AN ABSTRACT OF THE THESIS OF

Henry William Tieman for the degree of Master of Science

in Mathematics presented on April 25, 1991

Title: A Computer Subroutine for the Numerical Solution of Nonlinear Fredholm Equations



Abstract approved: Joel Davis

Computer subroutines are presented for the solution of nonlinear Fredholm integral equations. The solutions may be vector valued and an error bound is computed for the approximate solutions found. The computer subroutines written for this solution are described so that a reader may use them to solve specific problems.

Fredholm integral equations often come from partial differential equations. The integral equation,

$$x(s) - \lambda \int_a^b k(s,t)F(x(t))dt = y(s),$$

is a Fredholm equation. The desired solution is a function $x : [a,\ b] \rightarrow \Re^n$ that satisfies this equation.

# A Computer Subroutine for the Numerical Solution of Nonlinear Fredholm Equations

by

Henry William Tieman

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed April 25, 1991

Commencement June 1991

APPROVED:

Redacted for Privacy

Professor of Mathematics in charge of major

Redacted for Privacy

Chairman of Department of Mathematics

Redacted for Privacy

Dean of Graduate School

Date thesis is presented April 25, 1991

Typed by Henry William Tieman for Henry William Tieman

TABLE OF CONTENTS


| Chapter | Page |
|---|---|
| 1. Introduction | 1 |
| 2. Analysis Background | 3 |
| 3. Integral Operators | 7 |
| 4. Newton's Method In A Banach Space | 10 |
| 5. The Kantorovich Theorem | 13 |
| 6. Programming Considerations | 15 |
| 7. Description of Using Subroutine Package | 23 |
| 8. Example: Pendulum Problem | 34 |
| 9. Example: Stress/Strain in a Metal Cap | 37 |
| 10. Observations and Further Research | 40 |
| Bibliography | 41 |
| Appendix A. Subroutine Headings | 42 |
| Appendix B. Outline of Required Steps | 44 |
| Appendix C. Example Problem Code Listing | 47 |

# LIST OF FIGURES


| Figure | Page |
|---|---|
| 1. Graph of the solution for Pendulum position | 36 |
| 2. Stress as a function of pressure in a metal cap. | 38 |
| 3. Norm of the Inverse of $I - K_n F'(x)$. | 39 |

# A Computer Subroutine for the Numeric Solution of Nonlinear Fredholm Equations

## 1. Introduction

This paper describes a program written by the author to solve equations of the form

$$x(s) - \lambda \int_a^b k(s,t)F(x(t))dt = y(s),$$

with $s \in [a,b]$, for $x(s)$. This paper demonstrates a numerical solution technique for integral equations. It is the aim of this paper to provide mathematicians with some example solutions and subroutines so that a motivated reader might be able to attempt a numerical solution of an appropriate integral equation.

Integral equations arise naturally from physical systems. They are often equivalent to certain partial differential equations. The equation

$$x(s) - \lambda \int_a^b k(s,t)F(x(t))dt = y(s),$$

for $s \in [a,b]$ and $F$ nonlinear, is a nonlinear Fredholm equation of the second kind. Very few equations of this type have closed form solutions, therefore to solve these equations numerical methods are necessary. The method is straightforward but the background for understanding the steps is sometimes hard, so the paper contains background from analysis.

The chapters are arranged roughly in order from theoretical to applied, starting with Banach spaces and ending with the solution of different problems. Chapter 2 contains various definitions and terminology from analysis. This is included for completeness and clarity. Chapter 3 is on integral operators, much of the notation is simplified by considering integral equations in operator form, as opposed to the more classical notation. In chapter 4 Newton's method is presented in operator form. Newton's method as presented is suitable for use in finite Banach spaces. The theoretical error bound for Newton's method is presented in chapter 5. Chapter 6

describes the various approximations required for computer solution of the material found in chapters 4 and 5. Chaper 7 is a users guide for the Pascal subroutines. Chapters 8 and 9 are example problems. The final chapter, chapter 10 describes certain modifications that might be usefull in the future. There are two appendicies. Appendix A contains the subroutine headers for the provided Pascal subroutines. Appendix B is an outline of the required programming to solve an integral equation using the Pascal subroutines. Appendix C contains Pascal source code that will solve the example problem found in chapter 9.

## 2. Analysis Background

This chapter is a collection of definitions from functional analysis. These definitions are included in this paper to standardise the use of terminology, and attempt to start from a familiar mathematics base. Such concepts as Banach space, linear operators and Frechét derivative are described in this chapter.

A Banach space is defined as a complete normed linear space. This definition has three properties:

1.) It is a linear space, or all scalar multiples and finite sums of elements in the space are in the space.

2.) It has a defined norm or distance function.

3.) It is a complete space, that is, Cauchy sequences converge to elements in the space.

Examples of linear spaces:

1). $\Re^n$, for $n \geq 1$ is a linear space.

2). $\mathbf{C}[0,1]$, the set of continous functions $[0,1] \rightarrow \Re$ is a linear space.

A norm, $\| \circ \|$, defined on the space $S$ has three basic properties:

Given $a, b \in S$

1.) $\|a\| \geq 0$ ( and $\|a\| = 0$ iff $a = 0$ ).

2.) $\|\gamma a\| = \|\gamma\|\|a\|$

3.) $\|a + b\| \leq \|a\| + \|b\|$.

There are many such norms for each linear space:

1). If $a \in \Re^n$ then $\sum_{i=1}^{n} |a_i|$ is a norm.

2). If $a \in \Re^n$ then $\max_{i \in \{1,\ldots,n\}} |a_i|$ is a norm.

3). If $a \in \mathbf{C}[0..1]$ then $\int_0^1 |a(t)|dt$ is a norm.

4). If $a \in \mathbf{C}[0..1]$ then $max_{t \in [0,1]} |a(t)|$ is a norm.

The norms that will be used later in this paper are the max norms (2 and 4 above.)

A closed ball, $\mathbf{B}(x, r)$, is a subset of a space close to the point $x$ and defined by,

$$\mathbf{B}(x, r) = \{y \in \mathbf{X} : \|x - y\| \leq r\}$$

The set $\mathbf{C}$ is said to be convex if and only if, $x,\ y \in \mathbf{C} \implies x + \gamma(y - x) \in \mathbf{C}$, for $\gamma \in [0, 1]$.

A sequence, $\{s_i\} \subset \mathbf{C}$, is said to converge if $\forall\ \varepsilon > 0\ \exists\ N \in \mathbf{N}$ such that $j, > N$ implies that $\|s_j - S\| < \varepsilon$, for some $S$ in the space $\mathbf{C}$, where $\| \circ \|$ is the norm for the space.

A sequence, $\{s_i\} \in \mathbf{C}$, is said to be Cauchy if $\forall\ \varepsilon > 0\ \exists\ N \in \mathbf{N}$ such that $j,\ k > N$ implies that $\|s_j - s_k\| < \varepsilon$, where $\| \circ \|$ is the norm for the space.

A sequence of functions, $\{f_i\} \subset \mathbf{C}[0, 1]$, is said to converge uniformly if $\forall\ \varepsilon > 0\ \exists\ N \in \mathbf{N}$ such that $j > N$ implies that $|f_j(x) - f(x)| < \varepsilon$, for some $f$ in C[0,1], and all $x \in \mathbf{C}[0, 1]$.

A sequence of functions, $\{f_i\} \subset \mathbf{C}[0, 1]$, is said to converge pointwise if $\forall\ x \in \mathbf{C}[0, 1]$, $f_i(x)$ converges to $f(x)$.

Suppose there exists an operator $L : \mathbf{X} \to \mathbf{Y}$ both linear spaces of the same scalar field, $\mathbf{F}$, such that each $x \in \mathbf{X}$ maps to a unique $y \in \mathbf{Y}$.

1.) $L$ is called additive if $L(x_1 + x_2) = L(x_1) + L(x_2)$, $\forall\ x_1, x_2 \in \mathbf{X}$.

2.) $L$ is called homogeneous if $L(\lambda x) = \lambda L(x)$. $\forall\ x \in \mathbf{X}$ and $\lambda \in \mathbf{F}$.

If $L$ is both additive and homogeneous then $L$ is called linear. It is usual to write the composition of operators as multiplication and therefore multiplication is non-communtative, since in general composition is non-commutative.

There is notation associated with operators most of it is common to the notation of linear algebra. For linear operators it is customary to not have parenthesis around the operand (as in $Lx$ and not $L(x)$). Parenthesis are used with linear operators when there is a question of operand order. The symbol $I$ will refer to the identity operator, that function that returns its operand.

The norm of a linear operator is defined as,

$$\begin{aligned} \|L\| &= \sup_{\|x\| \leq 1} \|Lx\| \\ &= \sup_{\|x\| = 1} \|Lx\|. \end{aligned}$$

for $x$ in the domain of $L$.

Definition of the Frechét Derivative. Suppose $f$ is an operator, $f : \mathbf{X} \to \mathbf{Y}$, $\mathbf{X}$, $\mathbf{Y}$ Banach spaces. If

$$\lim_{\|h\| \to 0} \frac{\|f(x+h) - f(x) - Lh\|}{\|h\|} = 0,$$

for $h \in \mathbf{X}$ and $L$ a bounded linear operator then $L$ is called the derivative of $f$ at $x$, usually written as $f'(x) = L$. All derivatives for the rest of this paper are going to be assumed to be Frechét derivatives, denoted with a prime symbol. Frechét derivatives have some expected properties:

1.) If $L$ is a linear operator then $L'(x_0) = L$, for any $x_0 \in X$.

2.) If $R, S$ are operators $\mathbf{X} \to \mathbf{Y}$ then $(R+S)'(x_0) = R'(x_0) + S'(x_0)$.

3.) If $RS(x) = R(S(x))$ then $(RS)'(x_0) = R'(Sx_0)S'(x_0)$.

