ON DUALITY AND THE BI-CONJUGATE GRADIENT ALGORITHM

by

Kristin E. Harnett

M.A., University of Pittsburgh, 2006

B.S., University of Pittsburgh, 2003

Submitted to the Graduate Faculty of

the Department of Mathematics in partial fulfillment

of the requirements for the degree of

Master of science

University of Pittsburgh

2008

UNIVERSITY OF PITTSBURGH

MATHEMATICS DEPARTMENT

This thesis was presented

by

Kristin E. Harnett

It was defended on

April 24th 2008

and approved by

Dr. William Layton, Department of Mathematics

Dr. Mike Sussman, Department of Mathematics

Dr. Leo Rebholz

Thesis Advisor: Dr. William Layton, Department of Mathematics

ON DUALITY AND THE BI-CONJUGATE GRADIENT ALGORITHM

Kristin E. Harnett, M.S.

University of Pittsburgh, 2008

It is not uncommon to encounter problems that lead to large, sparse linear systems with coefficient matrices that are invertible and sparse, but have little other structure. In such problems the solution $u = A^{-1}f$ is typically calculated only to acurately compute functionals of the solution, $L(u)$. This paper determines a method that converges rapidly to the functional's value. Specifially, a modified bi-conjugate gradient algorithm is found to generate convergence to the solution of linear functionals, $L(u)$, much more rapidly than convergence to the linear system solution u.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1.0 INTRODUCTION

Problems in many applications lead to large sparse linear systems with coefficient matrices that are invertible and sparse, but have little other structure. For example, convective transport problems yield linear systems

$$Au = f, \quad A : N \times N \text{ invertible matrix}, \tag{1.1}$$

where N can often be on the order of millions. Typically all that is known is that the symmetric part of A, $(A + A^{tr})/2$, is positive definite. Problems in three body scattering theory lead to linear systems like (1.1) where N is of the order of 20 million and the coefficient matrix is very sparse but has little other discernible structure. In such problems the solution $u = A^{-1}f$ is typically calculated only to compute accurately further statistics or functionals of that solution, upon which decisions are taken. This is obviously necessary; sifting through 20 million data values of equal importance is humanly impossible. Further, only $\mathbb{R}^1$ is well-ordered. Thus a design "improving" a previous one can only be decided upon if one number gets larger or smaller. The functionals extracted can be linear (or nonlinear) functionals, leading to the problem

$$\begin{aligned} \text{for } A \quad &: \quad \mathbb{R}^N \to \mathbb{R}^N \text{ and } L : \mathbb{R}^N \to \mathbb{R}, \\ \text{solve } Au \quad &= \quad f, \text{ compute } L(u). \end{aligned} \tag{1.2}$$

In all cases a very interesting and practical question arises [1]: **Can we determine methods that converge very rapidly to the functional's value?**

$$l_n \to L(u) \text{ much more rapidly than } u_n \to u. \tag{1.3}$$

This is especially important when the coefficient matrix A is not symmetric positive definite (SPD) and when the lack of discernible structure in A makes constructing effective preconditioners difficult.

1.1 PRELIMINARY WORK

Preliminary research into this question has been completed by Trisha R. Butler [2]. The robustness of post-processing and descent algorithms was explored by augmenting the algorithms with the Jacobi method to solve for several functionals. The data showed that the usefulness of the algorithms depended largely on which functional was being calculated and in some cases, whether N (the size of the linear system) was even or odd. For example, a post-processing algorithm was used with Jacobi to evaluate an average temperature and a heat flux functional (these functionals will be discussed in greater detail in Chapter 3). When N was even, the average temperature functional converged 4-5 times faster than the original solution converged. In contrast, when N was odd, the heat flux functional converged 1.2-1.5 times faster than the original solution. Thus, for both the post-processing and the descent algorithms, the functional l_n converged faster than the linear system, but the extent of the improvement depended greatly on whether N was even or odd. No explanation was found for the differences in the convergence of odd verus even system sizes. The results from using acceleration formulas to increase the convergence of l_n are inconsistent, but they do indicate that there are many avenues yet to be explored in answering the question posed in (1.3) [2].

1.2 THE SPD CASE

Many methods exist to solve problems in the form of (1.1). The case where A is SPD has solution methods that capitalize on the existence of an inner product and norm based on the matrix A. That is, the A-inner product exists and can be written as

$$\langle x, y \rangle_A = x^{tr} A y,$$

which leads to the A-norm:

$$\|x\|_A = \sqrt{x^{tr} A x}.$$

An A-orthogonal basis can be generated, beginning with a Krylov subspace, using the orthogonalization of moments algorithm. Given an A-orthogonal basis

$$\begin{aligned} &\{\phi_1, \ldots, \phi_N\} \\ &\text{with } \langle \phi_i, \phi_j \rangle_A = 0, \ i \neq j \\ &\text{and } \langle \phi_i, \phi_j \rangle_A \neq 0, \ i = j, \end{aligned} \tag{1.4}$$

then the solution of $Au = f$ can be written down explicitly. We can expand

$$u = \sum_{i=1}^{N} \alpha_i \phi_i, \ \alpha_i \in \mathbb{R},$$

and substitute into the original problem:

$$\begin{aligned} Au &= f, \\ \sum_{i=1}^{N} \alpha_i A\phi_i &= f, \\ \sum_{i=1}^{N} \alpha_i \langle A\phi_i, \phi_j \rangle &= \langle f, \phi_j \rangle. \end{aligned}$$

Using Equation 1.4, this gives an expression for α_i and for u:

$$\begin{aligned} \alpha_i &= \frac{\langle f, \phi_i \rangle}{\langle \phi_i, \phi_i \rangle_A}, \\ u &= \sum_{i=1}^{N} \frac{\langle f, \phi_i \rangle}{\langle A\phi_i, \phi_i \rangle} \phi_i. \end{aligned} \tag{1.5}$$

The series in (1.5) can be summed until a given accuracy is attained as follows:

Algorithm 1. *Conjugate Directions for $Au = f$*

For A SPD and $\phi_i, \ldots, \phi_N$ A-conjugate vectors

$$u_1 = \frac{\langle f, \phi_1 \rangle}{\langle A\phi_1, \phi_1 \rangle} \phi_1$$

compute until satisfied

$$\begin{gathered} \textit{given } u_n \\ \alpha_{n+1} = \frac{\langle f, \phi_{n+1} \rangle}{\langle \phi_{n+1}, \phi_{n+1} \rangle_A} \\ u_{n+1} = u_n + \alpha_{n+1}\phi_{n+1} \\ \textit{if } \|f - Au_{n+1}\| < Tolerance, \textit{ stop} \\ \textit{otherwise } n = n + 1. \end{gathered}$$

This is the basic algorithm for the SPD case. Efficiency dictates numerous refinements of the algorithm which culminate in the conjugate gradient method (CG).

1.3 THE NON-SPD CASE

In the non-SPD case, extensions are based on: (i) passing to the normal equation $A^{tr}Au = A^{tr}f$, (ii) L^2 orthogonalization of ϕ_i by Gram-Schmidt instead of A-conjugancy, or (iii) replacing A-orthogonality by A-biorthogonality (bi-conjugacy).

The bi-conjugate gradient method (Bi-CG) is a solution method for problems in the form of (1.1) where A is not SPD. The Bi-CG method is an extension of CG that utlizes a bi-conjugate sequence of vectors.

Definition 2. *Given an inner product $\langle \cdot, \cdot \rangle$ the sequences ϕ_i,ψ_j are bi-orthogonal if*

$$\begin{aligned} \langle \phi_i, \psi_j \rangle &= 0, \ i \neq j, \\ \langle \phi_i, \psi_j \rangle &\neq 0, \ i = j. \end{aligned}$$

Definition 3. *Given an inner product* $\langle\cdot,\cdot\rangle$ *the sequences* ϕ_i,ψ_j *are bi-conjugate (or A-biorthogonal) if*

$$\begin{aligned}\langle\phi_i, A^{tr}\psi_j\rangle &= \langle A\phi_i, \psi_j\rangle = 0,\ i \neq j,\\ \langle\phi_i, A^{tr}\psi_j\rangle &= \langle A\phi_i, \psi_j\rangle \neq 0,\ i = j.\end{aligned}$$

The solution to the problem $Au = f$ can be written down explicitely, given bi-conjugate sequences $\{\phi_i\}$,$\{\psi_j\}$. Again consider an expansion of the vector u:

$$u = \sum_i \alpha_i \phi_i.$$

Substitute into the original problem:

$$\begin{aligned} Au &= f,\\ \sum_{i=1}^{N} \alpha_i A\phi_i &= f,\\ \sum_{i=1}^{N} \alpha_i \langle A\phi_i, \psi_j\rangle &= \langle f, \psi_j\rangle.\end{aligned}$$

This gives an expression for α_i and for u:

$$\begin{aligned}\alpha_i &= \frac{\langle f, \psi_i\rangle}{\langle A\phi_i, \psi_i\rangle},\\ u &= \sum_{i=1}^{N} \frac{\langle f, \psi_i\rangle}{\langle A\phi_i, \psi_i\rangle}\phi_i.\end{aligned} \tag{1.6}$$

The algorithm breaks down and must be restarted if

$$\langle A\phi_i, \psi_i\rangle = 0.$$

The series in (1.6) can be summed until a given accuracy is attained. Refinements of this idea lead to the Bi-CG algorithm.

Algorithm 4. *Bi-CG for* $Au = f$

Guess u^1,

Calculate $r^1 = f - Au^1$

Set $\tilde{r}^1 = d^1 = \tilde{d}^1 = r^1$

For $k = 1, 2, 3, \ldots$ *until satisfied*

Calculate (if denominator vanishes: restart)

$$\begin{aligned} \alpha_k &= \frac{\langle \tilde{r}^k, r^k \rangle}{\langle \tilde{d}^k, Ad^k \rangle}, \\ u^{k+1} &= u^k + \alpha_k d^k \\ r^{k+1} &= r^k - \alpha_k A d^k \\ \tilde{r}^{k+1} &= \tilde{r}^k - \alpha_k A^{tr} \tilde{d}^k \end{aligned}$$

$$\textit{if } \|f - Au_{n+1}\| < Tolerance, \textit{ stop}$$

$$\textit{otherwise } n = n + 1.$$

1.4 CALCULATING FUNCTIONALS

The CG and Bi-CG algorithms solve problems in the form of (1.1). However, this thesis considers the problem of finding methods that converge rapidly to *functionals* of the solution to problem (1.1). That is, we here consider solving the problem as posed in (1.2):

$$\text{solve } u = A^{-1} f, \text{ compute } L(u).$$

If one can solve $Au = f$ to high accuracy and with minimal cost, it is surely most reliable to first solve $Au = f$ and then compute $L(u)$. The error in $L(u)$ is bounded by the error in u:

$$|L(u) - L(u_n)| \ = \left| l^{tr}(u - u_n) \right| \leq \|l\| \ \ \|u - u_n\| .$$

On the other hand, there are simple and important problems $Au = f$ (mostly highly non-symmetric and/or indefinite) which require very many iterates for good solutions. There

are thus problems in which a high accuracy solution of $Au = f$ is not feasible because of time and resource limitations. In fact, the solution may not converge at all with traditional methods. Thus, considering (1.3) is interesting for many of these problems. Recall (1.3): we seek

$$l_n \to L(u) \text{ much more rapidly than } u_n \to u.$$

To see this more clearly, consider a simple convection-diffusion problem. Let $U(x, y)$ be the solution of

$$-\varepsilon \Delta U + \mathbf{b} \cdot \nabla U = f(x, y), \text{ in } \Omega = (0, 1)^2 \text{ and } U = g(x, y) \text{ on } \partial\Omega,$$

where ε is the diffusion coefficient, $\mathbf{b}$ is the convection field, $f(x, y)$ is the heat source or sink, and $g(x, y)$ is the boundary condition. As the diffusion coefficient, ε, decreases the cell Péclet number becomes large. The cell Péclet number is:

$$Pe = \frac{h\,|v|}{2\varepsilon}.$$

A large Pe indicates more "turbulence" in the convection-diffusion problem [3]. The physical process that is modeled by the A matrix is more turbulent, and A is "more non-symmetric" as ε becomes smaller.

It is reasonable to expect that when ε decreases more iterations will be required to attain a sufficient level of accuracy. Table 1 reports the number of iterations that are necessary to solve the linear system arising from the central difference discretization on a uniform mesh of the convection-diffusion problem using Bi-CG with a fixed h as ε decreases.

Table 1 suggests that calculating accurate solutions becomes more costly as the problem becomes more non-symmetric. It is in these cases that we should expect methods of the form (1.3) to be of interest.

This thesis considers (1.3), building an iterative method for the simplest case in which $L(u)$ is either a linear functional or a smooth nonlinear functional for which linearization

Table 1: Convection Diffusion Problem (N=100, Tol = 1E-5)

ε	numerical value of ε	number of iterates
1	1	212
$1/\sqrt{N}$	0.1	199
$2/N$	0.02	166
1/500	0.002	798
1/1000	0.001	20,000 (divergence)
1/1500	6.66×10^{-4}	20,000 (divergence)
$1/N^2$	1×10^{-4}	20,000 (divergence)

is an effective tool. If the functional is linear, the problem can be posed as solving the augmented linear system for u and l: Find u, l satisfying:

$$L(u) - \langle l, u \rangle = 0, \quad Au = f. \tag{1.7}$$

If this augmented system is solved by standard iterative methods, no new algorithms arise. This approach is equivalent to computing $l_n = \langle l, u_n \rangle$ at each step.

The basic approach we explore is to solve iteratively the coupled problems

$$Au = f, \quad \text{and } A^{tr}\phi = l. \tag{1.8}$$

Lemma 5. *Let $r_n = f - Au_n$ and $A^{tr}\phi = l$. Then $\langle l, u \rangle = \langle l, u_n \rangle + \langle \phi, r_n \rangle$.*

Proof.

$$\begin{aligned} \langle l, u \rangle &= \langle l, u_n \rangle + \langle l, e_n \rangle \text{ where } e_n \text{ is the error in } u_n, \text{ the calculated solution at iterate } n. \\ &= \langle l, u_n \rangle + \left\langle A^{tr}\phi, e_n \right\rangle \\ &= \langle l, u_n \rangle + \langle \phi, r_n \rangle . \end{aligned}$$

□

For smooth nonlinear functionals $n(u)$ with derivative $n'(u)$ this becomes the linearized coupled system

$$Au = f, \quad \text{and } A^{tr}\phi_n = n'(u_n). \tag{1.9}$$

2.0 ADAPTING BI-CG METHODS

The Bi-CG algorithm is a natural choice for considering (1.3) due to the dual problems involved in calculating functionals of a solution to a system. Therefore consider the coupled systems

$$Au = f, \quad \text{and } A^{tr}\phi = l. \tag{2.1}$$

Algorithm 4 describes Bi-CG for the problem $Au = f$. The method has the following properties [4].

