on using Full Multigrid cycle (FMG) to improve convergence

It was found that using FMG to determine a better initial guess solution before initiating the V cycle algorithm resulted in about 40% reduction in the number of iterations needed to convergence by the V cycle algorithm.

The following is a plot of one of the tests performed showing the difference in number of iterations needed to converge. All other parameters are kept the same. This shows that with FMG cycle, convergence reached in 4 iterations, while without FMG, it was reached in 7 cycles. The cost of the FMG cycle itself was not taken into account. It is estimated that the FMG correction cycle adds about $\frac{1}{2}$ the cost of one V cycle to the total cost.

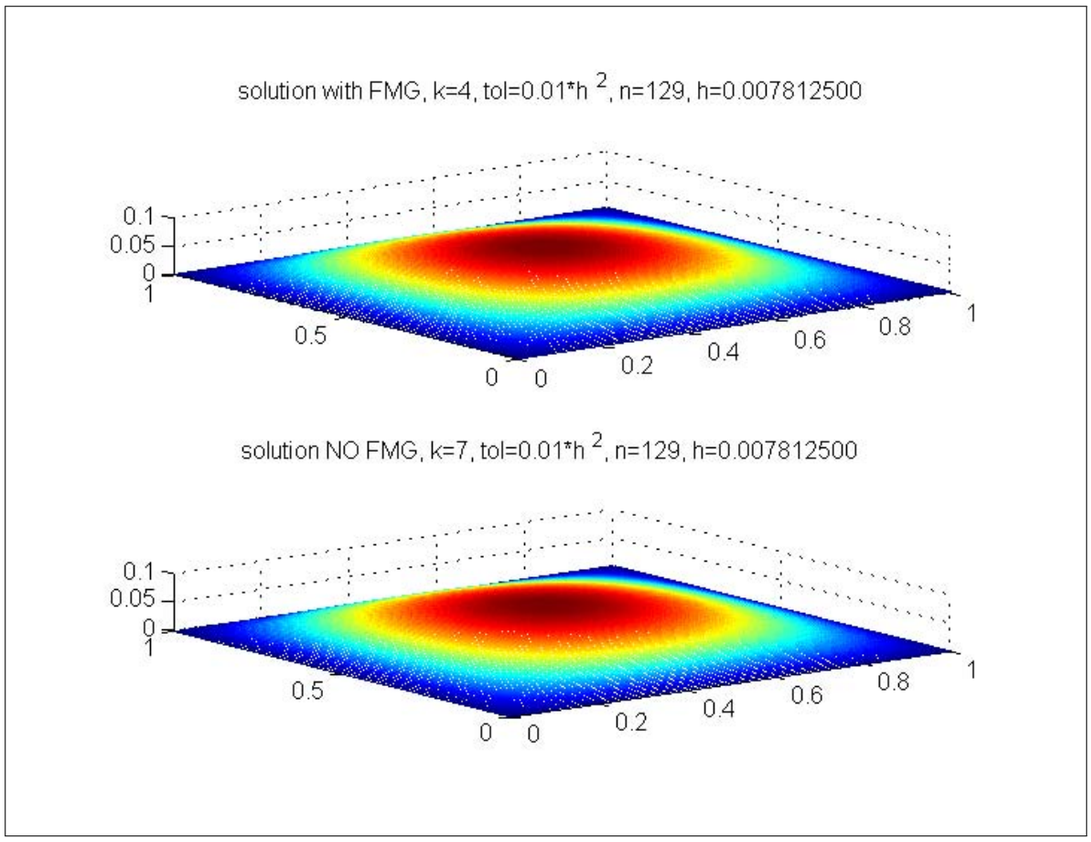


Figure 2.57: compare with FMG

The following is a small function written to compare convergence when using FMG and without using FMG which generated the above result. (see code on web page)

2.5.5 References

[1] Thor Gjesdal, Analysis of a new Red-Black ordering for Gauss-Seidel smoothing in cell-centered multigrid. ref no. CMR-93-A200007, November 1993.

[2] William L. Briggs, A multigrid tutorial, W-7405-Eng-48.

[3] Robert Guy, Lecture notes, Math 228a, Fall 2010. Mathematics dept. UC Davis.

[4] Jim Demmel, Lecture notes, CS267, 1996. UC Berkeley.

[5] P. Wesseling, A survey of Fourier smoothing analysis results. ISNM 98, Multigrid methods II page 105-106.

2.5.6 Source code listing

```
function nma_build_HW4()

list = dir('*.m');

if isempty(list)
    fprintf('no matlab files found\n');
    return
end

for i=1:length(list)
    name=list(i).name;
    fprintf('processing %s\n',name)
    p0 = fdep(list(i).name,'-q');
    [pathstr, name_of_matlab_function, ext] = fileparts(name);

    %make a zip file of the m file and any of its dependency
    p1=dir([name_of_matlab_function '.fig']);
```

```
    if length(p1)==1
        files_to_zip =[p1(1).name;p0.fun];
    else
        files_to_zip =p0.fun;
    end

    zip([name_of_matlab_function '.zip'],files_to_zip)

end

end
```

```
%-------------------------------------------------------
% This function is the prologation operator for 1D
% it takes 1D coarse grid of spacing 2h and generate 1D fine
% grid of spacing h by linear interpolation
%
% INPUT:
%  c the coarse 1D grid, spacing 2h
% OUTPUT:
%  f the fine 1D grid, spacing h
%
% EXAMPLE:
% nma_c2f_1D( [0 1 0] )
%         0    0.5000    1.0000    0.5000         0
%
% Nasser M. Abbasi
% Math 228a, UC Davis fall 2010
function  f = nma_c2f_1D( c )
if ndims(c) > 2
   error('nma_c2f_1D:: input number of dimensions too large');
end

[n,nCol]  = size(c(:));
if nCol>1
   error('nma_c2f_1D::input must be a vector');
end

valid_grid_points = log2(n-1);
if  round(valid_grid_points) ~= valid_grid_points
   error('nma_c2f_1D:: invalid number of grid points value');
end

if n < 3
   error('length of coarse grid must be at least 3');
end

fine_n     = 2*(n-2) + 3;
f          = zeros(fine_n ,1);
f(1:2:end) = c;
indx       = 2:2:fine_n;
f(indx)    = ( f(indx-1) + f(indx+1) )/2;
f          = f(:)';
end
```

```
%-----------------------------------------
% function for debugging
%
function nma_DEBUG(msg)
debug = false;
if debug

```

```
    fprintf(msg);
    fprintf('\n');
end
end
```

```
%-----------------------------------------------------------
% This function is the restriction full weight operator
% for mapping 1D fine grid to 1D coarse grid
%
% INPUT:
%   f the fine 1D grid, spacing h
% OUTPUT:
%   c the corase 1D grid, spacing 2h
%
% EXAMPLE
%nma_c2f_1D( [0 1 0] )
%         0    0.5000    1.0000    0.5000         0
%nma_f2c_1D(ans)
%         0    0.7500         0
%
% Nasser M. Abbasi
% Math 228a, UC Davis fall 2010

function c = nma_f2c_1D( f )
if ndims(f) > 2
   error('nma_f2c_1D:: input number of dimensions too large');
end

[n,nCol]  = size(f(:));
if nCol>1
   error('nma_f2c_1D::input must be a vector');
end

valid_grid_points = log2(n-1);
if  round(valid_grid_points) ~= valid_grid_points
   error('nma_f2c_1D:: invalid number of grid points value');
end

if n < 5
   error('length of fine grid cant be smaller than 5');
end

if 2*floor(n/2) == n
   error('length of fine grid must be odd number');
end

c          = zeros((n+1)/2,1);
indx       = 3:2:n-2;
c(2:end-1) = (1/4)*f(indx-1) + (1/2)*f(indx) + (1/4)*f(indx+1);
c          = c(:)';
end
```

```
function  nma_HW4_residue_animation

myforce = @(X,Y) -exp( -(X-0.25).^2 - (Y-0.6).^2 ); % RHS of PDE
N = 6;        % number of levels
n = 2^N+1;    % total number of grid points, always odd number
h = 1/(n-1);  % mesh spacing
[X,Y] = meshgrid(0:h:1,0:h:1);  % for plotting solution

% use a problem whose solution is known Lu=0, since B.C. are zero
f     = myforce(X,Y);   %  RHS of the problem
```

```
u     = 0.*X;  % initial guess of solution, use random

%-- select number of cycle to run. choose number not too large
number_of_cycles =10;

METHOD  = 1;  % GSRB
mu1=2; mu2=1;
k=0;


for i = 1:number_of_cycles

   k = k+1;

   u = nma_V_cycle(u,f,mu1,mu2,METHOD);  %--- CALL V CYCLE
   the_residue  = nma_find_residue(u,f);

   mesh(X,Y, u);  drawnow; title(sprintf('k=%d\n',k));
   pause(.01);
end
end
```

```
%---------------------------------------
%This function perform full mutligrid cycle
%to determine a good initial guess to use
%for initial solution to the V cycle interative
%solver
%
% INPUT
%   f: 2D grid, the force on the orginal fine grid
% OUTPUT:
%   u: estimated solution on the fine grid obtained
%      by running FMG cycle once
%
% Example use: see nma_test_FMG.m
%
% by Nasser M. Abbasi
% Math 228a, UC Davis, Fall 2010

function u = nma_initial_solution_guess_using_FMG( f )

nma_validate_dimensions_1(f); %asserts dimensions make sense
   [n,~] = size(f);
   if n<3
      error('initial_solution_guess_using_FMG:: size of f too small');
   end

   mu1 = 1;
   mu2 = 1;
   smoother = 1; %GSRB

   k = 3;
   while k <= n
      fk = f;
      [current_size,~] = size(fk);
      while current_size ~= k
            fk = nma_f2c(fk);
            [current_size,~] = size(fk);
      end

      if k == 3
         u      = zeros(3,3);
         u(2,2) = -0.25*(0.5)^2*fk(2,2);
```

```
      else
        u = nma_c2f(u);
        u = nma_V_cycle(u,fk,mu1,mu2,smoother);
      end
      k = 2*k - 1;
   end

end
```

```
%-----------------------------------------------------
% This function solves problem 2 in HW4
%
%LOGIC
%
%  LOOP over all combinations of mu1 and mu2
%       run V cycle 15 times on the problem
%       record the result in table, which contains
%       average convergence factor
%  END LOOP
%
% By Nasser M. Abbasi
% Math 228A, UC Davis
function nma_math_228_fall_2010_HW4_problem2

N = 6;        % number of levels
n = 2^N+1;    % total number of grid points, always odd number
h = 1/(n-1);  % mesh spacing

[X,Y] = meshgrid(0:h:1,0:h:1);
% use a problem whose solution is known Lu=0, since B.C. are zero

%force = @(X,Y) -2* ( (1-6*X.^2).*Y.^2.*(1-Y.^2)+(1-6*Y.^2).*X.^2.*(1-X.^2)); % RHS of PDE
%force = @(X,Y) -exp( -(X-0.25).^2 - (Y-0.6).^2 );  % RHS of PDE
%exact = @(X,Y) (X.^4-X.^2).*(Y.^4-Y.^2);
%f = force(X,Y);

table = zeros(20,6);
f     = zeros(n,n);   %  RHS of the problem
exact = zeros(n,n);   %  exact solution

initial_guess = rand(n,n);  % initial guess of solution, use random
initial_guess(:,1)=0; initial_guess(:,end)=0; %initialize B.C. to zero
initial_guess(1,:)=0; initial_guess(end,:)=0;

%-- select mu1 and mu2, these are the number of pre-smooth and
%-- number of post smooth to be used inside the V cycle function
mu1_values = 0:2;
mu2_values = 0:2;

%-- select number of cycle to run. choose number not too large
number_cycles = 10;

%-- select where to send the result, either to stdout or to a file
%fileID = fopen('table_result_work','w');
fileID  = 1;
METHOD  = 1;  % GSRB

for i = 1:length(mu1_values)

    mu1 = mu1_values(i);
    for j = 1:length(mu2_values)
        mu2   = mu2_values(j);

```

```
        table = run_V_cycle(initial_guess, ...
            exact,...
            f,...
            mu1,...
            mu2,...
            number_cycles,...
            h,...
            METHOD);

        print_table(table,mu1,mu2,fileID);
    end
end
%fclose(fileID);
end

%-------------
% This function runs the V cycle for k times, recording
% the average convergence rate at each step k. In the end
% it returns table containing the results found
%
%
function table = run_V_cycle(u,exact,f,mu1,mu2,number_of_cycles,h,METHOD)

%-- initialization of data and storage
table = zeros(number_of_cycles,7);  %initialize table for result collection

last_error_norm    = 0;
last_residue_norm  = 0;
initial_error_norm = nma_find_norm(u-exact);
k=0;
[X,Y] = meshgrid(0:h:1,0:h:1);  % for plotting solution

for i = 1:number_of_cycles

    k = k+1;

    u = nma_V_cycle(u,f,mu1,mu2,METHOD);  %--- CALL V CYCLE

    e = exact-u ;
    norm_error   = nma_find_norm(e);
    the_residue  = nma_find_residue(u,f);
    norm_residue = nma_find_norm(the_residue);

    % fill in table for analysis
    table(k,1) = k;
    table(k,2) = norm_residue;
    table(k,4) = norm_error;
    table(k,6) = (norm_error/initial_error_norm)^(1/k);

    if k > 1
        table(k,3) = norm_residue/last_residue_norm;
        table(k,5) = norm_error/last_error_norm;
        table(k,7) = -6/log10(table(k,6))*(3/4)*(7*(mu1+mu2)+13);
    end

    last_error_norm   = norm_error;
    last_residue_norm = norm_residue;

    mesh(X,Y, u);  drawnow; title(sprintf('k=%d\n',k));
end
end
%-----------------------------------------
% This function is called by HW4 problem 2 solver
```

```
% to format the results and print it. THe results
% are included in the final report.
%
% this also makes plots of the data
%
function print_table(table,mu1,mu2,fileID)

fprintf(fileID,'mu1=%d, mu2=%d\n',mu1,mu2);

% figure;
% subplot(5,1,1);
% plot(table(:,3),'-o');
% title('residual ratio');
%
% subplot(5,1,2);
% plot(table(:,5),'-o');
% title('error ratio');
%
% subplot(5,1,3);
% plot(table(:,2),'-o');
% title('residual norm');
%
% subplot(5,1,4);
% plot(table(:,4),'-o');
% title('error norm');
%
% subplot(5,1,5);
% plot(table(:,6),'-o');
% title('RAO(M)');

titles={'V-cycle','|residue|','ratio','|error|','ratio','C.F','work'};
wid    = 16;
fms    = {'d','.6e','.6f','.6e','.6f','.6f','.1f'};

nma_format_matrix(titles,table,wid,fms,fileID,true);

end
```

```
%-------------------------------------------------------
% This function solves problem 3 in HW4
% It uses the code generated in problem 1, which implements
% V cycle.
% By Nasser M. Abbasi
% Math 228A, UC Davis

function nma_math_228_fall_2010_HW4_problem3

%----------- INITIALIZATION SECTION -------------
myforce = @(X,Y) -exp( -(X-0.25).^2 - (Y-0.6).^2 ); % RHS of PDE
h       = 2^-7;          % mesh spacing
n       = 1/h+1;         % total number of grid points, always odd number
tol     = 1*h^2;         % tolerance used
[X,Y]   = meshgrid(0:h:1,0:h:1); % coordinates
f       = myforce(X,Y); % evaluate the force on the grid
u       = zeros(n,n);   % initial guess of solution
mu1     = 1;             % mu1 number of pre-smooth
mu2     = 1;             % mu2 number of post-smooth
METHOD  = 1;             % relaxation method for multigrid: Gause-Seidel B/R

[k, u ] = nma_solver_Vcycle(u,f,mu1,mu2,METHOD,tol,false);

[X,Y] = meshgrid(0:h:1,0:h:1); % coordinates
mesh(X,Y, u);  drawnow; title(sprintf ...
```

```
   ('HW4, problem 3 solution\nk=%d, h=%f, tol=%f, N=%d\n',k,h,tol,n));

end
```

```
%-------------------------------------
%This function compares the performance of multigrid
% with and without a FMG initial cycle by solving HW3 poisson
% problem and comparing the number of iterations needed
% to converge when using V cycle.
%
% by Nasser M. Abbasi
% Math 228a, UC Davis, Fall 2010
function nma_math_228_fall_2010_HW4_test_FMG()
%----------- INITIALIZATION SECTION -------------
myforce = @(X,Y) -exp( -(X-0.25).^2 - (Y-0.6).^2 ); % RHS of PDE
h       = 2^-7;          % mesh spacing
n       = 1/h+1;         % total number of grid points, always odd number
tol     = 0.01*h^2;      % tolerance used
[X,Y]   = meshgrid(0:h:1,0:h:1); % coordinates
f       = myforce(X,Y); % evaluate the force on the grid
mu1     = 1;             % mu1 number of pre-smooth
mu2     = 1;             % mu2 number of post-smooth
METHOD  = 1;             % relaxation method for multigrid: Gause-Seidel B/R


%------------ LOGIC SECTION -------------------

% Solve by doing initial FMG correction
u     = nma_initial_solution_guess_using_FMG(f);
[k,u] = nma_solver_Vcycle(u,f,mu1,mu2,METHOD,tol,false);
subplot(2,1,1);
mesh(X,Y, u);
title(sprintf('solution with FMG, k=%d, tol=%s, n=%d, h=%0.9f\n',k, ...
              '0.01*h^2',n,h));

% Solve without doing an initial FMG correction
u     = zeros(n);
[k,u] = nma_solver_Vcycle(u,f,mu1,mu2,METHOD,tol,false);
subplot(2,1,2);
mesh(X,Y, u);
title(sprintf('solution NO FMG, k=%d, tol=%s, n=%d, h=%0.9f\n',...
               k,'0.01*h^2',n,h));

end
```

2.6 HW 5

2.6.1 Introduction

Math 228A
Homework 5
Due Friday, 12/10/10, 4:00 P.M.

Homework must be turned in to Arcade before the deadline. You may email him a pdf file or put a hard copy in his mailbox.

Exam week office hours:
Bob, Monday 12-1
Arcade, Tuesday & Wednesday 1:30-2:20

1. Write a program to solve the discrete Poisson equation on the unit square using preconditioned conjugate gradient. Set up a test problem and compare the number of iterations and efficiency of using (i) no preconditioning and (ii) SSOR preconditioning. Run your tests for different grid sizes. How does the number of iterations scale with the number of unknowns as the grid is refined?

   Note that there are two typos in the PCG algorithm in our textbook. See your class notes, another textbook, or the author's webpage for the correct algorithm.

   SSOR preconditioning Symmetric SOR (SSOR) consists of one forward sweep of SOR followed by one backward sweep of SOR. For the discrete Poisson equation, one step of SSOR is

$$u_{i,j}^{k+1/2} = \frac{\omega}{4}(u_{i-1,j}^{k+1/2} + u_{i,j-1}^{k+1/2} + u_{i+1,j}^{k} + u_{i,j+1}^{k} - h^2 f_{i,j}) + (1-\omega)u_{i,j}^{k}$$
$$u_{i,j}^{k+1} = \frac{\omega}{4}(u_{i-1,j}^{k+1/2} + u_{i,j-1}^{k+1/2} + u_{i+1,j}^{k+1} + u_{i,j+1}^{k+1} - h^2 f_{i,j}) + (1-\omega)u_{i,j}^{k+1/2}.$$

   It can be shown that one step of SSOR in matrix form is equivalent to

$$\frac{1}{\omega(2-\omega)}(D-\omega L)D^{-1}(D-\omega U)(\boldsymbol{u}^{k+1} - \boldsymbol{u}^{k}) = \boldsymbol{f},$$

   where $A = D - L - U$.

   For the constant coefficient problem, this suggests the preconditioner.

$$M = (D-\omega L)(D-\omega U).$$

   Note: If you are interested, experiment with incomplete Cholesky factorization preconditioning and multigrid preconditioning. Incomplete Cholesky preconditioning requires that you form the matrix. Vary the amount of fill (in MATLAB use `cholinc` and vary the drop tolerance). Obviously, a factorization with more elements results in fewer iterations of CG, but it is more expensive to compute and to apply the preconditioner. To use MG as a preconditioner, the product $M^{-1}r$ is computed by applying one V-cycle with zero initial guess with right hand side r. If the smoother is symmetric and the number of pre and post smoothing steps are the same, this preconditioner is symmetric positive definite and may be used with CG.

Figure 2.58: problem description

The test problem used is

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

on the unit square $[(0,1),(0,1)]$ with zero boundary conditions.