The second derivative is defined as the derivative of the first derivative. Suppose $F : \mathbf{X} \to \mathbf{Y}$, an operator from $\mathbf{X}$ to $\mathbf{Y}$, both Banach spaces, is differentiable at $x_0$ and also in $\mathbf{B}(x_0, r), r > 0$. For each $x \in \mathbf{B}(x_0, r), F'(x_0)$ will be an element of the space $L(\mathbf{X}, \mathbf{Y})$ of bounded linear operators, $\mathbf{X} \to \mathbf{Y}$. Therefore, we can consider $F'$ to be $F' : \mathbf{B}(x_0, r) \to L(\mathbf{X}, \mathbf{Y})$. Then we can take the Frechét derivative of $F'$. If there exists a bounded linear operator $B : \mathbf{X} \to L(\mathbf{X}, \mathbf{Y})$ such that,

$$\lim_{\|\Delta x\| \to 0} \frac{\|F'(x_0 + \Delta x) - F'(x_0) - B\Delta x\|}{\|\Delta x\|} = 0,$$

for $\Delta x \in \mathbf{X}$ then we say $F'$ is differentiable at $x_0$ and $F''(x_0) = B$.

If $F''$ exists, for $F : \mathbf{X} \to \mathbf{Y}$ at $x = x_0$ then $F''$ is a symmetric bilinear operator from $\mathbf{X} \to \mathbf{Y}$. In general for $n \geq 2$, $F^{(n)}$, the $n^{th}$ derivative of $F$, is a symmetric multilinear operator.

A function, f, has a Lipschitz constant $b$ on a set $\mathbf{S}$ if

$$\|f(x) - f(y)\| \leq b\|x - y\| \; \forall \; x, y \in \mathbf{S}$$

If $\|f'(x)\| \leq b$ for all $x \in \mathbf{B}(x_0, r)$, then $f$ has a Lipschitz constant $b$. A differentiable function has a Lipschitz constant when its derivative is bounded in a convex region of space.

There is also a Lipschitz constant associated with first derivatives. We are interested in examining the derivative of a function in a closed region of space, $\mathbf{B}(x_0, r), r > 0$. Assume $g(x) = f'(x)$ then

$$\begin{aligned} \|f'(x) - f'(y)\| &= \|g(x) - g(y)\| \\ &\leq k\|x - y\|, \end{aligned}$$

for some constant $k$. Often $k$ is chosen to be a bound on the derivative of $g(x) = f'(x)$ and so is a bound on $f''(x)$.

## 3. Integral Operators

This chapter contains a fundamental introduction to integral operators. A basic understanding of integral operators will aid in the understanding of some of the following chapters.

Integral operators have domain and range in the set of continous functions. These operators can be viewed as elements of the space,

$$\mathbf{S} = \{ functions\ f\ s.t.\ f : \mathbf{C}[a,b] \rightarrow \mathbf{C}[a,b] \},$$

where $\mathbf{C}[a,b]$ means "the continous functions with domain [a,b] and range in $\Re^n$". This space has many different norms available but the often used norm in computer algorithms and will be used in this paper is the infinity norm, sup or max norm(all represent the same operation). If addition is in the usual sense($f(x) + g(x) = (f+g)(x)$) then it forms a linear space. An integral operator, $K$, is defined as

$$Kx = K(x)(s) = \int_a^b k(s,t)x(t)dt,$$

for some kernel function, $k$, and function, x(s).

The integral operators of interest here are contained in equations. Of particular interest are Fredholm Equations of the form,

$$x(s) - \lambda \int_a^b k(s,t)F(x)(t)dt = y(s).$$

The function $k(s,t)$ is referred to as the kernel function. The other function $F(x)(t)$ may be nonlinear and $\lambda$ is a scalar . Integral operators are sometimes refered to by some property of their kernel(such as "an integral operator with 'smooth' kernel" or "integral operator with singular kernel").

In addition to the terminology there is also a special notation associated with the study of integral equations. Which reduces the equation

$$x(s) - \lambda \int_a^b k(s,t)F(x)(t)dt = y(s).$$

to the more friendly $x - \lambda KF(x) = y$ or even the more compact $(I - \lambda KF)x = y$. It is customary to denote to the integral operator as the capital letter if the kernel is a lower case letter, ie.

$$Kx = \int_a^b k(s,t)x(t)dt.$$

The norm of a linear integral operator is a simple computation. There is even an easy bound that can be computed.

$$\begin{aligned}
\|K\| &= \sup_{\|x\|\leq 1} \|Kx\| \\
&= \sup_{\|x\|=1} \| \int_a^b k(s,t)x(t)dt\| \\
&= \sup_{\|x\|=1} \sup_{s\in[a,b]} \left| \int_a^b k(s,t)x(t)dt \right| \\
&\leq \sup_{s\in[a,b]} \left| \int_a^b k(s,t)dt \right| \\
&\leq \sup_{s\in[a,b]} \int_a^b |k(s,t)|\, dt.
\end{aligned}$$

So

$$\|K\| \leq \sup_{s\in[a,b]} \int_a^b |k(s,t)|dt.$$

Which gives a bound for the norm of an integral operator. In the section on numerical considerations this will be reviewed and approximated.

The computation of the derivative of an integral operator is fairly straight forward, since integral operators are linear. The derivative of an integral operator is just the integral operator(ie. $K'(x_0) = K$).

Recall, the interest is in using a computer to arrive at a solution. The usual method for integration on a computer is quadrature. Quadrature means to approximate the integration by a weighted sum. In general the form for such an approximation is,

$$Kx \approx K_n x = \sum_{i=1}^{n} w_i k(s,t_i)x(t_i),$$

where the $w_i$ and $t_i$ are defined by the integration approximation used.

There are theorems on the approximation of $K$ by $K_n$. The immediate thought is that as $n$ grows the approximation of $K$ by $K_n$ improves. In general it does not for it can be shown,

$$\lim_{n\to\infty} \|K_n - K\| \to 2\|K\|.$$

However, there is pointwise convergence at each $x$. Pointwise convergence refers to a dependence on the particular value of $x$ for convergence. In other words given any $K$,

$$\lim_{n\to\infty} \|K_n x - Kx\| \to 0,$$

for any particular choice of $x$.

## 4. Newton's Method In A Banach Space

This section describes Newton's method. Newton's method is used to find roots or solutions of an equation, $f(x^*) = y$ for $x^*$ and $f$ may be nonlinear. Newton's method can also be used to solve equations in multiple dimensions.

In an introductory numerical analysis class one of the methods for solving $f(x) = y$ in $\Re$ usually described is Newton's method,

$$x_{i+1} = x_i + \frac{y - f(x_i)}{f'(x_i)}.$$

This is based on Taylor's Theorem from first year calculus:

$$f(x) = f(x_0) + \sum_{i=1}^{q} \frac{f^{(i)}(x_0)(x - x_0)^i}{i!} + R_q(x_0, x),$$

where

$$R_q(x_0, x) = \int_{x_0}^{x} \frac{(x - t)^q}{q!} f^{(q+1)}(t)dt.$$

The term $f^{(i)}(x)$ refers to the $i^{th}$ derivative of $f$ evaluated at $x$. The function $R_q(x_o, x)$ is called the error term

It is possible to make a simple approximation to $f(x)$ which is reasonable for values of $x$ near $x_0$. Suppose it is desirable to solve for $x$ given $f(x) = y$. Starting from $y = f(x) \approx f(x_0) + f'(x_0)(x_1 - x_0)$ and rearranging the equation to get

$$x_1 = x_0 + \frac{y - f(x_0)}{f'(x_0)}.$$

The above equation is a first approximation, to improve the approximation try using the equation again,

$$x_2 = x_1 + \frac{y - f(x_1)}{f'(x_1)}.$$

This $x_2$ is often a better approximation than $x_0$ to the true value of $x$ that is the goal. In general we get,

$$x_{i+1} = x_i + \frac{y - f(x_i)}{f'(x_i)},$$

and in practice the iteration is performed until $|x_i - x_{i+1}|$ is sufficently small. There are three other possibilities that arise in practice:

1). $f'(x_i) = 0$. This would cause a divide by zero.

2). $x_i \to \pm\infty$. Computers don't deal well with infinity so this is an error condition.

3). $x_i$ oscillates or takes a large number of iterations to converge.

There is a Banach space generalization of Taylor's Theorem. Suppose that $P(x)$ is differentiable $q$ times in the ball $B(x_0, r)$, $r > 0$, and $P^{(q)}(x)$ is integrable from $x_0$ to any $x_1 \in U(x_0, r)$. Then

$$P(x_1) = P(x_0) + \sum_{k=1}^{q} \frac{1}{k!} P^{(k)}(x_0)(x_1 - x_0)^k + R_q(x_0, x_1),$$

where

$$R_q(x_0, x_1) = \int_{x_0}^{x_1} \frac{(x-\theta)^q}{q!} f^{(q+1)}(\theta) d\theta.$$

where $P^{(q+1)}$ is a multilinear operator, operating on $q$ vectors all equal to $(x_1 - x_0)$. The proof for this is found in Rall (pg. 124-125). Those interested in mathematical analysis will note that the sum is the same one presented in first year calculus. The remainder term however is very different from the usual one dimensional remainder term.

Using this as a starting point, we have,

$$y = G(x) \approx G(x_0) + G'(x_0)(x - x_0) + \textit{higher order terms}.$$

Neglecting the higher order terms, we get,

$$\begin{aligned} y &\approx G(x_0) + G'(x_0)(x - x_0) \\ &= G(x_0) + G'(x_0)x - G'(x_0)x_0 \\ G'(x_0)x &\approx -G(x_0) + y + G'(x_0)x_0 \\ &= G'(x_0)x_0 + y - G(x_0) \\ (G'(x_0))^{-1}G'(x_0)x &\approx (G'(x_0))^{-1}G'(x_0)x_0 + (G'(x_0))^{-1}(y - G(x_0)) \\ x &\approx x_0 + (G'(x_0))^{-1}(y - G(x_0)), \end{aligned}$$

if $G'(x_0)^{-1}$ exists. Giving a Newton's method iteration of

$$x_{i+1} = x_i + (G'(x_i))^{-1}(y - G(x_i)),$$

for $i = 0, 1, 2, ...$ and a guess $x_0$ for the real value of $x$.

At this point I would like to relate this technique to the problem at hand, the solution of integral equations.