1. The residuals and shadow residuals are bi-orthogonal

$$\left\langle \tilde{r}^i, r^j \right\rangle = 0, \ i \neq j.$$

2. The search directions are bi-conjugate

$$\begin{aligned} \left\langle d^i, A\tilde{d}^j \right\rangle &= 0, \ i \neq j, \\ \left\langle d^i, A^{tr}\tilde{d}^j \right\rangle &= 0, \ i \neq j. \end{aligned}$$

3. The residuals and search directions are bi-orthogonal

$$\begin{aligned} \left\langle r^i, d^j \right\rangle &= 0, \ j < i, \\ \left\langle \tilde{r}^i, \tilde{d}^j \right\rangle &= 0, \ j < i. \end{aligned}$$

Given the Bi-CG residuals and shadow residuals, it is possible to use them in a projected Bi-CG algorithm to solve the dual problem

$$A^{tr}\phi = l.$$

Indeed, expanding

$$\phi = \sum_{j=1}^{N} a_j \tilde{d}^j$$

and substituting the expansion into the dual system gives

$$\begin{aligned} \left\langle d^i, A^{tr}\phi \right\rangle &= \left\langle d^i, l \right\rangle \\ \sum_{j=1}^{N} a_j \left\langle d^i, A^{tr}\tilde{d}^j \right\rangle &= \left\langle d^i, l \right\rangle, \text{ due to bi-conjugacy,} \\ a_j &= \frac{\langle d^j, l \rangle}{\left\langle d^j, A^{tr}\tilde{d}^j \right\rangle},\ j = 1, \ldots N. \end{aligned}$$

This can be implemented simply as follows.

Algorithm 6. *Summing bi-orthogonal series*

Given bi-conjugate directions $d^i, \tilde{d}^j$
Calculate $a_1 = \frac{\langle d^1, l \rangle}{\langle d^1, A^{tr}\tilde{d}^1 \rangle}$.
Set $\phi^1 = a_1 \tilde{d}^1$.
For $k = 2, 3, \ldots$ *until satisfied*
Calculate $a_k = \frac{\langle d^k, l \rangle}{\langle d^k, A^{tr}\tilde{d}^k \rangle}$.
Set $\phi^k = \phi^{k-1} + a_k \tilde{d}^k$.

Naturally, it is more efficient to modify this to allow an initial guess and to use this idea in a bi-conjugate direction method. In the simplest form, this is equivalent to choosing an initial guess ϕ^0 and applying the above bi-orthogonal series algorithm to the equivalent system:

$$\begin{aligned} \rho^0 &= l - A^{tr}\phi^0, \\ A^{tr}(\phi - \phi^0) &= \rho^0. \end{aligned}$$

The associated series is

$$\begin{aligned} \phi &= \phi^0 + \sum_{j=1}^{N} a_j \tilde{d}^j, \\ a_j &= \frac{\langle d^j, \rho^0 \rangle}{\left\langle d^j, A^{tr}\tilde{d}^j \right\rangle}, \ j = 1, \ldots N. \end{aligned}$$

Algorithmically, this becomes the following.

Algorithm 7. *Projected bi-conjugate directions*

Given bi-conjugate directions $d^i, \tilde{d}^j$

Guess ϕ^0

Calculate the dual residual $\rho^0 = l - A^{tr}\phi^0$

Calculate $a_1 = \frac{\langle d^1, \rho^0 \rangle}{\langle d^1, A^{tr}\tilde{d}^1 \rangle}$.

Set $\phi^1 = \phi^0 + a_1 \tilde{d}^1$

For $k = 1, 2, 3, \ldots$ *until satisfied*

Calculate $a_k = \frac{\langle d^k, \rho^0 \rangle}{\langle d^k, A^{tr}\tilde{d}^k \rangle}$;

Set $\phi^k = \phi^{k-1} + a_k \tilde{d}^k$.

We can thus combine the Bi-CG method with the projected bi-conjugate diretion method as follows.

Algorithm 8. *Bi-CG for* $Au = f, L(u) = \langle u, l \rangle$

Guess

$$u^1, \ \phi^0.$$

Calculate the residual

$$r^1 = f - Au^1.$$

Calulate the dual residual

$$\rho^0 = l - A^{tr}\phi^0.$$

Set

$$\begin{aligned} \tilde{r}^1 &= \rho^0, \\ d^1 &= r^1, \\ \tilde{d}^1 &= \tilde{r}^1. \end{aligned}$$

For $k = 1, 2, 3, \ldots,$ until satisfied

Update original problem's solution.

Calculate (if denominator vanishes: restart)

$$\begin{aligned} \alpha_k &= \frac{\left\langle \tilde{r}^k, r^k \right\rangle}{\left\langle \tilde{d}^k, Ad^k \right\rangle}, \\ u^{k+1} &= u^k + \alpha_k d^k, \\ r^{k+1} &= r^k - \alpha_k A d^k, \\ \tilde{r}^{k+1} &= \tilde{r}^k - \alpha_k A^{tr} \tilde{d}^k. \end{aligned}$$

Update dual problem's solution.

Calculate (if denominator vanishes: restart)

$$a_k = \frac{\left\langle d^k, \rho^0 \right\rangle}{\left\langle d^k, A^{tr} \tilde{d}^k \right\rangle}.$$

Set

$$\phi^k = \phi^{k-1} + a_k \tilde{d}^k.$$

Update approximation to functional's solution.

Calculate:

$$L_{k+1} = \langle l, u_{k+1} \rangle + \left\langle \phi^k, r^{k+1} \right\rangle.$$

Update search directions and shadow search directions.

Calculate (if denominator vanishes: restart)

$$\begin{aligned} \beta_k &= \frac{\left\langle \tilde{r}^{k+1}, r^{k+1} \right\rangle}{\left\langle \tilde{r}^k, r^k \right\rangle}, \\ d^{k+1} &= r^k + \beta_k d^k, \\ \tilde{d}^{k+1} &= \tilde{r}^k + \beta_k \tilde{d}^k. \end{aligned}$$

Algorithm 8 has several implications that may not be obvious at first glance. First, the choice of the initial shadow residual, $\tilde{r}^1$, differs from the basic Bi-CG algorithm. This is a consequence of the dual nature of the problem being considered. Recall that the algorithm has been modified to solve the coupled problem (2.1). The original residual and update are associated with the original problem, $Au = f$. The shadow residual and shadow update are associated with the dual problem $A^{tr}\phi = l$. Thus the shadow residual must be initialized using the original guess of the dual problem: $\tilde{r}^1 = \rho^0 = l - A^{tr}\phi^0$.

The other subtle implication of the modified algorithm is that the solution of the original problem, u, will differ depending on the choice of functional, l. Intuitively we expect that an algorithm should require the same number of iterations to solve for u regardless of the functional that is being simultaneously calculated. However, recall that the shadow residual, $\tilde{r}$, is initialized using the original guess of the dual problem, which will result in a different $\tilde{r}^1$and a different sequence of $\tilde{r}^k$ iterates for each functional. While the shadow residuals are associated with the dual problem, they are also used in the update of the original search direction: $d^{k+1} = r^k + \frac{\langle \tilde{r}^{k+1}, r^{k+1} \rangle}{\langle \tilde{r}^k, r^k \rangle} d^k$. Thus the choice of functional will affect the number of iterations required to calculate the solution of the original problem, u. Some numerical examples of this implication can be found in Tables 2-10 in Chapter 4.

3.0 FUNCTIONALS

Three linear functionals are used to test the modified Bi-CG method that is described in Algorithm 7. We shall consider the problem of estimating these functionals in a thermal convection-diffusion problem. Thus, let $U(x, y)$ be the solution of

$$-\varepsilon \Delta U + \mathbf{b} \cdot \nabla U = f(x, y), \text{ in } \Omega = (0, 1)^2 \text{ and } U = g(x, y) \text{ on } \partial\Omega, \tag{3.1}$$

where ε is the diffusion coefficient, $\mathbf{b}$ is the convection field, $f(x, y)$ is the heat source or sink, and $g(x, y)$ is the boundary condition. An interesting functional associated with equation (3.1) is the average temperature:

$$L(U) = \int \int_{\Omega} U(x, y) dx dy.$$

In 2-D, the average temperature is calculated discretely on an $N \times N$ mesh by (note we change notation to represent U discretely: $U = (U_1, \ldots U_{N^2})^{tr}$):

$$L(U) = \frac{1}{N^2} \sum_{k=1}^{N^2} U_k.$$

Written in terms of dot products, $L(U)$ becomes

$$\begin{aligned} L(U) &= \left(\frac{1}{N^2} \cdots \frac{1}{N^2} \right) \cdot \begin{pmatrix} U_1 \\ \vdots \\ U_{N^2} \end{pmatrix} = \begin{pmatrix} \frac{1}{N^2} \\ \vdots \\ \frac{1}{N^2} \end{pmatrix}^{tr} \cdot \begin{pmatrix} U_1 \\ \vdots \\ U_{N^2} \end{pmatrix} \\ &= \frac{1}{N^2} \langle \mathbf{1}, U \rangle \text{ where } \mathbf{1} \text{ is the vector with all 1's.} \end{aligned} \tag{3.2}$$

The vector l is then $\frac{1}{N^2} (1 \quad \cdots \quad 1)^{tr}$ so that $L(U) = \langle l, U \rangle$.

Another linear functional addressed in this paper is the heat flux. Let Γ denote one face of the flow domain with outward unit normal $\hat{n}$. The problem is: given diffusion coefficient ε, the convection field $\mathbf{b}$, heat source or sink $f(x,y)$, and temperature on the boundary $g(x,y)$, find

$$L_F(U) = \int_\Gamma \varepsilon \nabla U \cdot \hat{n} ds.$$

In the case where $f(x,y) = 0, g(x,y) = x$, and the hot wall is the side $x = 1$, heat flux is represented using the equation

$$L_F(U) = \int_0^1 \varepsilon \frac{\partial U}{\partial x}(1,y) dy. \tag{3.3}$$

A linear functional for the heat flux is developed in Lemma 6 of [2]. A modification of Lemma 6 follows:

Lemma 9 (Linear Heat Flux Functional)**.** *Using equation (3.1) with* $f(x,y) = \mathbf{0}$ *and* $g(x,y) = x$*, and the hot wall side* $x = 1$*, the heat flux from equation (3.3) is equivalent to*

$$\begin{aligned} L_F(U) &= \varepsilon \left(h + N + \langle l, U \rangle\right), \text{ where} \\ l &= \begin{cases} 0, & k \leq N^2 - N \\ -1, & k > N^2 - N \end{cases} \end{aligned} \tag{3.4}$$

Proof. Let $\Psi = \varepsilon \frac{\partial U}{\partial x}(1,y)$ and $\Psi_j = \frac{1 - U_N^j}{h}$. Then, by the Trapezoid Rule, equation (3.3)

becomes

$$\begin{aligned}
L_F(U) &= \varepsilon \sum_{j=1}^{N+1} \frac{h}{2}(\Psi_{j-1} + \Psi_j) \\
&= \varepsilon \frac{h}{2} \left(\sum_{j=0}^{N} \Psi_j + \sum_{j=1}^{N+1} \Psi_j \right) \\
&= \varepsilon \frac{h}{2} \left(\Psi_0 + 2 \sum_{j=1}^{N} \Psi_j + \Psi_{N+1} \right) \\
&= \varepsilon \frac{h}{2} \left[(\Psi_0 + \Psi_{N+1}) + 2 \sum_{j=1}^{N} \Psi_j \right] \\
&= \varepsilon \frac{h}{2} (1+1) + \varepsilon h \sum_{j=1}^{N} \frac{1 - U_N^j}{h} \\
&= \varepsilon \left(h + \sum_{j=1}^{N} \left(1 - U_N^j\right) \right) \\
&= \varepsilon \left(h + N - \sum_{j=1}^{N} U_N^j \right).
\end{aligned}$$

For $k = (i-1)N + j$, we have $l_k = -1$ for $k > N^2 - N$. Then

$$L_F(U) = \varepsilon \left(h + N + \langle l, U \rangle \right).$$

Since $L_F(U)$ is affine but not linear, the iterative method calculates the linear functional $\langle l, U \rangle$ to the desired tolerance. In order to find the value of the heat flux, $L(U)$, one must then sum $\varepsilon(h + N + \langle l, U \rangle)$. □

The last functional that is considered here is the checkerboard functional. The checkerboard functional does not have a physical meaning as the average temperature and heat flux functionals do. However, it is a rigorous test of the method because it causes the algorithm to fail for reasons that are different from why it might fail for the average temperature or heat flux functionals.

The checkerboard functional used here is

$$\begin{aligned} \text{For } i &= 1, \ldots, N \text{ and } j = 1, \ldots, N \\ \text{let } m &= N(i-1) + j. \\ l^{(m)} &= \begin{cases} 2, \text{ if } m \text{ is even} \\ 0, \text{ if } m \text{ is odd} \end{cases} . \end{aligned} \tag{3.5}$$

The checkerboard functional is so named because if the mesh were colored red where $L(U)$ is zero and colored black where $L(U)$ is two, it would resemble a checkerboard.

Note that each functional, l, is an $N^2 \times 1$ vector. For the test problem considered herein, A is an $N^2 \times N^2$ matrix.

4.0 NUMERICAL RESULTS

4.1 ERROR ANALYSIS

Why should we expect a disparity in the number of iterations required for convergence of the functional versus the original solution? The following lemma shows the relationship between the error in the original solution and the error in the functional solution.