The above problem is solved using the numerical method of conjugate gradient iterative solver. The mesh spacings used is

$$h = \left\{ \frac{1}{16}, \frac{1}{32}, \frac{1}{64}, \frac{1}{128} \right\}$$

and the tolerance used to check for convergence is

$$\varepsilon = 10^{-6}$$

The solver terminates when the mesh norm of the residual becomes smaller than the above quantity using the following check

$$\sqrt{h}\left\| r^{(k)} \right\|_2 < \varepsilon$$

Where in the above, $r^{(k)}$ is the residual at the k^{th} iteration and h is the current value of the

mesh spacing.

The reason for using *zero* as the driving force on the RHS of the pde, is to allow the calculation and tracking of the error at each iteration as the exact solution u_{exact} for this problem is now known, which is zero. Now the error at each iteration k to be found using

$$\begin{aligned} e^{(k)} &= \left\| u_{exact} - u^{(k)} \right\| \\ &= \left\| u^{(k)} \right\| \end{aligned}$$

Where in the above $u^{(k)}$ represents the approximate solution at the k^{th} iteration.

A Matlab function *CG.m* was written to implement conjugate gradient solver. One of the parameters this function accepts is the name of the preconditioner to use. The following preconditioners are supported

NONE, SSOR, MultiGrid, IncompleteCholesky

When NONE is specified, then no preconditioning is done.

For each preconditioner, the solver was run to find the solution to the above test problem. The initial guess for the solution $u^{(0)}$ used was generated using Matlab rand() function.

For each preconditioner the following plots were generated

1. Plot of error $\left\| e^{(k)} \right\|$ per iteration k which showed how the rate of error reduction per iteration. The plot was generated in log and linear scale.
2. Plot of the residual $\left\| r^{(k)} \right\|$ per iteration which showed the rate of residual norm reduction per iteration, and also plotted in log and linear scale. The initial residual is defined as $r^{(0)} = f - Au^{(0)}$ and each subsequent iteration, the residual is defined as $r^{(k)} = r^{(k-1)} - \alpha A p^{(k-1)}$. The algorithm below illustrates this in more details.
3. The spectrum of the eigenvalues of A and the spectrum of the eigenvalues of $M^{-1}A$ are plotted using matlab's scatter() command to better see the effect on the condition number value when multiplying A by M^{-1}, where M is the preconditioner matrix.
4. Plot of the final solution found on a 3D mesh plot. The final solution was verified to be close to zero, which is the same as the exact solution.

In addition to the above plots, for each mesh spacing h, the actual result table is printed which tabulates the above values at each iteration. This table was used to generate the above plots. The printed tables also show the ratio of the value of norm of the residual at the current iteration to its value at the previous iteration, similarly for the error norm.

Due to the large size of these tables, the tables for all the spacings and for each solver are available in the appendix.

Conjugate gradient algorithm description

The idea of conjugate gradient is to use preconditioning matrix to speed up the convergence of the conjugate gradient method. The original problem

$$Ax = f$$

is transformed to a new problem

$$M^{-1}Ax = M^{-1}f$$

such that $M^{-1}A$ has a smaller condition number than A. For most iterative solvers, the rate of convergence increases as the condition number of the system matrix A decreases.

The conjugate gradient method works only on symmetric positive definite A matrix, and its speed of convergence is affected by the distribution of the eigenvalues of the A matrix. The estimate of convergence is more accurate if the distribution of eigenvalues is uniform. For the discrete 2D Poisson problem, this is the case, as verified by plots of the spectrum generated below for each case.

The preconditioning is used to modify the spectrum of A so that the eigenvalues of the new system matrix $M^{-1}A$ become more clustered together causing the condition number to become smaller and thus increasing the convergence rate.

The following table was generated to show the effect of preconditioning on lowering the condition number. It shows the condition number for CG (in other words, for the A matrix only), and then the condition number for $M^{-1}A$ for different solvers as mesh spacing is changed. It also shows below the condition number value, in a box, the maximum eigenvalue and the minimum eigenvalue. Notice that in the following table, if one tries to apply the $\frac{|\max \lambda|}{|\min \lambda|}$ to determine the condition number, then the result will not match the condition number as shown. The above formula do not apply in this case, as these are sparse matrices and the condition number was found by estimating its value using the Matlab function condest() and not by applying the above formula.

solver	$h=\frac{1}{16}, N=225$	$h=\frac{1}{32}, N=961$	$h=\frac{1}{64}, N=3969$	$h=\frac{1}{128}, N=16129$
NONE	150, (7.923, 0.0768)	603, (7.9807, 0.01926)	2413, (7.9995, 0.0048)	9655, (7.9987, 0.001205)
SSOR	33, (0.25, 0.018203)	128, (0.25, 0.004748)	511, (0.25, 0.0012)	
IncCholesky $\varepsilon = 10^{-2}$	32, (2.365, 0.2279)	108, (2.44, 0.0585)	422, (2.537, 0.0146)	
IncCholesky $\varepsilon = 10^{-3}$	48, (2.337, 0.508)	153, (2.526, 0.1603)	373, (2.592, 0.041756)	

This diagram below reflects the above table result to clearly show the reduction of the condition number as a result of preconditioning.

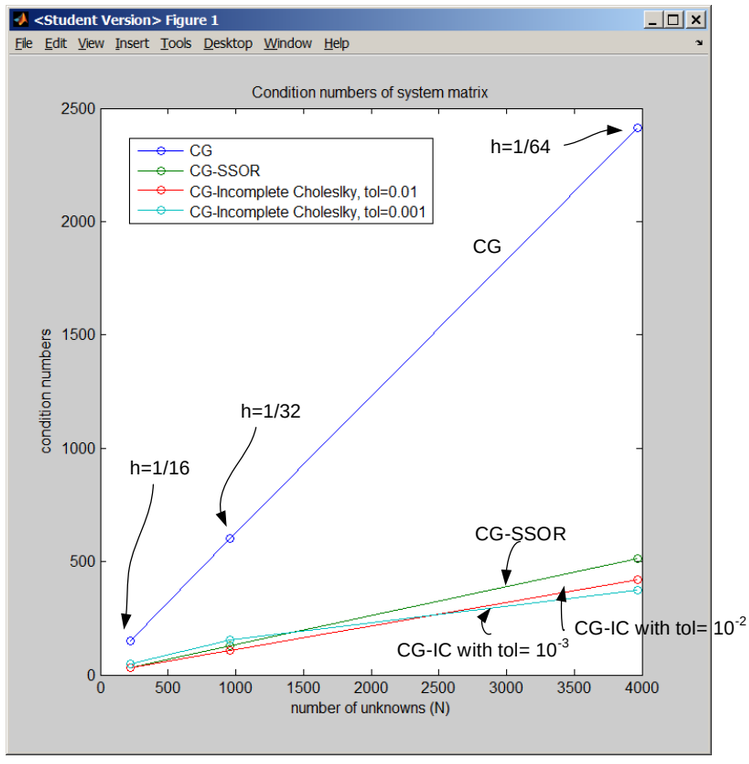


Figure 2.59: compare condition numbers

CG Algorithm pseudocode

The following is the algorithm used for the implementation of conjugate gradient with precondi-

tioning.

$$
\begin{aligned}
&\textit{Input}: \ A, f, tol, preconditionSolverName, dropTol\\
&\textit{Output}: \ \tilde{u} \ \textit{approximate solution to} \ Ax = 0\\
&\\
&u_0 = \text{rand}() \ (*\textit{initial guess of solution} *)\\
&r_0 = f - Au_0 \ (*\textit{initial residual}*)\\
&z_0 \leftarrow CALL \ \ preconditionSolver(r_0, A, preconditionSolverName, dropTol)\\
&p_0 = z_0\\
&FOR \ k = 1, 2, 3, \cdots\\
&\qquad e_{k-1} = \sqrt{h}\,\|u_{k-1}\|_2 \ \ (* \ \textit{the error since} \ u_{exact} \ \textit{is known to be zero}*)\\
&\qquad \omega_{k-1} = Ap_{k-1}\\
&\qquad \alpha_{k-1} = \frac{z_{k-1}^T r_{k-1}}{p_{k-1}^T \omega_{k-1}}\\
&\qquad u_k = u_{k-1} + \alpha_{k-1}\, p_{k-1}\\
&\qquad r_k = r_{k-1} - \alpha_{k-1}\omega_{k-1}\\
&\qquad IF \ \left(\sqrt{h}\,\|r_k\|_2\right) < tol \ THEN\\
&\qquad\qquad RETURN \ u_k\\
&\qquad END \ IF\\
&\qquad z_k \leftarrow CALL \ \ preconditionSolver(r_k, A, preconditionSolverName, dropTol)\\
&\qquad \beta_{k-1} = \frac{z_k^T r_k}{z_{k-1}^T r_{k-1}}\\
&\qquad p_k = z_k + \beta_{k-1} p_{k-1}\\
&END \ FOR
\end{aligned}
$$

The algorithm for the function *preconditionSolver*() is as follows

$$
\begin{aligned}
&\textit{Input}: \ r, A, preconditionSolverName, dropTol\\
&\textit{Output}: \ z \ \textit{approximate solution to} \ Mz = r\\
&CASE \ preconditionSolverName \ IS\\
&\qquad WHEN \ NONE \ z \leftarrow r \qquad \textit{// no preconditioning}\\
&\qquad WHEN \ MultiGrid\\
&\qquad\qquad \mu_1 = \mu_2 = 1 \ (* \ \textit{presmoother and postsmoother}*)\\
&\qquad\qquad z \leftarrow CALL \ VCYCLE(zeroInitialGuess, r)\\
&\qquad\qquad\quad \textit{//VCYCLE is one implemented in HW4 but changed to do}\\
&\qquad\qquad\quad \textit{//one forward Gauss – Seidel/red – black followed by}\\
&\qquad\qquad\quad \textit{//one reverse Gauss – Seidel/red – black}\\
&\qquad WHEN \ \ SSOR\\
&\qquad\qquad z \leftarrow CALL \ SOR \ forward \ followed \ by \ SOR \ in \ reverse\\
&\qquad WHEN \ IncompleteCholesky\\
&\qquad\ R = cholinc(A, dropTol)\\
&\qquad\ z \leftarrow R\backslash(R^T\backslash r)\\
&END \ CASE\\
&RETURN \ z
\end{aligned}
$$

2.6.2 Solvers efficiency and iterations count

In addition to finding the number of iterations needed for convergence by each solver, the problem also asked to compare the efficiency of each solver. This is done by finding the work needed by each solver to converge.

Work needed is defined as

$$Work = NumberOfIterations \ \times \ WorkPerIteration$$

Before determining the work for each solver, the following table lists the *cputime* used by each solver for the different spacings. The cpu time is measured using Matlab cputime function, and measures only the call to CG() and does not include any other calls such as plotting.

preconditioning	$h=\frac{1}{16}$ $N=225$	$h=\frac{1}{32}$ $N=961$	$h=\frac{1}{64}$ $N=3969$	$h=\frac{1}{128}$ $N=16129$
NONE	0.19	0.37	13.48	556.6
Multigrid	0.34	0.6	13.48	566
SSOR	0.23	0.5	13.3	564.6
Incomplete Cholesky $\varepsilon = 10^{-2}$	0.16	0.34	13.8	559
Incomplete Cholesky $\varepsilon = 10^{-3}$	0.19	0.5	13.3	559

Surprisingly, no appreciable difference can be seen between the different solvers in terms of cpu time. It was expected that NONE would have the largest CPU time as it has the lowest efficiency. This result can be attributed to using small number of N values, which was not large enough in the limit to reflect the difference. One needs to use much larger values of N to see the effect of preconditioning on CPU time difference. Due to memory limitation, this was not possible to implement at this time. Now the work per iteration is analyzed.

2.6.2.1 Work per iteration

All solvers perform similar work per iterations except for the step needed to apply the preconditioning to determine z_k. The only difference between not using preconditioning and using one, is in the step to solve for z in $Mz = r$. Using work per iteration as $O(N)$ for the base CG with no preconditioning, then the following can be defined for work per iteration for each solver:

1. No preconditioner is applied: no extra work is needed, as z is the same as r hence $O(N)$
2. *multigrid* : work needed to determine z adds an extra cost of one V cycle. Work for one V cycle was found from HW4 to be $\frac{4}{3}C\times N$ where N is number of unknowns and C is a constant estimated to be $(7(v_1+v_2)+13)$ where v_1, v_2 are the numbers of pre smooth and post smooth operations. These are both *one* in this case. The smoothing is done twice (forward and reverse), hence the above becomes $(2\times 7(v_1+v_2)+13)$, resulting in work per iteration of $\frac{4}{3}(2\times 7(v_1+v_2)+13)N$ or about $55N$. Adding the $O(N)$ from the above, this is still results in $O(N)$.
3. *SSOR* : The cost is twice one SOR step. One step of SOR work is $7N$, where N is number of unknowns, since it takes about 7 flop operations to smooth one grid point, and there are N of these. Hence for SSOR work is twice that or $14N$. As above, this is still an order of N.

2.6.2.2 Number of iterations

From lectures notes, it was found that the error rate in conjugate gradient (with no preconditioning) behaves as

$$\|e_k\|_A \leq 2\frac{\sqrt{\mathbb{k}(A)}-1}{\sqrt{\mathbb{k}(A)}+1}\|e_0\|_A$$

Where $\mathbb{k}(A)$ is the condition number of A, it was shown that $\mathbb{k}(A) = O\left(h^{-1}\right)$ where h is the mesh spacing. Hence, for fixed tolerance, which is the case here, the number of iterations is $O\left(h^{-1}\right) = O\left(N^{1/2}\right)$ where N is number of unknowns.

The results of the numerical experiment done agrees with the above, as shown below, for the case of NONE (which is the case of conjugate gradient with no preconditioning).

The following table was generated from result of running the program. In this table, N is the number of unknowns, $\mathbb{k}\left(M^{-1}A\right)$ is the condition number of $M^{-1}A$, and $\mathbb{k}(A)$ is the condition number of A. Since sparse matrices are used, the Matlab function condest() was used to find the condition numbers. In this table I.C. means Incomplete Cholesky

preconditioner	$h=\frac{1}{16}$ $N=225$	$\mathbb{k}\left(M^{-1}A\right)$	$\mathbb{k}(A)$	$h=\frac{1}{32}$ $N=961$	$\mathbb{k}\left(M^{-1}A\right)$	$\mathbb{k}(A)$
NONE	42	N/A	150	82	N/A	603
Multigrid	4		150	4		603
SSOR	18	33	150	30	128	603
IC $\varepsilon = 10^{-2}$	7	32	150	13	108	603
IC $\varepsilon = 10^{-3}$	4	48	150	6	153	603

preconditioner	$h=\frac{1}{64}$ $N=3969$	$\mathbb{k}\left(M^{-1}A\right)$	$\mathbb{k}\left(A\right)$	$h=\frac{1}{128}$ $N=16129$	$\mathbb{k}\left(M^{-1}A\right)$	$\mathbb{k}\left(A\right)$
NONE	157	N/A	2413	291	N/A	9655
Multigrid	4		2413	4		9655
SSOR	56	511	2413	103	Memory problem	9655
IC $\varepsilon=10^{-2}$	22	422	2413	39	Memory problem	9655
IC $\varepsilon=10^{-3}$	8	373	2413	14	Memory problem	9655

The following is a plot that represents the above results.

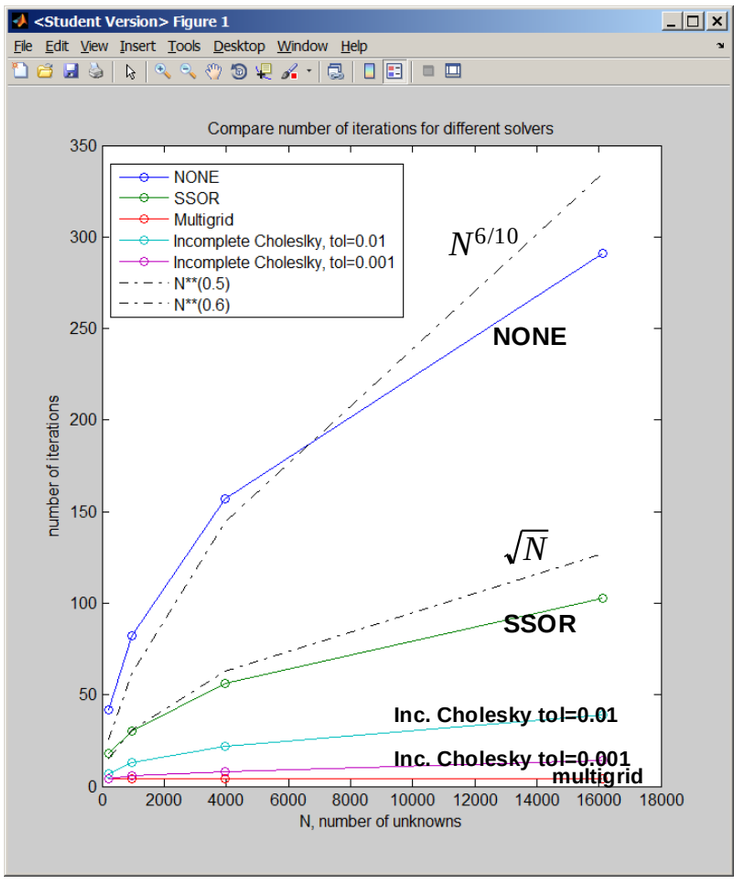


Figure 2.60: iterations plot

From the above one can see that multigrid has $O\left(1\right)$ for the number of iterations. The number of iterations was 4 for all the cases from $h=\frac{1}{16}$ to $h=\frac{1}{128}$. For SSOR, the number of iterations grew sublinear in terms of N, from the above one can estimate this to be $O\left(N^{1/4}\right)$, while for no preconditioning, the number of iterations grew as approximately as $O\left(N^{\frac{1}{2}}\right)$ as predicted by earlier analytical result.

2.6.3 Discussion of results and conclusions

The use of preconditioning on A caused a reduction of the number of iterations to convergence to the same fixed tolerance when compared to convergence with no preconditioning. This was due to reduction of the condition number of the system matrix as can be seen in the above table. By reducing the largest eigenvalue, the rate of convergence increased. However, preconditioning also adds an extra cost per iteration. The extra work however, was also of order N and hence the final efficiency was governed by the number of iterations for large N.

Therefore, this is the result of work efficiency for the main solvers, using the formula of

$$\text{Work} = \text{NumberOfIterations} \times \text{WorkPerIteration}$$

1. NONE: $O\left(N^{1/2}\right)\times O\left(N\right)=O\left(N^{3/2}\right)$
2. SSOR: $O\left(N^{1/4}\right)\times O\left(N\right)=O\left(N^{5/4}\right)$

3. Multigrid: $O(1) \times O(N) = O(N)$

Incomplete Cholesky was not added as it was hard to estimate from the curve above the number of iterations and since depend on the drop tolerance values.

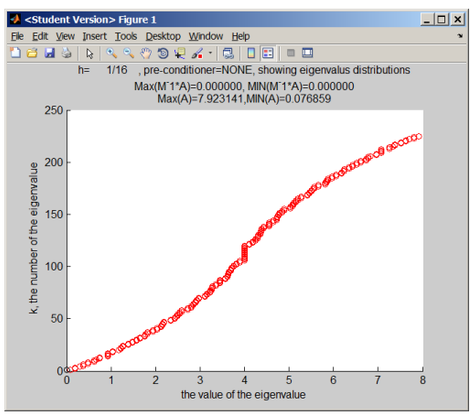
The above analysis showed that **Multigrid is most efficient**, followed by Incomplete Cholesky (but these depend on the tolerance drop term), followed by SSOR, and finally by CG with no preconditioning

In conclusion, using Multigrid for preconditioner for conjugate gradient seems to be the most effective solver.