The equation $(I - KF)x(s) = y(s)$ can be differentiated with respect to $x$, since we are in a Banach space. Define a new function $G(x) = (I - KF)x$, now find $G'(x)$.

$$\begin{aligned} G'(x) &= \frac{d}{dx}((I - KF)x) \\ &= \frac{d}{dx}(x - KF(x)) \\ &= \frac{d}{dx}x - \frac{d}{dx}(KF(x)) \\ &= I - K'(Fx)F'(x) \\ &= I - KF'(x) \end{aligned}$$

The last step is because K is a linear operator, and we are using the Frechet derivative. This derivative is in a function space instead of real(number) space, which means $\frac{d}{dx}x = I$.

This yields a Newton's method iteration of

$$\begin{aligned} x_{i+1} &= x_i + (G'(x_i))^{-1}(y - G(x_i)) \\ &= x_i + (I - KF'(x_i))^{-1}(y - (x_i - KF(x_i))) \end{aligned}$$

This last equation is a little messy, even in functional notation.

## 5. The Kantorovich Theorem

The Kantorovich theorem is unique in numerical analysis. It proves existence, uniqueness in $S$ and an error bound. All this is from a fairly simple set of analytical assumptions.

Suppose that $\mathbf{C}$ is an open convex region in a Banach space $\mathbf{X}$ and that $\mathbf{Y}$ is also a Banach space. Let $f : \mathbf{C} \to \mathbf{Y}$ be a differentiable function defined on $C$ whose derivative satisfies the Lipschitz condition:

$$\|f'(x) - f'(y)\| \leq M\|x - y\|,$$

for $x, y \in C$. Assume further for some $x_0 \in C$, $G_0 = [f'(x_0)]^{-1}$ exists, $\|G_0\| \leq b$ and $\|G_0 f(x_0)\| \leq \eta$. Define $h = b\eta M$; assume that $h \leq 1/2$. Let $t^* = (2\eta)/(1 + \sqrt{1-2h})$. Suppose that

$$S = \{x \in \mathbf{X} : \|x - x_0\| \leq t^*\} \subset C.$$

Then the Newton sequence,$\{x_i\}$, is defined, $x_i \in S$, $x_i$ converges to a root , $x^*$, of $f$ and the error bound

$$\|x^* - x_k\| \leq t^* \frac{(1 - \sqrt{1-2h})^{2^k - 1}}{2^k},$$

holds for $k = 0, 1, 2, 3, 4, ....$

If the conditions for the Kantorovich theorem are met then there exists an open ball $\mathbf{B}' = \mathbf{B}(x_0, r_0) \in \mathbf{C}$ and $x^*$ is unique in $\mathbf{B}'$, where $r_0$ is defined as

$$r_o = t^* \left( \frac{1 + \sqrt{1-2h}}{1 - \sqrt{1-2h}} \right).$$

There are several different forms for this theorem with different error bounds but, this is one of the more recent ones with a smaller error. This particular version of the error bound is found in Groetch(pg. 256-261).

Proofs for this theorem are in several different places, Kantorovich and Akilov(pg. 695-723), Groetch(pg. 252-262) and Rall(pg. 133-142).

In order to program this error bound one must decipher the meaning of the constants $M$, $b$, and $\eta$ in relation to Fredholm equations. The function $f(x)$ represents $x - KFx - y$, the function for which we want to find a root.

$M$ is given as a Lipschitz condition on the first derivative. From analysis this can also be considered a bound on the norm of the second derivative of $f$. For Fredholm integral equations this is $\|KF''(x_0)\| \leq M$.

The constant $b$ is a bound on the norm of $[f'(x_0)]^{-1}$. In the case of the Fredholm integral equation, this would be $\|(I - KF'(x))^{-1}\| \leq b$.

$\eta$ is given as $\|G_0 f(x_0)\| \leq \eta$, or substituting in earlier things,

$$\|(f'(x_0))^{-1} f(x_0)\| \leq \eta.$$

If one examines the Newton iteration, this is a bound on the norm of the change in $x$ on the first step of Newton's method.

$$\begin{aligned}\|(f'(x_0))^{-1} f(x_0)\| &= \|(I - KF'x_0)^{-1}(x_0 - KFx_0 - y)\| \\ &= \| - (I - KF'x_0)^{-1}(y - (x_0 - KFx_0))\| \\ &= \|x_1 - x_0\|\end{aligned}$$

We have $\|x_1 - x_0\| \leq \eta$ as the last inequality.

## 6. Programming Considerations

There is a collection of theory about the solution of integral equations. However, there is a semantic gap between the mathematical theory and the computing languages that are in common use today. In order to use a computer to approximate solutions to these equations one must reformulate the theoretical methods into appropriate methods for a computer. This section contains several approximations of this type.

The theory in the beginning of this paper holds for functions from arbitrary Banach spaces, but we are interested in integral equations. These integral equations are functions $f : \mathbf{C}[a,b] \rightarrow \mathbf{C}^n[a,b]$. This observation leads to a choice for the Banach space of

$$\mathbf{S} = \{functions\ f\ s.t.\ f : [a,b] \rightarrow \Re^n\},$$

where $n \in \{1,2,3,...\}$.

For any algorithm on a computer there are two different methods to represent functions as vectors of function values at sample points or algebraically as polynomials of analytic functions. The more prevalent method at this time is to represent functions as a vector of function values, this is also the representation used here.

In Chapter 4 Newton's method for the Fredholm equation was found to be,

$$x_{i+1} = x_i + (I - KF'(x_i))^{-1}(y - (x_i - KF(x_i))).$$

This equation is a little more theoretical than computers can handle. This equation has an inverse operator in it which must be converted to computer form. The desire is to arrive at a computable approximation to $x_i$. Define $rhs_i(s) = y(s) - (I - KF)(x_i(s))$, and $z_i = x_{i+1} - x_i$, these will simplify the equations.

$$\begin{aligned} x_{i+1} &= x_i + (I - KF'(x_i))^{-1}(y - (x_i - KF(x_i))) \\ &= x_i + (I - KF'(x_i))^{-1} rhs_i \\ (I - KF'(x_i))(x_{i+1} - x_i) &= rhs_i \\ (I - KF'(x_i))z_i &= rhs_i \end{aligned}$$

This equation is better since it linear and involves the solution of a linear system. In this last equation, everything is known except $z_i$. This equation now reads $(I - KF'(x_i))z_i = rhs_i$, this is a linear equation(by definition of the derivative) and we wish to solve it for $z_i(s)$. Then $x_{i+1} = x_i + z_i$.

At this point, sample the domain of the functions $z_i(s)$ and $rhs_i(s)$ at $n$ discrete points, say $s_j$'s turning the functions $z_i$ and $rhs_i$ into vectors. Let us assume that these points are equally spaced in the interval $[a,\ b]$. Giving a system of equations,

$$(I - KF'(x_i))z(s_j) = rhs(s_j).$$

$$z(s_j) - (KF'(x_i))z(s_j) =$$

Solve for the function $(KF'(x_i))$ operating on the vector(or function) $z$, since this is the only unknown in the system of equations. Select an integration method: the trapezoid rule is one of many different quadrature methods that would work in this situation.

Since $z$ and $rhs$ are both vectors we can easily assume that $KF'(x_i)$ is a matrix. With this little bit of knowledge, it can be observed that the computation for the integration is

$$\begin{aligned} KF'xz &= \lambda \sum_{l=2}^{n} \frac{k(s_j,t_l)F'(x(t_l))z(t_l)}{2n} + \frac{k(s_j,t_{l-1})F'(x(t_{l-1}))z(t_{l-1})}{2n} \\ &= \frac{\lambda}{n} \sum_{l=2}^{n} \frac{k(s_j,t_l)F'(x(t_l))z(t_l)}{2} + \frac{k(s_j,t_{l-1})F'(x(t_{l-1}))z(t_{l-1})}{2} \\ &= \frac{\lambda}{n} \left( \sum_{l=2}^{n-1} k(s_j,t_l)F'(x(t_l))z(t_l) \right) + \\ &\qquad \frac{\lambda k(s_j,t_1)F'(x(t_1))z(t_1)}{2n} + \frac{\lambda k(s_j,t_n)F'(x(t_n))z(t_n)}{2n}. \end{aligned}$$

where $t_l = \frac{l-1}{n}(b-a) + a$. Let $s_j = t_j = \frac{j-1}{n}(b-a) + a$. There is a way to rewrite the above sum as a matrix equation, but it is of more interest to write

$(I - KF'(x))z = rhs$ as a matrix equation. Define $w = \frac{\lambda}{n}$. Then the matrix approximation to $I - KF'(x)$ is

$$\begin{pmatrix} 1 - \frac{w}{2}k(s_1,s_1)F'(x(s_1)) & wk(s_1,s_2)F'(x(s_2)) & \dots & \frac{w}{2}k(s_1,s_n)F'(x(s_n)) \\ \frac{w}{2}k(s_2,s_1)F'(x(s_1)) & & \dots & \frac{w}{2}k(s_2,s_n)F'(x(s_n)) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{w}{2}k(s_n,s_1)F'(x(s_1)) & wk(s_n,s_2)F'(x(s_2)) & \dots & 1 - \frac{w}{2}k(s_n,s_n)F'(x(s_n)) \end{pmatrix}$$

If the function $x$ is vector valued then the matrix has a form similar to

$$\begin{pmatrix} 1 - \frac{w}{2}k_1(s_1,s_1)\frac{\partial}{\partial x_1}F_1(x(s_1)) & \frac{w}{2}k_1(s_1,s_1)\frac{\partial}{\partial x_2}F_1(x(s_1)) & \dots \\ \frac{w}{2}k_2(s_1,s_1)\frac{\partial}{\partial x_1}F_2(x(s_1)) & 1 - \frac{w}{2}k_2(s_1,s_1)\frac{\partial}{\partial x_2}F_2(x(s_1)) & \dots \\ \vdots & \vdots & \ddots \end{pmatrix}.$$

The observant reader will note, that I am trying to indicate that each of the elements in the first matrix is replaced with a submatrix containing $I - KF'(x_j)(s_i)$ or $-KF'(x_j)(s_i)$ for the element in the row $i$ column $j$ position. This representation of the matrix implies that the vectors representing the functions $x$, $y$, $z$ and any others used in this method should be understood to be in the form

$$x = \begin{pmatrix} x_1(s_1) \\ x_2(s_1) \\ \vdots \\ x_n(s_1) \\ x_1(s_2) \\ \vdots \\ x_n(s_m) \end{pmatrix}.$$

At this point, there is an approximation to $I - KF'(x)$ and a representation of $x_i$. The formulation for a Newton iteration under these conditions is reasonable. Given a way of computing $(I - KF)x$ for any value of $x$, and an initial guess for

$x$, called $x_0$, and some tolerance, called $z_{min}$. Set $i = 0$. Then repeat the following steps.