Lemma 10 (Error Analysis)**.** *The error in the functional is quadratic in the error of the solution of the original problem.*

$$\begin{aligned} L(u) &= \langle l, u_n \rangle + \langle r_n, \phi \rangle \\ L_n &= \langle l, u_n \rangle + \langle r_n, \phi_n \rangle \\ |L(u) - L_n| &= |\langle r_n, \phi - \phi_n \rangle| = |\langle A\left(u - u_n\right), \phi - \phi_n \rangle| . \end{aligned}$$

Notice that the error of the functional, $L(u) - L_n$, is equivalent to an inner product which contains the error of the solution to the original linear system, $u - u_n$. As such, the error of the functional is a quadratic function of the error of the original solution. The implication of Lemma 10 is that the error in the functional will not only be smaller than the error in the solution of the original linear system, it will also decrease at a faster rate. We should expect to find that fewer iterations are needed to calculate the functional than are needed to calculate the solution to the original problem. We should also expect to see a faster convergence rate in the functional solution than in the original problem solution. Section 4.2 is a numerical exploration of the number of iterations required to calculate both the functional and the original solution. Chapter 5 focuses on estimating and comparing convergence rates of the functional and of the solution of the original linear system.

4.2 NUMERICAL EXPLORATION

The adapted Bi-CG method performs well when calculating the three test functionals. In all convergent cases, the functional converges faster than the solution of the original problem. That is, we observe $l_n \to L(u)$ more rapidly than $u_n \to u$.

The tables below report the number of iterations required for (i) convergence of the solution of the original problem and (ii) convergence of the functional. The notation *its* (e_o) indicates the number of iterations required for the relative error of the solution of the original problem,

$$\frac{\|u_{true} - u_n\|}{\|u_{true}\|},$$

to reach the desired tolerance of 1×10^{-5}. The notation *its* (e_f) indicates the number of iterations required for the relative error of the functional,

$$\frac{\|L(u_{true}) - L_n\|}{\|L(u_{true})\|},$$

to reach the desired tolerance of 1×10^{-5}. (Stopping criteria will be discussed in detail in Chapter 6.)

The data in Tables 2, 3, and 4 reveal that fewer iterations are required for convergence of the functional than for convergence of the solution of the original problem. Table 2 reports the data for the case in which the diffusion coefficient is large ($\varepsilon = 1$), thus the matrix A is nearly symmetric. Tables 3 and 4 report the results for cases in which A becomes increasingly non-symmetric: $\varepsilon = \frac{1}{\sqrt{N}}$ and $\varepsilon = \frac{1}{N}$. Tables 5, 6, and 7 report the same information for the Heat Flux functional and Tables 8, 9, and 10 report the information for the Checkerboard functional.

The results indicate that the adapted Bi-CG method starts to fail as the system becomes larger (as N gets larger) and less symmetric (as ε shrinks). The notation "DIV" indicates that the method diverged. Divergence is signaled when the method reaches the maximum number of iterations, here $MaxIts = 10,000$ iterations.

Table 2: Average Temperature Functional (eps = 1/N)

h	N	$its\ (e_o)$	$its\ (e_f)$
1/11	10	22	14
1/64	63	110	76
1/95	94	450	172
1/152	151	DIV	DIV
1/301	300	DIV	DIV

Table 3: Average Temperature Functional (eps = 1/sqrtN)

h	N	$its\ (e_o)$	$its\ (e_f)$
1/11	10	23	15
1/64	63	133	75
1/95	94	185	101
1/152	151	282	154
1/301	300	582	251

Table 4: Average Temperature Functional (eps = 1/N)

h	N	$its\ (e_o)$	$its\ (e_f)$
1/11	10	22	14
1/64	63	110	76
1/95	94	450	172
1/152	151	DIV	DIV
1/301	300	DIV	DIV

Table 5: Heat Flux Functional (eps = 1)

h	N	$its\ (e_o)$	$its\ (e_f)$
1/11	10	23	15
1/64	63	135	89
1/95	94	201	132
1/152	151	321	207
1/301	300	636	403

Table 6: Heat Flux Functional (eps = 1/sqrtN)

h	N	$its\ (e_o)$	$its\ (e_f)$
1/11	10	23	15
1/64	63	128	61
1/95	94	181	113
1/152	151	293	164
1/301	300	563	162

Table 7: Heat Flux Functional (eps = 1/N)

h	N	$its\ (e_o)$	$its\ (e_f)$
1/11	10	22	13
1/64	63	108	80
1/95	94	202	154
1/152	151	DIV	DIV
1/301	300	DIV	DIV

Table 8: Checkerboard Functional (eps = 1)

h	N	$its\ (e_o)$	$its\ (e_f)$
1/11	10	25	14
1/64	63	138	86
1/95	94	199	128
1/152	151	339	196
1/301	300	DIV	353

Table 9: Checkerboard Functional (eps = 1/sqrtN)

h	N	$its\ (e_o)$	$its\ (e_f)$
1/11	10	23	15
1/64	63	121	79
1/95	94	188	101
1/152	151	280	151
1/301	300	582	251

Table 10: Checkerboard Functional (eps = 1/N)

h	N	$its\ (e_o)$	$its\ (e_f)$
1/11	10	22	14
1/64	63	103	73
1/95	94	DIV	DIV
1/152	151	DIV	DIV
1/301	300	DIV	DIV

Notice the last row of Table 8: the solution of the original problem diverged, but the functional converged in only 353 iterations. The modified Bi-CG algorithm allowed us to accurately solve the aspect of the problem that we are taking decisions on (the functional of the system's solution) even though the method did not compute an accurate solution of the original linear system!

5.0 CONVERGENCE

5.1 HOW MANY ITERATIONS DOES AN ITERATIVE METHOD USE BEFORE CONVERGING?

Suppose the iterative operator is T. Then we know

$$e_{n+1} = Te_n \tag{5.1}$$

where e_n is the error at iteration step n. Equivalently we can say

$$e_{n+1} = T^n e_0.$$

By the multiplicative property of operator norms we have

$$\begin{aligned} \|e_n\| &\leq \|T\|^n \|e_0\|, \\ \frac{\|e_n\|}{\|e_0\|} &\leq \|T\|^n. \end{aligned}$$

This inequality can be used to bound the relative error. In order to bound the relative error by some number ϵ, the following condition must hold:

$$\frac{\|e_n\|}{\|e_0\|} < \epsilon.$$

It is thus sufficient to have:

$$\|T\|^n < \epsilon.$$

After taking logs this condition can be restated as

$$n \ln(\|T\|) < \ln(\epsilon).$$

A sufficient condition for convergence of the iterative method is $\|T\| < 1$. If this is the case, then we know that $\ln(\|T\|) < 0$. After rearranging we get the following condition for the number of iterations, n, required for a convergent iterative method to reduce the error by ϵ :

$$n \geq \frac{\ln(\epsilon)}{\ln(\|T\|)}. \tag{5.2}$$

We can use Equation (5.2) to determine how many iterations of the Bi-CG algorithm are needed to obtain one significant digit of accuracy by taking $\epsilon = 0.1$. However, we must first estimate the norm of the iteration operator, $\|T\|$. From (5.1) we can estimate $\|T\|$ as

$$\|T\| \simeq \frac{\|e_{n+1}\|}{\|e_n\|}. \tag{5.3}$$

According to (5.3), to estimate $\|T\|$ we need an estimate of *true error*(k) where k is the number of iterations. The next section deals with the problem of estimating $\|T\|$.

5.2 ESTIMATING THE TRUE ERROR FUNCTION AND THE ITERATION OPERATOR

5.2.1 Average Temperature Functional with Epsilon = 1

Consider the Average Temperature functional with N=200 and $\varepsilon = 1$. Figure 1 is a semi-log plot (the y-axis has a logarithmic scale) of the true error of the original problem $Au = f$ and the true error of the functional $L(u)$. The figure indicates that the true error of the functional converges faster and at a faster rate than the true error of the original problem of finding $u = A^{-1}f$.

Upon inspection, it appears that there is a bend in the functional error plot at iterate number 250. Because of the bend in the error plots, the estimation problem will be treated in two parts. One pair of convergence rates will be estimated for the two errors (the original problem error and the functional error) before the bend occurs at iterate 250. A second pair of convergence rates will be estimated for the two errors after the convergence rate appears

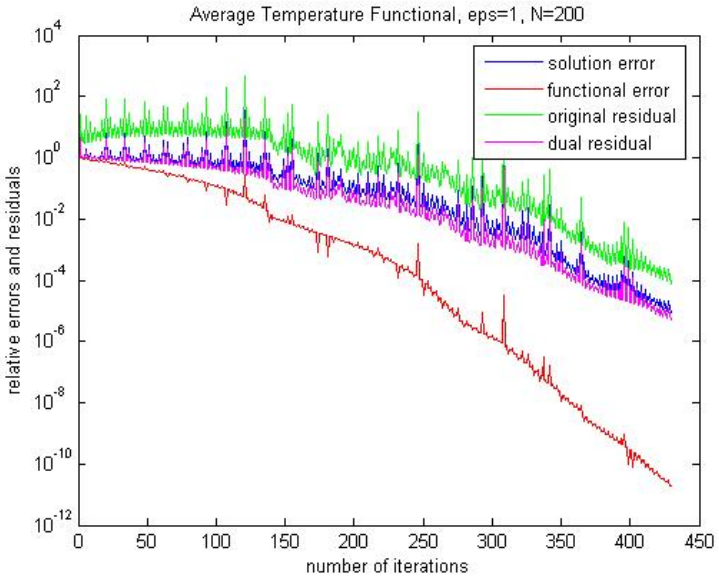

Figure 1: Average Temperature Functional

to increase at the bend in the plot. The second pair of convergence rates corresponds to iterates 250 through 430.

The slopes of these two line segments are each estimated using a linear fit to the logarithm of the true error data. The functional form of the fitted curve is

$$e(k) = a * k + b$$

where e is the log of the error associated with iterate k. Figure 2 and Figure 3 show the error data and the associated linear fitted curves:

The estimated linear equations for the first 250 iterates are

$$\begin{gathered} \text{Original Problem} \\ e_{original}(k) = -0.006 * k + 0.351 \\ and \\ \text{Functional} \\ e_{functional}(k) = -0.016 * k + 0.437. \end{gathered} \tag{5.4}$$

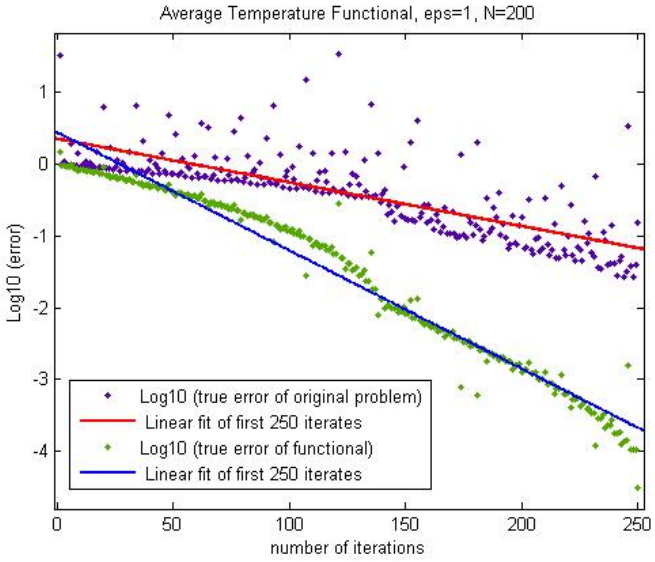

Figure 2: First 250 Iterates of the Average Temperature Funcitonal

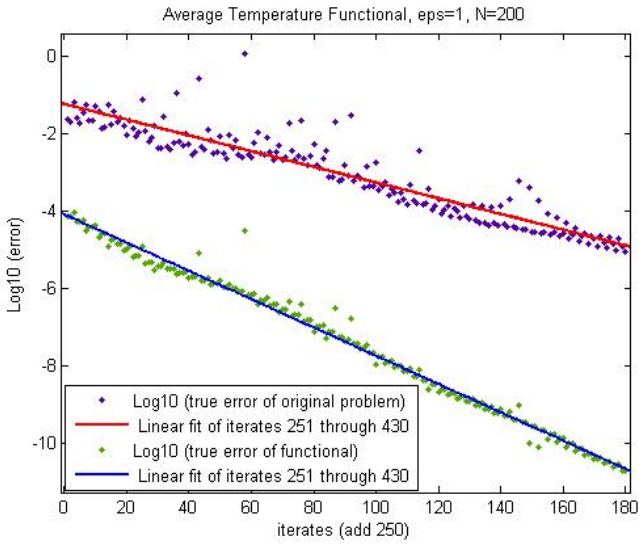

Figure 3: Iterates (251:430) of Average Temperature Functional

The estimated linear equations for iterates 250 through 430 are

$$\begin{gathered} \text{Original Problem} \\ e_{original}(k) = -0.020 * k - 1.224 \\ and \\ \text{Funtional} \\ e_{functional}(k) = -0.037 * k - 4.079. \end{gathered} \tag{5.5}$$

Using these two estimated equations, we can derive estimates for the norm of the iterative operator for each of these two problems.

Conclusion 11 (Estimating $\|T\|$)**.**

$$\begin{aligned} Let\ e_i(k) &= \log(true\ error_i)\ where\ i = original,\ functional \\ e_i(k) &= a_i * k + b_i \\ e_i(k+1) &= a_i * (k+1) + b_i \\ e_i(k+1) &= a_i * k + a_i + b_i \\ e_i(k+1) &= a_i + e_i(k) \\ true\ error_i(k+1) &= 10^{a_i} * true\ error_i(k) \\ true\ error_i(k+1) &= \|T\|_i * true\ error_i(k) \end{aligned}$$

From (5.3) and the estimated equations (5.4) and (5.5) we now have estimates of $\|T\|_i$ for both segments of the errors:

$$\begin{gathered} \text{First 250 iterates} \\ \|T\|_{original} = 10^{-0.006} \\ \|T\|_{functional} = 10^{-0.016} \end{gathered}$$

$$\begin{gathered} \text{Iterates 250 through 430} \\ \|T\|_{original} = 10^{-0.020} \\ \|T\|_{functional} = 10^{-0.037}. \end{gathered}$$

It is useful to know how many iterations are required of the Bi-CG iterative method to obtain one significant digit of accuracy. This is equivalent to requiring that the relative error be bounded by $\epsilon = 0.1$. Inserting the estimated norms of the iterative method into (5.2)

provides the condition for the number of iterations that are needed to gain one significant digit of accuracy:

First 250 iterations

$$n_{original} \geq \frac{\ln(0.1)}{\ln(10^{-0.006})} \simeq 167 \text{ iterations per significant digit of accuracy}$$

$$n_{functional} \geq \frac{\ln(0.1)}{\ln(10^{-0.016})} \simeq 63 \text{ iterations per significant digit of accuracy.}$$

Iterations 250 through 430

$$n_{original} \geq \frac{\ln(0.1)}{\ln(10^{-0.020})} \simeq 50 \text{ iterations per significant digit of accuracy}$$

$$n_{functional} \geq \frac{\ln(0.1)}{\ln(10^{-0.037})} \simeq 27 \text{ iterations per significant digit of accuracy.}$$

5.2.2 Average Temperature Functional with Epsilon = $1/\sqrt{N}$

As we saw in Table 1, changing the diffusion coefficient, ε, changes the number of iterations required for convergence. How does a change in ε affect convergence rates? Let us consider the same problem of estimating the iteration operators and number of iterations required to gain one significant digit of accuracy with the Average Temperature Functional, $N = 200$, but with a smaller diffusion coefficient: $\varepsilon = 1/\sqrt{N} = 1/\sqrt{200}$.