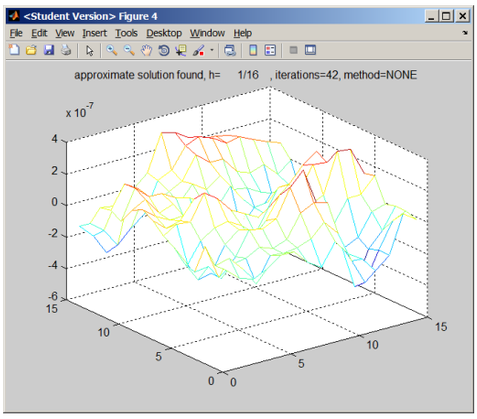
2.6.4 Appendix

2.6.4.1 Result for CG with no preconditioner

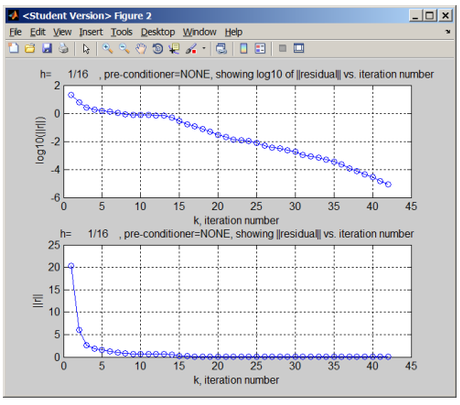
Plots h=1/16



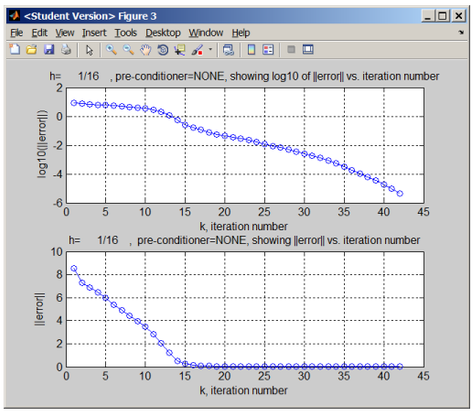
Eigenvalues of A



Final solution



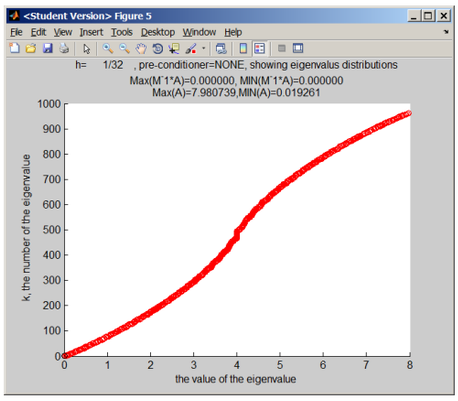
Residual per iteration



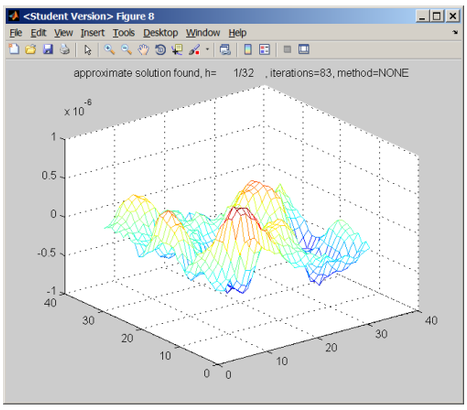
ERROR per iteration

Figure 2.61: solver none plots 16

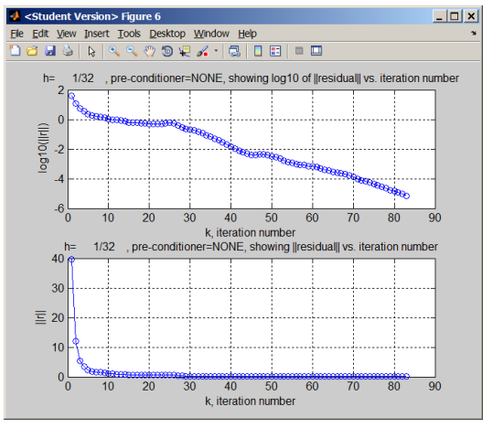
Plots for h=1/32



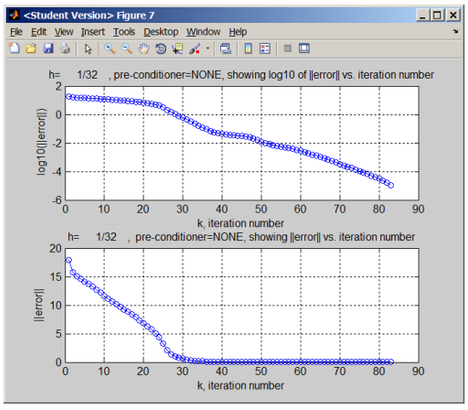
Eigenvalues of A



Final solution



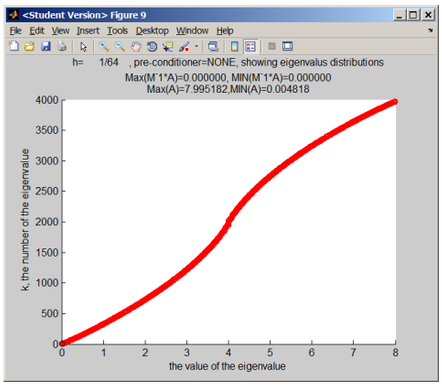
Residual per iteration



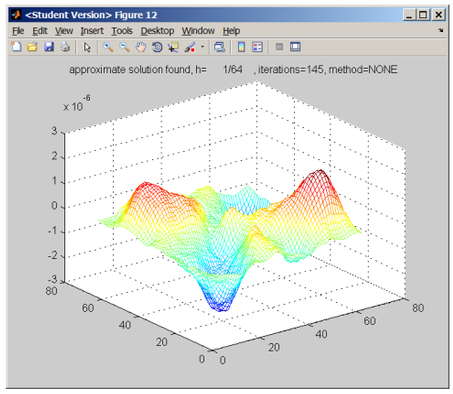
ERROR per iteration

Figure 2.62: solver none plots 32

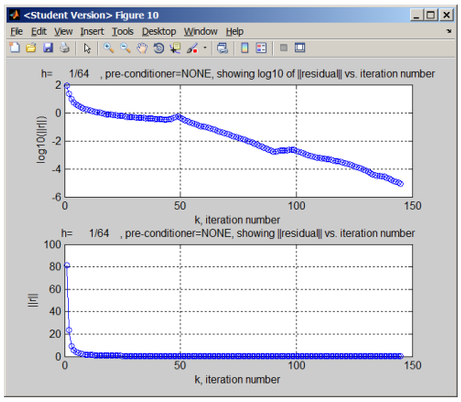
Plot for h=1/64



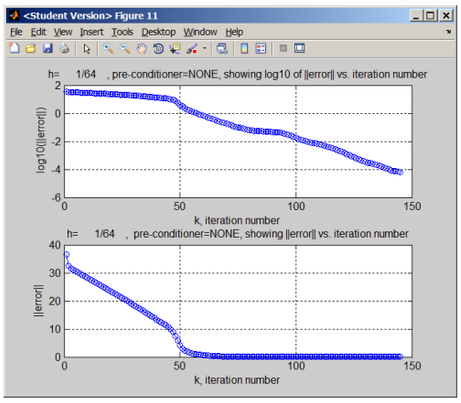
Eigenvalues of A



Final solution



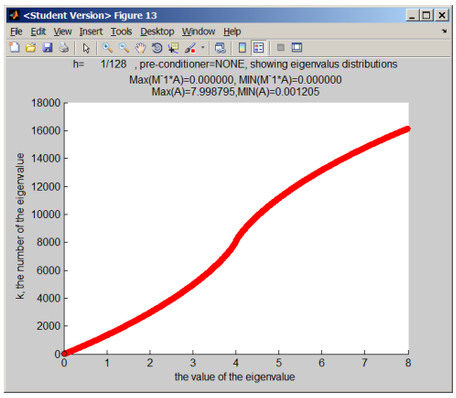
Residual per iteration

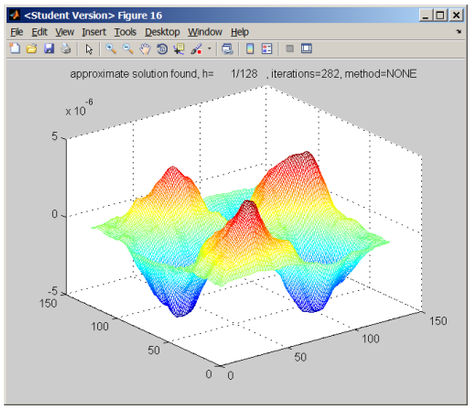


ERROR per iteration

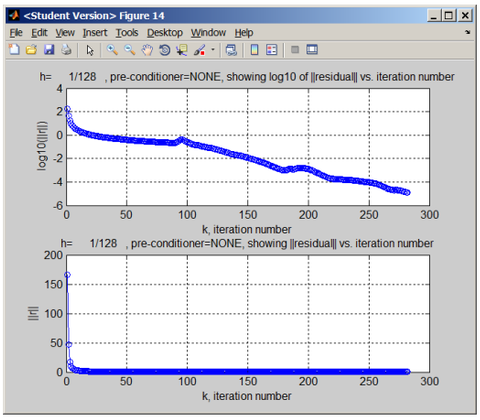
Figure 2.63: solver none plots 64

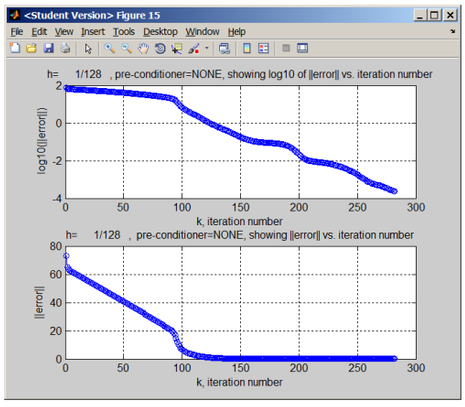
Plots for h=1/128





Eigenvalues of A **Final solution**



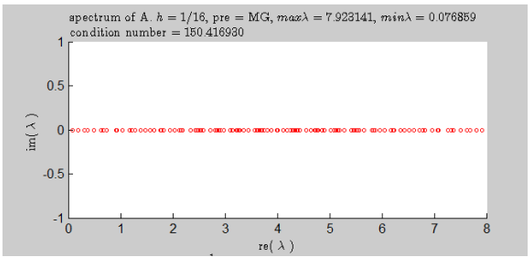


Residual per iteration **ERROR per iteration**

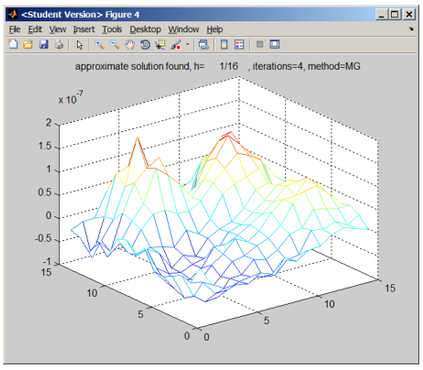
Figure 2.64: solver none plots 128

2.6.4.2 Result for CG with Multigrid preconditioner

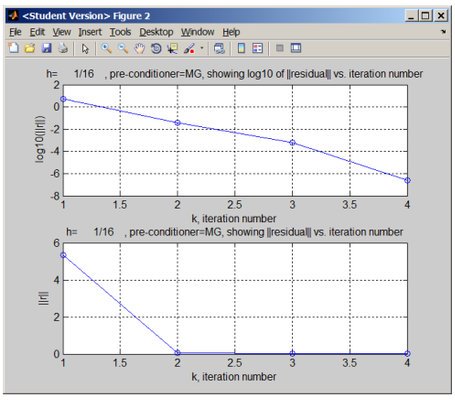
Plots for h=1/16



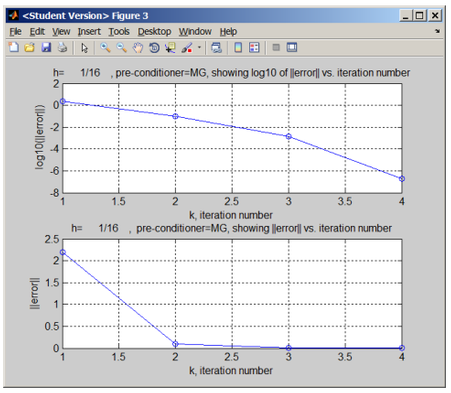
Eigenvalues of A



Final solution



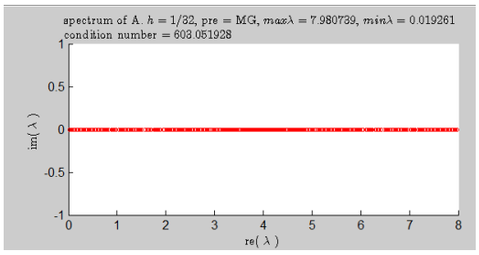
Residual per iteration



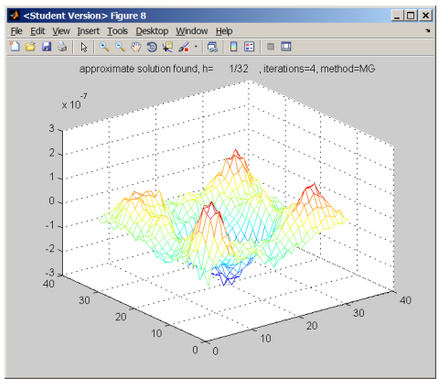
ERROR per iteration

Figure 2.65: solver MG plots 16

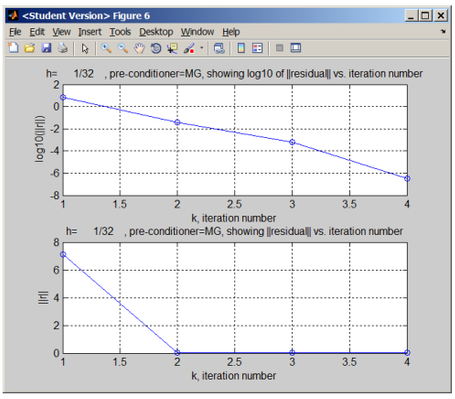
Plots for h1/32



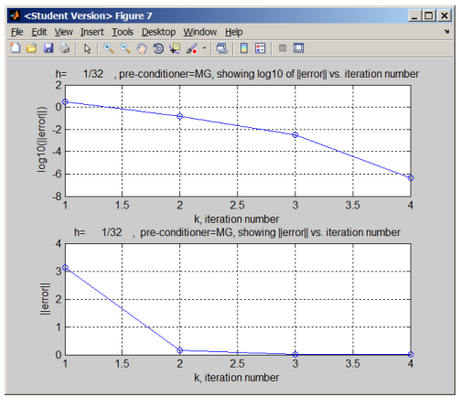
Eigenvalues of A



Final solution



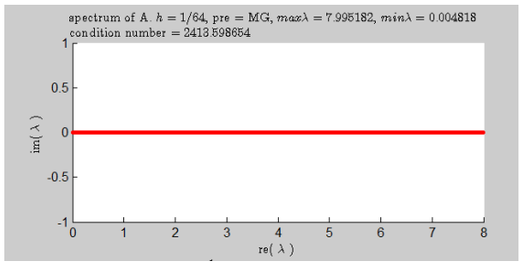
Residual per iteration



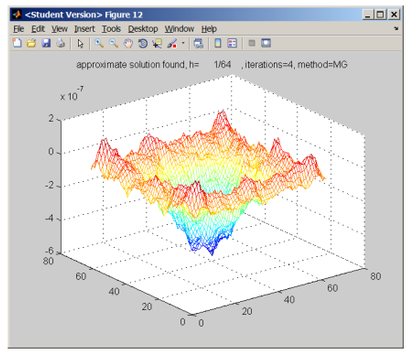
ERROR per iteration

Figure 2.66: solver MG plots 32

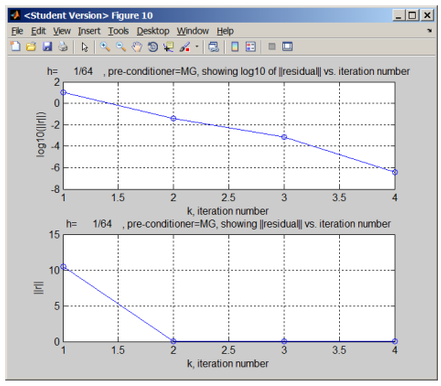
Plots for h=164



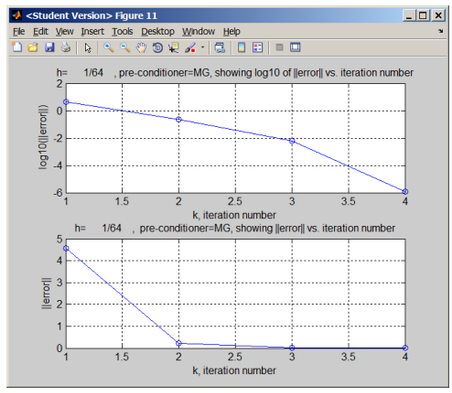
Eigenvalues of A



Final solution



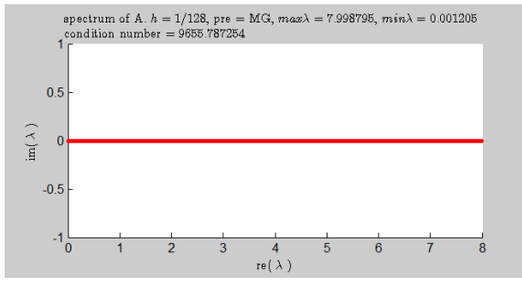
Residual per iteration



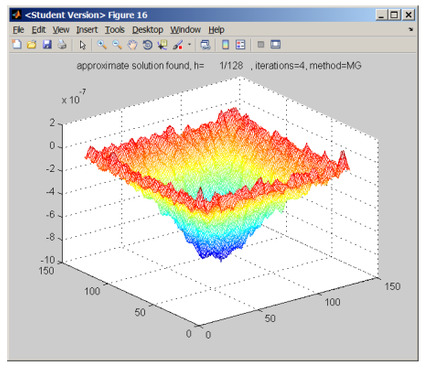
ERROR per iteration

Figure 2.67: solver MG plots 64

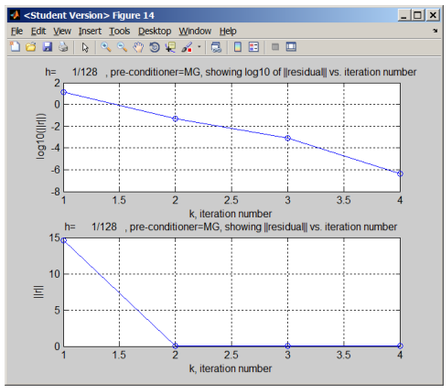
Plots for h=1/128



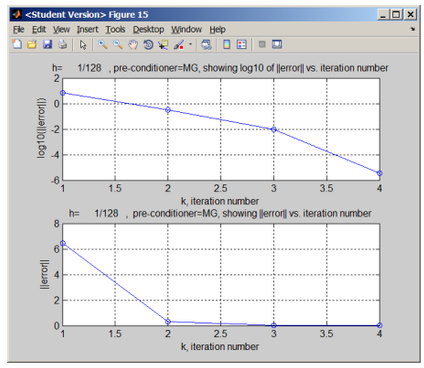
Eigenvalues of A



Final solution



Residual per iteration

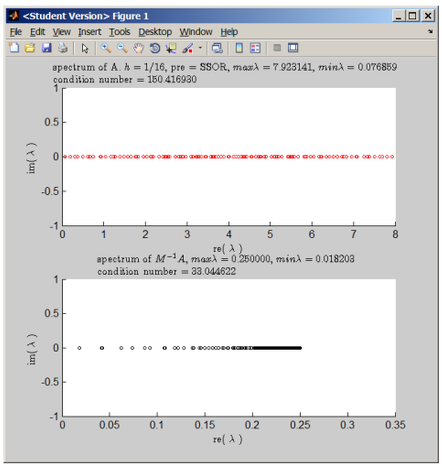


ERROR per iteration

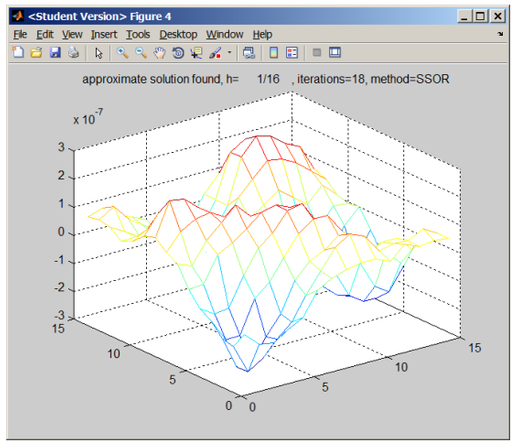
Figure 2.68: solver MG plots 128

2.6.4.3 Result for CG with SSOR preconditioner