1.) Set $rhs_i = y - (I - KF)x_i$ by using the quadrature approximation used for the operator $KF'$.

2.) Solve for $z$ in $(I - KF'(x_i))z = rhs_i$.

3.) $x_{i+1} = x_i + z$.

4.) If $\|z\| < z_{min}$ then $x_{i+1}$ is an approximate solution to the discrete problem and the iteration is complete. Otherwise, set $i = i + 1$ and goto step 1.

This is a common representation for Newton's method, except for the use of the operators $I - KF'(x_i)$ and $(I - KF)x_i$.

Newton's method has been proven to work, under certain conditions(see Chapter 5). The initial guess, $x_0$, must be "close" in order for Newton's method to reach a solution. This may not be a major problem when dealing with a well understood system, but if it is an experimental system the initial guess may be very hard to find with the accuracy required to converge to a solution.

One method around this problem has been called practical homotopy. This method revolves around the idea that although a solution to the exact problem may not be possible to guess well, there is a problem close to it that can be solved. In practice it is used for tracing a solution function $x$ depending on a parameter $\gamma$. Suppose there is a function $G(x, \gamma) = y$ that is easily solved at $\gamma_0$ for $x$. Given the values $\Delta$, $x_0$ a solution of $G$ at $\gamma_0$ and a $\min \Delta$. Set $i = 1$. Then while $\gamma_i$ less than the final gamma value perform the following steps:

1.) if $\Delta < \min \Delta$ quit with error message.

2.) $P_1 = \frac{\partial}{\partial x} G(x_{i-1}, \gamma_{i-1})$.

3.) $P_2 = \frac{\partial}{\partial \gamma} G(x_{i-1}, \gamma_{i-1})$.

4.) Solve $P_1 z = -\Delta P_2$. If this can not be solved for $z$ for any reason set $\Delta = \Delta/2$, and go to step 1.(Note: Because of the subscripts this is equivalent to backing up to the last $x$ that was a solution.) This value $z$ is sometimes called a residue.

5.) $x_i = x_{i-1} + z$.

6.) $\gamma_i = \gamma_{i-1} + \Delta$.

7.) Solve $G(x_i, \gamma_i) = y$ using Newton's method. If Newton's method does not find a solution set $\Delta = \Delta/2$, and go to step 1.(Note: Because of the subscripts this is equivalent to backing up to the last $x$ that was a solution.)

8.) i = i + 1.

This method is very good at following solutions, along $\gamma$. This method may take significant time, if $\Delta$ gets small.

There is a mathematical reason behind the method of practical homotopy. Given an equation $G(x, \gamma) = y$. If there is a known solution at say $\gamma = \gamma_0$ of $x_0$ then there is an approximation at any point nearby,

$$\begin{aligned} y &= G(x_0 + z, \gamma_0 + \Delta) \\ &\approx G(x_0, \gamma_0) + \frac{\partial}{\partial x} G(x_0, \gamma_0) z + \frac{\partial}{\partial \gamma} G(x_0, \gamma_0) \Delta \\ &= y + \frac{\partial}{\partial x} G(x_0, \gamma_0) z + \frac{\partial}{\partial \gamma} G(x_0, \gamma_0) \Delta \\ 0 &\approx \frac{\partial}{\partial x} G(x_0, \gamma_0) z + \frac{\partial}{\partial \gamma} G(x_0, \gamma_0) \Delta \\ \frac{\partial}{\partial x} G(x_0, \gamma_0) z &\approx -\frac{\partial}{\partial \gamma} G(x_0, \gamma_0) \Delta. \end{aligned}$$

This last equation gives information on what small changes in $\gamma$ do to $x$. We can use this equation to motivate the algorithm for tracing a solution along the parameter $\gamma$.

The computation of the Kantorovich error bound is not as trivial as a first glance might lead one to think. There are three norms to compute and two of them

are not obvious. These these norms are bounds on the following qualities from chapter 5, $M$, $\eta$ and $b$.

In this subroutine set the functions $x$ and $y$ are represented as finite vectors of numbers, under these conditions the max norm can be defined as,

$$\|x\| = \max_{j\in\{1,n\}} |x[j]|,$$

if $x[j]$ indicates the $j^{th}$ element of the vector $x$, a $n$ dimensional vector. From linear algebra, the compatible matrix norm for the max norm is,

$$\|L\| = \max_{j\in\{1,n\}} \sum_{k=1}^{n} |L[j,k]|,$$

for $L$ an $nxn$ matrix. This particular matrix norm can be thought of as the max row sum. There is also a three dimensional matrix form for compatible with the max norm(presented without proof),

$$\|A\| \leq \max_{j\in\{1,n\}} \sum_{l=1}^{n}\sum_{k=1}^{n} |A[j,k,l]|,$$

where $A$ is a three dimensional matrix, indexed as $A[$ *row*, *column*, *depth*$]$. If one thinks of the three dimensional matrix as a stack of square plates of elements, then this is kind of a max plate sum.

Of the three different norms that need to be computed, only one is computed just like the definition. That one is $\eta$. This value is computed in the Newton's method calculation as part of the stopping criteria. This value is just the maximum of the absolute values of the components of $z$, the update to $x_i$.

The next value computed is $b$. This is the matrix norm of the inverse matrix, $\|(I - KF'(x_i))\|$. Computing the inverse of a matrix is relatively difficult, about three times as complex as doing an LU factorization. This is almost enough reason to find another method that might be easier. I chose to compute each of the vectors, sum these vectors and take the maximum row. In more mathematical notation,

$$\begin{aligned} b &= \|(I - KF'(x_i)\| \\ &= \max_{row} \sum_{j=1}^{n} |(I - KF'(x_i))^{-1} e_j|, \end{aligned}$$

where $\max_{row}$ means taking the maximum row value and $e_j$ is a vector of zeros(0's) with a one in the $j^{th}$ row.

The final norm to be computed is $M$. This value is the Lipschitz constant that bounds the variation in the first derivative. Define $G(x) = (I - KF)x$. One of the hypothesis of the Kantorovich theorem is that,

$$\|G'(x)) - G'(y)\| \leq M\|x - y\|,$$

for $x$, $y$ in an open convex region. The theory tells us that $M$ is an upper bound on $\|G''(x)\|$ for $x$ in an open region about $x_0$.

$$\begin{aligned}\|G''(x_i)\| &= \| - KF'(x_i)\| \\ &\leq \|K\|\|F'(x_i)\|\end{aligned}$$

This last line qualifies as an upper bound on $G''$ so this is out choice for $M$.

From Chapter 3 we have,

$$\|K\| = \int_a^b \left|k(s_j, s_i)\right| dt$$

Which is a bound on the norm of $K$. Working from this with our approximation $K_n$ of the operator $K$ we arrive at an approximation to $\|K\|_\infty$ that is reasonable to compute.

$$\begin{aligned}\|K\| &= \int_a^b \left|k(s_j, s_i)\right| dt \\ &\approx \max_{j\in\{1,\ldots,n\}} \sum_{i=1}^{n} w_i \left|k(s_j, s_i)\right|\end{aligned}$$

Similarly, we wish to compute an approximation to $\|F''(x(s_i))\|$. It is known that the representation for $F''(x(s_i))$ is a three dimensional matrix, which we have a formula to compute the norm.

$$\|F''(x_i)\| \approx \max_{i\in\{1,m\}} \max_{j\in\{1,n\}} \sum_{l=1}^{n}\sum_{k=1}^{n} |F''(x(s_i))[j,k,l]|$$

The Kantorovich theorem gives a worst case error bound for Newton's method after $n$ iterations. In theory, when computing this error bound there is a value, $h$, that is an estimate of how close the guess is to the region of convergence. If this $h$ value is larger than 0.5 then the theory does not guarantee convergence. In practice this initial value for $h$, has little to do with the eventual convergence of Newton's method. I have noted on many occasions that $h$ was greater than 1 and the method still converged. With this in mind, instead of basing the error estimate on the initial guess, use the "x" vector that we know will be the $n-1$ iterate. This is a much nicer guess for the solution to use, we know that it is in the region of convergence and we know that it will only iterate once so the equation for the error bound is simplified.

The equation for the error bound from chapter 5 has a value $k$. This value $k$ represents the number of iterations performed after the computation of $t^*$ and $h$ for the final approximation to $x$. Because of the choice of when this computation will be performed, it can be assumed that $k = 1$.

$$\|x^* - x_k\| \leq t^* \frac{(1-\sqrt{1-2h})^{2^k-1}}{2^k}$$

$$\|x^* - x_1\| \leq t^* \frac{(1-\sqrt{1-2h})}{2}$$

# 7. Description of Using Subroutine Package

This chapter is a users guide for procedures and functions written for the solution of Fredholm integral equations. The subroutine package is in three parts, each part is a Pascal source file. The source files are intended to be included in the users program by use of compiler control statements. The parts are matrix solution, Newton's method and practical homotopy. There is an example program using these subroutines in appendix C and an outline of the steps required to write a program in appendix B.

The method described in this paper reduces the solution of integral equation to a Newton like iteration on a system of non-linear equations. A linear equation solution package must be the base for any program that attempts to solve integral equations in the manner described.

There are two major parts to getting the subroutines to solve a system, first writing the procedures that define the integral equation desired and second using the calling sequence to get the solution out. For the discussion, parameter values will be in quotation marks (i.e., $x$ is the theoretical value and "x" is a parameter to a function or procedure.)

