AN ABSTRACT OF THE THESIS OF

Lynn Taylor Winter for the degree of Master of Science

in Computer Science presented on NOVEMBER 4, 1976

Title: COMPUTER TECHNIQUES YIELDING AUTOMATIC AND RIGOROUS SOLUTIONS TO LINEAR AND NONLINEAR INTEGRAL EQUATIONS

Redacted for privacy

Abstract approved: _____ Joel Davis

Interval arithmetic is applied to the problem of obtaining rigorous solutions to integral equations on a computer. The integral equations considered are the linear Fredholm equation of the second kind and the nonlinear Urysohn equation. Techniques are presented which enable the computer to find an approximate solution, prove the existence of an exact solution and bound the discrepancy. These techniques were implemented as computer programs and the results from test problems are given.

COMPUTER TECHNIQUES YIELDING AUTOMATIC AND RIGOROUS SOLUTIONS TO LINEAR AND NONLINEAR INTEGRAL EQUATIONS

by

Lynn Taylor Winter

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

MASTER OF SCIENCE

June 1977

APPROVED:

Redacted for privacy

Associate Professor of Mathematics
in charge of major

Redacted for privacy

Chairman of Department of Computer Science

Redacted for privacy

Dean of Graduate School

Date thesis is presented NOVEMBER 4, 1976

Typed by Leona M. Nicholson for Lynn Taylor Winter

ACKNOWLEDGEMENT

I take this opportunity to thank Dr. Davis for the time and effort that he has devoted to this project. His suggestions, encouragement and enthusiasm all played key roles in the final result. Also, I have taken several valuable background courses from Dr. Davis and found his teaching to be excellent and his concern for the students to be genuine.

I would also like to thank my wife, Joan, who never doubted that my efforts were worthwhile.

TABLE OF CONTENTS

COMPUTER TECHNIQUES YIELDING AUTOMATIC AND RIGOROUS SOLUTIONS TO LINEAR AND NONLINEAR INTEGRAL EQUATIONS

CHAPTER 1

INTRODUCTION

The rigorous numerical solution of a mathematical problem often involves two aspects. One is a computational scheme which yields an approximate solution to the problem. The other being an analytical expression which serves to bound the error between the approximate solution and an exact solution. When using a computer to carry out the calculations a third aspect is introduced; namely, the problem of round-off error which is present with computer arithmetic. Usually, both evaluating the error bound and allowing for round-off error present significant difficulties.

The above situation is encountered in the numerical solution of integral equations. In [1], P. M. Anselone presents an approximation theory which is applicable to linear Fredholm integral equations of the second kind. Yungen [11] applied this theory to specific problems and obtained rigorous solutions. Hand computation was used to evaluate the error bound. Round-off error was allowed for by analyzing the accuracy of computer arithmetic and then bounding the error in key operations such as inner products. Bryan [2] gives a numerical procedure applicable to Urysohn integral equations. An

example was given where the error bound was computed by hand. Round-off error was not mentioned.

The goal of this thesis is to further automate the rigorous solution of these integral equations. Techniques are found which enable the computer to evaluate the error bound and to bound its own round-off error. These techniques employ interval arithmetic for which the main reference is Moore [8].

Chapter 2 deals with interval arithmetic and those applications which were found to be relevant. In particular, Section 2.2 contains the description of a computer implementation of interval arithmetic which was programmed for this thesis.

Chapter 3 deals with the Fredholm equation. With this problem, computation of the error bound presents the greatest difficulty. In this regard, it is found that the computer allows use of a sharper bound than the one which was practical for hand calculation. Results on test problems are given in Section 3.5.

Chapter 4 deals with the Urysohn equation. The main tools are Newton's method and a theorem of Kantorovich concerning the existence of solutions to nonlinear equations. It is found that the hypothesis of this important theorem can be verified automatically. Section 4.5 contains results from two test problems.

CHAPTER 2

INTERVAL ARITHMETIC

2.1 Exact and Rounded Interval Arithmetic

Interval arithmetic is designed for computation with quantities for which an exact value is not available. An interval of values which contains the exact value is used instead. This section deals with intervals of numbers and their arithmetic. Later sections introduce interval vectors, matrices and functions. A superscript I will be used to distinguish interval quantities from their ordinary counterparts.

The set of intervals consists of all subsets of the real numbers which are both compact and connected. An interval can be specified by giving its left and right endpoints. For example, if $a^I = \{x : a^- \leq x \leq a^+\}$, then the notation $[a^-,a^+]$ will be used for a^I. A real number, say x, can be considered as the interval [x,x] and so the set of intervals can be considered as an extension of the real numbers. The fact that a real number x lies in an interval a^I will be expressed by $x \in a^I$. The fact that an interval b^I is contained in an interval a^I will be expressed by $b^I \subset a^I$, where the possibility that $b^I = a^I$ is allowed.