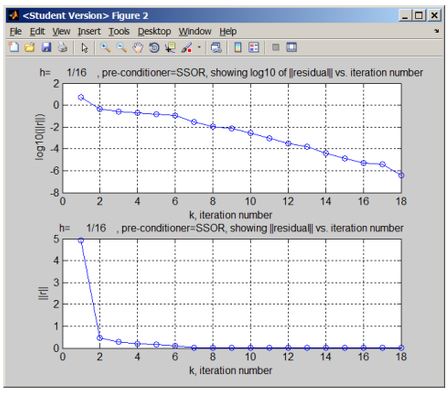
Plots for h=1/16



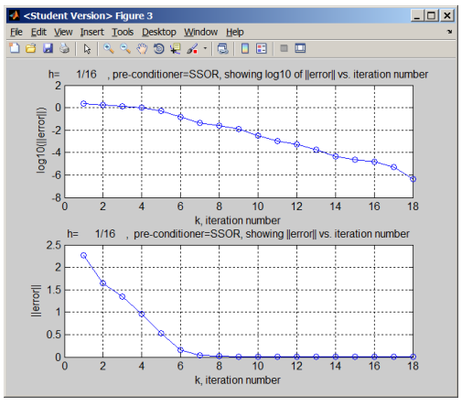
Eigenvalues of A and M^{-1}*A



Final solution



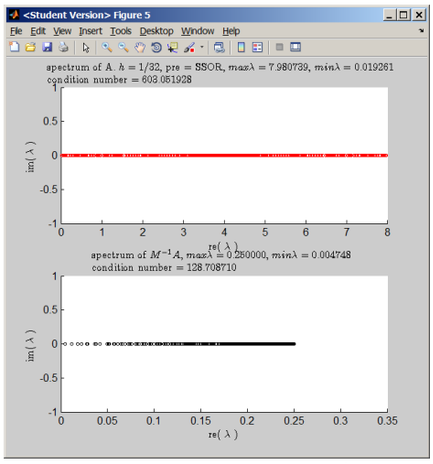
Residual per iteration



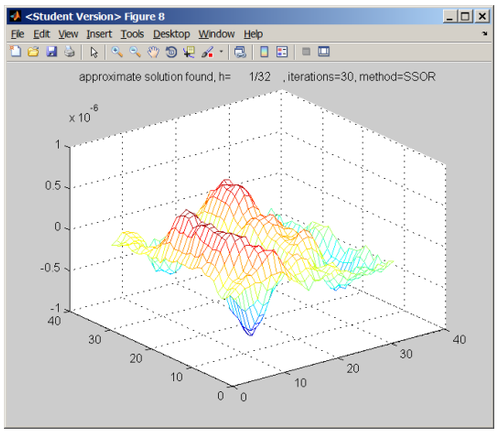
ERROR per iteration

Figure 2.69: solver SSOR plots 16

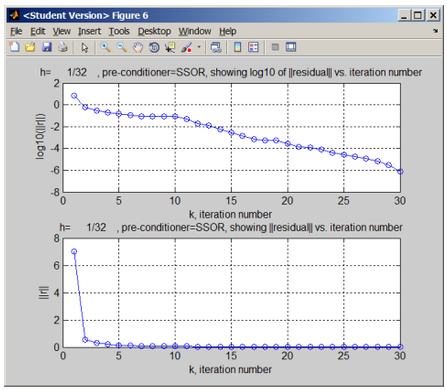
Plots for h=1/32



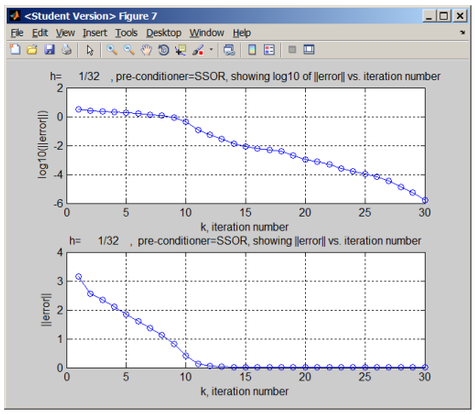
Eigenvalues of A and M^{-1}*A



Final solution



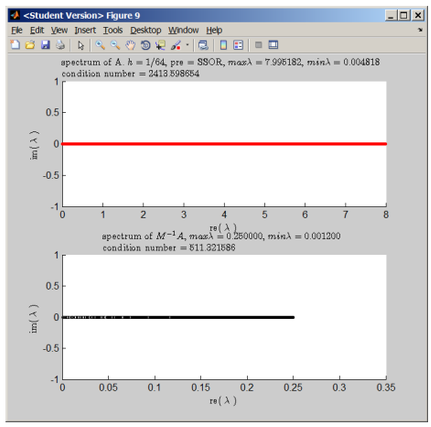
Residual per iteration



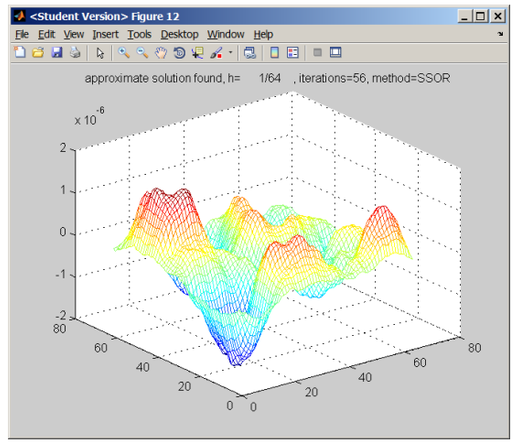
ERROR per iteration

Figure 2.70: solver SSOR plots 32

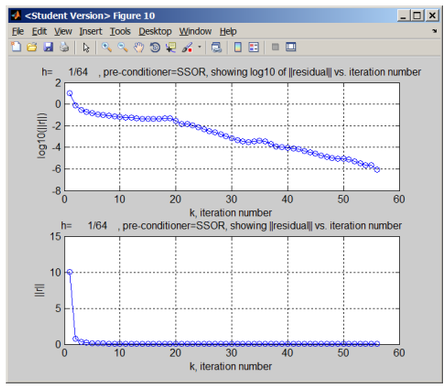
Plots for h=1/64



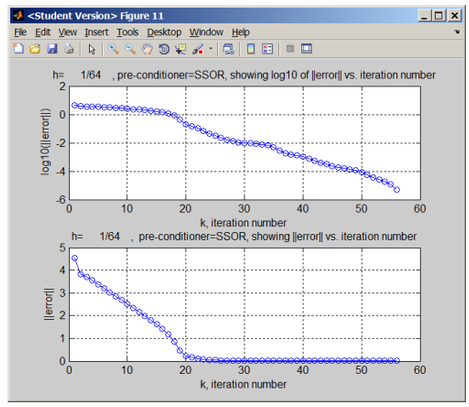
Eigenvalues of A and M^{-1}*A



Final solution



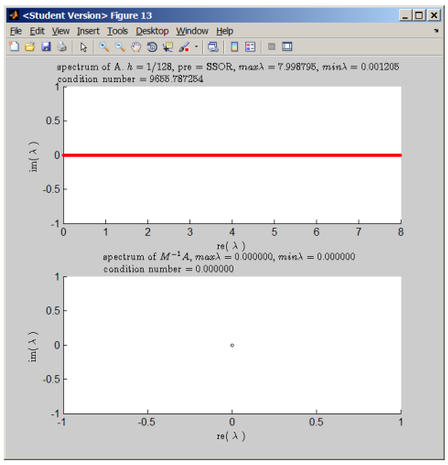
Residual per iteration



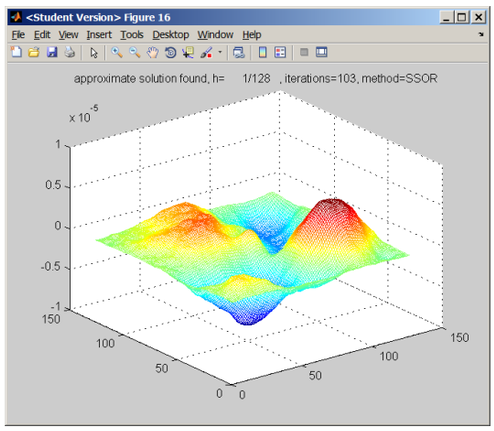
ERROR per iteration

Figure 2.71: solver SSOR plots 64

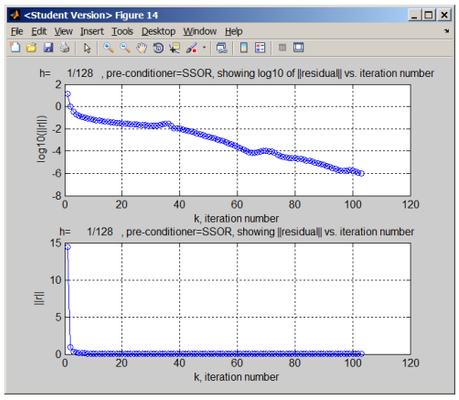
2.6.4.4 Plots for h=1/128



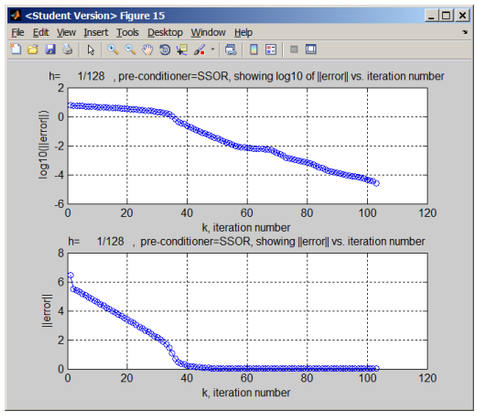
Eigenvalues of A



Final solution



Residual per iteration

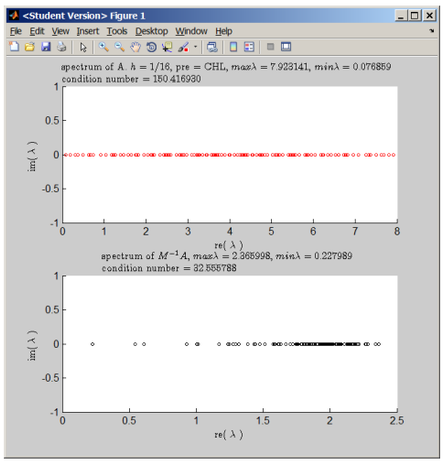


ERROR per iteration

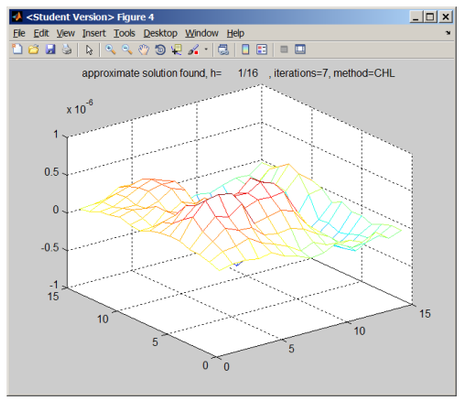
Figure 2.72: solver SSOR plots 128

2.6.4.5 Result for CG with incomplete cholesky preconditioner $\varepsilon = 10^{-2}$

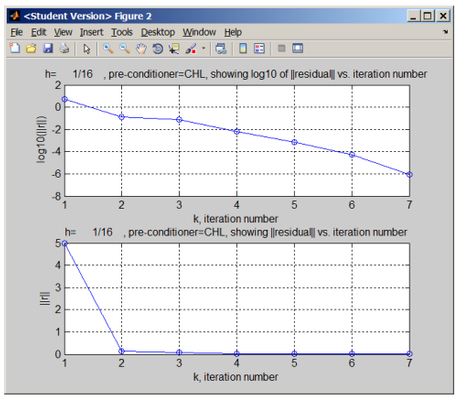
Plots for h=1/16



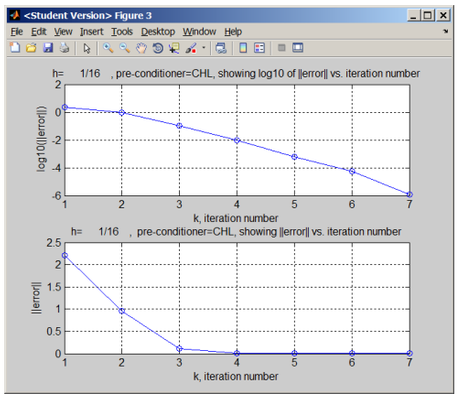
Eigenvalues of A and M^{-1}*A



Final solution



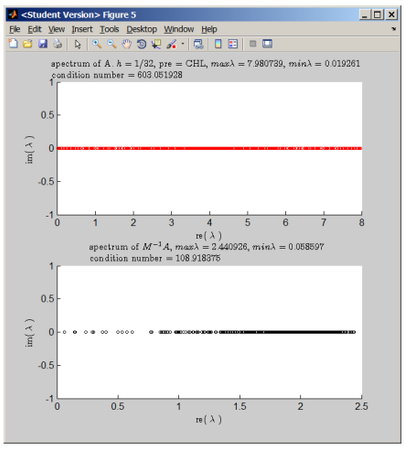
Residual per iteration



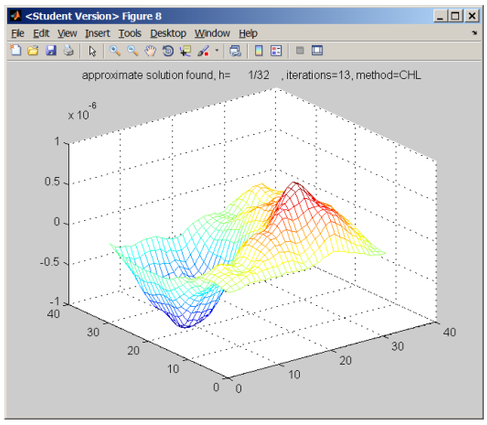
ERROR per iteration

Figure 2.73: solver incomplete cholesky plots 16

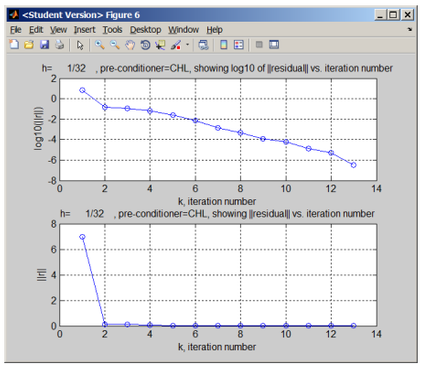
Plots for h=1/32



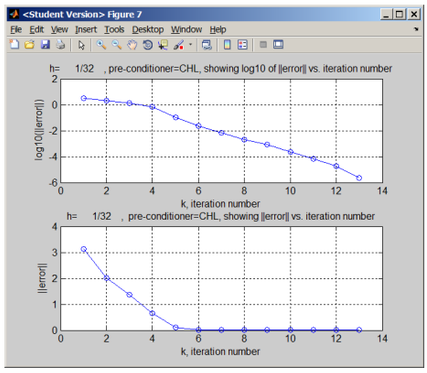
Eigenvalues of A and M^{-1}*A



Final solution



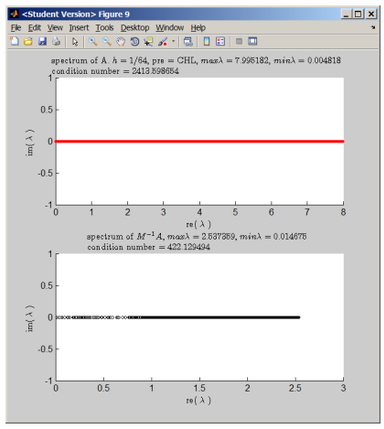
Residual per iteration



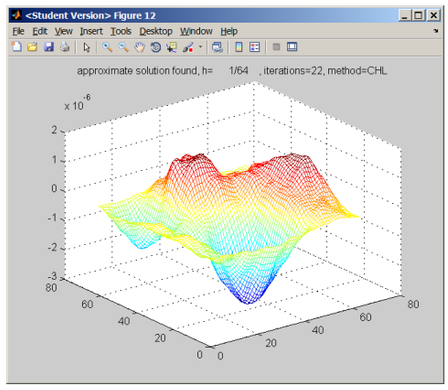
ERROR per iteration

Figure 2.74: solver incomplete cholesky plots 32

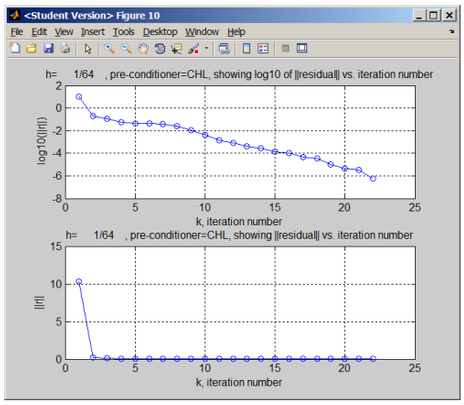
Plots for h=1/64



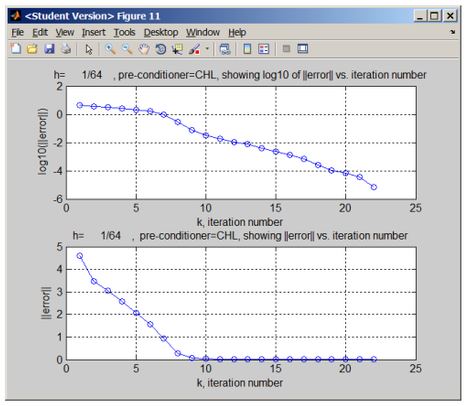
Eigenvalues of A and M^{-1}*A



Final solution



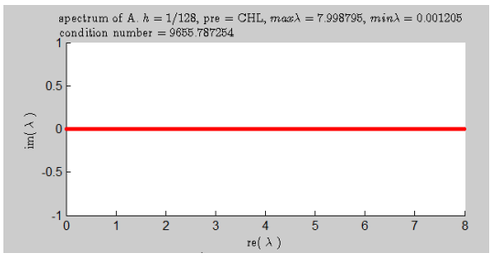
Residual per iteration



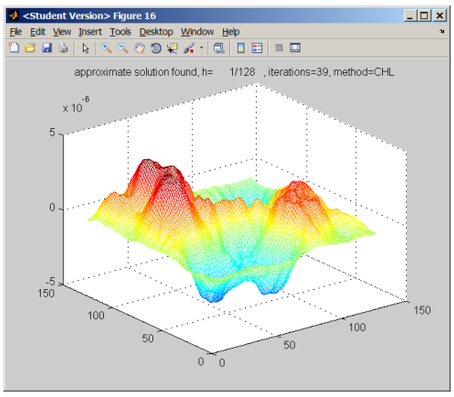
ERROR per iteration

Figure 2.75: solver incomplete cholesky plots 64

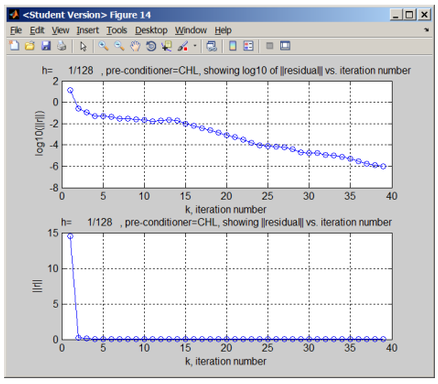
Plots for h=1/128



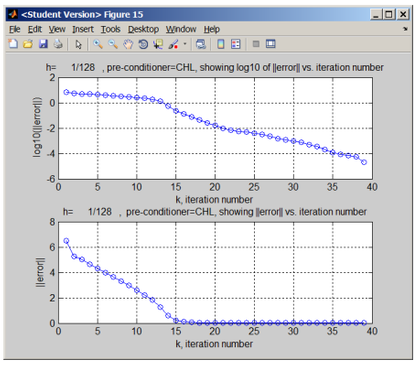
Eigenvalues of A



Final solution



Residual per iteration

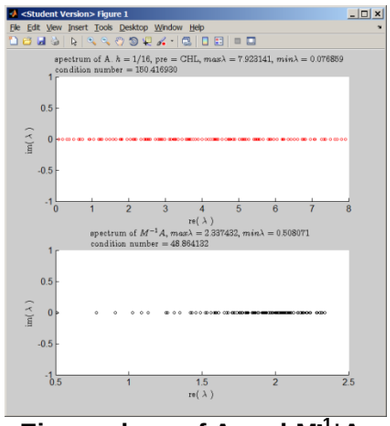


ERROR per iteration

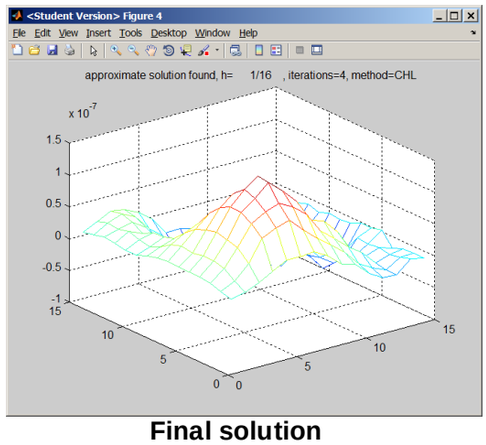
Figure 2.76: solver incomplete cholesky plots 128