The subroutines are written in Pascal to be used with the Borland's Turbo Pascal. With minor modifications they could be used with most other Pascal compilers. There are three extensions used in these subroutines that may cause problems in the conversion:

1). Some Pascal compilers might be case sensitive. This means that some compilers want reserved words in all capital letters or lower case letters, and compare capitalization of user defined words.

2). The include files use type "extended". "Extended" variables are extended precision floating point numbers. For compilers without type "extended", it these should be changed to "real."

3). Forward declarations are a defined part of the Pascal language but different the compilers implementation of this vary. The difference is with the formal parameters, that is the variables in the parenthesis after the procedure name and before the semicolon. There are two common formats:

   a). Turbo Pascal allows the repeat of the formal parameters in the subroutine heading.

   b). The Pascal definition does not allow for the repeat of the formal parameters.

Making these changes to the include files takes less than an hour using text editor with a search and replace command.

In Pascal programming you must declare the variables that you use. This means that you must define the variables that you need before you use them. Programmers are allowed multi-letter names for identifiers.

The first step is to determine the constants for the program. At present there are only two values that must absolutely be constants. They are "arsize" and "func_dimen".

1.) The linear equation routines require that "arsize" must be declared an integer constant before the inclusion of the file "matrix.pas." The dimensions of the types "matrix," "vector" and "ivector" are all related to"arsize." The maximum number of sample points used for the quadrature method is "arsize" / "func_dimen." A good starting guess is 50.

2.) The value of "func_dimen" represents the range dimension of the system of integral equations, and as such should be an integer constant. This value is the dimention of the range of the function "x." The constant "func_dimen" corresponds to the theoretical value $n$ in the preceding chapter. Just as the size of some types is tied to "arsize" there are also some types whose size is tied to "func_dimen." These types are "x_vect," "square" and "cube."

There are six different data types used by the subroutines and defined in the include files. Three are defined for the solution of linear equations and the other three are used to represent function values in multiple dimensions. The three for linear equations are:

1). "Matrix" is an "arsize"x"arsize" array of type extended.

2). "Vector" is a "arsize"x1 array of type extended.

3). "Ivector" is a "arsizex1 array or type integer. Variables of this type are used as pivot vectors for the LU-decomposition procedure.

The other three defined data types are used to represent function evaluations at points.

1). "X_vect" is used to represent multidimensional function evaluations at a particular node point. "X_vect" is "func_dimen"x1 array of type extended.

2). "Square" is used to represent multidimensional derivative evaluations at a particular node point. "Square" is "func_dimen"x"func_dimen" array of type extended.

3). "Cube" is used to represent multidimensional second derivative evaluations at a particular node point. "Cube is a three dimensional martix "func_dimen" on a side.

Then, declare the variables for the procedures. Most of these names could be either variables or constants. There are only two names that must represent variables, "f_prime" and "mat_inv_norm."

1.) "F_prime" must be declared before the inclusion of the file "int_eqn.pas" as type "matrix." This variable is used internally for the matrix $(I - KF'(x))$. It is the only matrix used by the subroutines, and is defined globally because of stack limitations on 80x86 type computers.

2.) "Mat_inv_norm" should be declared at the same time as "f_prime" as a type "extended" or "real." This variable is set by the function KANTOROVICH to the approximation of,

$$\text{"}mat\_inv\_norm\text{"} = \|(I - KF'(x_n))^{-1}\|.$$

There are several other names that must be defined before the inclusion of the files that define the subroutines, these names may specify ether constants of variables. They are required by the various different procedures in "int_eqn.pas," here is a list "lu_epsilon", "newton_err," "lambda" and "num_eqn."

1.) "Lu_epsilon" is a value used for a singularity check when a matrix is factored into LU form. It should represent a small floating point value compared to the norm of $i - KF(x)$. Try $1\text{x}10^{-8}$. as a starting point for values of "lu_epsilon"

2.) The value of "lambda" is from the integral equation, and should be apparent to the user. In most cases "lambda" is a floating point value.

3.) "Newton_err" is the tolerance for the procedure NEWTON_METHOD, this value is the maximum $\|z\|$ value that will cause the procedure NEWTON_METHOD to iterate. This should be a floating point value and depends on the problem being solved.

4.) "Num_eqn" is the number of defined elements in the vector "s." This means that there are "num_eqn" * "func_dimen" defined values in the vector "x," so "arsize" > "num_eqn" * "func_dimen." This is the number of node points used in the quadrature method.

There are two names required for the procedure HOMOTOPY.

1.) "Newton_iter" is a name that may reference ether a constant or a variable, but it must be declared before the inclusion of the file "homotopy.pas." It

controls the number of iterations that NEWTON_METHOD will take when called from the procedure HOMOTOPY, and should be of type integer.

2.) "Gamma" is the $\gamma$ value for the method of practical homotopy. "Gamma" must represent a floating point variable, since this value is changed by the procedure HOMOTOPY.

Writing the procedures requires the user to understand how the rest of the code expects to find the values in the $x$ values. The vector called "x" in most procedures represents an approximation to the function $x$ from the theory. It may be a vector valued function. If $x$ is a vector valued function then the values for a particular "s" value are stored in a group in the function. Suppose that $x$ has a range in $\Re^3$, this means that "func_dimen" = 3. To access $x_i(s_j)$, one references the value in "x[(j - 1) * func_dimen + i]" or "x[(j-1)*3 +i]." To examine this relationship compare the functions in chapter 9 and the procedures in appendix B(eg. look at the functions $F_1$ and $F_2$ and the procedure Cap_F.)

There are five procedures that specify the integral equation to the program and each represents a function. The functions are: the function $F(x)$ and its first two derivatives($F'(x)$ and $F''(x)$), the function $k(s,t)$ and the right hand side function $y(s)$. The following is a annotated list of the procedure headings to explain the parameters to be passed.

```
PROCEDURE Cap_F(  VAR val : x_vect;
                      i : integer;
                  VAR  x : vector;
                  VAR  s : vector);
```

This procedure represents the function $F(x)$ in the Fredholm integral equation. If it is a system of integral equations then the return will be a short vector, "val". The value of each entry should contain the value for the corresponding $F(x)$ function.(ie. $val[j] = F_j(x(s_i))$).

```
PROCEDURE D_Cap_F(  VAR d_f : square;
                        i : integer;
                    VAR x : vector;
                    VAR s : vector);
```

D_Cap_F is the derivative of the preceding function, $F$, evaluated at "x($s_i$)." The return value is stored in "d_f". This is, in general, an asymmetric matrix and represents a Jacobean matrix evaluated at the current guess for the vector "x".

$$d\_f[j,k] = \frac{\partial F_j}{\partial x_k}(x(s_i))$$

```
PROCEDURE DD_Cap_F(  VAR dd_f : cube;
                         i : integer;
                     VAR x : vector;
                     VAR s : vector);
```

DD_Cap_F is the second derivative of the function, $F$, evaluated at "x($s_i$)." The return value is stored in "dd_f". This is a three dimensional matrix and represents a second derivative evaluated at the current guess for the vector "x".

$$dd\_f[j,k,m] = \frac{\partial^2 F_j}{\partial x_k \partial x_m}(x(s_i))$$

```
PROCEDURE K(  VAR val : x_vect);
                  s,t : extended);
```

A procedure that represents the kernel function of the Fredholm equation to be solved. The function values are to be stored in the short vector called "val'. If the equation to be solved is a system then the user must place the values of each separate kernel in the corresponding locations in "val". (ie. $val[i] = k_i(s,t)$).

```
PROCEDURE Y_FUNC(  VAR val : x_vect;
                       s : extended);
```

This represents function $y$ in the integral equation $(I - KF)(x) = y$. As before, the user should use the corresponding subscripts for the short return vector "val" as the function y(ie. $val[i] = y_i(s)$). Caution: In this procedure, unlike the other functions, the "s" value passed is a real value and not a vector.

There are three procedures needed for the homotopy method that are not needed for the strait forward newton iteration. They are two output procedures and one function procedure. The output procedures are needed since the HOMOTOPY procedure does not return control to the user until either the final "gamma" value is reached or the method fails. In both cases there are a lot of solutions that the user will not have access to if the method does not call procedures that the user has control over.

```
PROCEDURE Cap_FDG(  VAR val : x_vect;
                        i : integer;
                    VAR x : vector;
                    VAR s : vector);
```

This procedure is the derivative of the function $F$ with respect to the parameter "gamma." The parameter "gamma" may have significance in the problem, or it may be an artificial value to enable the use of the procedure HOMOTOPY. As is usual with these procedures the return value "val" contains the derivative of the function evaluated at "x($s_i$)."

$$val[j] = \frac{\partial}{\partial \gamma} F_j(x(s_i))$$

```
PROCEDURE HOMOTOPY_OK_OUT(      gamma : extended;
                            err_est : extended;
                        VAR       x : vector;
                        VAR       s : vector);
```

The procedure HOMOTOPY_OK_OUT is a procedure called by the procedure HOMOTOPY to allow the user to produce output while the method is iterating.

Enough data is passed to this procedure that it can print most of the interesting data about the function at this particular "gamma."

```
PROCEDURE HOMOTOPY_ERR_OUT(  gamma : extended;
                             why   : integer);
```

The procedure HOMOTOPY_ERR_OUT is a output procedure to allow the user to print messages to inform about the failure of the HOMOTOPY procedure in the attempt to advance $\gamma$ by the current "delta" value. The values for "why" are 1 through 4. All but 4 correspond to values returned by NEWTON_METHOD.

1: Newton's method found a singular matrix for $(I - KF'(x))$. This causes the procedure to stop, because NEWTON_METHOD is required to use this matrix to solve for the next "x" vector.

2: The iteration count exceeded the allowed number in to the procedure NEWTON_METHOD.

3: The value that would be added to an "$x_i$" would cause numeric difficulties. This is most probably caused by a step size too large, or a near singular matrix has been found.

4: $(I - KF'(x))$ was found to be singular during the calculation of the residue in the practical homotopy algorithm.