The following functions of intervals are convenient:

(a) Center: $c([a^-,a^+]) = (a^-+a^+)/2$.

(b) Width: $w([a^-,a^+]) = a^+ - a^-$.

(c) Norm: $\|[a^-,a^+]\|^I = \max\{|a^-|,|a^+|\}$.

(d) Absolute Value:

$$|[a^-,a^+]|^I = \begin{cases} [a^-,a^+] & a^- \geq 0, \\ [-a^+,-a^-] & a^+ \leq 0, \\ [0, \max\{-a^-,a^+\}] & \text{otherwise.} \end{cases}$$

Notice that the center, width and norm functions are real valued while the absolute value yields an interval. In fact,

$$|[a^-,a^+]|^I = \{|x| : a^- \leq x \leq a^+\}.$$

Interval arithmetic operations are defined so that the interval result equals the set of all possible results obtained by using one real value from each operand. Let a^I and b^I be intervals and let o be one of the operations $+$, $-$, $\cdot$, $/$, then the interval arithmetic operations are defined by $a^I o b^I = \{xoy : x\varepsilon a^I \text{ and } y\varepsilon b^I\}$, except that a^I/b^I is undefined if $0\varepsilon b^I$. The operation of negation is defined by $-a^I = \{-x : x\varepsilon a^I\}$.

It follows from the definition that addition and multiplication are associative and commutative. The additive identity is $0 = [0,0]$ and the multiplicative identity is $1 = [1,1]$. The distributive law does not hold in general; however, interval arithmetic has the property that $a^I \cdot (b^I + c^I) \subset a^I \cdot b^I + a^I \cdot c^I$, which is called subdistributivity.

Interval arithmetic is inclusion monotonic in the sense that, if $a^I \subset c^I$ and $b^I \subset d^I$, then $a^I \; o \; b^I \subset c^I \; o \; d^I$ where o represents

+ , - , · , or / . The following proposition is a consequence of this monotonicity.

Proposition 2.1.1. If f^I is a rational expression in the interval variables $x_1^I, x_2^I, \ldots, x_n^I$, then $y_1^I \subset x_1^I$, $y_2^I \subset x_2^I, \ldots, y_n^I \subset x_n^I$ implies that $f^I(y_1^I, y_2^I, \ldots, y_n^I) \subset f^I(x_1^I, x_2^I, \ldots, x_n^I)$ for every set of intervals $x_1^I, x_2^I, \ldots, x_n^I$ for which the operations in f^I are defined.

In the above proposition, the operations indicated in f^I are to be carried out in a specified order using interval arithmetic. A frequent and important application of this proposition occurs in the evaluation of a rational function, say $f(x_1, x_2, \ldots, x_n)$. Suppose that the value of f is desired at $(y_1, y_2, \ldots, y_n)$, but that it is known only that $y_i \in Y_i^I$ for $i = 1,2,\ldots,n$. By the above proposition, an interval, a^I, containing $f(y_1, y_2, \ldots, y_n)$ can be found by replacing y_i with Y_i^I, $i = 1,2,\ldots,n$, in the expression for f and then using interval arithmetic to carry out the specified operations. In fact, $\{f(y_1, y_2, \ldots, y_n) : y_i \in Y_i^I, \ i = 1,2,\ldots,n\} \subset a^I$.

For actual computation of interval arithmetic operations the following formulas are used:

Let $a^I = [a^-, a^+]$ and $b^I = [b^-, b^+]$,

$$
\begin{aligned}
a^I + b^I &= [a^- + b^- \ , \ a^+ + b^+] \\
-a^I &= [-a^+ \ , \ -a^-] \\
a^I - b^I &= a^I + (-b^I) \\
a^I \cdot b^I &= [\min S \ , \ \max S]
\end{aligned}
\qquad (2.1\text{-}1)
$$

where $S = \{a^- b^-, a^- b^+, a^+ b^-, a^+ b^+\}$

$$a^I / b^I = [\min S' , \max S']$$

where $S' = \{a^-/b^-, a^-/b^+, a^+/b^-, a^+/b^+\}$

This interval arithmetic, which will be called exact, cannot be programmed directly on a computer. The presence of round-off error in computer arithmetic means that the above formulas would not yield the exact endpoints for the result. However, by analyzing the accuracy of the arithmetic operations on a given machine, it may be possible to "round" the machine computed endpoints in such a way as to insure containment of the exact interval result. This machine dependent interval arithmetic will be called rounded interval arithmetic (RIA).