2.6.5 Result for CG with incomplete cholesky preconditioner $\varepsilon = 10^{-3}$

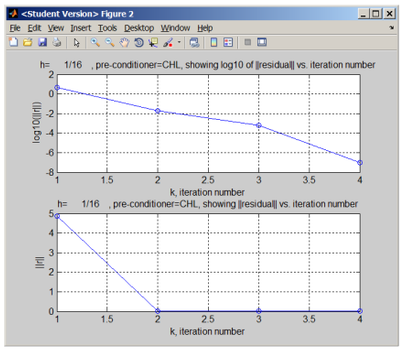
Plots for h=1/16



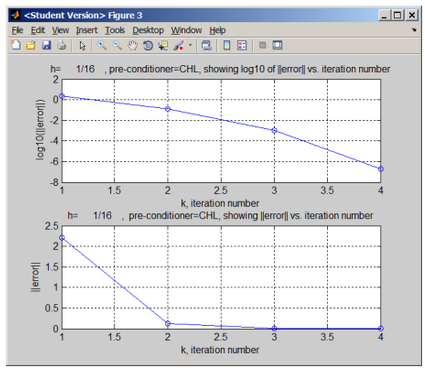
Eigenvalues of A and M^{-1}*A



Final solution



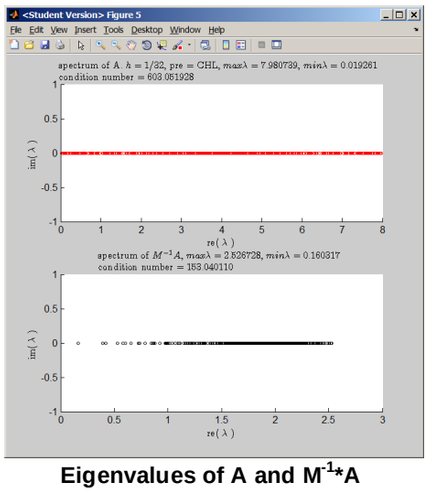
Residual per iteration

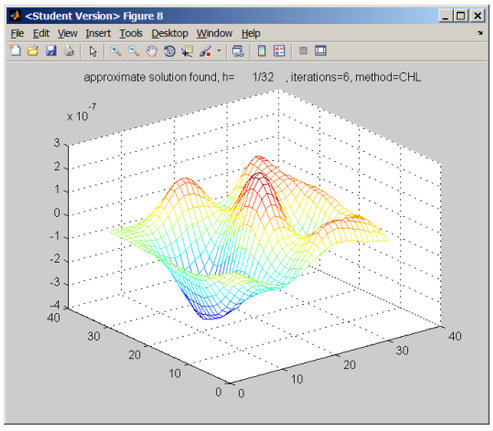


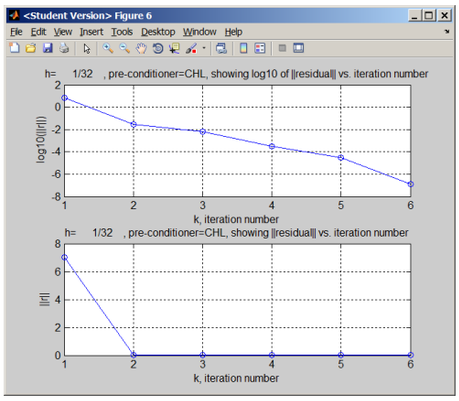
ERROR per iteration

Figure 2.77: solver CG with incomplete cholesky preconditioner 16

underlinePlots for h=1/32







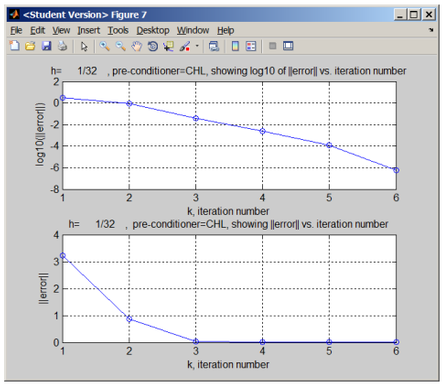
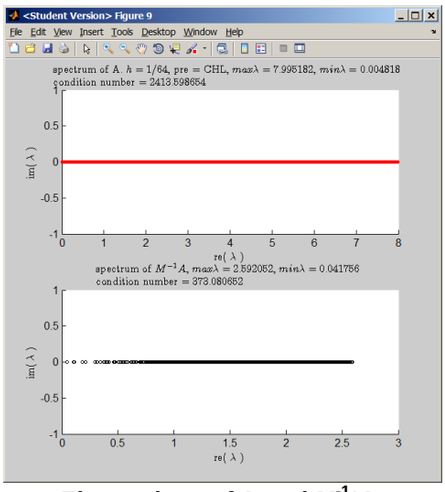
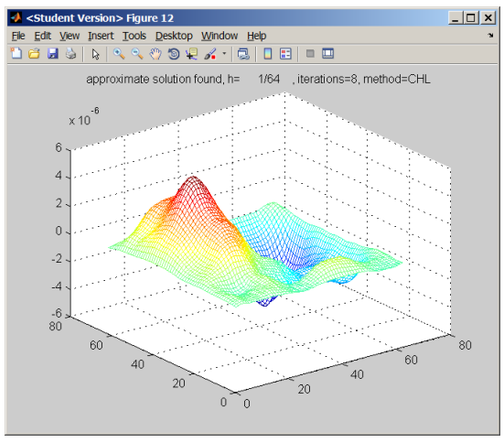


Figure 2.78: solver CG with incomplete cholesky preconditioner 32

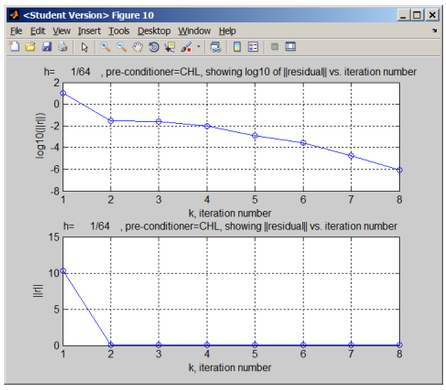
Plots for $h = \frac{1}{64}$



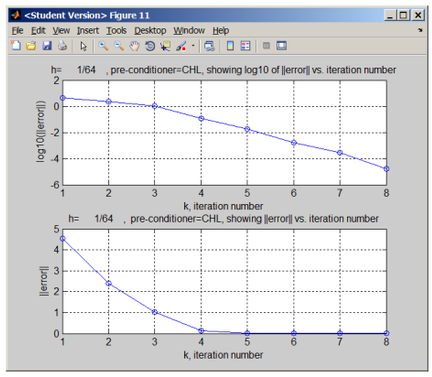
Eigenvalues of A and M^{-1}*A



Final solution



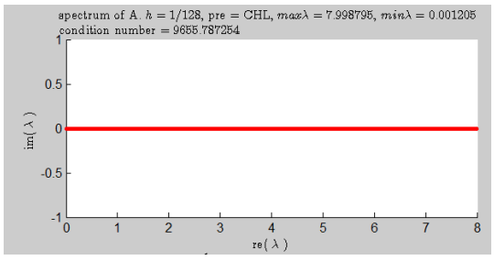
Residual per iteration



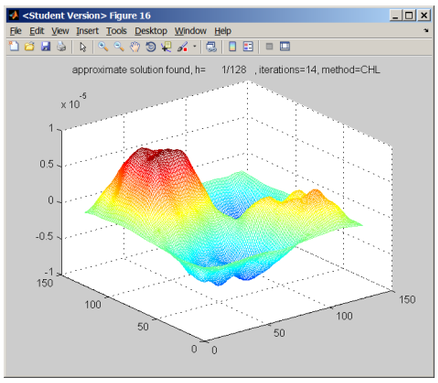
ERROR per iteration

Figure 2.79: solver CG with incomplete cholesky preconditioner 64

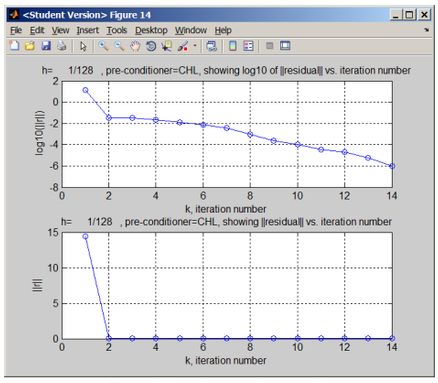
Plots for h=1/128



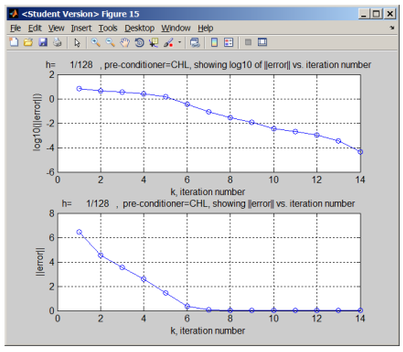
Eigenvalues of A



Final solution



Residual per iteration



ERROR per iteration

Figure 2.80: solver CG with incomplete cholesky preconditioner 128

2.6.5.1 References

1. R. J. LeVeque. Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems. SIAM, 2007.

2.6.5.2 Computation Tables

table for CG with no preconditioner

table for CG with no preconditioner $h = \frac{1}{16}$

Tolerance=10^{-6} method=NONE, Iterations = 42, condition number A=150.4167

k	|e|	ratio	|r|	ratio
1	2.1830924	0.0000000	4.7210582	0.0000000
2	1.8864972	0.8641399	1.4184825	0.3004586
3	1.7843339	0.9458449	0.6424897	0.4529416
4	1.6861080	0.9449509	0.4396146	0.6842360
5	1.5849468	0.9400032	0.3336348	0.7589256
6	1.4553992	0.9182637	0.3163522	0.9481991
7	1.2957415	0.8902998	0.2736152	0.8649068
8	1.1409611	0.8805468	0.2420287	0.8845587
9	0.9977301	0.8744646	0.2038050	0.8420696
10	0.8499980	0.8519318	0.2051285	1.0064937
11	0.6651595	0.7825425	0.1974297	0.9624684
12	0.4864479	0.7313251	0.1747116	0.8849309
13	0.2960128	0.6085191	0.1779989	1.0188155
14	0.1451740	0.4904313	0.1199788	0.6740421
15	0.0935858	0.6446460	0.0678391	0.5654261
16	0.0756963	0.8088434	0.0408147	0.6016397
17	0.0635298	0.8392727	0.0298095	0.7303614
18	0.0529339	0.8332136	0.0218161	0.7318495
19	0.0429735	0.8118324	0.0195722	0.8971481
20	0.0322133	0.7496102	0.0163238	0.8340275
21	0.0237202	0.7363457	0.0127609	0.7817369
22	0.0157525	0.6640959	0.0107769	0.8445245
23	0.0090001	0.5713465	0.0081136	0.7528675
24	0.0049041	0.5448943	0.0056646	0.6981578
25	0.0029273	0.5968981	0.0032193	0.5683292
26	0.0021383	0.7304863	0.0017564	0.5455764
27	0.0016007	0.7485962	0.0012297	0.7001280
28	0.0011635	0.7268656	0.0008660	0.7042553
29	0.0008013	0.6886615	0.0006405	0.7396406
30	0.0005365	0.6695259	0.0005187	0.8097440
31	0.0003581	0.6674555	0.0002751	0.5304362
32	0.0002692	0.7517878	0.0001920	0.6979471
33	0.0001902	0.7066626	0.0001491	0.7762729
34	0.0001158	0.6085937	0.0001143	0.7665146
35	0.0000648	0.5601207	0.0000746	0.6531095
36	0.0000377	0.5807215	0.0000446	0.5983047
37	0.0000238	0.6316540	0.0000268	0.5993857
38	0.0000137	0.5773472	0.0000177	0.6619224
39	0.0000077	0.5599609	0.0000104	0.5863875
40	0.0000043	0.5545265	0.0000060	0.5767504
41	0.0000023	0.5432272	0.0000034	0.5745691
42	0.0000005	0.2307583	0.0000009	0.2689011

Table CG with no preconditioner for $h = \frac{1}{32}$

Tolerance=10^{-6} method=NONE, Iterations =82, condition number A=603

k	|e|	ratio	|r|	ratio	
1	3.1348156	0.0000000	7.2393385	0.0000000	
2	2.7330660	0.8718427	2.0520496	0.2834582	
3	2.6175107	0.9577196	0.9150415	0.4459159	
4	2.5284570	0.9659777	0.6101126	0.6667595	
5	2.4421554	0.9658679	0.4386752	0.7190070	
6	2.3586462	0.9658051	0.3380251	0.7705588	
7	2.2766645	0.9652421	0.2853482	0.8441627	
8	2.1887867	0.9614006	0.2476061	0.8677331	
9	2.1039473	0.9612391	0.2079568	0.8398696	
10	2.0254995	0.9627140	0.1877177	0.9026765	

	11	1.9387591	0.9571758	0.1699200	0.9051889
	12	1.8539303	0.9562459	0.1545855	0.9097544
	13	1.7682926	0.9538075	0.1400435	0.9059291
	14	1.6834581	0.9520246	0.1278646	0.9130349
	15	1.6022568	0.9517652	0.1160407	0.9075285
	16	1.5217100	0.9497291	0.1094328	0.9430547
	17	1.4395506	0.9460085	0.1027356	0.9388016
	18	1.3562155	0.9421104	0.0942611	0.9175109
	19	1.2759187	0.9407934	0.0898504	0.9532076
	20	1.1866336	0.9300229	0.0872327	0.9708661
	21	1.0934573	0.9214784	0.0826619	0.9476025
	22	1.0024063	0.9167311	0.0788741	0.9541777
	23	0.8992085	0.8970499	0.0830311	1.0527040
	24	0.7631293	0.8486678	0.0970541	1.1688879
	25	0.5530580	0.7247239	0.1109983	1.1436745
	26	0.3359324	0.6074089	0.0961082	0.8658534
	27	0.2191717	0.6524281	0.0639449	0.6653430
	28	0.1626895	0.7422922	0.0484804	0.7581592
	29	0.1228014	0.7548210	0.0407252	0.8400342
	30	0.0917790	0.7473771	0.0335094	0.8228158
	31	0.0700724	0.7634914	0.0273614	0.8165289
	32	0.0540805	0.7717795	0.0214580	0.7842445
	33	0.0432646	0.8000041	0.0163945	0.7640291
	34	0.0364310	0.8420503	0.0126036	0.7687676
	35	0.0309924	0.8507149	0.0103112	0.8181153
	36	0.0268480	0.8662776	0.0080396	0.7796959
	37	0.0236146	0.8795683	0.0066579	0.8281358
	38	0.0209965	0.8891317	0.0052897	0.7945024
	39	0.0191928	0.9140930	0.0040810	0.7714996
	40	0.0177298	0.9237770	0.0033815	0.8285992
	41	0.0164277	0.9265582	0.0027478	0.8126066
	42	0.0154290	0.9392033	0.0020770	0.7558581
	43	0.0144978	0.9396484	0.0018150	0.8738906
	44	0.0134250	0.9260010	0.0016909	0.9316005
	45	0.0121487	0.9049280	0.0017116	1.0122194
	46	0.0103132	0.8489185	0.0019431	1.1352723
	47	0.0076740	0.7440911	0.0020798	1.0703591
	48	0.0052118	0.6791481	0.0017795	0.8556085
	49	0.0036674	0.7036863	0.0012802	0.7194204
	50	0.0028794	0.7851283	0.0009362	0.7313235
	51	0.0023421	0.8133830	0.0007764	0.8292528
	52	0.0019119	0.8163410	0.0006295	0.8108465
	53	0.0016151	0.8447753	0.0004538	0.7207790
	54	0.0014025	0.8683421	0.0003797	0.8368191
	55	0.0011871	0.8464182	0.0003408	0.8975492
	56	0.0009760	0.8221407	0.0002934	0.8609202
	57	0.0007699	0.7888788	0.0002736	0.9323579
	58	0.0005719	0.7428557	0.0002359	0.8624696
	59	0.0004284	0.7489784	0.0001812	0.7678420
	60	0.0003281	0.7659409	0.0001493	0.8242995
	61	0.0002531	0.7714986	0.0001175	0.7865263
	62	0.0002060	0.8136716	0.0000861	0.7334368
	63	0.0001797	0.8726498	0.0000546	0.6339584
	64	0.0001644	0.9147896	0.0000414	0.7578420
	65	0.0001500	0.9121801	0.0000324	0.7822544
	66	0.0001359	0.9062892	0.0000282	0.8704977
	67	0.0001196	0.8799064	0.0000263	0.9342924
	68	0.0001024	0.8564000	0.0000246	0.9353246
	69	0.0000828	0.8079831	0.0000233	0.9453090
	70	0.0000620	0.7488319	0.0000226	0.9703960
	71	0.0000452	0.7287425	0.0000167	0.7383479
	72	0.0000366	0.8101289	0.0000122	0.7285159
	73	0.0000299	0.8162470	0.0000101	0.8335502
	74	0.0000239	0.8015867	0.0000085	0.8430243
	75	0.0000186	0.7787371	0.0000074	0.8676639
	76	0.0000140	0.7525703	0.0000060	0.8134688
	77	0.0000104	0.7435247	0.0000050	0.8225143
	78	0.0000076	0.7246112	0.0000039	0.7889123
	79	0.0000055	0.7232358	0.0000030	0.7583253
	80	0.0000040	0.7287703	0.0000023	0.7618901
	81	0.0000029	0.7237151	0.0000017	0.7706604
	82	0.0000015	0.5224390	0.0000010	0.5519726

Table for CG with no preconditioner $h = \frac{1}{64}$

Tolerance=10^{-6} method=NONE, Iterations = 157, condition number A=2413

k	|e|	ratio	|r|	ratio
1	4.5618175	0.0000000	10.2041977	0.0000000
2	4.0484423	0.8874626	2.9223637	0.2863884
3	3.9296490	0.9706570	1.1769802	0.4027494
4	3.8598809	0.9822457	0.7112536	0.6043038
5	3.7995456	0.9843686	0.4805222	0.6755989
6	3.7431529	0.9851580	0.3718902	0.7739293
7	3.6863983	0.9848378	0.2978303	0.8008555
8	3.6292174	0.9844887	0.2541673	0.8533964
9	3.5685722	0.9832897	0.2226181	0.8758725
10	3.5069581	0.9827342	0.2001179	0.8989292
11	3.4429589	0.9817508	0.1826321	0.9126225
12	3.3782370	0.9812017	0.1713830	0.9384053
13	3.3146909	0.9811896	0.1522840	0.8885599
14	3.2564054	0.9824160	0.1328174	0.8721691
15	3.2008849	0.9829504	0.1224334	0.9218176
16	3.1455152	0.9827018	0.1137015	0.9286800
17	3.0882964	0.9818094	0.1060581	0.9327769
18	3.0308365	0.9813943	0.0996429	0.9395123
19	2.9722551	0.9806716	0.0956821	0.9602499
20	2.9114870	0.9795549	0.0903023	0.9437739
21	2.8528462	0.9798588	0.0851183	0.9425936
22	2.7941605	0.9794291	0.0812703	0.9547925
23	2.7330511	0.9781296	0.0777886	0.9571584
24	2.6717953	0.9775870	0.0762112	0.9797225
25	2.6080786	0.9761521	0.0724646	0.9508390
26	2.5466310	0.9764395	0.0682339	0.9416164
27	2.4869367	0.9765595	0.0647979	0.9496439
28	2.4272004	0.9759800	0.0620220	0.9571603
29	2.3688169	0.9759461	0.0598978	0.9657516
30	2.3065758	0.9737248	0.0596884	0.9965033
31	2.2416075	0.9718334	0.0576136	0.9652406
32	2.1770400	0.9711959	0.0558695	0.9697274
33	2.1148235	0.9714215	0.0530152	0.9489108
34	2.0513467	0.9699848	0.0526473	0.9930604
35	1.9830861	0.9667240	0.0520339	0.9883494
36	1.9164540	0.9663998	0.0501573	0.9639341
37	1.8482870	0.9644306	0.0495531	0.9879541
38	1.7802122	0.9631687	0.0470089	0.9486572
39	1.7180330	0.9650720	0.0434971	0.9252956
40	1.6579511	0.9650286	0.0428035	0.9840528
41	1.5950758	0.9620765	0.0419171	0.9792931
42	1.5314235	0.9600945	0.0407596	0.9723845
43	1.4665547	0.9576415	0.0403056	0.9888615
44	1.3990502	0.9539707	0.0399968	0.9923390
45	1.3251017	0.9471438	0.0414126	1.0353981
46	1.2399275	0.9357225	0.0446116	1.0772483
47	1.1274063	0.9092518	0.0518707	1.1627170
48	0.9608210	0.8522402	0.0624979	1.2048781
49	0.7395289	0.7696844	0.0669797	1.0717122
50	0.5283032	0.7143780	0.0585964	0.8748381
51	0.3950155	0.7477062	0.0442497	0.7551608
52	0.3199062	0.8098571	0.0353741	0.7994194
53	0.2652143	0.8290380	0.0311022	0.8792360
54	0.2243041	0.8457467	0.0261245	0.8399568
55	0.1942000	0.8657888	0.0227046	0.8690920
56	0.1684643	0.8674782	0.0210588	0.9275151
57	0.1453235	0.8626370	0.0187644	0.8910446
58	0.1265923	0.8711071	0.0167577	0.8930590
59	0.1097734	0.8671408	0.0154345	0.9210396
60	0.0947833	0.8634453	0.0138393	0.8966502
61	0.0823391	0.8687086	0.0123950	0.8956317
62	0.0713607	0.8666691	0.0111566	0.9000953
63	0.0615239	0.8621533	0.0099908	0.8954998
64	0.0531284	0.8635401	0.0089677	0.8976025
65	0.0460486	0.8667428	0.0078037	0.8701995