Figure 4 is a semi-log plot of the true error of the original problem $Au = f$ and the true error of the functional $L(u)$. The figure indicates that the true error of the functional converges faster than the true error of the original problem.

Upon inspection, it again appears that there is a bend in the functional error plot at iterate number 250. Thus, again one pair of convergence rates will be estimated for the first 250 iterates and another pair of convergence rates will be estimated for iterates 250 through 365.

The slopes of these two line segments are each estimated using a linear fit to the logarithm of the true error data. The functional form of the fitted curve is again

$$e(k) = a * k + b$$

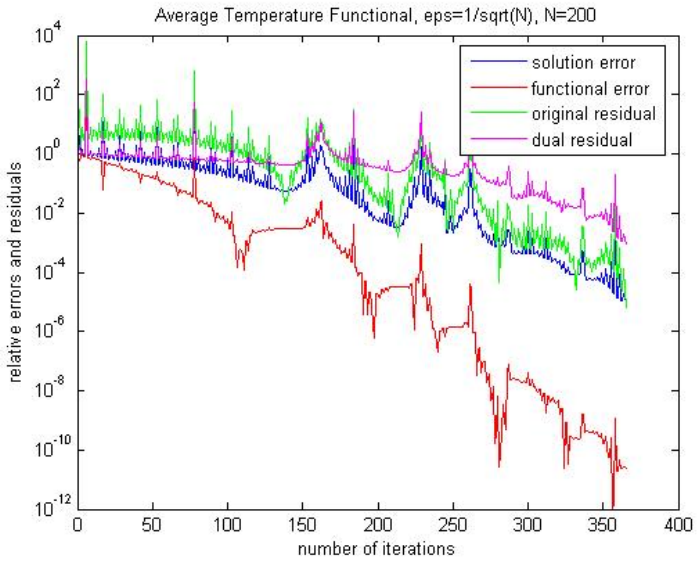

Figure 4: Average Temperature Functional: epsilon = 1/sqrtN

where e is the log of the error associated with iterate k. Figure 5 and Figure 6 show the error data and the associated linear fitted curves:

The estimated linear equations for the first 250 iterates are

$$\begin{gathered} \text{Original Problem} \\ e_{original}(k) = -0.009 * k + 0.323 \\ and \\ \text{Functional} \\ e_{functional}(k) = -0.022 * k + 0.245. \end{gathered} \tag{5.6}$$

The estimated linear equations for iterates 250 through 365 are

$$\begin{gathered} \text{Original Problem} \\ e_{original}(k) = -0.023 * k - 2.009 \\ and \\ \text{Functional} \\ e_{functional}(k) = -0.038 * k - 6.169. \end{gathered} \tag{5.7}$$

Using these two estimated equations, we can derive estimates for the norm of the iterative operator for each of these two problems.

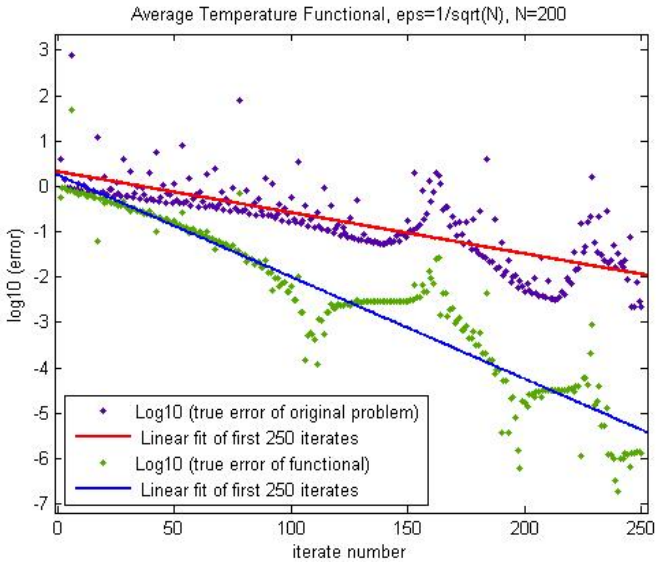

Figure 5: First 250 iterates of Average Temperature Functional

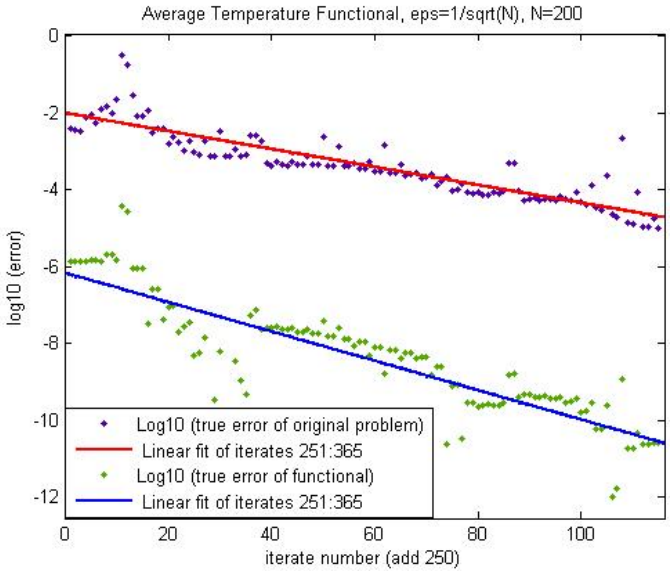

Figure 6: Iterates (251 : 365) of Average Temperature Functional

From (5.3) and the estimated equations (5.6) and (5.7) we now have estimates of $\|T\|_i$ for both segments of the errors:

First 250 iterates

$$\|T\|_{original} = 10^{-0.009}$$

$$\|T\|_{functional} = 10^{-0.022}$$

Iterates 250 through 365

$$\|T\|_{original} = 10^{-0.023}$$

$$\|T\|_{functional} = 10^{-0.038}.$$

Again, (5.2) provides the condition for the number of iterations that are needed to gain one significant digit of accuracy:

First 250 iterations

$$n_{original} \geq \frac{\ln(0.1)}{\ln(10^{-0.009})} \simeq 111 \text{ iterations per significant digit of accuracy}$$

$$n_{functional} \geq \frac{\ln(0.1)}{\ln(10^{-0.022})} \simeq 45 \text{ iterations per significant digit of accuracy.}$$

Iterations 250 through 365

$$n_{original} \geq \frac{\ln(0.1)}{\ln(10^{-0.023})} \simeq 43 \text{ iterations per significant digit of accuracy}$$

$$n_{functional} \geq \frac{\ln(0.1)}{\ln(10^{-0.038})} \simeq 26 \text{ iterations per significant digit of accuracy.}$$

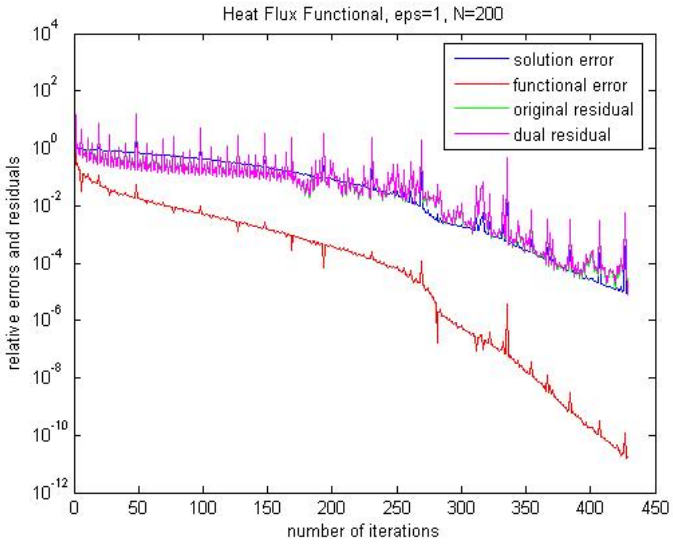

Figure 7: Heat Flux Functional

5.2.3 Heat Flux Functional with Epsilon = 1

This same analysis can be applied to the Heat Flux Functional with $\varepsilon = 1$ and $N = 200$.

Figure 7 is a semi-log plot of the true error of the original problem $Au = f$ and the true error of the functional $L(u)$. The figure indicates that the true error of the functional converges faster than the true error of the original problem.

Upon inspection, it again appears that there is a bend in the functional error plot near iterate number 250. Thus, again one pair of convergence rates will be estimated for the first 250 iterates and another pair of convergence rates will be estimated for iterates 250 through 429. Figure 8 and Figure 9 show the error data and the associated linear fitted curves:

The estimated linear equations for the first 250 iterates are

$$\begin{gathered} \text{Original Problem} \\ e_{original}(k) = -0.006 * k + 0.199 \\ and \\ \text{Functional} \\ e_{functional}(k) = -0.012 * k - 1.008. \end{gathered} \tag{5.8}$$

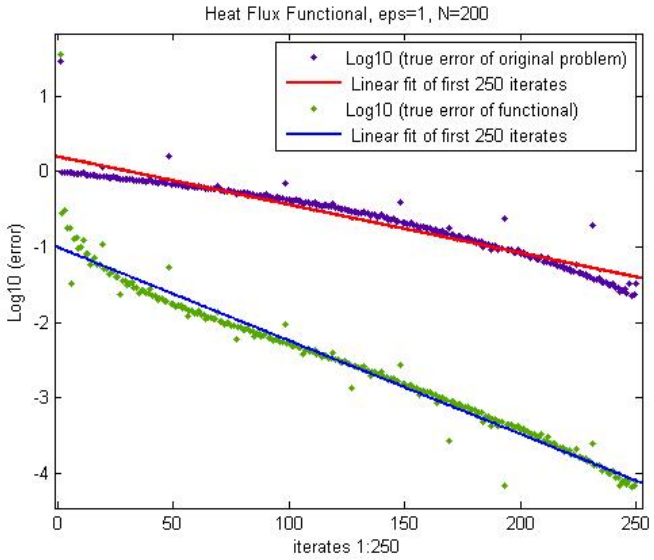

Figure 8: First 250 iterates of Heat Flux Functional

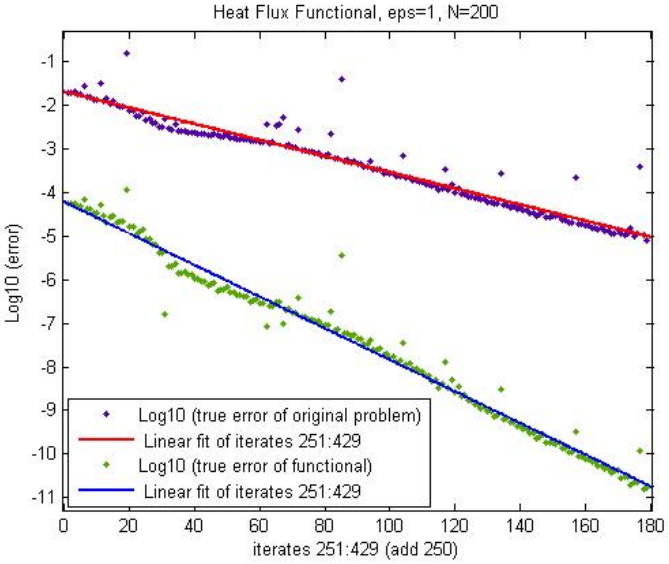

Figure 9: Iterates (251:429) of Heat Flux Functional

The estimated linear equations for iterates 250 through 429 are

$$\begin{gathered}\text{Original Problem}\\ e_{original}(k) = -0.019 * k - 1.688\\ and\\ \text{Functional}\\ e_{functional}(k) = -0.036 * k - 4.220.\end{gathered} \tag{5.9}$$

From (5.3) and the estimated equations (5.8) and (5.9) we now have estimates of $\|T\|_i$ for both segments of the errors:

$$\begin{gathered}\text{First 250 iterates}\\ \|T\|_{original} = 10^{-0.006}\\ \|T\|_{functional} = 10^{-0.012}\end{gathered}$$

$$\begin{gathered}\text{Iterates 250 through 429}\\ \|T\|_{original} = 10^{-0.019}\\ \|T\|_{functional} = 10^{-0.036}.\end{gathered}$$

Again, (5.2) provides the condition for the number of iterations that are needed to gain one significant digit of accuracy:

$$\begin{gathered}\text{First 250 iterations}\\ n_{original} \geq \frac{\ln(0.1)}{\ln(10^{-0.006})} \simeq 167 \text{ iterations per significant digit of accuracy}\\ n_{functional} \geq \frac{\ln(0.1)}{\ln(10^{-0.012})} \simeq 83 \text{ iterations per significant digit of accuracy.}\end{gathered}$$

$$\begin{gathered}\text{Iterations 250 through 429}\\ n_{original} \geq \frac{\ln(0.1)}{\ln(10^{-0.019})} \simeq 53 \text{ iterations per significant digit of accuracy}\\ n_{functional} \geq \frac{\ln(0.1)}{\ln(10^{-0.036})} \simeq 28 \text{ iterations per significant digit of accuracy.}\end{gathered}$$

5.3 ACCELERATING CONVERGENCE RATES

There are some features of this problem that we can utilize to try to accelerate the convergence rates. The obvious place to look for possible accelerations is in the initial guesses of the original and dual solutions: u_0 and ϕ_0. Choosing $u_0 = (0, \ldots, 0)$, an initial guess of zero for the solution of the original problem, is standard practice to which we will adhere. Perhaps the choice of ϕ_0 can be modified to result in accelerations of the convergence rate.

$$\begin{aligned} & \text{Given } u_0 \\ r_0 &= f - Au_0 \\ \text{Select } \phi_0 &= \arg\min \left\{ \left\| l - A^{tr}\phi \right\| : \phi = \alpha r_0 \right\} \\ \alpha &= \frac{\langle A^{tr} r_0, l \rangle}{\langle A^{tr} r_0, A^{tr} r_0 \rangle} \\ \phi_0 &= \alpha r_0. \end{aligned}$$

This choice of ϕ_0 is the best possible initial guess for the dual problem in $span\{r_0\}$. Unfortunately this choice of ϕ_0 does not accelerate the convergence of the Bi-CG algorithm. The new choice of ϕ_0 turns out to be zero, which is the same initial guess that we were using before! The denominator of the coefficient α is extremely large, leading to a multiplication by zero when calculating ϕ_0.