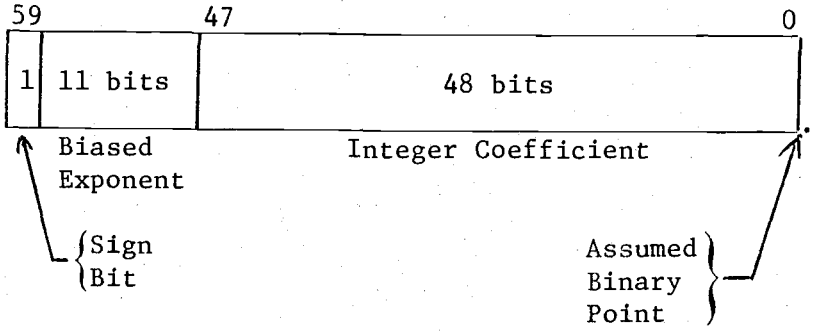
Suppose that a^I and b^I are intervals and that o indicates one of the four arithmetic operations. The essential condition of RIA is that $a^I o b^I \subset a^I \tilde{o} b^I$ where $\tilde{o}$ indicates the use of RIA. It follows from this monotonicity that, if f^I is a rational expression in the interval variables $x_1^I, x_2^I, \ldots, x_n^I$, then $f^I(x_1^I, x_2^I, \ldots, x_n^I) \subset \tilde{f}^I(x_1^I, x_2^I, \ldots, x_n^I)$ where $\tilde{f}^I$ indicates the use of RIA in the evaluation of f^I.

2.2 A Computer Implementation of RIA

Subroutines to perform RIA were written for a CONTROL DATA CYBER 70 - MODEL 73 digital computer. These subroutines were written in COMPASS and designed for use in FORTRAN programs. Appendix A contains information on how this RIA is called from FORTRAN. Under this implementation, machine intervals comprise all intervals which

can be formed using normalized single precision floating-point numbers as endpoints. The endpoint calculations utilize unrounded single precision floating-point arithmetic together with a special rounding procedure.

Floating-point number representation allows numbers of the form $p \cdot 2^n$ to be expressed exactly in the computer as long as the magnitude of the integer coefficient, p, or the integer exponent, n, is not too large. On the CYBER, single precision floating-point numbers occupy one 60-bit computer word in the following format:



Positive numbers are packed in this format with their true coefficient, with bit 59 set to zero, and with positive exponents biased by adding 1024 and negative exponents biased by adding 1023. Negative numbers are packed by first packing their absolute value and then complementing all 60 bits. A nonzero number is normalized if bit 47 is different from bit 59. The normalized form of zero is all zero bits.

The range of exponents allowed by this format is -1023 to 1023. Exponential overflow is said to occur if the result of an operation has an exponent which is greater than 1023. In the case of

overflow, the machine returns an infinite form which, if used in a subsequent calculation, results in a CPU error exit. Exponential underflow occurs when the result of an operation has an exponent which is less than -1023. In this case, the machine returns normalized zero without signaling any error condition. Thus, in RIA, overflow can be left for the computer to detect but underflow must be checked for when necessary to keep the arithmetic rigorous.

The unrounded normalize operation and the unrounded single precision add, multiply and divide operations form the basis of RIA. The following information on these operations is based on [3] and verified by experimentation.

(a) Unrounded Normalize Instruction (OCTAL CODE 24):

The operand is a single precision floating-point number. In general, the coefficient is shifted to the left until bit 47 is different from bit 59 and the exponent is decreased by the number of positions shifted. Normalized zero is returned if the coefficient is zero or if exponential underflow occurs. Underflow is recognized when a nonzero operand normalizes to zero. In the absence of underflow, the value of the operand is unchanged by normalization.

(b) Unrounded Single Precision Floating-Point Add Instruction (OCTAL CODE 30):

Suppose that the operands are $A = p \cdot 2^n$ and $B = q \cdot 2^m$ with $n \geq m$. The coefficient of A is entered in the upper 48 bits of a 96-bit register. Next, the coefficient of B is entered in the top half of

another 96-bit register and shifted to the right n-m bits which serves to align the binary points. In the shifting process, part or all of the coefficient of B can be lost off the right end of the double length register. The contents of the two 96-bit registers are then added. If overflow occurs on this addition then the entire sum is right shifted one bit and the exponent is increased to n+1. The upper 48 bits are then returned as the coefficient of the sum and the exponent is either n or n+1. Exponential underflow cannot occur, however the result may not be normalized.