66	0.0401432	0.8717562	0.0068652	0.8797327	
67	0.0349347	0.8702530	0.0061224	0.8918082	
68	0.0305072	0.8732635	0.0053390	0.8720462	
69	0.0267317	0.8762420	0.0047623	0.8919769	
70	0.0234572	0.8775033	0.0041094	0.8628988	
71	0.0207112	0.8829387	0.0036173	0.8802590	
72	0.0181979	0.8786512	0.0032920	0.9100613	
73	0.0159488	0.8764068	0.0028970	0.8800222	
74	0.0140658	0.8819366	0.0025342	0.8747771	
75	0.0125049	0.8890299	0.0021879	0.8633171	
76	0.0111985	0.8955273	0.0019189	0.8770658	
77	0.0101148	0.9032251	0.0016804	0.8757329	
78	0.0091865	0.9082280	0.0015038	0.8948967	
79	0.0084158	0.9160976	0.0013190	0.8770684	
80	0.0078196	0.9291617	0.0011071	0.8393543	
81	0.0073758	0.9432389	0.0009256	0.8360613	
82	0.0070385	0.9542812	0.0007869	0.8502076	
83	0.0067629	0.9608446	0.0006653	0.8453718	
84	0.0065430	0.9674723	0.0005542	0.8330332	
85	0.0063606	0.9721362	0.0004631	0.8356598	
86	0.0062041	0.9753943	0.0003904	0.8429988	
87	0.0060628	0.9772213	0.0003301	0.8455336	
88	0.0059316	0.9783588	0.0002793	0.8462168	
89	0.0058038	0.9784528	0.0002434	0.8714097	
90	0.0056707	0.9770673	0.0002200	0.9036392	
91	0.0055215	0.9736957	0.0002182	0.9922456	
92	0.0053270	0.9647725	0.0002329	1.0671837	
93	0.0050613	0.9501202	0.0002563	1.1003357	
94	0.0047075	0.9300953	0.0002754	1.0744267	
95	0.0043063	0.9147708	0.0002692	0.9778062	
96	0.0039172	0.9096464	0.0002592	0.9626214	
97	0.0035245	0.8997507	0.0002627	1.0135967	
98	0.0030928	0.8774993	0.0002717	1.0343359	
99	0.0026357	0.8522177	0.0002652	0.9760185	
100	0.0022317	0.8467214	0.0002384	0.8988006	
101	0.0019165	0.8587692	0.0002124	0.8912480	
102	0.0016658	0.8691736	0.0001896	0.8925872	
103	0.0014748	0.8853198	0.0001562	0.8237657	
104	0.0013425	0.9102889	0.0001341	0.8584851	
105	0.0012338	0.9190797	0.0001229	0.9163269	
106	0.0011364	0.9210448	0.0001110	0.9029318	
107	0.0010529	0.9264833	0.0000981	0.8837443	
108	0.0009822	0.9329029	0.0000884	0.9015361	
109	0.0009173	0.9338643	0.0000805	0.9108633	
110	0.0008605	0.9381542	0.0000706	0.8763963	
111	0.0008106	0.9420035	0.0000648	0.9185117	
112	0.0007612	0.9390822	0.0000609	0.9395127	
113	0.0007143	0.9383693	0.0000557	0.9139756	
114	0.0006696	0.9374446	0.0000519	0.9329107	
115	0.0006260	0.9348144	0.0000481	0.9264502	
116	0.0005854	0.9351858	0.0000443	0.9199678	
117	0.0005460	0.9326180	0.0000424	0.9576760	
118	0.0005044	0.9239461	0.0000413	0.9738142	
119	0.0004627	0.9172921	0.0000391	0.9473491	
120	0.0004208	0.9094667	0.0000381	0.9752708	
121	0.0003784	0.8992550	0.0000372	0.9763676	
122	0.0003372	0.8911407	0.0000350	0.9403920	
123	0.0002996	0.8885168	0.0000322	0.9205549	
124	0.0002665	0.8895264	0.0000296	0.9186267	
125	0.0002382	0.8936247	0.0000267	0.9014817	
126	0.0002145	0.9005476	0.0000235	0.8810575	
127	0.0001949	0.9087432	0.0000205	0.8702270	
128	0.0001793	0.9195943	0.0000191	0.9333716	
129	0.0001645	0.9175205	0.0000165	0.8627026	
130	0.0001529	0.9299627	0.0000145	0.8810031	
131	0.0001428	0.9334836	0.0000126	0.8674787	
132	0.0001338	0.9370194	0.0000113	0.8986161	
133	0.0001252	0.9354832	0.0000105	0.9248925	
134	0.0001161	0.9280017	0.0000101	0.9641272	
135	0.0001061	0.9132398	0.0000104	1.0276371	
136	0.0000941	0.8872630	0.0000108	1.0403204	
137	0.0000809	0.8591660	0.0000106	0.9825534	

138	0.0000692	0.8553820	0.0000095	0.9003622
139	0.0000600	0.8674816	0.0000080	0.8360480
140	0.0000535	0.8925543	0.0000066	0.8225625
141	0.0000486	0.9079807	0.0000057	0.8611308
142	0.0000443	0.9113730	0.0000052	0.9135594
143	0.0000402	0.9066825	0.0000048	0.9354110
144	0.0000361	0.8975802	0.0000045	0.9304066
145	0.0000323	0.8956690	0.0000041	0.9034467
146	0.0000288	0.8917449	0.0000038	0.9321324
147	0.0000255	0.8865527	0.0000034	0.9105714
148	0.0000226	0.8841495	0.0000031	0.9002286
149	0.0000200	0.8862625	0.0000028	0.8885652
150	0.0000177	0.8849574	0.0000025	0.8993699
151	0.0000155	0.8766179	0.0000024	0.9491487
152	0.0000133	0.8572703	0.0000022	0.9393268
153	0.0000113	0.8482148	0.0000020	0.9135275
154	0.0000095	0.8451107	0.0000018	0.8915801
155	0.0000081	0.8501849	0.0000015	0.8470707
156	0.0000070	0.8664622	0.0000013	0.8602920
157	0.0000054	0.7724505	0.0000010	0.7331119

Table for CG with no preconditioner $h = \frac{1}{128}$

Tolerance=10^{-6} method=NONE, Iterations = 291, condition number A=9655

k	|e|	ratio	|r|	ratio
1	6.5216900	0.0000000	14.4577169	0.0000000
2	5.8211800	0.8925877	4.0764931	0.2819597
3	5.6842162	0.9764715	1.5799015	0.3875639
4	5.6198883	0.9886831	0.8968116	0.5676377
5	5.5711714	0.9913313	0.5778805	0.6443723
6	5.5285647	0.9923523	0.4303215	0.7446548
7	5.4869080	0.9924652	0.3341897	0.7766047
8	5.4468461	0.9926986	0.2785535	0.8335192
9	5.4044559	0.9922175	0.2424388	0.8703491
10	5.3616029	0.9920708	0.2146822	0.8855108
11	5.3191442	0.9920810	0.1862262	0.8674507
12	5.2782868	0.9923188	0.1677396	0.9007302
13	5.2363954	0.9920634	0.1524839	0.9090513
14	5.1941353	0.9919295	0.1420818	0.9317825
15	5.1513426	0.9917614	0.1293917	0.9106846
16	5.1099665	0.9919679	0.1195224	0.9237248
17	5.0683715	0.9918600	0.1119117	0.9363248
18	5.0255870	0.9915585	0.1066056	0.9525867
19	4.9819254	0.9913121	0.0994276	0.9326674
20	4.9398031	0.9915450	0.0934835	0.9402165
21	4.8972751	0.9913908	0.0895284	0.9576918
22	4.8541439	0.9911928	0.0849488	0.9488477
23	4.8107070	0.9910516	0.0811183	0.9549077
24	4.7673483	0.9909871	0.0782140	0.9641979
25	4.7234798	0.9907981	0.0742769	0.9496615
26	4.6807711	0.9909582	0.0714271	0.9616324
27	4.6372353	0.9906990	0.0684405	0.9581873
28	4.5948580	0.9908615	0.0651097	0.9513333
29	4.5530996	0.9909119	0.0621499	0.9545408
30	4.5112658	0.9908120	0.0601683	0.9681159
31	4.4687574	0.9905773	0.0584916	0.9721339
32	4.4262843	0.9904955	0.0566503	0.9685198
33	4.3834625	0.9903256	0.0547127	0.9657966
34	4.3405454	0.9902093	0.0529867	0.9684541
35	4.2983731	0.9902841	0.0507715	0.9581929
36	4.2562271	0.9901949	0.0496163	0.9772479
37	4.2138748	0.9900493	0.0481310	0.9700646
38	4.1708118	0.9897807	0.0472816	0.9823505
39	4.1266023	0.9894003	0.0466718	0.9871047
40	4.0823351	0.9892727	0.0454224	0.9732297
41	4.0380267	0.9891463	0.0441588	0.9721808
42	3.9946964	0.9892694	0.0426145	0.9650276
43	3.9522103	0.9893644	0.0409349	0.9605877
44	3.9106277	0.9894786	0.0398300	0.9730069
45	3.8684976	0.9892268	0.0392197	0.9846780
46	3.8257317	0.9889451	0.0385564	0.9830880

	47	3.7830507	0.9888437	0.0372870	0.9670762
	48	3.7403946	0.9887244	0.0366904	0.9839996
	49	3.6974958	0.9885310	0.0359622	0.9801539
	50	3.6549289	0.9884876	0.0352283	0.9795926
	51	3.6113640	0.9880805	0.0345719	0.9813650
	52	3.5675354	0.9878637	0.0340535	0.9850064
	53	3.5231205	0.9875503	0.0336276	0.9874943
	54	3.4787130	0.9873954	0.0329058	0.9785343
	55	3.4345867	0.9873153	0.0323069	0.9817996
	56	3.3901581	0.9870643	0.0316810	0.9806276
	57	3.3461400	0.9870159	0.0306893	0.9686978
	58	3.3036285	0.9872954	0.0298158	0.9715352
	59	3.2612089	0.9871597	0.0294984	0.9893541
	60	3.2175113	0.9866008	0.0293399	0.9946276
	61	3.1732557	0.9862454	0.0288105	0.9819557
	62	3.1292108	0.9861200	0.0282986	0.9822332
	63	3.0851351	0.9859147	0.0277461	0.9804746
	64	3.0412129	0.9857633	0.0274147	0.9880581
	65	2.9964828	0.9852920	0.0267788	0.9768022
	66	2.9522621	0.9852425	0.0265317	0.9907736
	67	2.9083386	0.9851221	0.0258464	0.9741706
	68	2.8648102	0.9850332	0.0254640	0.9852068
	69	2.8199937	0.9843562	0.0255037	1.0015562
	70	2.7742200	0.9837682	0.0252014	0.9881474
	71	2.7287776	0.9836197	0.0245978	0.9760481
	72	2.6844169	0.9837434	0.0237590	0.9659003
	73	2.6412743	0.9839285	0.0235033	0.9892381
	74	2.5968246	0.9831711	0.0234332	0.9970183
	75	2.5517313	0.9826352	0.0230106	0.9819663
	76	2.5070630	0.9824949	0.0225744	0.9810401
	77	2.4633246	0.9825539	0.0220316	0.9759583
	78	2.4188791	0.9819571	0.0219368	0.9956962
	79	2.3736882	0.9813174	0.0216703	0.9878504
	80	2.3279918	0.9807488	0.0217378	1.0031152
	81	2.2808231	0.9797385	0.0215495	0.9913383
	82	2.2336755	0.9793287	0.0212204	0.9847292
	83	2.1862533	0.9787694	0.0208521	0.9826415
	84	2.1387752	0.9782833	0.0206835	0.9919146
	85	2.0909984	0.9776616	0.0203045	0.9816770
	86	2.0438264	0.9774405	0.0199742	0.9837320
	87	1.9962370	0.9767155	0.0198473	0.9936479
	88	1.9483597	0.9760162	0.0195917	0.9871222
	89	1.8995855	0.9749666	0.0195412	0.9974214
	90	1.8476324	0.9726503	0.0202739	1.0374967
	91	1.7887809	0.9681476	0.0215423	1.0625600
	92	1.7179957	0.9604282	0.0239573	1.1121081
	93	1.6232429	0.9448469	0.0281514	1.1750654
	94	1.4854374	0.9151048	0.0338549	1.2025994
	95	1.2901182	0.8685107	0.0389859	1.1515605
	96	1.0644065	0.8250457	0.0390577	1.0018411
	97	0.8691133	0.8165238	0.0342085	0.8758440
	98	0.7330221	0.8434137	0.0280872	0.8210594
	99	0.6386873	0.8713069	0.0245142	0.8727891
	100	0.5637403	0.8826547	0.0221816	0.9048483
	101	0.5029967	0.8922489	0.0199448	0.8991569
	102	0.4540556	0.9027011	0.0177812	0.8915224
	103	0.4150739	0.9141476	0.0160900	0.9048868
	104	0.3813251	0.9186921	0.0152002	0.9446998
	105	0.3512848	0.9212212	0.0140495	0.9243007
	106	0.3250302	0.9252614	0.0132617	0.9439243
	107	0.3006703	0.9250533	0.0127023	0.9578196
	108	0.2782780	0.9255254	0.0118652	0.9340934
	109	0.2583980	0.9285605	0.0111782	0.9421066
	110	0.2399280	0.9285211	0.0107487	0.9615730
	111	0.2224847	0.9272978	0.0101863	0.9476738
	112	0.2067140	0.9291155	0.0095446	0.9370026
	113	0.1922750	0.9301500	0.0090068	0.9436612
	114	0.1789377	0.9306343	0.0085970	0.9544985
	115	0.1659538	0.9274389	0.0084277	0.9803081
	116	0.1532174	0.9232532	0.0081486	0.9668770
	117	0.1413960	0.9228457	0.0076599	0.9400255
	118	0.1307287	0.9245573	0.0071955	0.9393847

119	0.1209090	0.9248852	0.0067734	0.9413369	
120	0.1120288	0.9265547	0.0063259	0.9339284	
121	0.1039599	0.9279741	0.0059769	0.9448358	
122	0.0964983	0.9282266	0.0056345	0.9427016	
123	0.0896672	0.9292102	0.0053026	0.9411039	
124	0.0833125	0.9291304	0.0049828	0.9396944	
125	0.0776324	0.9318209	0.0046406	0.9313226	
126	0.0724280	0.9329618	0.0043473	0.9367799	
127	0.0676853	0.9345182	0.0040083	0.9220193	
128	0.0635380	0.9387263	0.0036864	0.9197089	
129	0.0597311	0.9400845	0.0034773	0.9432675	
130	0.0560418	0.9382361	0.0033230	0.9556383	
131	0.0525073	0.9369301	0.0031496	0.9478249	
132	0.0491855	0.9367361	0.0029710	0.9432938	
133	0.0460666	0.9365890	0.0027967	0.9413153	
134	0.0431344	0.9363485	0.0026568	0.9499861	
135	0.0403771	0.9360763	0.0024800	0.9334519	
136	0.0378184	0.9366311	0.0023383	0.9428438	
137	0.0354510	0.9374012	0.0021922	0.9375558	
138	0.0332424	0.9376994	0.0020377	0.9294867	
139	0.0312191	0.9391363	0.0019028	0.9338130	
140	0.0293648	0.9406007	0.0017822	0.9366054	
141	0.0275654	0.9387233	0.0017140	0.9617729	
142	0.0258209	0.9367154	0.0016189	0.9444659	
143	0.0242216	0.9380629	0.0015084	0.9317937	
144	0.0227180	0.9379219	0.0014412	0.9554332	
145	0.0212899	0.9371386	0.0013528	0.9386613	
146	0.0199738	0.9381832	0.0012813	0.9471259	
147	0.0187271	0.9375794	0.0012066	0.9417288	
148	0.0175963	0.9396166	0.0011173	0.9259961	
149	0.0165930	0.9429837	0.0010360	0.9271850	
150	0.0156685	0.9442818	0.0009697	0.9359917	
151	0.0148093	0.9451670	0.0009143	0.9429451	
152	0.0140387	0.9479640	0.0008446	0.9236841	
153	0.0133481	0.9508081	0.0007828	0.9269097	
154	0.0127077	0.9520237	0.0007435	0.9498209	
155	0.0121136	0.9532515	0.0007005	0.9420813	
156	0.0115697	0.9551005	0.0006583	0.9397314	
157	0.0110828	0.9579138	0.0006133	0.9317308	
158	0.0106451	0.9605075	0.0005721	0.9327397	
159	0.0102507	0.9629485	0.0005351	0.9353541	
160	0.0099019	0.9659712	0.0005003	0.9348993	
161	0.0095977	0.9692748	0.0004553	0.9101756	
162	0.0093373	0.9728771	0.0004165	0.9146442	
163	0.0091078	0.9754214	0.0003875	0.9305895	
164	0.0089051	0.9777442	0.0003573	0.9218909	
165	0.0087271	0.9800029	0.0003282	0.9185575	
166	0.0085684	0.9818138	0.0003015	0.9188234	
167	0.0084309	0.9839583	0.0002719	0.9016052	
168	0.0083109	0.9857609	0.0002449	0.9008619	
169	0.0082058	0.9873564	0.0002206	0.9008608	
170	0.0081135	0.9887523	0.0001955	0.8858579	
171	0.0080298	0.9896858	0.0001764	0.9025415	
172	0.0079505	0.9901290	0.0001610	0.9127841	
173	0.0078748	0.9904714	0.0001459	0.9060863	
174	0.0078020	0.9907636	0.0001330	0.9115998	
175	0.0077322	0.9910514	0.0001201	0.9027629	
176	0.0076646	0.9912555	0.0001102	0.9181004	
177	0.0075980	0.9913082	0.0000994	0.9021361	
178	0.0075343	0.9916206	0.0000903	0.9080043	
179	0.0074697	0.9914240	0.0000843	0.9338806	
180	0.0073991	0.9905492	0.0000836	0.9916163	
181	0.0073144	0.9885474	0.0000873	1.0443408	
182	0.0072065	0.9852455	0.0000944	1.0807980	
183	0.0070661	0.9805186	0.0001041	1.1030488	
184	0.0068861	0.9745304	0.0001130	1.0857231	
185	0.0066684	0.9683929	0.0001181	1.0445829	
186	0.0064350	0.9649952	0.0001170	0.9906712	
187	0.0062072	0.9645942	0.0001121	0.9587357	
188	0.0059900	0.9650072	0.0001088	0.9702163	
189	0.0057630	0.9621089	0.0001137	1.0446340	
190	0.0054891	0.9524693	0.0001251	1.1007868	