Let us consider why the denominator is so large. Recall that we are trying to solve the coupled problem

$$Au = f, \text{ and } A^{tr}\phi = l \tag{5.10}$$

where A is large and sparse. If $u_0 = (0, \ldots, 0)$, then $r_0 = f$. The choice of f here leads to $\|A^{tr} f\|$ being very large (on the order of 10^5).

Acceleration by optimal choice of ϕ_0 may yet be useful. If u_0 is chosen as something other than $u_0 = (0, \ldots, 0)$, then the acceleration may improve convergence. Suppose $u_0 = (1, \ldots, 1)$. In this case α again turns out to be very small for the same reason: the denominator $\langle A^{tr} r_0, A^{tr} r_0 \rangle$ is very large, which leads to α very small (on the order of 10^{-18}).

Using the acceleration formula for choosing ϕ_0 does not slow down the convergence of the algorithm for two different choices of u_0. There may be some unexplored scenarios in which choosing $\phi_0 = \alpha r_0$ does accelerate convergence. Since there aren't any losses associated with this choice but there are potential gains, choosing $\phi_0 = \alpha r_0$ is a better choice than choosing the generic $\phi_0 = (0, \ldots, 0)$.

6.0 STOPPING CRITERION

How do we know when the Bi-CG iterative method has converged? There are three possible options for stopping criterion in the algorithm:

- small residual
- small update
- too many iterations.

The "small residual" criterion signals convergence to the true solution because as the residual approaches zero the error does as well. That is, if the residual is smaller than some tolerance level, then we signal convergence (if $\|r_n\| \leq tol$ then stop and converges).

The "small update" criterion signals convergence (less reliably than small residual) to the true solution because as the updates gets smaller and smaller it indicates that the iteration is Cauchy. In other words, it is not adding much new information - thus provided the update is small enough the solution is unchanging within a pre-specified number of significant digits. Thus, if the update is smaller than some tolerance level, then we signal convergence (if $\|update\| \leq tol$ then stop and converges).

The "too many iterations" criterion signals divergence. This criterion is used in conjunction with either the "small residual" or "small update" criterion. If the method doesn't converge according to one of the above criterion within a pre-specified number of iterations, then we signal divergence. Thus, if the number of iterations reaches some maximum tolerance level, then we signal divergence (if $iter \geq MaxIts$ then stop and diverges).

The question then becomes, which criterion do we use? Which gives us a more accurate picture of what is happening to the true error in the problem - a small residual or a small update? We can use our test problem to provide an answer to this question because we know

the true error in our test problem. However, another question arises in this application: Do we use the residual and updates of the original problem, of the dual problem, or of the functional? Again, we can examine our test problem to see what happens in all of the different cases.

Section 6.1.1 tests the original residual as a stopping criterion and Section 6.1.2 tests the dual residual as a stopping criterion. Section 6.2.1 tests the update of the original problem as a stopping criteron and Section 6.2.2 tests the update of the functional as a stopping criterion.

6.1 RESIDUALS

The fundamental equation of numerical linear algebra tells us that there is a direct connection between the residual and the error:

$$Ae = r.$$

Thus we know that bounding r will in some way bound the error:

$$\|A\| \, \|e\| \leq \|r\| \, .$$

There are two problems with this stopping criterion. First, if the residual is not bounded, then we can't say anything about the error. Second, if the problem is ill-conditioned, then the residual converging to zero is not indicative of the error converging to zero. That is, the residual can be small, but the error can still be large. The condition number of the system gives an indication of whether controlling the residual will also control the error. The following theorem formalizes this relationship:

Theorem 12. *If $Ax = b$ and $r = b - A\hat{x}$, then*

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq cond(A) \frac{\|r\|}{\|b\|}.$$

Proof. Since $Ae = r$,

$e = A^{-1}r$.

We have that $\|e\| \leq \|A^{-1}\| \|r\|$.

Also, $b = Ax$.

Thus we have $\|b\| \leq \|A\| \|x\|$.

Combining these two inequalities gives $\frac{\|e\|}{\|A\|\|x\|} \leq \frac{\|A^{-1}\|\|r\|}{\|b\|}$.

Rearranging yields $\frac{\|e\|}{\|x\|} \leq \|A\| \|A^{-1}\| \frac{\|r\|}{\|b\|} = cond(A)\frac{\|r\|}{\|b\|}$. □

Thus if $cond(A)$ is large, the system is ill-conditioned and having a tight bound on the relative residual doesn't give a tight bound on the relative error. Example 13 calculates $cond(A)$ for our test problem.

Example 13. *For the discretized thermal convection-diffusion problem described in Equation (3.1) with* $N = 150, \mathbf{b} = [1, 0]$, *and* $\varepsilon = 1$

$cond(A) = \|A\| \|A^{-1}\| = 1.3376e(+004)$.

This condition number was calculated using the MATLAB function "cond(A)", which returns the 2-norm condition number.

This system is ill-conditioned, thus theory does not give us any reason to believe that bounding the residual will bound the true error. How to bound the error then becomes an empirical question.

6.1.1 Residual of the Original Problem

For our test problem, we can directly examine this relationship because we know the true solution and we therefore know the true error. What happens if we use the residual of the original problem to signal convergence? Figure 10 illustrates the relationship between the original residual and the true errors of both the original problem and the functional for the Average Temperature functional. Figure 11 illustrates the same relationship for the Heat Flux functional. The stopping criterion used here is

$$original\ residual = \frac{\|r\|}{\|r_0\|} \leq tol$$

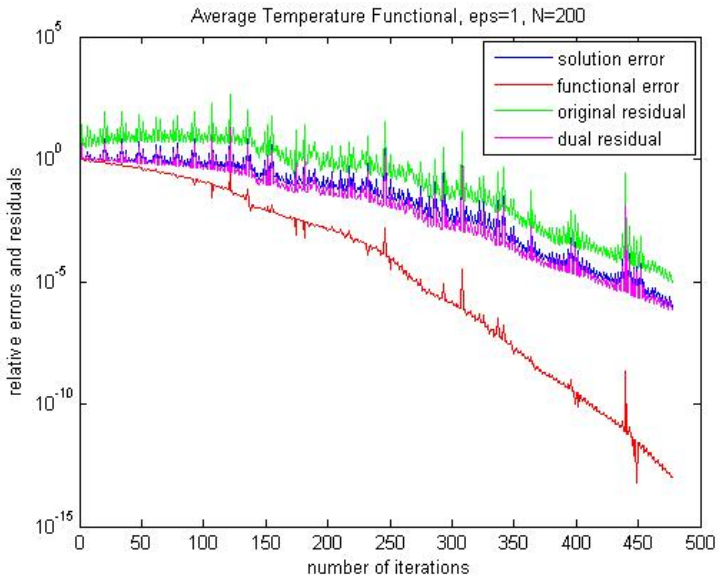

Figure 10: Average Temperature Functional, $eps = 1, N = 200$

where $tol = 1 \times 10^{-5}$ and $\|r\|$ is the residual of the original problem for the most recent iterate. This stopping criterion requires that the relative residual be equal to zero with five significant digits of accuracy.

For all three functionals, the relative residual of the original problem is a sufficient bound for controlling the relative error of the original problem, even though $cond(A) \gg 1$. Even though the system is ill-conditioned, the relative residual of the original problem provides a sufficient bound for both the true error in the original problem and the true error in the functional. As pointed out in Section 5, the true error in the functional converges much faster than does the true error in the original problem. This results in the original residual being a loose bound for the true error in the fuctional ($error_{functional} \simeq 10^{-12}$ when $original\ residual \simeq 10^{-5}$).

The residual continues to be a sufficient bound for both errors even when the system becomes less symmetric (that is, when ε decreases). Figures 13 and 14 show what happens with the average temperature functional when the diffusion coefficient, ε, gets smaller. The same patterns hold for the heat flux and checkerboard functionals. The size of the system is also smaller in these examples because the algorithm fails for large systems as ε shrinks.

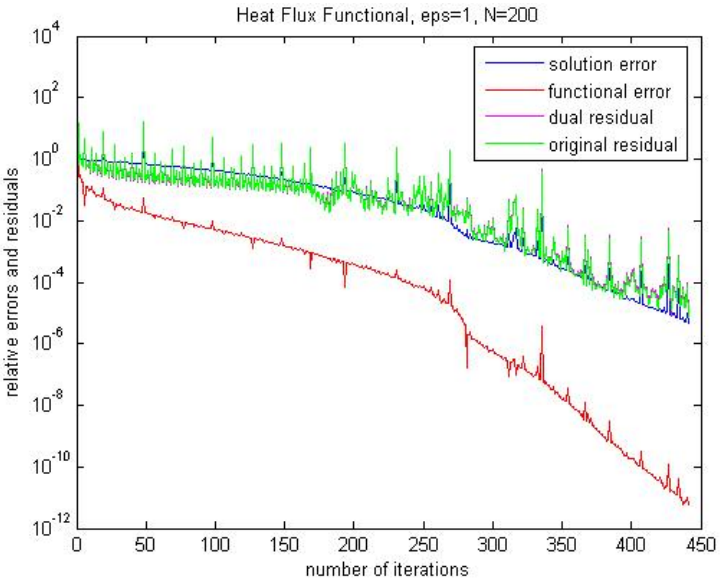

Figure 11: Heat Flux Functional, $eps = 1, N = 200$

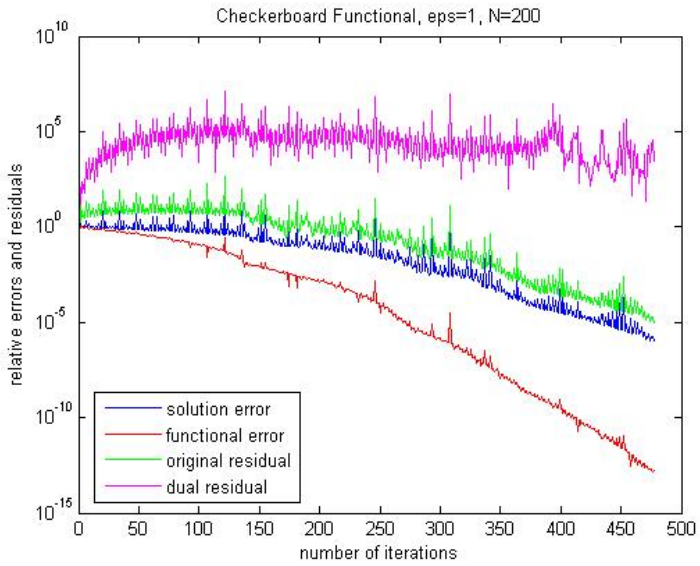

Figure 12: Checkerboard Functional, $eps = 1, N = 200$

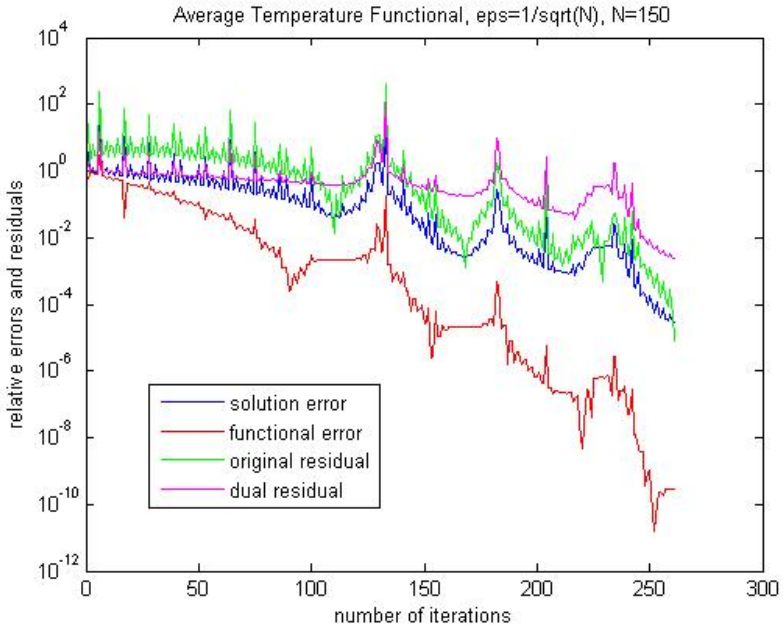

Figure 13: Average Temperature Functional, eps=1/sqrtN, $N = 150$

The residual of the original problem, $\frac{\|r\|}{\|r_0\|}$, does bound the true error of both the original problem and the functional. The residual is a much tighter bound for the error in the original problem than it is for the functional error. However, the residual of the original problem never gives a false signal for convergence or for divergence and is therefore a reliable stopping criterion. In fact, the patterns suggest that no matter what tolerance is chosen for the residual, the true error in the functional will be yet smaller than that tolerance.

6.1.2 Residual of the Dual Problem

The other residual stopping criterion we have available in this problem is the residual of the dual problem, $\frac{\|s\|}{\|s_0\|}$. What is the behavior of the residual of the dual problem, and does it bound the error in some constructive way? Figure 15, Figure 16, and Figure 17 show how the residual of the dual problem behaves in relation to the true error of the original problem and the true error of the functional.