The following procedures are supplied to the user, and intended to be the interface between the user and the code that implements the solution technique. At first glance these procedures seem to have unduly complex parameter passing, but the added functionality gained is superior to having multiple procedures perform such similar tasks.

```
PROCEDURE MAKE_DOMAIN(  VAR  s : vector;
                           a,b : extended;
                             n : integer);
```

A provided procedure to set up the "s" vector of sample points. The elements in this vector are the node points of the quadrature approximation used for integration. The return value, "s," is a vector of "n" equidistant values, such that "S[1]" = "a," "S[n]" = "b". It must be called before NEWTON_METHOD.

```
PROCEDURE MAKE_Y_FUNC(  VAR y : vector;
                            s : vector);
```

This provided procedure sets up the "y" vector, by calling the procedure Y_FUNC. the vector y is used by the procedure NEWTON_METHOD. This procedure depends on the values computed by MAKE_DOMAIN.

```
PROCEDURE FUNC_X(   VAR out : x_vect;
                        at_s : extended;
                    VAR   x : vector;
                    VAR   s : vector);
```

This supplied function allows the user to evaluate the value of the function x at any particular value of s. The user specifies the point the parameter "at_s" and passes the vectors "x" and "s". The return value is a short vector. It should be called only after Newton's method has converged.

```
PROCEDURE NEWTON_METHOD(  VAR   err_est : extended;
                          VAR         x : vector;
                          VAR     error : integer;
                              max_iter : integer;
                                   use : boolean;
                          VAR         y : vector;
                          VAR         s : vector);
```

This procedure implements Newton's method for the solution of integral equations. The input vectors are "x", "y" and "s". The vector "x" defines the starting location in the space to perform the solution. And "y" and "s" are defined by the

problem to be solved. The maximum number of iterations allowed by the procedure is passed in as "max_iter". The boolean flag value "use" is set to "TRUE" when the user wants the Kantorovich error estimate computed. The solution vector is returned in the vector "x", and the Kantorovich error estimate is returned in "err_est". The error estimate is computed without considering numeric truncation or roundoff. In case of an error in the execution of the routine the value "error" will be non-zero.

There are four values for "error" to take on:

0: Newton's method had no difficulties, and has computed a new vector "x."

1: Newton's method found a singular matrix for $(I - KF'(x))$. This causes the procedure to stop, because NEWTON_METHOD is required to use this matrix to solve for the next "x" vector.

2: The iteration count exceeded the value passed in to the procedure in the parameter "max_iter."

3: The value that would be added to an "$x_i$" would cause numeric difficulties. This condition is usually caused by initial guess, "x," not in the region of convergence for Newton's method.

The next procedure requires more support procedures than the direct Newton's method and is in a separate file. The user does not have to worry about those procedures if the file containing the procedure HOMOTOPY is not included in the compilation.

```
PROCEDURE HOMOTOPY(  VAR          x : vector;
                             in_delta : extended;
                            min_delta : extended;
                          max_gamma : extended;
                     VAR          y : vector;
                     VAR          s : vector);
```

This supplied procedure traces a series of solutions through a range of gamma values. The input vector "x" is a guess for the value of $x$ at gamma = 0 and Newton's method is used to improve "x". The value of "in_delta" is the initial step size for gamma, and "min_delta" is the minimum step size allowed. If the algorithm reduces the update to "gamma" below "min_delta" the procedure terminates. In addition to these values one must also pass in the values for "y" and "s".

The last step in using these procedures is to write the main program to call the subroutines provided. This includes defining certain variables and determining the calling sequence for the various procedures provided.

1.) Call MAKE_DOMAIN with the values that you are interested. This procedure sets up the "s" vector that most procedures want as a parameter.(eg. "MAKE_DOMAIN( s, 0, 1, NUM_EQN);")

2.) Call MAKE_Y_FUNC with the names of the variables that you are using. (eg. "MAKE_Y_FUNC( y, s);")

3.) Make an initial guess vector and assign to a vector. If the equation is linear then this is not hard, and almost anything will work. If it is nonlinear then this might be the hardest part of using these procedures to solve an integral equation.

4.) Call the appropriate solution procedure. The options are either NEWTON_METHOD or HOMOTOPY. The user choose one of the procedures to call. The values and parameters in these calls are only examples. The values depend on the equation to be solved and the variable names used in the solution program.

    a.) "NEWTON_METHOD( k_est, x, error, 100, TRUE, y, s);"

    b.) "HOMOTOPY( x, 0.05, 1e-6, 6, y, s);"

Please note that the values in these procedure calls are only example values and may be very different for any particular integral equation that you may have.

## 8. Example: Pendulum Problem.

The pendulum equation is a standard first year physics example of a differential equation. In order for the standard solution methods of differential equations to apply, one must first linearize the problem. With the method described in this paper, formulated as an integral equation the pendulum equation may be solved for the non-linear case.

To derive the integral equation, start from the differential equation,

$$\theta'' + c\theta' + \omega^2 \sin(\theta) = 0.$$

The value $c$ is the coefficient of friction. The parameter $\omega^2$ represents the gravitational force divided by the distance from the pivot to the weight$(g/l)$. The function $\theta$ is the angular position.

Choose a constant for $\omega$ and to neglect friction, $\omega^2 = 4$ and $c = 0$. In order to solve this equation, boundary conditions are needed. The ones chosen are $\theta(0) = 0$ and $\theta(1) = 1$. Rearange the equation,

$$\theta'' + \omega^2 \sin(\theta) = 0$$

to get

$$\theta''(s) + \omega^2\theta(s) = \omega^2(\theta(s) - \sin(\theta(s))) = f(s).$$

Define a function, $f(s) = \omega^2(\theta(s) - \sin(\theta(s)))$. The solution to the homogenous equation is

$$\theta = A\sin(\omega s) + B\cos(\omega s).$$

Let $A$ and $B$ be functions of $s$ then,

$$\theta' = A'\sin(\omega s) + B'\cos(\omega s) + A\omega\cos(\omega s) - B\omega\sin(\omega s).$$

Assume $A'\sin(\omega s) + B'\cos(\omega s) = 0$, so that

$$\theta' = A\omega\cos(\omega s) - B\omega\sin(\omega s)$$

Taking one more derivative, the equation is,

$$\begin{aligned}\theta'' &= A'\omega\cos(\omega s) - B'\omega\sin(\omega s) - A\omega^2\sin(\omega s) - B\omega^2\cos(\omega s)\\ &= A'\omega\cos(\omega s) - B'\omega\sin(\omega s) - \omega^2\theta.\end{aligned}$$

Rewriting the original equation in terms of the homogeneous equation gives,

$$\begin{aligned}\theta'' + \omega^2\theta &= A'\omega\cos(\omega s) - B'\omega\sin(\omega s) - \omega^2\theta + \omega^2\theta\\ &= A'\omega\cos(\omega s) - B'\omega\sin(\omega s) = f(s).\end{aligned}$$

This gives two equations in two unknowns,

$$A'\sin(\omega s) - B'\cos(\omega s) = 0$$

$$A'\omega\cos(\omega s) - B'\omega\sin(\omega s) = f(s).$$

The the values for $A'$ and $B'$ are,

$$\begin{aligned}A' &= \frac{f(s)}{\omega}\cos(\omega s)\\ B' &= \frac{-f(s)}{\omega}\sin(\omega s)\end{aligned}$$

Integrate from 0 to $s$ to regain the original functions

$$\begin{aligned}A(s) &= \int_0^s \frac{f(t)}{\omega}\sin(\omega t)dt\\ B(s) &= -\int_0^s \frac{f(t)}{\omega}\cos(\omega t)dt\end{aligned}$$

Combining this into the homogeneous equation, the final equation for $\theta$ is,

$$\theta = \int_0^s \frac{f(t)}{\omega}\sin(\omega(s-t))dt + A_0\sin(\omega s) + B_0\cos(\omega s).$$

The initial condition $\theta(0) = 0$ implies that $B_0 = 0$. Substitute in the initial condition $\theta(1) = 1$ and rearrange to get the equation

$$A_0\sin(\omega s) = \frac{\sin(\omega s)}{\sin(\omega)} - \int_0^1 f(t)\frac{\sin(\omega(1-t))}{\sin(\omega)}\sin(\omega s)dt.$$

Combining these last two equations,

$$\theta(s) = \int_0^s f(t)\frac{\sin(\omega(s-t))}{\omega}dt - \int_0^1 f(t)\frac{\sin(\omega(1-t))}{\sin(\omega)}\sin(\omega s)dt + \frac{\sin(\omega s)}{\sin(\omega)}.$$

Define the kernel function piecewise as the function in the integral that multiplies the function $f(t)$.

Thus, the integral equation form for this physical system is

$$\theta(s) - \int_0^1 k(s,t)(x(t) - \sin(\theta(t)))dt = \frac{\sin(\omega s)}{\sin(\omega)}$$

with

$$k(s,t) = tmp - \frac{\sin(\omega(1-t))\sin(\omega s)}{\omega \sin(\omega)}$$

and if $t < s$ then $tmp = \frac{\sin(\omega(s-t))}{\omega}$, otherwise $tmp = 0$.

Applying the method described in this paper to this equation we get a solution shown in figure 1.

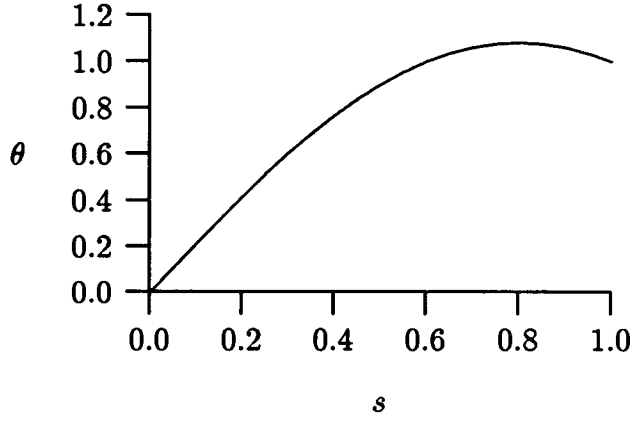


Fig. 1. Graph of the solution for Pendulum position

The Figure 1 represents a graph of the solution function $\theta$, with $\omega = 2$ and the number of divisions of $s$ is 21. The error estimate from the Kantorovich theorem is $3\text{x}10^{-12}$. The Kantorovich error estimate is computed without considering round-off error or truncation error in the computation. This is the solution that arises from almost any starting guess, $\theta_0(s)$. All the starting guesses that I have tried have converged to a solution that is similar to this particular one(sign, leading digits, and order of magnitude.)