(c) Unrounded Single Precision Floating-Point Multiply Instruction (OCTAL CODE 40):

Suppose that the operands are $A = p \cdot 2^n$ and $B = q \cdot 2^m$ and assume that A and B are nonzero and normalized. (If either operand is zero then normalized zero is returned as the product.) The 48-bit integer coefficients p and q are multiplied forming a 96-bit product with the assumed binary point at the right. The exponent of this product is n+m. Since A and B were normalized, the bit structure of $p \cdot q$ is either 1xx...x. or 01xx...x. . If the bit structure is 1xx...x., then the upper 48 bits are returned as the coefficient with exponent n+m+48. The addition of 48 is needed to move the binary point left 48 positions. If the bit structure is 01xx...x., then the 48 bits starting with the one are returned as the coefficient with exponent n+m+47. The binary point only needs to be moved 47 positions in this case. The result is always normalized but exponential underflow or overflow can occur.

(d) Unrounded Single Precision Floating-Point Divide Instruction (OCTAL CODE 44):

Suppose that the operands, $A = p \cdot 2^n$ and $B = q \cdot 2^m$, are nonzero normalized floating-point numbers and that A/B is performed. (If A = 0, then normalized zero is returned.) Long division takes place until 49 bits of the quotient p/q are generated. Since A and B were normalized, the quotient has either the form 1.xx...x or 0.1xx...x with exponent n-m. If the form is 1.xx...x, then the upper 48 bits are returned as the coefficient and the exponent is n-m-47. If the form is 0.1xx...x, then the lower 48 bits are returned as the coefficient and the exponent is n-m-48. The result is always normalized, but exponential underflow or overflow can occur.

Using the above information, it is possible to analyze the relationship between the machine computed value and the exact value of an arithmetic operation performed on normalized operands. The usual situation is that the machine computed value is the exact value truncated at the assumed binary point. By this it is meant that, if the machine computed value is $p \cdot 2^n$ (p a binary integer), then the exact value has the form $(p.xx...) \cdot 2^n$ where .xx... represents a binary fraction. Thus, if $p \geq 0$, then the exact value lies in the interval $[p \cdot 2^n, p \cdot 2^n + 2^n]$ and if $p < 0$, then the exact value lies in the interval $[p \cdot 2^n - 2^n, p \cdot 2^n]$.

The add operation on normalized operands yields a truncated exact answer in most cases. An exception occurs when the operands have opposite signs and exponents which differ by 96 or more. Since

the coefficient of the number with the smaller exponent is shifted completely off and lost during the addition process the machine returned value, say $p\cdot 2^n$, is exactly the operand with the larger exponent. Hence, if $p > 0$, then the exact value lies in the interval $[p\cdot 2^n - 2^n,\ p\cdot 2^n]$ and if $p < 0$, then the exact value lies in the interval $[p\cdot 2^n,\ p\cdot 2^n + 2^n]$. In the absence of exponential underflow or overflow, the multiply and divide operations always yield a truncated exact answer.

Regardless of the sign of p, if $A = p\cdot 2^n$ is a machine number, then the numbers $p\cdot 2^n + 2^n$ and $p\cdot 2^n - 2^n$ can be formed exactly in the computer. Forming $p\cdot 2^n + 2^n$ will be called rounding A up while forming $p\cdot 2^n - 2^n$ will be called rounding A down. The four RIA operations will now be described. Assume that formulas (2.1-1) have been used with unrounded single precision floating-point arithmetic.

(a) Rounded Interval Addition:

First certain special cases are considered. Let $[A^-, A^+]$ and $[B^-, B^+]$ be the machine interval operands. If $A^- = 0$ or $B^- = 0$, then the other one is returned as the left endpoint. Similarly for the right endpoint if $A^+ = 0$ or $B^+ = 0$. If $A^- = -B^-$, then zero is returned as the left endpoint and if $A^+ = -B^+$, then zero is returned as the right endpoint. In the absence of such cases, the following procedures are used.

Negative left endpoints and positive right endpoints are rounded down and up, respectively, and then normalized. It is

considered a fatal error if underflow occurs during normalization.

When negative right endpoints and positive left endpoints are truncated versions of the exact endpoint, they are simply normalized. In the exceptional case when the operands have opposite signs and exponents that differ by 96 or more, they are rounded up and down, respectively, and then normalized. Underflow in these cases is ignored.

(b) Rounded Interval Subtraction:

This is accomplished by negation, which is done exactly in the computer, followed by rounded interval addition.

(c) Rounded Interval Multiplication:

Endpoints which are zero are either the exact answer or underflow has occurred. In the first case they are left as zero with the underflow case treated as below.

Negative left endpoints and positive right endpoints are rounded down and up, respectively. It is considered a fatal error if underflow occurs on the calculation of a left endpoint which would be negative or on the calculation of a right endpoint which would be positive if exact arithmetic were used.

Negative right endpoints and positive left endpoints are left unchanged. Underflow is ignored on the calculation of left endpoints which would be positive or on the calculation of right endpoints which would be negative if exact arithmetic were used.

(d) Rounded Interval Division:

A fatal error is signaled if the divisor contains zero. Otherwise, the same rounding rules are used as with rounded interval multiplication.