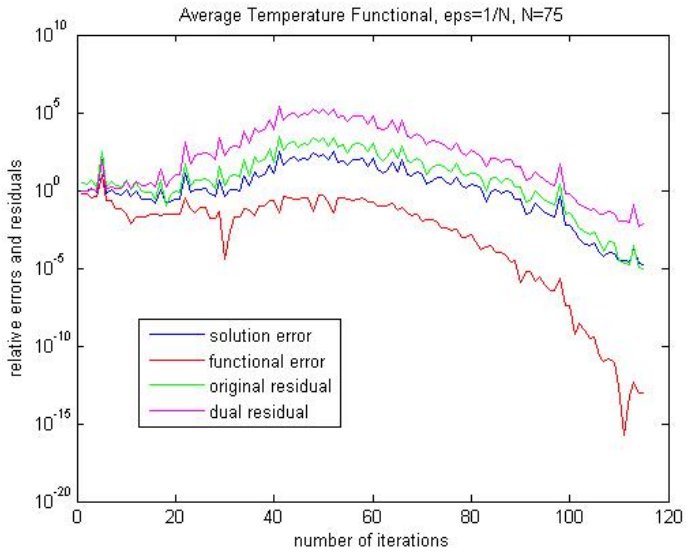

Figure 14: Average Temperature Functional, $eps = 1/N$, $N = 75$

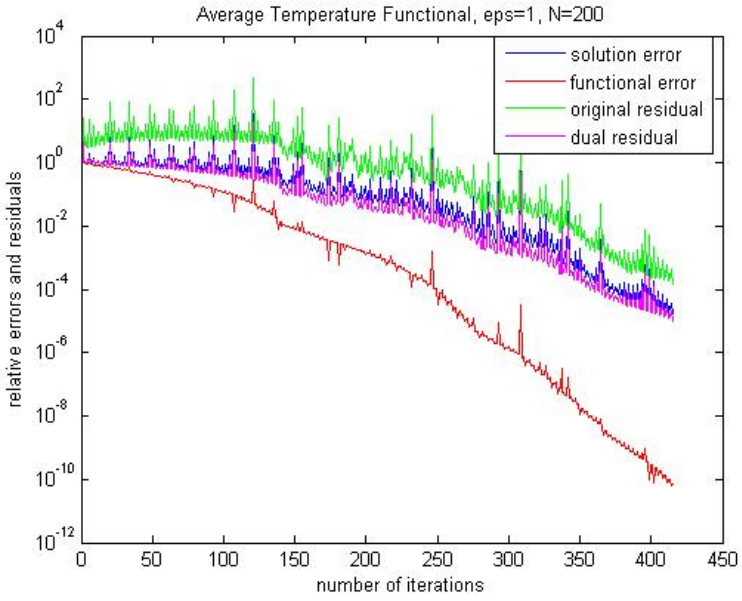

Figure 15: Average Temperature, $eps = 1$, $N = 200$

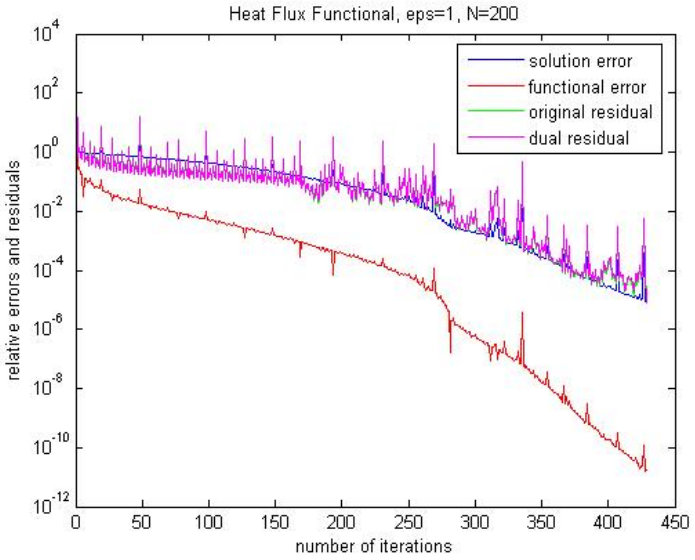

Figure 16: Heat Flux Functional, $eps = 1$, $N = 200$

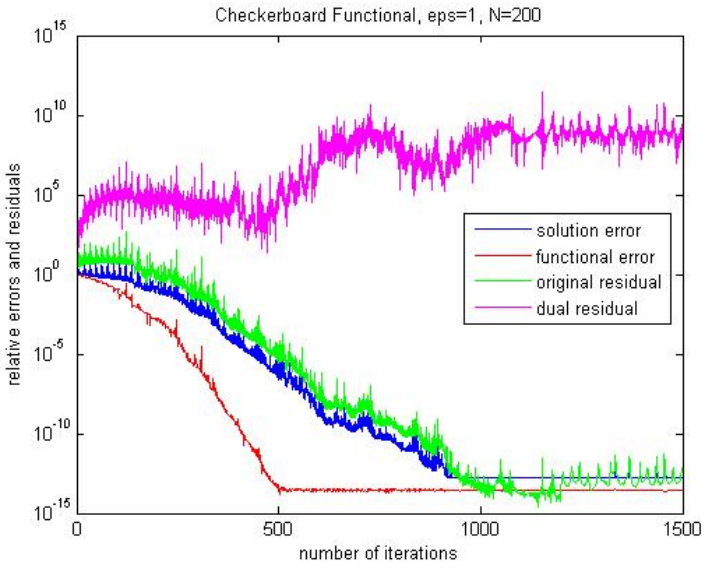

Figure 17: Checkerboard Functional, $eps = 1$, $N = 200$

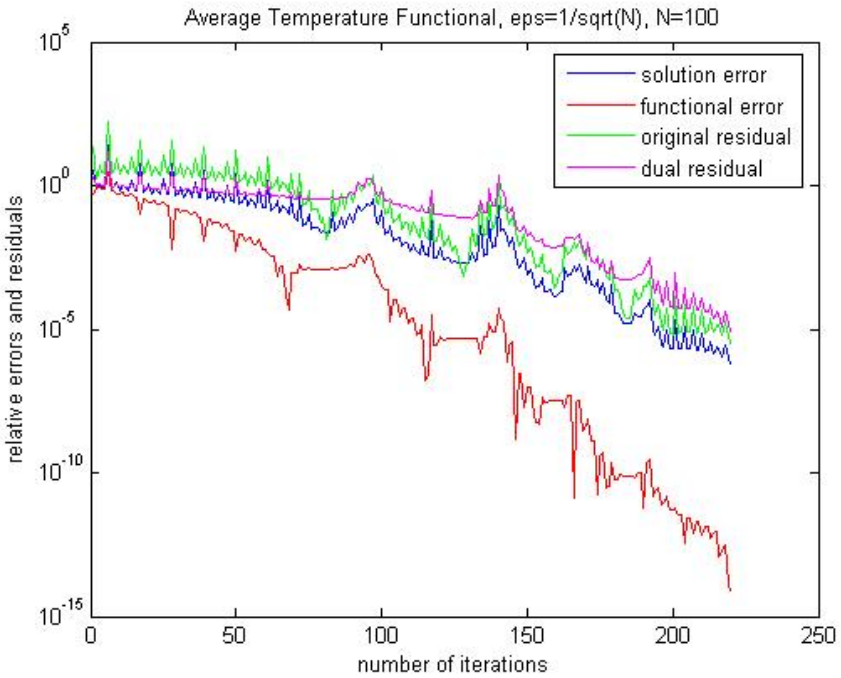

Figure 18: Average Temperature Functional, $eps = 1/sqrtN$, $N = 100$

When $\varepsilon = 1$ the dual residual is a good bound for the average temperature and heat flux functionals. The dual residual signals divergence for the checkerboard functional even though the true errors and original residual all converge to zero. When ε decreases, the dual residual begins to falsely signal divergence for all three functionals. Figures 18 and 19 portray the average temperature functional with $\varepsilon = \frac{1}{\sqrt{N}}$ but with two different choices for N: $N = 150$ and $N = 100$. In the case where $N = 100$ the dual residual correctly signals convergence, but in the case where $N = 150$ the dual residual falsely signals divergence.

The dual residual is a good bound for the true errors of the average temperature and heat flux functionals when ε is large. In such cases the dual residual is very close to the original residual. As ε shrinks the dual residual begins to diverge in certain cases (such as for large N but not for small N), even though the true errors are converging to zero. The dual residual certainly has no advantage over the original residual, and in fact is a much less reliable stopping criterion than the original residual.

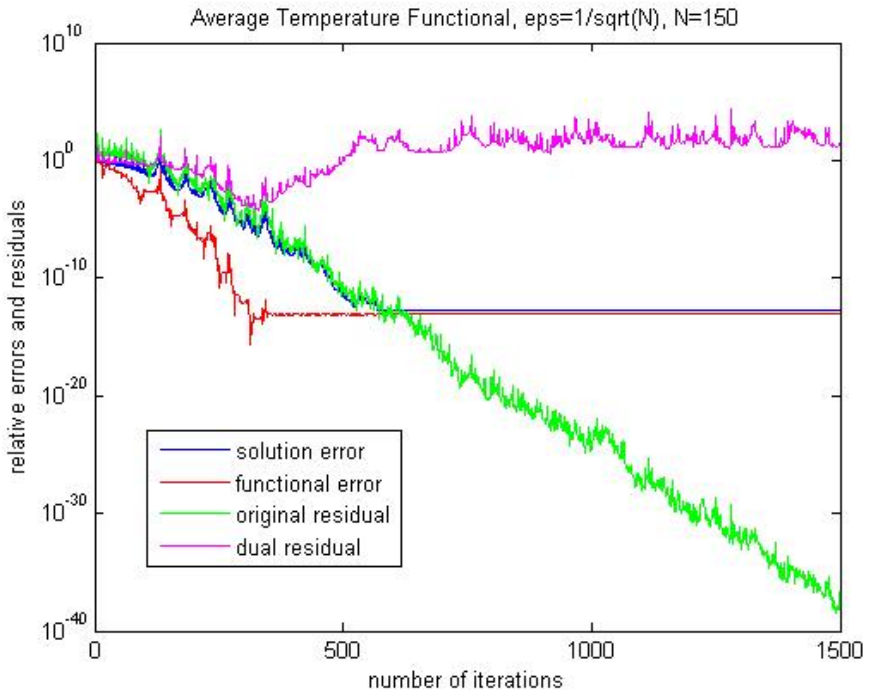

Figure 19: Average Temperature Functional, $eps = 1/sqrtN$, $N = 150$

6.2 SMALL UPDATES

The other set of stopping criteria are small updates. If the updates of either the original problem or of the functional are small - that is, are bounded above by some tolerance - then we can ensure that the solutions to the original problem and to the functional are unchanging for some pre-specified number of significant digits. For example, if the functional update is bounded by $tol = 1 \times 10^{-5}$, then we can be sure that the first four significant digits of the functional solution are unchanging and the iterative method isn't providing any new information for those digits. The "small update" stopping criterion will fail if the method gets stuck on a wrong answer for more than one iteration. That is, if the method provides an answer where the error is large, but the method doesn't provide a large update in the next iteration. We can again look at this question empirically. How does the error behave relative to the updates?

6.2.1 Updates to the Original Solution

The original solution is updated every iteration by amount

$$\frac{\langle \tilde{r}^k, r^k \rangle}{\langle \tilde{d}^k, Ad^k \rangle} d^k.$$

The small update stopping criterion will signal convergence if

$$\left\| \frac{\langle \tilde{r}^k, r^k \rangle}{\langle \tilde{d}^k, Ad^k \rangle} d^k \right\| \leq tol.$$

Consider the test problem when $\varepsilon = 1$ and $N = 200$. Figures 20, 21, and 22 show the behavior of the original solution update and the true errors of the original problem and the functional for the three functionals.

The update of the original problem follows the same basic trend as the true error of the original problem. The update is much more volatile than the error, which means that a large downward spike causes the update to be small enough to signal convergence. In all three cases the update signaled convergence before the error of the original problem reached