## 9. Example: Stress/Strain in Metal Cap.

The integral equation that represents the stress and strain in a spherical metal cap is a problem that has been studied. It is an example of a "hard" integral equation for certain parameter values. It has been shown to be multiple valued with respect to the parameter that corresponds to pressure. The method of practical homotopy as presented in this paper does not handle the this multiple valued case.

Let $\mathbf{X}$= { continous functions on $[0,1]$, with range in $\Re^2$}, and $x \in \mathbf{X}$. The value for $x$ desired is the fixed point defined by $x - KF(x) = y$ where $y \in \mathbf{X}$. Defined by the following equations

$$
\begin{aligned}
y_1(s) &= \frac{1}{4}(s - s^3) \\
y_2(s) &= 0 \\
F_1(x)(s) &= \mu s x_2(s) - \gamma \mu^2 x_1(s) x_2(s) \\
F_2(x)(s) &= -\mu s x_1(s) + \frac{1}{4}\gamma \mu^2 (x_1(s))^2 \\
K_1(x)(s) &= \frac{1}{2}\int_0^1 k_1(s,t) x_1(t) dt \\
K_2(x)(s) &= \frac{1}{2}\int_0^1 k_2(s,t) x_2(t) dt
\end{aligned}
$$

with the kernel function, $k_i$. Let $T = \min(s,t)$ and $S = \max(s,t)$ then

$$
\begin{aligned}
k_1(s,t) &= T(1/S - S) \\
k_2(s,t) &= T(1/S + 2S)
\end{aligned}
$$

along with the provision that $k = 0$ if $s = t = 0$. In this formulation of the problem $\gamma$ is proportional to pressure and $\mu$ is inversely proportional to the thickness of the plate.

This is a coupled system of integral equations, that is the value of the solution for each equation depends on the solution of the other equation. These equations

are

$$x_1(s) - \frac{1}{2}\int_0^1 k_1(s,t)(\mu s x_2(t) - \gamma\mu^2 x_1(t)x_2(t))dt = \frac{1}{4}(s - s^3)$$

$$x_2(s) - \frac{1}{2}\int_0^1 k_2(s,t)(\mu s x_1(t) + \frac{1}{4}\gamma\mu^2(x_1(t))^2)dt = 0.$$

The function $x_1$ is proportional to stress and $x_2$ is proportional to strain.

Applying the method described in this paper to these equations can be solved for small values of $\mu$.

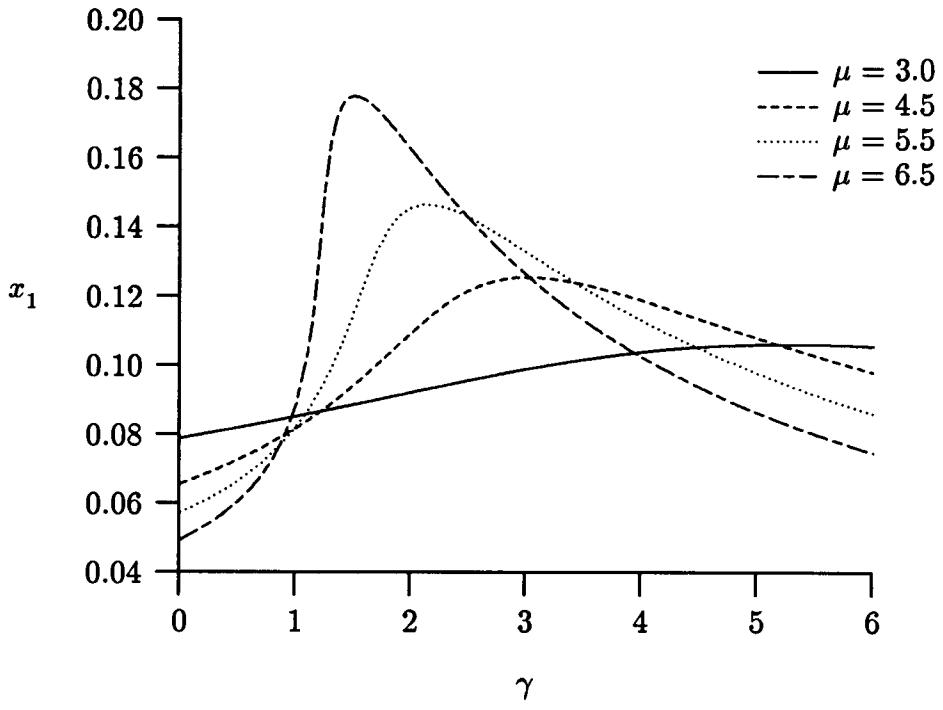


Fig. 2. Stress as a function of pressure in a metal cap.

Figure 2 represents the graphs of the curves for the function $x_1$ various values of $\mu$. The values presented in this graph are for $x_1(0.5)$. The major settings for the solution were newton_iter = 50, newton_error = $1\text{x}10^{-6}$ and num_eqn = 20. The error bound varies greatly on this graph, from about $10^{-9}$ to $10^{-21}$.

The solutions for this problem become more interesting for larger values of $\mu$, but the solutions are not as accesable. The reason for this is that the operator

$I - K_n F'(x)$ becomes singular for large values $\mu$. The derivative with respect to $x$ becomes singular for some $\gamma$ between $\mu = 7.3$ and $\mu = 7.35$. The figure 3 is a graph of the norm of the inverse operator for four different values of $\mu$.

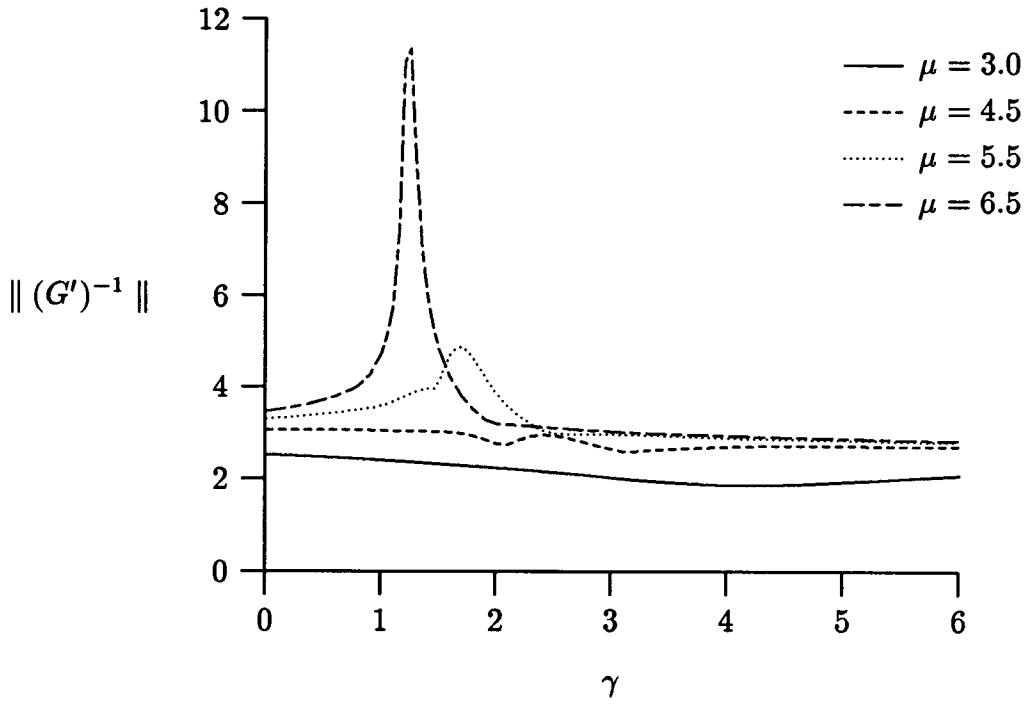


Fig. 3. Norm of the Inverse of $I - K_n F'(x)$.

For cases with simple singularities there is an extension of practical homotopy, called change of parameters. This exention of practical homotopy traces the solution through simple singularities in the derivative. This method uses a new function $Q(x, \rho)$, with the same $x$ but a new parameter $\rho$. Define,

$$Q(x, \rho) = G(x, \gamma(x, \rho)).$$

$G$ is the same function that was used to motivate practical homotopy in chapter 6. This method is more expensive to compute and so is only used to to follow the solution through the areas where the derivative with respect to $\gamma$ is close to singular. In this way solution functions may be found that are multiple valued for $\gamma$.

## 10. Observations and Further Research

I have programmed an extension of Newton's method for approximating the solution of nonlinear Fredholm equations. There are three Turbo Pascal files that will allow the reader to experiment with solutions to this type of equations.

In the body of this paper I made several assumptions that a reader may wish to change. First, I chose the trapezoid rule for integration, when in fact there are many other methods that may yield a closer approximation to the actual solution of the continous problem. The trapezoid rule has its merits, ease of programming and simplicity in explanation are two that come to mind. Second, my version of the method of practical homotopy assumes that the function $y$ does not depend on the parameter $\gamma$. In some cases this could be a major defect in the code, as one could consider making a function $p(\gamma)$ such that $p(0) = 0$ and $p(1) = y$ and then using practical homotopy to find a solution of $(I - KF)x = p(\gamma)$ at $\gamma = 1$. I did not compute the radius of convergence for the approximate solutions, this could be added with a small amount of coding. A further modification to the implementation of practical homotopy could be the addition of change of parameters to the algorithm.

The fundamental assumption made was that all solutions would be computed as finite vectors. There is another method for computer representation of functions, the algebraic representation. An implementation of these algorithms in algebraic form is possible. The results of algebraic methods would be series solutions.

I hope that my programming of this solution technique for integral equations will allow others to persue more theoretical investigations by crafting a tool to evaluate integral equations with greater ease.

# Bibliography

[1] Anselone, P. M., Editor, "Nonlinear Integral Equations," The University Of Wisconsin Press, Madison, 1964.

[2] Anselone, P. M., and R. E. Moore, "An Extension of the Newton-Kantorovich Method for Solving Nonlinear Equations with an Application to Elasticity", Journal of Mathematical Analysis and Applications, 1966.