	191	0.0051459	0.9374773	0.0001347	1.0766640
	192	0.0047715	0.9272480	0.0001344	0.9978854
	193	0.0044069	0.9235887	0.0001313	0.9767634
	194	0.0040493	0.9188441	0.0001316	1.0022126
	195	0.0036749	0.9075451	0.0001353	1.0284495
	196	0.0032688	0.8895021	0.0001380	1.0193791
	197	0.0028661	0.8768005	0.0001344	0.9740698
	198	0.0024955	0.8707062	0.0001284	0.9558213
	199	0.0021550	0.8635418	0.0001229	0.9570292
	200	0.0018567	0.8615758	0.0001137	0.9250923
	201	0.0016166	0.8706679	0.0001009	0.8877237
	202	0.0014368	0.8887817	0.0000897	0.8888228
	203	0.0012963	0.9022463	0.0000810	0.9033006
	204	0.0011838	0.9131675	0.0000720	0.8884030
	205	0.0010994	0.9286959	0.0000635	0.8813002
	206	0.0010326	0.9392711	0.0000578	0.9112359
	207	0.0009772	0.9463790	0.0000524	0.9068811
	208	0.0009315	0.9532468	0.0000471	0.8981476
	209	0.0008948	0.9606123	0.0000418	0.8876419
	210	0.0008645	0.9660678	0.0000379	0.9073230
	211	0.0008375	0.9687479	0.0000348	0.9164967
	212	0.0008140	0.9720156	0.0000308	0.8855352
	213	0.0007936	0.9749646	0.0000281	0.9143270
	214	0.0007742	0.9755578	0.0000264	0.9377285
	215	0.0007555	0.9758417	0.0000243	0.9211895
	216	0.0007382	0.9770402	0.0000225	0.9238694
	217	0.0007210	0.9766651	0.0000216	0.9599043
	218	0.0007031	0.9752798	0.0000205	0.9529276
	219	0.0006856	0.9749951	0.0000197	0.9592020
	220	0.0006669	0.9727617	0.0000196	0.9934542
	221	0.0006474	0.9706996	0.0000188	0.9599738
	222	0.0006279	0.9699750	0.0000182	0.9670444
	223	0.0006080	0.9682281	0.0000178	0.9778561
	224	0.0005876	0.9665155	0.0000170	0.9586794
	225	0.0005677	0.9661052	0.0000167	0.9784560
	226	0.0005461	0.9619817	0.0000169	1.0115654
	227	0.0005230	0.9576032	0.0000169	1.0017682
	228	0.0004988	0.9538113	0.0000169	0.9996731
	229	0.0004738	0.9498157	0.0000167	0.9877224
	230	0.0004487	0.9470574	0.0000163	0.9743509
	231	0.0004240	0.9450280	0.0000160	0.9872214
	232	0.0003992	0.9414172	0.0000157	0.9805466
	233	0.0003747	0.9386238	0.0000154	0.9774359
	234	0.0003508	0.9362417	0.0000150	0.9738462
	235	0.0003281	0.9354138	0.0000143	0.9557997
	236	0.0003072	0.9361931	0.0000135	0.9455793
	237	0.0002887	0.9398553	0.0000125	0.9270502
	238	0.0002722	0.9427474	0.0000119	0.9470597
	239	0.0002566	0.9425761	0.0000115	0.9644387
	240	0.0002423	0.9445275	0.0000106	0.9283254
	241	0.0002296	0.9476198	0.0000099	0.9320668
	242	0.0002183	0.9506384	0.0000093	0.9384921
	243	0.0002078	0.9516694	0.0000088	0.9457131
	244	0.0001979	0.9526251	0.0000083	0.9406033
	245	0.0001890	0.9550989	0.0000078	0.9394368
	246	0.0001807	0.9558598	0.0000074	0.9484184
	247	0.0001727	0.9559939	0.0000072	0.9728821
	248	0.0001647	0.9537430	0.0000070	0.9768819
	249	0.0001571	0.9536549	0.0000067	0.9498763
	250	0.0001499	0.9544458	0.0000064	0.9553444
	251	0.0001433	0.9558753	0.0000060	0.9370424
	252	0.0001374	0.9585791	0.0000056	0.9316401
	253	0.0001320	0.9606996	0.0000052	0.9306494
	254	0.0001272	0.9637858	0.0000048	0.9190719
	255	0.0001228	0.9655622	0.0000044	0.9347832
	256	0.0001188	0.9668738	0.0000044	0.9940088
	257	0.0001143	0.9625490	0.0000043	0.9723134
	258	0.0001102	0.9637233	0.0000041	0.9537761
	259	0.0001063	0.9649779	0.0000038	0.9226164
	260	0.0001030	0.9687202	0.0000035	0.9165359
	261	0.0001000	0.9705944	0.0000032	0.9210341
	262	0.0000972	0.9725780	0.0000030	0.9262509

```
            263       0.0000947       0.9740029       0.0000027       0.9186089
            264       0.0000923       0.9752465       0.0000025       0.9217604
            265       0.0000902       0.9762555       0.0000023       0.9262124
            266       0.0000880       0.9765195       0.0000022       0.9434152
            267       0.0000859       0.9755254       0.0000021       0.9638640
            268       0.0000836       0.9736855       0.0000021       0.9837715
            269       0.0000812       0.9707770       0.0000021       0.9930902
            270       0.0000785       0.9675646       0.0000021       1.0173215
            271       0.0000755       0.9611090       0.0000022       1.0450837
            272       0.0000719       0.9527471       0.0000023       1.0645178
            273       0.0000678       0.9424429       0.0000024       1.0398748
            274       0.0000632       0.9323311       0.0000025       1.0201026
            275       0.0000585       0.9260217       0.0000024       0.9839147
            276       0.0000541       0.9241385       0.0000023       0.9540136
            277       0.0000500       0.9240218       0.0000022       0.9530567
            278       0.0000462       0.9245567       0.0000021       0.9527424
            279       0.0000426       0.9230466       0.0000020       0.9612610
            280       0.0000394       0.9231135       0.0000019       0.9543941
            281       0.0000364       0.9245357       0.0000018       0.9222267
            282       0.0000339       0.9301992       0.0000016       0.9061832
            283       0.0000317       0.9376877       0.0000015       0.9005013
            284       0.0000299       0.9433252       0.0000013       0.9244644
            285       0.0000283       0.9433824       0.0000013       0.9573062
            286       0.0000266       0.9404446       0.0000013       0.9814851
            287       0.0000249       0.9364105       0.0000012       0.9666827
            288       0.0000232       0.9340297       0.0000012       0.9701371
            289       0.0000216       0.9308285       0.0000012       0.9738573
            290       0.0000200       0.9257747       0.0000011       0.9852032
            291       0.0000171       0.8523176       0.0000010       0.8703063
```

tables for CG with Multigrid preconditioner

```
h =      1/16   eps=0.000001    method=MG
k=4, cond A=150.416930
               k             |e|           ratio             |r|           ratio
               1       2.1998630       0.0000000       5.3253083       0.0000000
               2       0.0956633       0.0434860       0.0363016       0.0068168
               3       0.0014271       0.0149181       0.0006556       0.0180607
               4       0.0000002       0.0001388       0.0000002       0.0003796

h =      1/32           eps=0.000001   method=MG
k=4,  cond A=603.051928
               k             |e|           ratio             |r|           ratio
               1       3.1204692       0.0000000       7.1238671       0.0000000
               2       0.1461191       0.0468260       0.0360066       0.0050544
               3       0.0031325       0.0214382       0.0006083       0.0168931
               4       0.0000004       0.0001403       0.0000003       0.0005447

h =      1/64    ,n=65   eps=0.000001   method=MG
k=4, cond A=2413.598654
               k             |e|           ratio             |r|           ratio
               1       4.5764854       0.0000000      10.4620417       0.0000000
               2       0.2180254       0.0476404       0.0388724       0.0037156
               3       0.0058026       0.0266142       0.0006767       0.0174076
               4       0.0000012       0.0002040       0.0000004       0.0005660

h =      1/128   ,n=129  eps=0.000001   method=MG
k=4, cond A=9655.787254
               k             |e|           ratio             |r|           ratio
               1       6.4563811       0.0000000      14.6329466       0.0000000
               2       0.3174234       0.0491643       0.0482766       0.0032992
               3       0.0094526       0.0297790       0.0008327       0.0172495
               4       0.0000034       0.0003597       0.0000004       0.0005216
```

Tables for CG with SSOR preconditioner

Tables for CG with SSOR preconditioner h=1/16

```
               k             |e|           ratio             |r|           ratio
```

1	2.2693639	0.0000000	4.9322226	0.0000000
2	1.6468398	0.7256835	0.4605221	0.0933701
3	1.3513665	0.8205816	0.2730178	0.5928442
4	0.9606275	0.7108564	0.1964313	0.7194818
5	0.5252730	0.5468019	0.1561994	0.7951855
6	0.1534885	0.2922070	0.1084332	0.6941974
7	0.0416077	0.2710804	0.0275143	0.2537445
8	0.0262239	0.6302655	0.0103129	0.3748198
9	0.0123709	0.4717420	0.0068234	0.6616409
10	0.0033437	0.2702897	0.0029164	0.4274017
11	0.0011039	0.3301359	0.0008844	0.3032476
12	0.0005418	0.4908182	0.0003122	0.3530697
13	0.0001777	0.3279708	0.0001683	0.5389642
14	0.0000471	0.2650202	0.0000421	0.2501429
15	0.0000242	0.5133066	0.0000142	0.3362731
16	0.0000147	0.6099509	0.0000054	0.3833065
17	0.0000053	0.3600554	0.0000040	0.7457782
18	0.0000004	0.0818492	0.0000004	0.0934337

Tables for CG with SSOR preconditioner h=1/32

k	|e|	ratio	|r|	ratio
1	3.1617307	0.0000000	7.0150453	0.0000000
2	2.5617890	0.8102490	0.5539254	0.0789625
3	2.3542794	0.9189982	0.2962589	0.5348354
4	2.1065890	0.8947914	0.1986559	0.6705483
5	1.8518713	0.8790853	0.1422168	0.7158951
6	1.6156673	0.8724512	0.1133796	0.7972310
7	1.3789468	0.8534844	0.0910121	0.8027200
8	1.1345409	0.8227590	0.0843711	0.9270315
9	0.8224095	0.7248831	0.0847173	1.0041038
10	0.4180835	0.5083641	0.0825150	0.9740034
11	0.1235549	0.2955268	0.0482795	0.5850996
12	0.0551450	0.4463198	0.0203644	0.4218024
13	0.0278858	0.5056810	0.0122853	0.6032745
14	0.0128799	0.4618815	0.0057640	0.4691758
15	0.0081751	0.6347162	0.0026993	0.4683099
16	0.0058303	0.7131826	0.0013998	0.5185673
17	0.0047541	0.8154064	0.0006696	0.4783424
18	0.0038683	0.8136734	0.0005167	0.7716602
19	0.0021487	0.5554644	0.0005517	1.0677534
20	0.0011032	0.5134135	0.0002821	0.5114078
21	0.0007772	0.7045585	0.0001444	0.5119507
22	0.0005067	0.6519440	0.0001267	0.8768386
23	0.0002475	0.4884356	0.0000763	0.6020652
24	0.0001620	0.6545659	0.0000398	0.5221399
25	0.0001077	0.6648438	0.0000249	0.6246745
26	0.0000674	0.6260178	0.0000172	0.6916799
27	0.0000335	0.4974052	0.0000121	0.7029690
28	0.0000133	0.3976940	0.0000067	0.5550799
29	0.0000054	0.4034871	0.0000029	0.4265622
30	0.0000016	0.2990445	0.0000007	0.2473623

Tables for CG with SSOR preconditioner h=1/64

k	|e|	ratio	|r|	ratio
1	4.5490436	0.0000000	10.0541553	0.0000000
2	3.8362026	0.8432987	0.7280693	0.0724148
3	3.7003001	0.9645737	0.3053137	0.4193471
4	3.5423025	0.9573014	0.1926717	0.6310615
5	3.3783165	0.9537064	0.1437188	0.7459256
6	3.1995620	0.9470877	0.1180847	0.8216375
7	3.0218250	0.9444496	0.1001557	0.8481680
8	2.8564678	0.9452790	0.0828611	0.8273226
9	2.6921619	0.9424794	0.0706543	0.8526838
10	2.5198220	0.9359846	0.0632266	0.8948726
11	2.3462801	0.9311293	0.0583966	0.9236077
12	2.1638926	0.9222652	0.0533587	0.9137299
13	1.9814273	0.9156773	0.0489931	0.9181837
14	1.8029129	0.9099062	0.0433065	0.8839308
15	1.6149689	0.8957553	0.0422280	0.9750970
16	1.4112614	0.8738629	0.0412639	0.9771686
17	1.1749024	0.8325193	0.0429641	1.0412016

18	0.8528075	0.7258539	0.0493421	1.1484509
19	0.4518308	0.5298157	0.0484776	0.9824796
20	0.2132808	0.4720368	0.0278557	0.5746100
21	0.1501244	0.7038816	0.0146585	0.5262282
22	0.1111903	0.7406542	0.0140617	0.9592857
23	0.0662040	0.5954116	0.0111642	0.7939485
24	0.0437040	0.6601420	0.0069109	0.6190224
25	0.0303526	0.6945044	0.0050944	0.7371500
26	0.0221190	0.7287351	0.0033023	0.6482254
27	0.0165263	0.7471507	0.0024898	0.7539705
28	0.0129979	0.7865014	0.0016588	0.6662198
29	0.0110448	0.8497327	0.0011217	0.6762264
30	0.0098942	0.8958298	0.0007244	0.6457952
31	0.0091291	0.9226731	0.0004742	0.6545639
32	0.0084946	0.9304946	0.0003511	0.7405437
33	0.0076879	0.9050339	0.0003237	0.9219407
34	0.0065529	0.8523626	0.0003485	1.0763733
35	0.0047519	0.7251674	0.0003915	1.1234523
36	0.0028376	0.5971538	0.0003422	0.8742031
37	0.0018887	0.6656051	0.0002004	0.5855785
38	0.0015408	0.8157954	0.0001243	0.6204067
39	0.0012973	0.8419573	0.0001008	0.8108109
40	0.0010583	0.8157939	0.0000853	0.8463996
41	0.0007990	0.7549702	0.0000787	0.9219365
42	0.0005529	0.6919610	0.0000666	0.8468646
43	0.0003932	0.7111390	0.0000455	0.6826150
44	0.0003029	0.7704465	0.0000339	0.7445688
45	0.0002333	0.7702420	0.0000257	0.7580561
46	0.0001916	0.8211070	0.0000175	0.6821221
47	0.0001600	0.8349544	0.0000136	0.7744449
48	0.0001358	0.8490333	0.0000095	0.7032869
49	0.0001119	0.8239944	0.0000090	0.9472819
50	0.0000825	0.7372094	0.0000083	0.9171382
51	0.0000541	0.6554525	0.0000073	0.8784722
52	0.0000349	0.6460945	0.0000048	0.6616077
53	0.0000248	0.7089250	0.0000034	0.7101718
54	0.0000182	0.7361531	0.0000023	0.6772298
55	0.0000124	0.6811877	0.0000021	0.9093606
56	0.0000048	0.3868446	0.0000009	0.4194420

Tables for CG with SSOR preconditioner h=1/128

k	|e|	ratio	|r|	ratio
1	6.4457756	0.0000000	14.4540779	0.0000000
2	5.5139993	0.8554439	0.9798537	0.0677908
3	5.4088913	0.9809380	0.3403982	0.3473970
4	5.3026393	0.9803560	0.2034492	0.5976800
5	5.1853961	0.9778897	0.1483529	0.7291887
6	5.0654954	0.9768772	0.1202717	0.8107138
7	4.9451401	0.9762402	0.1012563	0.8418966
8	4.8245020	0.9756047	0.0858063	0.8474169
9	4.7066959	0.9755817	0.0722637	0.8421718
10	4.5874237	0.9746590	0.0649999	0.8994827
11	4.4629279	0.9728615	0.0605125	0.9309626
12	4.3391818	0.9722724	0.0545367	0.9012460
13	4.2186913	0.9722320	0.0488740	0.8961687
14	4.1021192	0.9723677	0.0440151	0.9005828
15	3.9854067	0.9715483	0.0412976	0.9382589
16	3.8680259	0.9705473	0.0382945	0.9272820
17	3.7513422	0.9698338	0.0359069	0.9376523
18	3.6295795	0.9675415	0.0344123	0.9583746
19	3.5086283	0.9666763	0.0316764	0.9204971
20	3.3903746	0.9662963	0.0299359	0.9450520
21	3.2691604	0.9642475	0.0290988	0.9720394
22	3.1436705	0.9616140	0.0277443	0.9534507
23	3.0186217	0.9602220	0.0262770	0.9471139
24	2.8957417	0.9592927	0.0248096	0.9441578
25	2.7732455	0.9576978	0.0238982	0.9632612
26	2.6506297	0.9557862	0.0223533	0.9353568
27	2.5323184	0.9553648	0.0216356	0.9678923
28	2.4045866	0.9495594	0.0212703	0.9831139
29	2.2772037	0.9470250	0.0200174	0.9410971

	30	2.1483420	0.9434123	0.0196727	0.9827790
	31	2.0149086	0.9378901	0.0192478	0.9784036
	32	1.8686594	0.9274165	0.0199859	1.0383474
	33	1.6895903	0.9041724	0.0217554	1.0885392
	34	1.4442876	0.8548153	0.0258210	1.1868754
	35	1.0931431	0.7568736	0.0302817	1.1727536
	36	0.6837108	0.6254540	0.0283683	0.9368158
	37	0.4418863	0.6463058	0.0184521	0.6504463
	38	0.3518847	0.7963243	0.0114809	0.6221984
	39	0.3006824	0.8544912	0.0105466	0.9186222
	40	0.2372194	0.7889368	0.0114586	1.0864744
	41	0.1840178	0.7757283	0.0085694	0.7478555
	42	0.1528291	0.8305127	0.0068315	0.7971989
	43	0.1237482	0.8097158	0.0067367	0.9861302
	44	0.0985025	0.7959918	0.0053549	0.7948776
	45	0.0801615	0.8138015	0.0046556	0.8694177
	46	0.0645647	0.8054324	0.0038580	0.8286741
	47	0.0538188	0.8335634	0.0030348	0.7866214
	48	0.0443159	0.8234283	0.0027050	0.8913447
	49	0.0369617	0.8340510	0.0021314	0.7879483
	50	0.0306704	0.8297872	0.0018818	0.8828969
	51	0.0251646	0.8204853	0.0015780	0.8385403
	52	0.0206586	0.8209407	0.0013350	0.8459954
	53	0.0170433	0.8249991	0.0010816	0.8101736
	54	0.0140243	0.8228592	0.0009494	0.8778358
	55	0.0116435	0.8302376	0.0007679	0.8087564
	56	0.0099038	0.8505901	0.0006226	0.8107796
	57	0.0087211	0.8805767	0.0004813	0.7731395
	58	0.0078802	0.9035759	0.0003957	0.8221169
	59	0.0072607	0.9213899	0.0003211	0.8113631
	60	0.0068350	0.9413714	0.0002460	0.7660613
	61	0.0065434	0.9573309	0.0001854	0.7536923
	62	0.0063343	0.9680457	0.0001386	0.7475188
	63	0.0061677	0.9736986	0.0001019	0.7352154
	64	0.0060189	0.9758793	0.0000781	0.7662600
	65	0.0058564	0.9730031	0.0000697	0.8929152
	66	0.0056418	0.9633473	0.0000711	1.0197906
	67	0.0053083	0.9408880	0.0000812	1.1420576
	68	0.0047907	0.9024899	0.0000954	1.1756096
	69	0.0040810	0.8518580	0.0001007	1.0553830
	70	0.0033234	0.8143729	0.0000974	0.9667177
	71	0.0026225	0.7891023	0.0000930	0.9553683
	72	0.0019626	0.7483464	0.0000878	0.9441616
	73	0.0014856	0.7569965	0.0000692	0.7874535
	74	0.0012467	0.8391488	0.0000479	0.6925092
	75	0.0011161	0.8952418	0.0000366	0.7631700
	76	0.0010177	0.9118797	0.0000315	0.8615686
	77	0.0009285	0.9123426	0.0000269	0.8534135
	78	0.0008523	0.9179524	0.0000225	0.8376764
	79	0.0007739	0.9080042	0.0000224	0.9961889
	80	0.0006799	0.8785639	0.0000216	0.9629637
	81	0.0005881	0.8648625	0.0000201	0.9290428
	82	0.0004882	0.8301801	0.0000208	1.0351272
	83	0.0003913	0.8015060	0.0000176	0.8481698
	84	0.0003242	0.8285988	0.0000142	0.8060784
	85	0.0002702	0.8334427	0.0000129	0.9107954
	86	0.0002246	0.8311510	0.0000109	0.8432222
	87	0.0001916	0.8532194	0.0000089	0.8125177
	88	0.0001647	0.8592511	0.0000077	0.8643326
	89	0.0001439	0.8739769	0.0000063	0.8266711
	90	0.0001263	0.8773506	0.0000057	0.8934908
	91	0.0001118	0.8853776	0.0000047	0.8270530
	92	0.0001006	0.8995960	0.0000040	0.8506773
	93	0.0000920	0.9145468	0.0000031	0.7799253
	94	0.0000856	0.9305279	0.0000025	0.8191927
	95	0.0000800	0.9350646	0.0000021	0.8149292
	96	0.0000752	0.9394217	0.0000017	0.8363557
	97	0.0000698	0.9282142	0.0000017	0.9814936
	98	0.0000632	0.9062999	0.0000017	1.0138043
	99	0.0000547	0.8654208	0.0000018	1.0641346
	100	0.0000461	0.8418280	0.0000016	0.8914509
	101	0.0000391	0.8481096	0.0000014	0.8598060