Another source of error encountered on the computer occurs in the form of input-output errors which may occur during binary-decimal conversions. In the programs of this thesis, these errors must be compensated for by the user.

2.3 Interval Extensions of Functions

Suppose that f is a real valued function defined on the interval a^I. Depending on the nature of f there may be several problems associated with finding the exact value of f(t), $t \varepsilon a^I$, on a computer. For one thing, t might not be known or representable exactly and hence might naturally be replaced by an interval containing t. Another problem is that f might involve arithmetic operations leading to round-off errors or might involve irrational functions whose value can only be approximated on the computer.

To deal with these problems, the concept of interval extension for functions is introduced. An interval valued function f^I will be called an <u>interval extension</u> of f on a^I if, for every interval $b^I \subset a^I$, $f^I(b^I)$ is defined and $\{f(t) : t \varepsilon b^I\} \subset f^I(b^I)$. This concept extends to real functions of more than one variable by allowing each argument to range in an interval. For example, suppose that g is a real valued function defined on the product $a_1^I \times a_2^I$. The interval valued function

g^I is an interval extension of g if $g^I(b_1^I,b_2^I)$ is defined for all pairs b_1^I, b_2^I such that $b_1^I \subset a_1^I$ and $b_2^I \subset a_2^I$ and

$\{g(t_1,t_2) : t_1 \epsilon b_1^I \text{ and } t_2 \epsilon b_2^I\} \subset g^I(b_1^I,b_2^I)$.

Two problems will now be considered. The first is the problem of finding an f^I for a given f and the second is the problem of finding the range of values of f, over an interval, given f^I. For more information on both problems, see Moore [8].

If f is a continuous rational function on a^I, then, as mentioned following Proposition 2.1.1, an interval extension of f can be found by direct use of exact interval arithmetic. If the constants appearing in f are machine numbers or are replaced by containing machine intervals, then RIA can be used. As an example, consider $f(t) = t(1-t)$ on $a^I = [0,1]$. The interval function f^I defined by $f^I(b^I) = b^I \cdot ([1,1]-b^I)$ for all $b^I \subset a^I$ is an interval extension of f using either exact or rounded interval arithmetic. This approach for obtaining f^I has the disadvantage that the interval $f^I(b^I)$ can be significantly larger than $\{f(t) : t \epsilon b^I\}$. This occurs in this example where the exact range of values of f on a^I is $[0,\frac{1}{4}]$ but $f^I(a^I) = [0,1]$.

If f is known to be monotone on a^I, then in theory it is easy to obtain an interval extension which gives the exact range of values of f. For example, if f is monotone increasing and $b^I = [b^-,b^+] \subset a^I$ then define $f^I(b^I) = [f(b^-),f(b^+)]$. In practice, suppose that f_1^I is an interval extension of f obtained by the direct method of the previous paragraph. Then the intervals $c^I = f_1^I([b^-,b^-])$ and $d^I = f_1^I([b^+,b^+])$ can be computed using RIA and will contain $f(b^-)$ and

$f(b^+)$, respectively. Now define $f^I(b^I)$ = [left endpt. of c^I, right endpt. of d^I]. Since the widths of $[b^-,b^-]$ and $[b^+,b^+]$ are zero, it is likely that the widths of c^I and d^I will be small and hence that a useful interval extension will be found. This method can be adapted for use with piecewise monotonic f provided that a^I is subdivided into intervals where f is either increasing or decreasing.

Using monotonicity and a computer subroutine of known accuracy, it may also be possible to find interval extensions for certain irrational functions. As an example, consider the exponential function $f(t) = e^t$ which is monotone increasing. Suppose that the computer subroutine for the exponential function is denoted by EXP and is known to have a relative error of less than 2^n. Then $EXP(t) - 2^n \cdot EXP(t) \le e^t \le EXP(t) + 2^n \cdot EXP(t)$ for all t in the domain of EXP. Thus, an interval extension e^I of e can be defined by $e^I([b^-,b^+]) = [1-2^n , 1+2^n] \cdot [EXP(b^-) , EXP(b^+)]$. Information on the accuracy of many of the library functions for the CYBER can be found in the Fortran Mathematical Library Documentation available at the Oregon State University Computer Center.