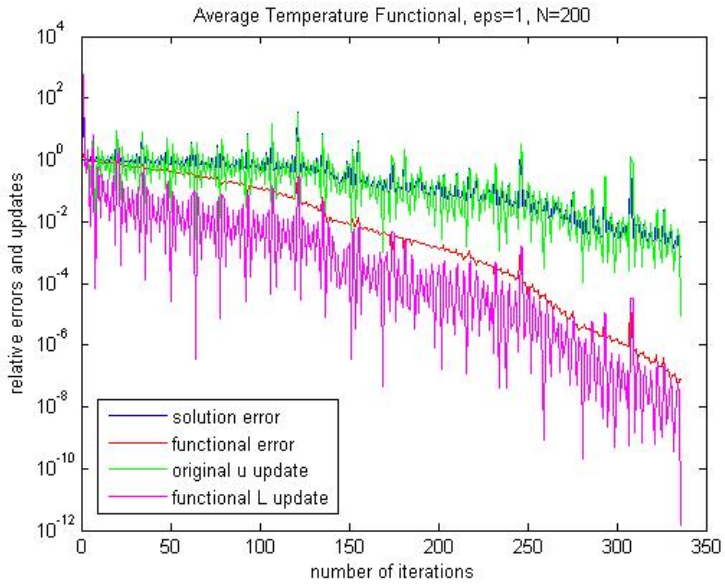

Figure 20: Average Temperature Functional, $eps = 1$, $N = 200$

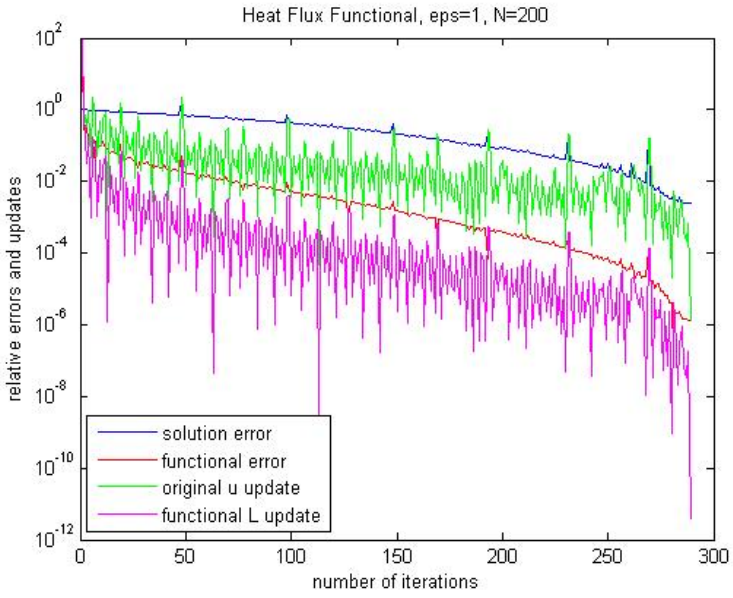

Figure 21: Heat Flux Functional, $eps = 1$, $N = 200$

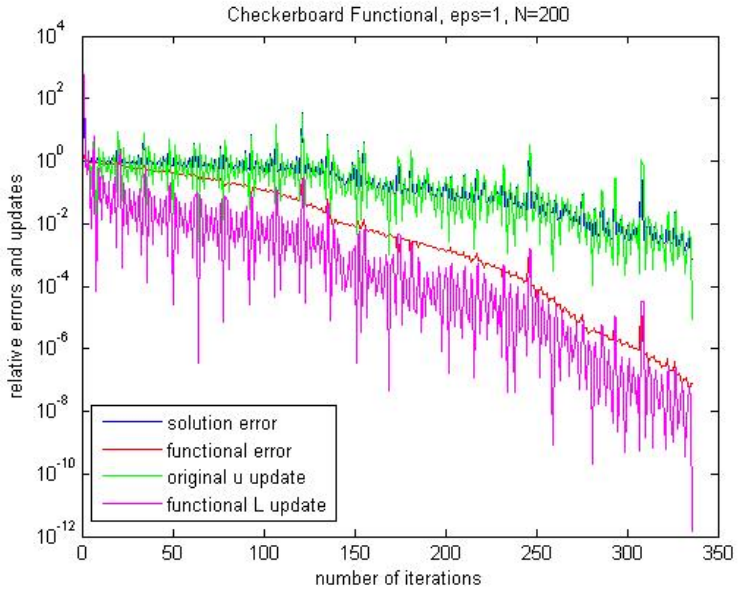

Figure 22: Checkerboard Functional, $eps = 1$, $N = 200$

the tolerance level. However, the error of the functional converges much more rapidly than the error of the original problem does and is below the tolerance before the update signals convergence. The update of the original problem is a good bound for the true error of the functional in this case where $\varepsilon = 1$.

Chapter 5 showed that changing ε results in different convergence rates. How effective is the update of the original problem as a stopping criterion for very small ε? Figures 23 and 24 show the relationship between the original update and the true errors for the Average Temperature Functional when ε shrinks. Notice that the size of the system also changes when ε gets smaller because the algorithm fails for large systems when A is highly non-symmetric, as described by the cell Péclet number.

It is again the case that the original update follows the trend of the original error, but doesn't quite bound it because convergence is signaled when the update suddenly spikes down below the tolerance. Yet the functional error converges fast enough that it is well below the tolerance by the time the update signals convergence. This pattern holds true for the Heat Flux and Checkerboard functionals as well. Thus even with a small ε the update

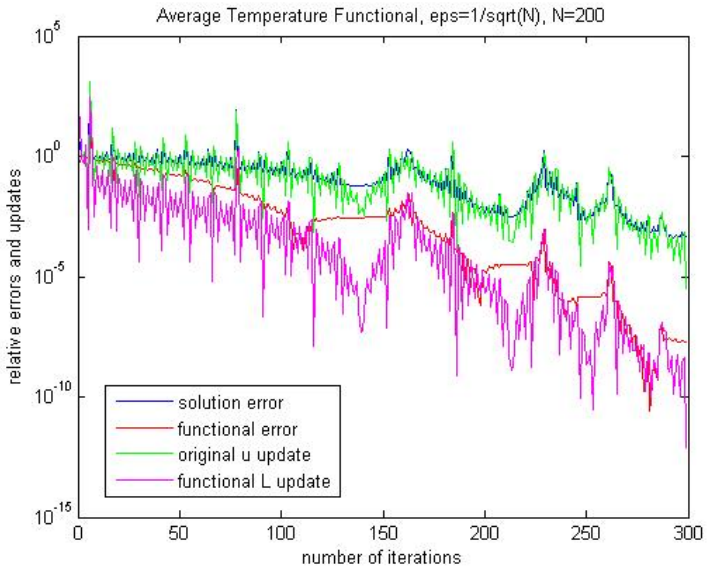

Figure 23: Average Temperature Functional, $eps = 1/sqrtN$, $N = 200$

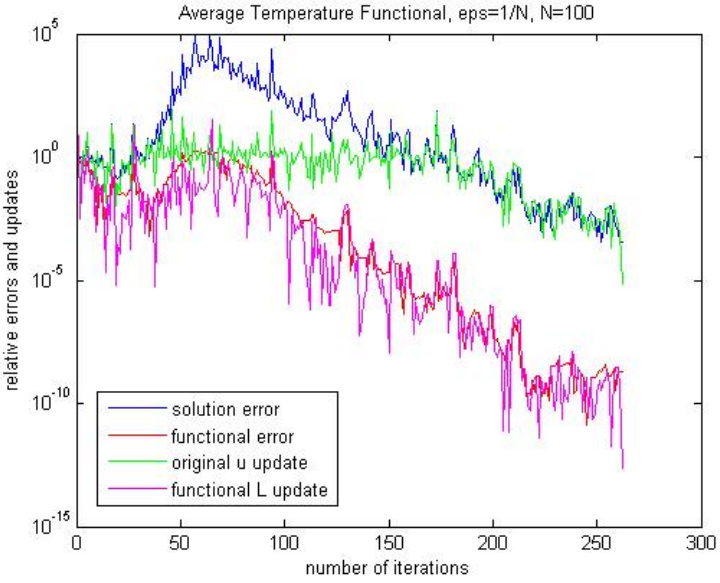

Figure 24: Average Temperature Functional, $eps = 1/N$, $N = 100$

of the original problem is a reasonable bound for the true error of the functional. If one also desires to bound the error of the original problem, a very tight tolerance on the update must be specified.

6.2.2 Updates to the Functional

The functional update is also available as a stopping criterion. In iteration $k+1$, the functional solution is updated by amount

$$\langle \phi^k, r_{k+1} \rangle.$$

The small update stopping criterion will signal convergence if

$$\left| \langle \phi^k, r_{k+1} \rangle \right| \leq tol.$$

Again consider the test problem with $\varepsilon = 1$ and $N = 200$ for all three functionals. Is the functional update a sufficient bound for the errors? Figures 25, 26, and 27 show the behavior of the functional solution update and the true relative errors of the original problem and the functional when $tol = 1 \times 10^{-5}$.

.

Clearly the functional update does not bound either error. This is again an artifact of the volatility of the update. We saw in Figures 20, 21, 22, 23, and 24 that both updates follow the same trends as the errors, but that the updates are significantly more volatile than the error. This has a much more dramatic efffect in the case where the functional update is used as the stopping criterion. In each instance a large downward spike causes the update to signal convergence long before either error is below the desired tolerance level.

How does the functional update behave as ε shrinks? As one might expect, the functional update continues to signal convergence long before either error is below the desired tolerance. Figures 28 and 29 show what happens with the Average Temperature Functional for decreasing ε.

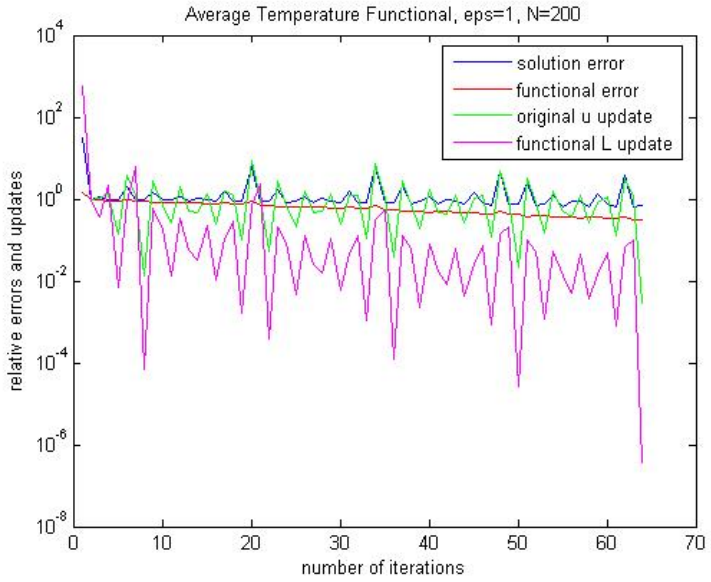

Figure 25: Average Temperature Functional, $eps = 1$, $N = 200$

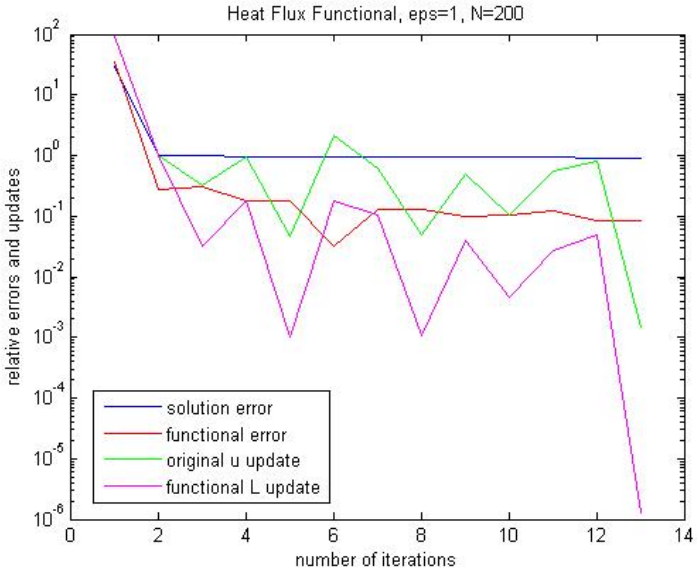

Figure 26: Heat Flux Functional, $eps = 1$, $N = 200$

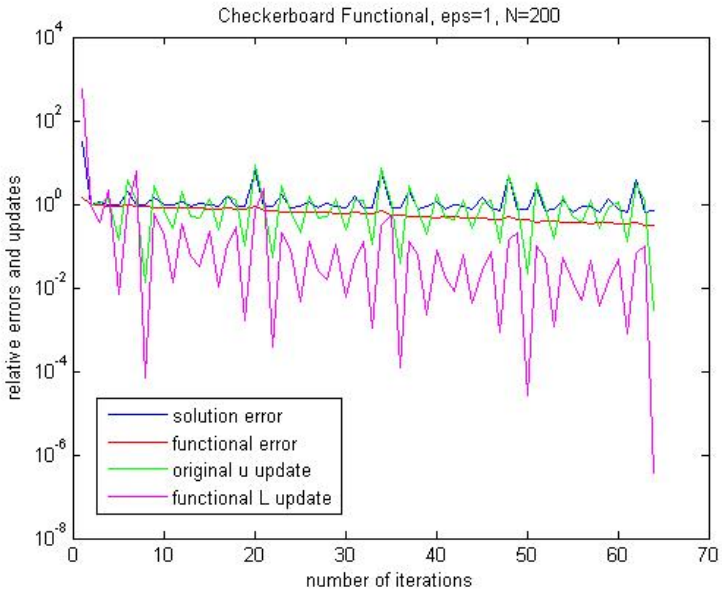

Figure 27: Checkerboard Functional, $eps = 1$, $N = 200$

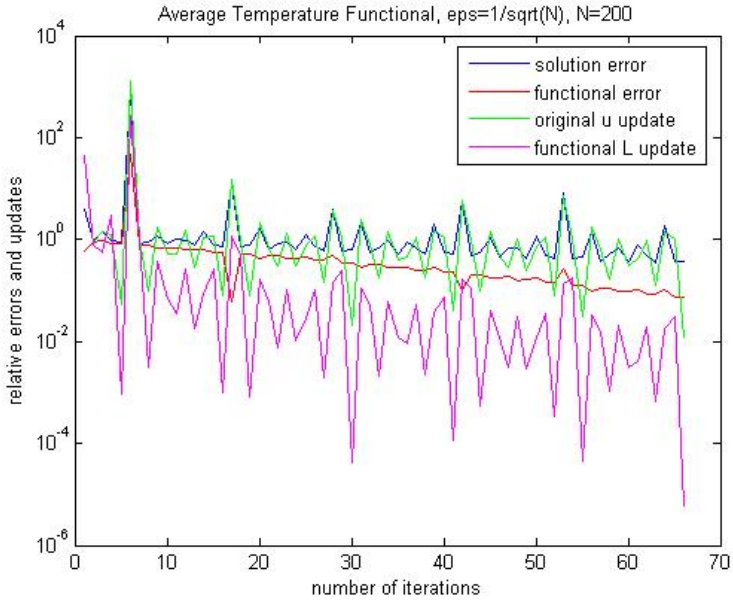

Figure 28: Average Temperature Functional, $eps = 1/sqrtN$, $N = 200$

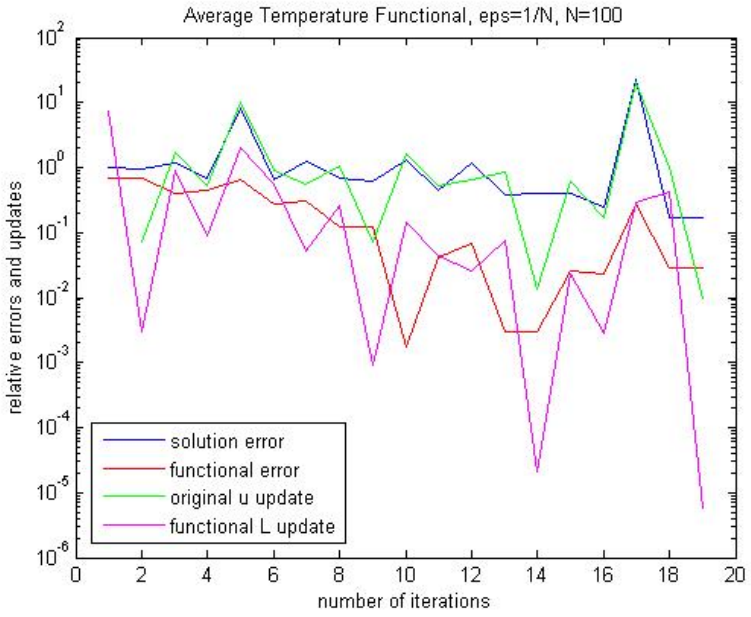

Figure 29: Average Temperature Functional, $eps = 1/N$, $N = 100$

This pattern holds for the Heat Flux and Checkerboard functionals as well. Clearly the functional update is not a tight enough bound on the error to use it as a stopping criterion. What is more, it is not simply a case of requiring a very tight tolerance for the functional update to work. There are some scenarios in which the functional update signals convergence while the errors are actually diverging! It appears to be the case that the functional update falsely signals convergence for cases where ε is one of the smallest diffusion-coefficients that we tested (for example, $\varepsilon = \frac{1}{N}$) and N is one of the largest systems that we tested (for example, $N = 200$, which yields a $40,000 \times 40,000$ system). Figures 30 and 31 represent two such scenarios with $N = 200$ and $\varepsilon = \frac{1}{N} = \frac{1}{200}$.