[3] Baker, C. T. H. "The Numerical Treatment Of Integral Equations," Oxford University Press, Oxford England, 1977.

[4] Davis, J. "The Solution of Nonlinear Operator Equations with Critical Points", Tech. Report No. 25, Dept. of Mathematics, Oregon State University, Corvallis, Oregon, 1966.

[5] Groetch, C. W. "Elements Of Applicable Functional Analysis," Marcel Dekker, New York, 1980.

[6] Kantorovich, L. V. and G. P. Akilov, "Functional Analysis in Normed Spaces," D. E. Brown, Trans., The Macmillian Co., New York, 1964.

[7] Kolmogorov, A. N. and S. V. Fomin, "Introductory Real Analysis," R. A. Silverman, Trans., Dover Publications, New York, 1970.

[8] Rall, L. B. "Computational Solution Of Nonlinear Operator Equations," John Wiley and Sons, New York, 1969.

[9] Winter, L. T. "Computer Techniques Yielding Automatic and Rigerous Solutions to Linear and Nonlinear Integral Equations", Tech. Report No. 50, Dept. of Mathematics, Oregon State University, Corvallis, Oregon, 1976.

# APPENDICES

# Appendix A. Subroutine Headings

The following are the procedure headings from the Integral Equation solution subroutines. They all in one place, to make finding the appropriate calling sequence easier.

```
PROCEDURE MAKE_DOMAIN(    VAR   s : vector;
                              a,b : extended;
                                n : integer);

PROCEDURE MAKE_Y_FUNC(    VAR y : vector;
                              s : vector);

PROCEDURE FUNC_X(    VAR out : x_vect;
                       at_s : extended;
                     VAR   x : vector;
                     VAR   s : vector);

PROCEDURE FUNC_EVAL(    VAR out : vector;
                        VAR   x : vector;
                        VAR   s : vector);

PROCEDURE MATRIX_EVAL(    VAR out : matrix;
                          VAR   x : vector;
                          VAR   s : vector);

FUNCTION KANTOROVICH(          eta_0 : extended;
                       VAR      lu : matrix;
                       VAR       p : ivector;
                       VAR       x : vector;
                       VAR       s : vector);
```

```
PROCEDURE NEWTON_METHOD(   VAR    err_est : extended;
                           VAR          x : vector;
                           VAR      error : integer;
                               max_iter : integer;
                                    use : boolean;
                           VAR          y : vector;
                           VAR          s : vector);

PROCEDURE HOMOTOPY(  VAR          x : vector;
                          in_delta : extended;
                         min_delta : extended;
                       max_gamma : extended;
                     VAR          y : vector;
                     VAR          s : vector);
```

# Appendix B. Outline of Required Steps

This appendix is a concise outline for the solution of integral equations using the subroutines described in this paper.

1). Define program constants.

    A). Arsize.

    B). Func_dimen.

2.) Include the file "matrix.pas".

3.) Newton's method.

    A). Define variables or constants

        1). Lu_epsilon.

        2). Lambda.

        3). Newton_iter.

        3). Num_eqn.

    B). Define variables.

        1). Newton_err.

        2). F_prime.

        3). F_prime_inv_norm.

    C). Include the file "inteqn.pas".

4). Practical Homotopy, may be skipped if only using Newton's method.

    A). Define variables or constants

1). Newton_iter.

B). Define variables.

1). Gamma.

C). Include the file "homotopy.pas".

5). Procedures to be written.

A). Needed for Newton's method.

1). Cap_F.

2). D_Cap_F.

3). DD_Cap_F.

4). K.

5). Y_FUNC.

B). Needed for practical homotopy, may be skipped if using only Newton's method.

1). HOMOTOPY_OK_OUT.

2). HOMOTOPY_ERR_OUT.

3). Cap_FDG.

6). Write the main procedure.

A). Make_domain.

B). Make_y_func.

C). Make an initial guess, $x_0$.

D). Call a solution function:

1). Newton_method.

2). Homotopy.

# Appendix C. Example Problem Code Listing

The following is an example of the use of the integral equation solution package.

```
program integral_equation(output);
        (* written by Henry Tieman *)


const
        arsize = 50;                    (* define a maximum N for matrix sizes *)
        func_dimen = 2;                 (* the dimension of the function x() *)
        newton_err = 1e-6;              (* min error tolerated by newton procedure *)
        newton_iter = 50;               (* max number of iterations for newtons method *)
        lu_epsilon = 1e-8;              (* singular check in lu decomp *)

        lambda = 0.5;                   (* lambda - number in front of the integral sign *)

{$I matrix.pas }                        (* include file with matrix routines *)
                                        (* also define types vector, ivector, matrix *)

var
        num_eqn : integer;              (* the number of equations in the matrix *)
        f_prime : matrix;               (* here because of memory limitations *)
        f_prime_inv_norm : extended; (*the norm of f^ (-1) *)

{$I int_eqn.pas }                       (* include file to solve integral equations *)
                                        (* also define types x_vect, square, cube *)
                                        (* requires the previous definition of the *)
                                        (* variables : *)
```

```
                                (* f_prime : matrix *)
                                (* num_eqn : integer*)

var
        gamma : extended;       (* pressure *)

{$I homotopy.pas}               (* include the practical homotopy method *)

var
        mu : extended;          (* thinness of plate *)


procedure k(    var val  : x_vect;
                    s, t  : extended);
var
        a,b : extended;
begin
        if ( s < t ) then begin
                a := t; b := s; end
        else begin
                a := s; b := t; end;
        if ( a = b) and (a = 0) then
                begin
                val[1] := 0;
                val[2] := 0;
                end
        else
                begin
                val[1] := b*(1/a - a);
                val[2] := b*(1/a + 2*a);
```

```
            end;
end;

procedure cap_f(   var val  : x_vect;
                       i    : integer;
                   var  x   : vector;
                   var  s   : vector);
(* function to be evaluated at a particular t *)
var
      loc_1, loc_2 : integer;
begin
      loc_1 := 2 * i - 1;
      loc_2 := 2 * i;
      val[1] := mu * x[loc_2] * (s[i] - gamma * mu * x[loc_1]);
      val[2] := mu * x[loc_1] * ( -s[i] + 0.25 * gamma * mu * x[loc_1]);
end;

procedure d_cap_f(   var d_f  : square;
                         i    : integer;
                     var  x   : vector;
                     var  s   : vector);
var
      loc_1, loc_2 : integer;
      t : extended;
begin
      loc_1 := 2 * i - 1;
      loc_2 := 2 * i;
      d_f[1,1] := -gamma * mu * mu * x[loc_2]; (* df1/dx1 *)
      d_f[1,2] := mu * ( s[i] - gamma*mu*x[loc_1]); (* df1/dx2 *)
      t := 0.5 * gamma * mu * x[loc_1];
      d_f[2,1] := mu * ( -s[i] + t);        (* df2/dx1 *)
```

```
        d_f[2,2] := 0;                          (* df2/dx2 *)
end;

procedure dd_cap_f(   var dd_f  : cube;
                          i     : integer;
                      var   x   : vector;
                      var   s   : vector);
var
        loc_1, loc_2 : integer;
begin
        loc_1 := 2 * i - 1;
        loc_2 := 2 * i;
        dd_f[1,1,1] := 0;                       (* ddf1/dx1dx1 *)
        dd_f[1,1,2] := -gamma * mu * mu;(* ddf1/dx1dx2 *)
        dd_f[1,2,1] := -gamma * mu * mu;(* ddf1/dx2dx1 *)
        dd_f[1,2,2] := 0;                       (* ddf1/dx2dx2 *)
        dd_f[2,1,1] := 0.5 * gamma * mu * mu;   (* ddf2/dx1dx1 *)
        dd_f[2,1,2] := 0;                       (* ddf2/dx1dx2 *)
        dd_f[2,2,1] := 0;                       (* ddf2/dx2dx1 *)
        dd_f[2,2,2] := 0;                       (* ddf2/dx2dx2 *)
end;

procedure y_func(   var val  : x_vect;
                        s    : extended);
begin
        val[1] := 0.25 * s * ( 1 - s * s );
        val[2] := 0;
end;
```

```
procedure cap_fdg(  var val  : x_vect;
                        i    : integer;
                    var  x   : vector;
                    var  s   : vector);
(* derivative of function F with respect to gamma *)
var
      loc_1, loc_2 : integer;
      t : extended;
begin
      loc_1 := 2 * i - 1;
      loc_2 := 2 * i;
      val[1] := mu * x[loc_2] * (s[i] - gamma * mu * x[loc_1]);
      t := gamma * mu * x[loc_1];
      val[2] := mu * x[loc_1] * ( -s[i] + 0.25 * t);
end;

procedure homotopy_err_out(  gamma  : extended;
                             why    : integer);
var
      msg : string;
begin
      write('gamma = ');
      str(gamma:10:6,msg);
      writeln('error: no');
      writeln(why);
end;

procedure homotopy_ok_out(      gamma   : extended;
                                err_est : extended;
                           var      x   : vector;
                           var      s   : vector);
```

```
var
        msg : string;
        ev_x : x_vect;
begin
        write('gamma = ');
        str(gamma:10:6,msg);
        write(msg);
        write(' - || f || = ');
        str(f_prime_inv_norm:10:6,msg);
        write(msg);
        write(' - err bnd = ');
        str(err_est:12,msg);
        write(msg);
        write(' - x = ');
        func_x(ev_x,0.5,x,s);
        write(ev_x[1]:12:8);
        writeln(ev_x[2]:12:8);
end;
(* Begin Main Program *)
var
        x : vector;
        y : vector;
        s : vector;
        i : integer;
begin
        num_eqn := 20;
        mu := 4.5;
        make_domain(s, 0, 1, num_eqn);
        make_y_func(y, s);
(* sorry but making the guess is kind of tough *)
```

```
        for i := 1 to num_eqn do
                begin
                x[(i-1)*func_dimen+1] := sin(pi*(i-1)/num_eqn) / 100;
                x[(i-1)*func_dimen+2] := (i-1)/100;
                end;
        homotopy(x, 0.05, 1e-6, 6, y, s);
end.
```