Next, consider the problem of finding, as closely as possible, the range of values of f on an interval a^I given an interval extension f^I. By definition of an interval extension, the interval $f^I(a^I)$ contains the set $S = \{f(t) : t \in a^I\}$, but nothing is known about the "closeness" of $f^I(a^I)$ and S. In general nothing more can be said, however, the situation illustrated by the example $f(t) = t(1-t)$ and $f^I(b^I) = b^I \cdot ([1,1]-b^I)$ is fairly common. In this case it is possible

to improve the bound on the range of values of f. This is done by subdividing a^I and taking the union of the images of f^I on each subinterval. As an illustration, consider this example with a^I subdivided by $[0,\frac{1}{2}] \cup [\frac{1}{2},1]$. Then the range of values of f on a^I is contained in $f^I([0,\frac{1}{2}]) \cup f^I([\frac{1}{2},1]) = [0,\frac{1}{2}]$. This is an improvement over $f^I(a^I) = [0,1]$, but is still larger than the actual range of values. When f is a continuous rational function and f^I is the interval extension formed by direct use of exact interval arithmetic, then Moore [8] proves that, by subdividing the domain into sufficiently fine subintervals, it is possible to obtain an interval arbitrarily close to the actual range of values of f. In the applications to come later, the relationship between f and f^I will not be known and this subdivision technique will be used in an empirical way.

As seen above, interval numbers motivated the definition of interval extension for functions whose domains and ranges were sets of numbers. Now suppose that T is an operator such that, if f is a real function, then T(f) is again a real function. It is now possible to define interval extensions for such T. An interval extension T^I of T is an operator such that for any interval extension f^I of f it is the case that $T^I(f^I)$ is an interval extension of T(f). As a simple example, an interval extension of the identity operator, I(f) = f, would be the operator $I^I(f^I) = f^I$. The concept of interval extension for operators can be applied to any operator whose domain and range have interval extensions. An example where the domain and range are spaces of vectors occurs in the next section.

The following specific properties of interval extensions of real functions and operators will be needed later. Suppose that f_1 and f_2 are real valued functions defined on the interval a^I. Let f_1^I and f_2^I be interval extensions for f_1 and f_2 and define $(f_1^I + f_2^I)(b^I) = f_1^I(b^I) + f_2^I(b^I)$ for all $b^I \subset a^I$, the last sum being an interval sum.

Proposition 2.3.1. With f_1, f_1^I, f_2 and f_2^I as above, it follows that $f_1^I + f_2^I$ is an interval extension of $f_1 + f_2$.

Proof: Let b^I be any interval contained in a^I. By definition, $(f_1^I + f_2^I)(b^I)$ is defined. The containments,

$$\{(f_1 + f_2)(t) : t\epsilon b^I\} = \{f_1(t) + f_2(t) : t\epsilon b^I\}$$

$$\subset \{f_1(t) : t\epsilon b^I\} + \{f_2(t) : t\epsilon b^I\}$$

$$\subset f_1^I(b^I) + f_2^I(b^I) = (f_1^I + f_2^I)(b^I) ,$$

complete the proof that $f_1^I + f_2^I$ extends $f_1 + f_2$.

Now suppose that T_1 and T_2 are operators which map the set of real valued functions defined on a^I into itself. Let T_1^I and T_2^I be interval extensions for T_1 and T_2 and define $(T_1^I + T_2^I)(f^I) = T_1^I(f^I) + T_2^I(f^I)$ where f^I extends a real valued function f on a^I.

Proposition 2.3.2. Let T_1, T_1^I, T_2 and T_2^I be as above. Then the interval operator $T_1^I + T_2^I$ extends $T_1 + T_2$.

Proof: Let f be any real valued function on a^I and let f^I be an interval extension. It must be shown that $(T_1^I + T_2^I)(f^I) = T_1^I(f^I) + T_2^I(f^I)$ extends $(T_1 + T_2)(f) = T_1(f) + T_2(f)$. This is true by

Proposition 2.3.1 since $T_1^I(f^I)$ extends $T_1(f)$ and $T_2^I(f^I)$ extends $T_2(f)$.

Another situation which will occur is the composition of interval extensions of certain operators. Suppose that T_1 and T_2 with interval extensions T_1^I and T_2^I are as above. Define $(T_1^I T_2^I)(f^I) = T_1^I(T_2^I(f^I))$ whenever $T_1(T_2(f))$ is defined.

Proposition 2.3.3. In the above setting, $T_1^I T_2^I$ extends $T_1 T_2$.

Proof: By definition, $T_2^I(f^I)$ extends $T_2(f)$ and again by definition $T_1^I(T_2^I(f^I))$ extends $T_1(T_2(f))$. Hence, $(T_1^I T_2^I)(f^I)$ extends $(T_1 T_2)(f)$ completing the proof.

2.4 Interval Matrix Inversion

The problem discussed in this section is the rigorous solution of an equation of the form $A \cdot \bar{x} = \bar{b}$ where A is a real n x n matrix and $\bar{b}$ is a real n-dimensional vector. If it is allowed that the elements of A and $\bar{b}$ are not known exactly, then the following definitions are natural. An interval matrix or interval vector is a matrix or vector whose elements are intervals. Arithmetic operations between interval vectors and matrices are assumed to follow the usual rules except that exact or rounded interval arithmetic replaces real arithmetic. Also, containment is understood to be componentwise.