The functional update signals convergence long before the errors converge to the tolerance and sometimes signals convergence when the errors actually diverge. The functional update is an unreliable stopping criterion.

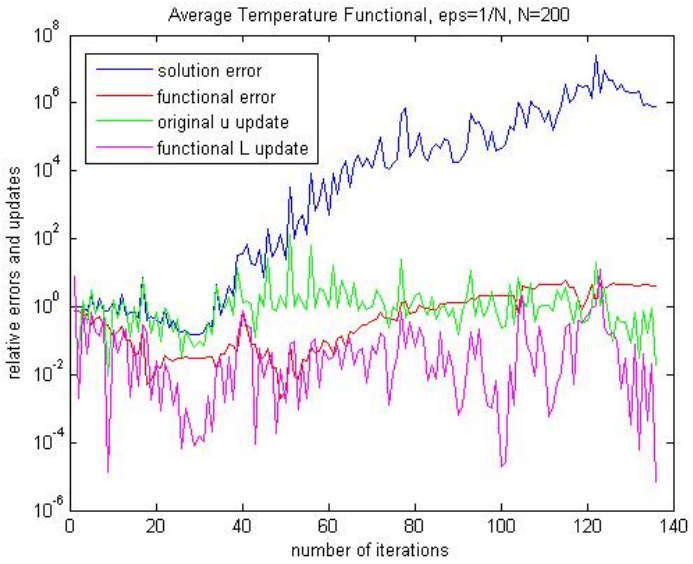

Figure 30: Average Temperature Functional, $eps = 1/N$, $N = 200$

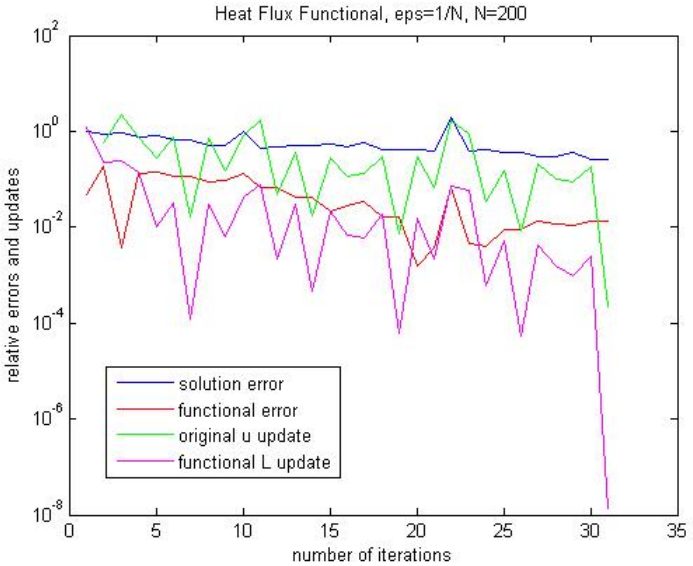

Figure 31: Heat Flux Functional, $eps = 1/N$, $N = 200$

6.3 STOPPING CRITERION SUMMARY

A small residual is the most reliable stopping criterion. The residual of the original problem is much more reliable than the residual of the functional. In fact, the original residual always signals convergence well after the functional has converged to the desired tolerance. A small update may also be utilized as a stopping criterion. Updates of both the original problem and the functional follow the same basic trends as the errors, but they are much more volatile. One consequence of the volatility of the updates is that the tolerance is reached and convergence is signaled when the update has a downward spike. Convergence is signaled before the trend reaches the desired tolerance, thus convergence is often signaled before the error (which has a smoother trend than the updates) reaches the tolerance. This problem is true of both the update of the original problem and of the functional, but it is especially poignant with the functional update.

Residuals and updates of the original problem are more reliable stopping criteria than residuals and updates of the functional. There are cases in which stopping criteria associated with the functional falsely signal either convergence or divergence. Furthermore, a small residual is a more reliable signal of convergence than a small update. This points to the residual of the original problem as being the best possible stopping criterion.

7.0 CONCLUSION

This paper explored the problem of determining methods that result in the functional's solution converging ($l_n \to L(u)$) more rapidly than the original solution ($u_n \to u$). A thermal convection-diffusion test problem was used to explore the convergence of three different linear functionals: the average temperature, the heat flux, and a checkerboard transformation. A modified Bi-CG algorithm was developed as a solution method for the coupled problem

$$Au = f \text{ and } A^{tr}\phi = l.$$

The Bi-CG method sucessfully converged to the functional's value, $L(u)$, much more rapidly than it converged to the original solution, u, for three different linear functionals.

Every case in which the Bi-CG method converged resulted in each of the three functionals converging more rapidly than the original solution. The method began failing for large systems ("large" in the context of this test problem - systems of size $10,000 \times 10,000$ and larger began failing first) as the coefficient matrix A became more non-symmetric (as described by the cell Péclet number) through a decrease in the diffusion coefficient, ε.

Further avenues for improving the modified Bi-CG algorithm may yet exist. One unsuccesful attempt at improving the initial guess of ϕ_0 was explored. However, more informed initial guesses for the original problem, u_0, and the dual problem, ϕ_0, can potentially accelerate the convergence of the algorithm. Another avenue of research is an exploration of how the diffusion coefficient, ε, is related to the failure of the algorithm. The data generated using the thermal convection-diffusion test problem indicate that a smaller ε leads to more rapid failure of the algorithm, but no theoretical connection was examined.

BIBLIOGRAPHY

[1] Layton, William, “Calculating Functionals of Solutions of Large, Sparse Systems”.

[2] Butler, Trisha R., “Calculating Functionals of Solutions of Large, Sparse Systems”, 2006.

[3] Reynolds, Osborne. "An experimental investigation of the circumstances which determine whether the motion of water shall be direct or sinuous, and of the law of resistance in parallel channels". Philosophical Transactions of the Royal Society 174: 935-982, 1883.

[4] Barrett, R., et. al. “Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods”, SIAM, 1994.

[5] Atkinson, Kendall E., “An Introduction to Numerical Analysis”, Wiley & Sons, 1989.

APPENDIX

MATLAB CODE

```
function A=SparseAmatrix1(N,epsilon,b)
    h=1/(N+1);
    A=sparse(N^2,N^2);
    for m=1:N^2
    A(m,m)=4*epsilon/(h^2);
    if m-1>0&(mod(m-1,N)~=0)
    A(m-1,m)=(-epsilon/(h^2))+(b(2)/(2*h));
    end
    if (m+1<=N^2)&(mod(m+1,N)~=1)
    A(m+1,m)=(-epsilon/(h^2))-(b(2)/(2*h));
    end
    if m-N>0
    A(m-N,m)=(-epsilon/(h^2))+(b(1)/(2*h));
    end
    if m+(N)<=N^2
    A(m+N,m)=(-epsilon/(h^2))-(b(1)/(2*h));
    end
    end
```

```
function f=RHS1(N,epsilon,b)
h = 1/(N+1);
xx=linspace(0,1,N+2);
x=xx(2:N+1);
yy=linspace(0,1,N+2);
y=yy(2:N+1);
% Set the boundary conditions:
TBottom = x; % x = 0
TLeft = zeros(size(y)); % y = 0
TTop = x; % x = 1
TRight = ones(size(y)); % y = 1
% Initialize the RHS vector:
f=zeros(N^2,1);
%left boundary
j=1;
for i=1:N
k=(j-1)*N+i;
f(k)=f(k)+ ( (b(1)/(2*h)) + (epsilon/(h^2)) )*TLeft(i);
end
% bottom boundary
i=1;
for j=1:N
k=(j-1)*N+i;
f(k)=f(k)+( (b(2)/(2*h)) + epsilon/(h^2)) *TBottom(j);
end
% top boundary
i=N;
for j=1:N
k=(j-1)*N+i;
f(k)= f(k)+( epsilon/(h^2) - (b(2)/(2*h)) ) *TTop(j);
```

```
end
% right boundary
j=N;
for i=1:N
k=(j-1)*N+i;
f(k)=f(k)+ ( epsilon/(h^2) - (b(1)/(2*h)) ) *TRight(i);
end
```

```
function l=flux(N)
% The function l=flux(N) forms the vector l, or the average temperature and
% is used when calculating the heat flux.
% Kristin Harnett
% October 30, 2007
for i=1:N
for j=1:N
k=N*(i-1)+j;
if k<=N^2-N
l(k)=0;
else
l(k)=-1;
end
end
end
l=l.';
```

```
function l=checkers2(N)

% Kristin Harnett

% January 23, 2007

% The function l=checkers(N) forms the vector l, or the linear functional

% that assigns every even/odd and odd/even coordinate pair a value of 1 and

% every even/even and odd/odd coordinate pair a value of -1.

for i=1:N

for j=1:N

k=i+j;

m=N*(i-1)+j;

if mod(k,2)==0

l(m)=2;

else

l(m)=0;

end

end

end

l=l.';
```

```
function [u_new,uTrue,L_new,resid1,resid2,resid3,resid4,error1,error2,error3,u_update,L_update,iter
% Kristin Harnett
% February 10, 2008
% The function bicg(u,N,MaxIt,epsilon,b,tol) uses a modified bi-cg
% algorithm to calculate the solution to the original problem and the dual
% problem (u and L, respectively). The code is based on the bi-cg pseudo-
% code presented in the book "Templates for the Solution of Linear Systems:
% Building Blocks for Iterative Methods" by Barrett, et al.
% Inputs:
% u: initial guess for solution to original problem
% N: size of problem
% MaxIt: maximum number of iterations before timing out
% epsilon: diffusion coefficient
% b: convection field
% tol: error tolerance. The relative residual must be at least as small as
% tol before convergence is signaled.
% Outputs:
% u: estimated solution to original problem
% error: relative residual error norm(r)/norm(uTrue)
% error1: relative true error of original problem norm(u-uTrue)/norm(uTrue)
% error2: relative true error of dual problem norm(L-LTrue)/norm(LTrue)
% iter: number of iterations before convergence
t=cputime;
iter=1;
A=SparseAmatrix1(N,epsilon,b);
f=RHS1(N,epsilon,b);
l = (1/N^2)*ones(N^2,1); %average temperature functional
%l = flux(N); %heat flux functional
%l = checkers2(N); %checker-board functional
uTrue = A\f;
```

```
phiTrue = A.'\l;
LTrue = dot(l,uTrue); %true solution of linear functional
rho=zeros(2,1); %initialize rho (need to keep time t and t-1
%versions of rho: rho(2) is old)
r0 = f - A*u_old; %original residual - used for relative residuals
r_old = r0; %residual of original problem - used in algorithm
%r_tilde = r_old; Changed Per Mike Sussman
% Initial guess for phi: use Jacobi best guess (or use guess of zero)
a1 = dot(A.'*r0,l);
a2 = dot(A.'*r0,A.'*r0);
a=a1/a2;
phi = a*r0;
%phi = zeros(N^2,1);
% Calculate dual residual (for use in algorithm) and functional value:
s0 = l - A.'*phi; %original residual - used in algorithm
L_old = dot(l,u_old) + dot(phi,r_old);
r_tilde = s0; %added per Mike Sussman
for i=1:MaxIt
rho(2) = dot(r_old',r_tilde);
if (i==1)
d=r_old;
d_tilde=r_tilde;
else
beta = rho(2)/rho(1);
d = r_old + beta*d;
d_tilde = r_tilde + beta*d_tilde;
end
%Bi-CG algorithm:
q = A*d;
q_tilde = A.'*d_tilde;
```

```
denom = dot(d_tilde',q);
alpha = rho(2)/denom;
alpha2 = dot(d',s0)/denom;
u_new = u_old + alpha*d;
phi = phi + alpha2*d_tilde;
rho(1)=rho(2);
r_new = r_old - alpha*q;
r_tilde = r_tilde - alpha*q_tilde;
s = l - A.'*phi;
L_new = dot(l,u_new) + dot(phi,r_new);
%Relative residuals
resid1(iter)=norm(r_new)/norm(r0);
resid2(iter)=norm(s)/norm(s0);
%Relative residuals - how to normalize?
resid3(iter)=norm(r_new)/norm(uTrue);
resid4(iter)=norm(s)/norm(LTrue);
%Relative true errors
error1(iter) = norm(uTrue-u_new)/norm(uTrue);
error2(iter) = norm(LTrue-L_new)/norm(LTrue);
error3(iter) = norm(phiTrue-phi)/norm(phiTrue);
%Relative updates
u_update(iter)=norm(u_new-u_old)/norm(u_old);
L_update(iter)=norm(L_new-L_old)/norm(L_old);
%Update residual and functional estimate.
r_old=r_new;
L_old=L_new;
u_old=u_new;
% if abs(resid1(iter))<=tol
% if abs(resid2(iter))<=tol
% if abs(resid3(iter))<=tol
```

```
% if abs(resid4(iter0)<=tol
if abs(error1(iter))<=tol
% if abs(error2(iter))<=tol
% if abs(error3(iter))<=tol
% if abs(u_update(iter))<=tol
% if abs(L_update(iter))<=tol
break;
else
if iter==MaxIt
disp('iterations maxed out');
break;
else
iter=iter+1;
end
end
end
cpu_time=cputime-t
```