102	0.0000338	0.8637455	0.0000011	0.8164981
103	0.0000254	0.7524275	0.0000009	0.8216432

2.6.5.3 Table CG with incomplete cholesky

Table CG with incomplete cholesky preconditioner $\epsilon = 10^{-2}$, $h = \frac{1}{\cdot}$

k	|e|	ratio	|r|	ratio
1	2.2117568	0.0000000	4.9847338	0.0000000
2	0.9605500	0.4342928	0.1236319	0.0248021
3	0.1101739	0.1146987	0.0740761	0.5991666
4	0.0098178	0.0891114	0.0064913	0.0876300
5	0.0006397	0.0651529	0.0007141	0.1100128
6	0.0000566	0.0885013	0.0000495	0.0693702
7	0.0000012	0.0210676	0.0000009	0.0178494

Table CG with incomplete cholesky preconditioner $\epsilon = 10^{-2}$, $h = \frac{1}{\cdot}$

k	|e|	ratio	|r|	ratio
1	3.1411335	0.0000000	6.9640117	0.0000000
2	2.0214097	0.6435287	0.1420103	0.0203920
3	1.3623817	0.6739760	0.1149900	0.8097304
4	0.6575397	0.4826399	0.0682371	0.5934171
5	0.1011955	0.1539003	0.0247907	0.3633033
6	0.0225210	0.2225494	0.0069472	0.2802356
7	0.0067767	0.3009055	0.0015084	0.2171254
8	0.0019548	0.2884659	0.0004569	0.3028904
9	0.0008448	0.4321526	0.0001217	0.2664539
10	0.0002326	0.2753055	0.0000595	0.4889375
11	0.0000681	0.2927750	0.0000127	0.2125976
12	0.0000188	0.2754409	0.0000052	0.4094558
13	0.0000022	0.1195568	0.0000003	0.0640489

Table CG with incomplete cholesky preconditioner $\epsilon = 10^{-2}$, $h = \frac{1}{\cdot}$

k	|e|	ratio	|r|	ratio
1	4.6047158	0.0000000	10.3310633	0.0000000
2	3.4648962	0.7524669	0.1937203	0.0187512
3	3.0608082	0.8833766	0.1211410	0.6253400
4	2.5757427	0.8415237	0.0563599	0.4652418
5	2.0650650	0.8017358	0.0413553	0.7337726
6	1.5529332	0.7520021	0.0446463	1.0795781
7	0.9346100	0.6018353	0.0369208	0.8269618
8	0.2831630	0.3029745	0.0265590	0.7193503
9	0.0732691	0.2587522	0.0112939	0.4252372
10	0.0321756	0.4391431	0.0039872	0.3530414
11	0.0178107	0.5535469	0.0013842	0.3471708
12	0.0110120	0.6182812	0.0008543	0.6171309
13	0.0073396	0.6665038	0.0004321	0.5057645
14	0.0039218	0.5343410	0.0002636	0.6101306
15	0.0023472	0.5984915	0.0001345	0.5102407
16	0.0013514	0.5757587	0.0001004	0.7467383
17	0.0007088	0.5244614	0.0000461	0.4586731
18	0.0002407	0.3396233	0.0000342	0.7426365
19	0.0001089	0.4524637	0.0000103	0.3000293
20	0.0000698	0.6407427	0.0000045	0.4406339
21	0.0000343	0.4918772	0.0000034	0.7510210
22	0.0000068	0.1977073	0.0000005	0.1584427

Table CG with incomplete cholesky preconditioner $\epsilon = 10^{-2}$, $h = \frac{1}{\cdot}$

k	|e|	ratio	|r|	ratio
1	6.4885732	0.0000000	14.5067213	0.0000000
2	5.2612798	0.8108531	0.2641937	0.0182118
3	5.0111119	0.9524511	0.1164715	0.4408564
4	4.6620201	0.9303364	0.0500341	0.4295820
5	4.3222195	0.9271130	0.0496910	0.9931436
6	3.9989939	0.9252177	0.0423171	0.8516054
7	3.6528553	0.9134436	0.0289715	0.6846290
8	3.2996211	0.9032991	0.0270517	0.9337335
9	2.9620134	0.8976829	0.0260842	0.9642352
10	2.6075430	0.8803279	0.0207816	0.7967142

```
              11       2.2325951       0.8562064       0.0176972       0.8515802
              12       1.8177887       0.8142044       0.0202955       1.1468181
              13       1.2782021       0.7031632       0.0227175       1.1193341
              14       0.5725682       0.4479481       0.0191123       0.8413068
              15       0.2181351       0.3809767       0.0101596       0.5315716
              16       0.1256071       0.5758222       0.0059499       0.5856484
              17       0.0759987       0.6050512       0.0036252       0.6092867
              18       0.0443639       0.5837452       0.0024628       0.6793574
              19       0.0240991       0.5432141       0.0014339       0.5822123
              20       0.0156123       0.6478383       0.0008604       0.6000823
              21       0.0094219       0.6034940       0.0005069       0.5891044
              22       0.0066705       0.7079759       0.0002955       0.5830443
              23       0.0054586       0.8183167       0.0001677       0.5673221
              24       0.0047031       0.8615927       0.0000873       0.5205250
              25       0.0040218       0.8551456       0.0000745       0.8537794
              26       0.0030834       0.7666791       0.0000688       0.9227595
              27       0.0021837       0.7082150       0.0000606       0.8814875
              28       0.0014407       0.6597450       0.0000421       0.6948456
              29       0.0011682       0.8108344       0.0000212       0.5036729
              30       0.0009944       0.8512118       0.0000176       0.8296109
              31       0.0007347       0.7388340       0.0000182       1.0336879
              32       0.0005166       0.7031852       0.0000114       0.6264216
              33       0.0003559       0.6888947       0.0000107       0.9380414
              34       0.0002062       0.5794708       0.0000081       0.7584751
              35       0.0001239       0.6006305       0.0000052       0.6377109
              36       0.0000837       0.6759086       0.0000032       0.6095393
              37       0.0000659       0.7871051       0.0000017       0.5390215
              38       0.0000539       0.8184570       0.0000013       0.7405635
              39       0.0000202       0.3752176       0.0000009       0.7459009
```

Tables for CG with incomplete cholesky preconditioner $\epsilon = 10^{-3}$

```
h =      1/16    ,n=17   eps=0.000001   method=CHL
               k             |e|           ratio             |r|           ratio
               1       2.2113015       0.0000000       4.8698707       0.0000000
               2       0.1268927       0.0573837       0.0196604       0.0040372
               3       0.0011025       0.0086886       0.0005919       0.0301065
               4       0.0000002       0.0001684       0.0000001       0.0001565

h =      1/32    ,n=33   eps=0.000001   method=CHL
               k             |e|           ratio             |r|           ratio
               1       3.2189095       0.0000000       7.0357482       0.0000000
               2       0.8589679       0.2668506       0.0306470       0.0043559
               3       0.0363666       0.0423375       0.0064895       0.2117505
               4       0.0025347       0.0696990       0.0003268       0.0503545
               5       0.0001276       0.0503368       0.0000319       0.0976751
               6       0.0000006       0.0047450       0.0000001       0.0037884
h =      1/64    ,n=65   eps=0.000001   method=CHL
               k             |e|           ratio             |r|           ratio
               1       4.5281055       0.0000000      10.2891303       0.0000000
               2       2.3962171       0.5291876       0.0305161       0.0029659
               3       1.0190972       0.4252942       0.0239474       0.7847488
               4       0.1236226       0.1213060       0.0102118       0.4264245
               5       0.0186393       0.1507755       0.0012096       0.1184530
               6       0.0016236       0.0871065       0.0002558       0.2114490
               7       0.0002756       0.1697321       0.0000182       0.0713307
               8       0.0000163       0.0592016       0.0000009       0.0502623
h =      1/128   ,n=129  eps=0.000001   method=CHL
               k             |e|           ratio             |r|           ratio
               1       6.4658650       0.0000000      14.3821767       0.0000000
               2       4.5500543       0.7037039       0.0349350       0.0024290
               3       3.5640524       0.7832989       0.0316566       0.9061590
               4       2.5832848       0.7248167       0.0229909       0.7262586
               5       1.4701895       0.5691163       0.0119903       0.5215250
               6       0.3578766       0.2434221       0.0077001       0.6421915
               7       0.0850789       0.2377325       0.0036799       0.4779069
               8       0.0274688       0.3228626       0.0009809       0.2665421
               9       0.0121215       0.4412808       0.0002445       0.2492235
              10       0.0034751       0.2866930       0.0001041       0.4258668
              11       0.0019516       0.5615987       0.0000341       0.3280329
              12       0.0010720       0.5492590       0.0000211       0.6186532
              13       0.0003595       0.3353743       0.0000058       0.2766025
```

```
41                14          0.0000439          0.1220343          0.0000010          0.1636738
```

Chapter 3

Study notes

3.1 note on finding expressions for centered difference of higher order

3.1.1 Introduction

To find centered difference approximation for $f'(x)$, we can processed as follows.

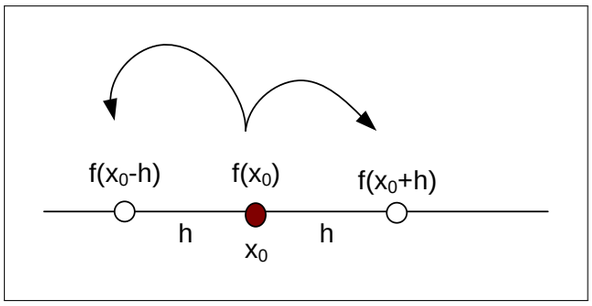


Figure 3.1: centered difference

3.1.2 approximation for $f'(x_o)$

Since

$$f(x_o + h) = f(x_o) + hf'(x_o) + O(h)$$

Then

$$f'(x_o) \approx \frac{1}{h}\left[f(x_o + h) - f(x_o)\right] \tag{1}$$

But we also know that

$$f(x_o - h) = f(x_o) - hf'(x_o) + O(h)$$

The trick is to find $f(x_o)$ from the above and plug it in (1). From the above we find

$$f(x_o) \approx f(x_o - h) + hf'(x_o)$$

substituting the above in (1) gives

$$\begin{aligned} f'(x_o) &\approx \frac{1}{h}\left[f(x_o + h) - \left(f(x_o - h) + hf'(x_o)\right)\right] \\ &\approx \frac{f(x_o + h) - f(x_o - h)}{h} - f'(x_o) \end{aligned}$$

Hence

$$f'(x_o) \approx \frac{f(x_o + h) - f(x_o - h)}{2h}$$

3.1.3 approximation for $f''(x_o)$

We can do the same trick to find centered difference approximation for $f''(x_o)$. Since

$$f(x_o+h) = f(x_o) + hf'(x_o) + \frac{h^2}{2}f''(x_o) + O\left(h^2\right)$$

Then

$$f''(x_o) \approx \frac{2}{h^2}\left(f(x_o+h) - f(x_o) - hf'(x_o)\right) \tag{2}$$

But we also know that

$$\begin{aligned} f(x_o-h) &= f(x_o) - hf'(x_o) + \frac{h^2}{2}f''(x_o) + O\left(h^2\right) \\ hf'(x_o) &\approx -f(x_o-h) + f(x_o) + \frac{h^2}{2}f''(x_o) \\ f'(x_o) &\approx \frac{f(x_o) - f(x_o-h)}{h} + \frac{h}{2}f''(x_o) \end{aligned}$$

Sustituting the above into (2), we find

$$\begin{aligned} f''(x_o) &\approx \frac{2}{h^2}\left[f(x_o+h) - f(x_o) - h\left(\frac{f(x_o) - f(x_o-h)}{h} + \frac{h}{2}f''(x_o)\right)\right] \\ &\approx \frac{2}{h^2}\left[f(x_o+h) - f(x_o) - \left(f(x_o) - f(x_o-h) + \frac{h^2}{2}f''(x_o)\right)\right] \\ &\approx \frac{2}{h^2}\left[f(x_o+h) - 2f(x_o) + f(x_o-h) - \frac{h^2}{2}f''(x_o)\right] \\ &\approx \frac{2}{h^2}\left(f(x_o+h) - 2f(x_o) + f(x_o-h)\right) - f''(x_o) \\ 2f''(x_o) &\approx \frac{2}{h^2}\left(f(x_o+h) - 2f(x_o) + f(x_o-h)\right) \end{aligned}$$

Solving for $f''(x_0)$ from the above gives

$$f''(x_o) \approx \frac{f(x_o+h) - 2f(x_o) + f(x_o-h)}{h^2}$$

This method can be used to find approximations for higher derivatives.

3.2 Generating Error Table, Handout of oct 8,2010 for the $u_{xx} = -\sin(3\pi x)$

Generating finite difference error table, Handout of oct 8,2010

Math 228A, UC Davis

Nasser M. Abbasi, 10/11/2010

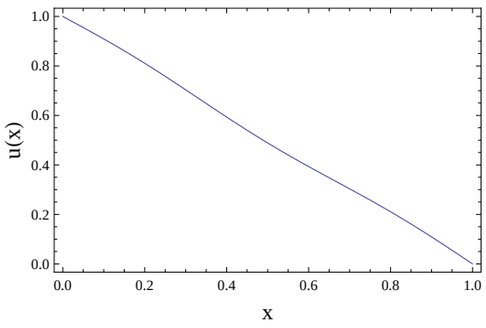
Problem: Generate error table for finite difference approximation to solution of $u_{xx} = \sin(3\pi x)$, u(0)=1,u(1)=0

In[6]:=
```
len = 1; (*length of element *)
h = len / 16; (*Initial spacing *)
α = 1; (* left Boundary conditions *)
β = 0; (* right Boundary conditions *)

(*generate spacings by halving it each time *)
nIter = 7;
hValues = Table[h / 2^i, {i, 0, nIter}];

(*---{nPoints,h,errorNorm2,errorNorm1,errorNormInfinity,errorNormFrobenius}--*)
tbl = Table[ process[N[hValues[[i]], len, x], {i, nIter + 1}];

(*we are done, format the table *)
tbl = Table[{
    tbl[[i, 1]],
    se[tbl[[i, 2]]],
    se[tbl[[i, 3]]], If[i == 1, 0, se[tbl[[i - 1, 3]] / tbl[[i, 3]]]],
    se[tbl[[i, 4]]], If[i == 1, 0, se[tbl[[i - 1, 4]] / tbl[[i, 4]]]],
    se[tbl[[i, 5]]], If[i == 1, 0, se[tbl[[i - 1, 5]] / tbl[[i, 5]]]],
    se[tbl[[i, 6]]], If[i == 1, 0, se[tbl[[i - 1, 6]] / tbl[[i, 6]]]]
   }, {i, 1, Length[tbl]}];
heading =
  {"pts", "h", "‖e‖₂", "ratio", "‖e‖₁", "ratio", "‖e‖∞", "ratio", "‖e‖frobenius", "ratio"};

Framed[TableForm[tbl, TableHeadings → {None, heading},
  TableSpacing → {1, 3}], ImageSize → 870]
```

$$\begin{pmatrix}
-2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -2
\end{pmatrix}$$

solution to u_{xx}=−sin(3πx), u(0)=1,u(1)=0

$$u(x) = -x + \frac{\sin(3\pi x)}{9\pi^2} + 1$$



Out[15]=

pts	h	$\|e\|_2$	ratio	$\|e\|_1$	ratio	$\|e\|_\infty$
15.	6.2500e−2	2.3423e−4	0	2.1020e−4	0	3.3125e−
31.	3.1250e−2	5.7795e−5	4.0527e	5.1992e−5	4.0429e	8.1734e−
63.	1.5625e−2	1.4402e−5	4.0131e	1.2964e−5	4.0106e	2.0367e−
127.	7.8125e−3	3.5975e−6	4.0033e	3.2387e−6	4.0027e	5.0876e−
255.	3.9063e−3	8.9919e−7	4.0008e	8.0955e−7	4.0007e	1.2717e−
511.	1.9531e−3	2.2479e−7	4.0002e	2.0238e−7	4.0002e	3.1790e−
1023.	9.7656e−4	5.6196e−8	4.0000e	5.0594e−8	4.0000e	7.9473e−
2047.	4.8828e−4	1.4049e−8	4.0000e	1.2648e−8	4.0000e	1.9868e−

- **Function process**

In[1]:=
```
process[h_, len_, x_] :=
  Module[{de, sol, solExact, pExact, u, f, A, approxSolution, data, pApprox, i,
    nPoints, error, errorNorm2, errorNorm1, errorNormInfinity, errorNormFrobenius},
   nPoints = len/h - 1;
   de = u''[x] == -Sin[3 Pi x];
   sol = First@DSolve[{de, u[0] == 1, u[1] == 0}, u[x], x];
   solExact = Simplify[(u[x] /. sol)];
   pExact = Plot[solExact, {x, 0, 1},
     Frame → True,
     PlotRange → All,
     FrameLabel → {{Style["u(x)", 14], None},
       {Style["x", 14], Column[{Style["solution to u_xx=-sin(3πx), u(0)=1,u(1)=0", 14],
          Style[Row[{"u(x) = ", solExact}], 14]}, Alignment → Center]}}
   ];

   A = makeA[nPoints];


   f = makeF[nPoints, α, β, rhs, len];
   approxSolution = LinearSolve[1/h^2 A, f];
   data = Table[{(i - 1) * h,
      If[i == 1, α, If[i == nPoints + 2, β, approxSolution[[i - 1]]]]}, {i, 1, nPoints + 2}];
   pApprox = ListPlot[data, Joined → True, AxesOrigin → {0, 0},
     Frame → True, PlotStyle → Red, PlotRange → All];

   If[h == 0.0625,
    {Print[MatrixForm[A]];
     Print[pExact];
    }];

   error = Table[approxSolution[[i]] - solExact /. x → (i * h), {i, 1, nPoints}];
   errorNorm2 = Norm[error, 2] * Sqrt[h];
   errorNorm1 = Norm[error, 1] * h;
   errorNormInfinity = Norm[error, Infinity];
   errorNormFrobenius = Norm[error, "Frobenius"] * Sqrt[h];

   {nPoints, h, errorNorm2, errorNorm1, errorNormInfinity, errorNormFrobenius}

  ];
```

- **Function to format decimal numbers like the handout**

In[2]:=
```
se[n_] := Module[{},
   Style[ScientificForm[N[n, $MachinePrecision], {6, 4},
     NumberFormat → (Row[{#1, "e", #3}] &), NumberPadding → {"", "0"}], 14]];
```

- **Function to construct A matrix**

In[3]:=
```
makeA[nPoints_] := Module[{A},
   fill[i_, j_] := If[i == j, -2, If[i == j + 1 || i == j - 1, 1, 0]];
   A = Table[fill[i, j], {i, nPoints}, {j, nPoints}]
  ];
```

- **Function to construct the f matrix**

In[4]:=
```
makeF[nPoints_, α_, β_, f_, len_] := Module[{h, rhs, i},
   h = len/(nPoints + 1);
   rhs = Table[If[i == 1, f[i * h] - α/h^2, If[i == nPoints, f[i * h] - β/h^2, f[i * h]]], {i, nPoints}]
  ];
```

- **Function to evaluate the rhs at a grid point**

In[5]:=
```
rhs[x_] := Module[{}, -Sin[3 π x]];
```

3.3 generate table 1 in textbook on approximation of derivatives

HTML

3.4 looking at eigenvalues of weighted jacobian iteration matrix

HTML