Let A^I be an interval matrix such that $A \varepsilon A^I$ and let $\bar{b}^I$ be an interval vector such that $\bar{b} \varepsilon \bar{b}^I$. The method given below gives the computer the capability of proving that A^{-1} exists and of finding an

interval matrix, $(A^I)^{-1}$, such that $A^{-1} \epsilon (A^I)^{-1}$. Hence, the computer generated vector, $\bar{x}^I = (A^I)^{-1} \cdot \bar{y}^I$, is known to contain $\bar{x}$. The method is adapted from [6].

In the following chapters, matrices arise in the role of linear operators on finite dimensional vector spaces. A matrix A defines the operator which takes $\bar{x}$ into $A \cdot \bar{x}$. If $\bar{x} \epsilon \bar{x}^I$ and $A \epsilon A^I$, then the interval operator which takes $\bar{x}^I$ into $A^I \cdot \bar{x}^I$ is an interval extension of the operator defined by A in the sense of the previous section. Hence, the interval operator defined by $(A^I)^{-1}$ is an interval extension of the operator defined by A^{-1}.

Let $E = (e_{ij})_{i,j=1}^{n}$ be a real n x n matrix. The matrix norm which will be used is the maximum absolute row sum norm defined by

$$\|E\| = \max_{1 \le i \le n} \sum_{j=1}^{n} |e_{ij}| .$$

If $\|E\| < 1$, then it is known that $(I-E)^{-1}$ exists and $\|(I-E)^{-1} - S_m\| \le \|E\|^{m+1}/(1 - \|E\|)$ where $S_m = I + E + E^2 + \cdots + E^m$. If a matrix A has either a right or left inverse, then A is invertible. That is, if there exists a matrix B such that AB = I or BA = I, then A^{-1} exists and $A^{-1} = B$. Using these facts, the method for obtaining $(A^I)^{-1}$ can be derived.

Let B be a real matrix which is an approximate inverse to the real matrix whose elements are the centers of the elements of A^I. Define E = I - AB. If $\|E\| < 1$, then the above mentioned facts show that A^{-1} exists and equals $B(I - E)^{-1}$. Thus the problem of finding

A^{-1} has been exchanged for the problem of finding $(I - E)^{-1}$. For each m, $\|(I - E)^{-1} - S_m\| \leq \|E\|^{m+1}/(1 - \|E\|)$ and, since the norm is the maximum absolute row sum, it follows that an element of S_m cannot differ from the corresponding element of $(I - E)^{-1}$ by more than $p_m = \|E\|^{m+1}/(1 - \|E\|)$. Hence, if P_m^I is the interval matrix with each element equal to $[-p_m, p_m]$, then $(I - E)^{-1}$ is contained in $S_m + P_m^I$ so that $A^{-1} \varepsilon B(S_m + P_m^I)$.

In practice, the matrix A^I (not A) is known explicitly and so the definition of E must be modified. Let I^I and B^I be the matrices I and B considered as interval matrices and define $E^I = I^I - A^I B^I$ where RIA is to be used. Since $A \varepsilon A^I$, it follows that $E \varepsilon E^I = (e_{ij}^I)_{i,j=1}^n$ and hence a bound on $\|E\|$ can be found by evaluating $\|E^I\|^I = \max_{1 \leq i \leq n} \sum_{j=1}^n \|e_{ij}^I\|^I$ with RIA. If $\|E^I\|^I < 1$, then it follows that A^{-1} exists and is contained in $B^I(S_m^I + P_m^I)$ where $S_m^I = I^I + E^I + (E^I)^2 + \cdots + (E^I)^m$ and again RIA is used.

Only the choice of m remains. The value m = 1 was chosen over m = 0 because of the prospect of better results at relatively little cost since the matrix E^I must be computed in either case. Experimentation with m = 1 on the matrices encountered in this thesis showed that sufficient accuracy was obtained. Larger values of m were ruled out as they would require many more arithmetic operations. Thus, $(A^I)^{-1}$ is taken to be $B^I(S_1^I + P_1^I) = B^I + B^I(E^I + P_1^I)$. The matrix B is computed using single precision arithmetic with an inversion scheme based on Gauss elimination as described in [5].

2.5 Integration and Quadrature Errors

In this section, interval arithmetic is applied to the problem of bounding the error in certain numerical integration formulas. Let f be a continuous function defined on the interval [a,b] and denote the Riemann integral of f by $\int_a^b f(t)dt$. For each m assume that [a,b] is the union of $P^I_{m1}, P^I_{m2}, \ldots, P^I_{mm}$ formed as follows. Assume for the moment that a and b are machine numbers. Using single precision arithmetic, machine numbers p_{mi}, i = 1,2,...,m-1, are calculated by the formula $p_{mi} = a + i((b - a)/m)$. Set $p_{m0} = a$ and $p_{mm} = b$. Then the P^I_{mi} are defined by $P^I_{mi} = [p_{m,i-1}, p_{mi}]$, i = 1,2,...,m. Note that each P^I_{mi} is a machine interval and has width approximately equal to (b - a)/m. Let $W^I_{mi} = [p_{mi}, p_{mi}] - [p_{m,i-1}, p_{m,i-1}]$ for i = 1,2,...,m, computed with RIA, so that W^I_{mi} is an interval containing the exact width of P^I_{mi}. If a or b is not a machine number, then it may be possible to put a narrow machine interval $[a^-,a^+]$ around a or $[b^-,b^+]$ around b and still apply the following methods. This would involve extending the integrand to the larger interval $[a^-,b^+]$ while preserving any assumed smoothness. This situation has been allowed for in the programs but will not be discussed further here.

From the definition of Riemann integral, it follows that the value of $\int_a^b f(t)dt$ lies between the upper and lower Riemann sums of f based on any subdivision of [a,b]. Hence, if f^I is an interval extension of f, then for each m, $\int_a^b f(t)dt \in \sum_{j=1}^m f^I(P^I_{mj}) \cdot W^I_{mj}$ where RIA can be used to compute the right hand side. This technique for

obtaining an interval containing the value of an integral will be called the Riemann sum method. In the applications, the integral will appear in a function of the form: $K(s) = \int_a^b k(s,t)dt$. If k^I is an interval extension of k, then the interval function K^I defined by $K^I(s^I) = \sum_{j=1}^m k^I(s^I,P_{mj}^I)\cdot W_{mj}^I$ is easily verified to be an interval extension of K and can be evaluated with RIA.

Next, the problem of bounding quadrature error is treated for the repeated midpoint and Simpson integration rules. Two smoothness hypotheses are considered for the midpoint rule.

(a) A First Order Method:

If L is a Lipschitz constant for f on [a,b], then for the simple midpoint rule it is known that

$$\left| \int_a^b f(t)dt - f(c([a,b]))\cdot w([a,b]) \right| \leq (w([a,b]))^2\cdot L/4.$$

Thus the quadrature error is contained in the interval $((w([a,b]))^2/4)\cdot[-L,L]$. Suppose that L_1^I is an interval valued function defined on all $[a_1,b_1] \subset [a,b]$ and having the property that $L_1^I([a_1,b_1])$ contains an interval $[-L_1,L_1]$ where L_1 is a Lipschitz constant for f on $[a_1,b_1]$. Using the function L_1^I the error in the simple midpoint rule lies in $((w([a,b]))^2/4)\cdot L_1^I([a,b])$. Repeating the midpoint rule on the subdivision $P_{m1}^I, P_{m2}^I, \ldots, P_{mm}^I$ gives the following containment of the quadrature error:

$$\int_a^b f(t)dt - \sum_{j=1}^m f(c(P_{mj}^I))\cdot w(P_{mj}^I) \in \sum_{j=1}^m ((W_{mj}^I)^2/4)\cdot L_1^I(P_{mj}^I).$$

Here RIA can be used to evaluate the right hand side. An interval

extension of f' could be used in place of L_1^I.

(b) A Second Order Method:

If f is twice continuously differentiable on [a,b], then the error in the simple midpoint rule obeys

$$\int_a^b f(t)dt - f(c([a,b]))\cdot w([a,b]) = ((w([a,b]))^3/24)\cdot f''(\xi)$$

where ξ is some point such that $a < \xi < b$. Hence, if $(f'')^I$ is an interval extension of f'', then the error lies in the interval $((w([a,b]))^3/24)\cdot(f'')^I([a,b])$. And for the repeated rule

$$\int_a^b f(t)dt - \sum_{j=1}^m f(c(P_{mj}^I))\cdot w(P_{mj}^I) \in \sum_{j=1}^m ((W_{mj}^I)^3/24)(f'')^I(P_{mj}^I)$$

where RIA can be used to evaluate the right hand side.

(c) A Fourth Order Method:

Assuming f to be four times continuously differentiable on [a,b], the error in Simpson's rule is

$$\int_a^b f(t)dt - (w([a,b])/6)\cdot(f(a) + 4f(c([a,b])) + f(b)) =$$
$$-((w([a,b]))^5/2880)\cdot(f^{(4)})(\xi) \text{ where } a < \xi < b.$$

As above, if $(f^{(4)})^I$ is an interval extension of $f^{(4)}$, then the error made by repeated Simpson's rule lies in the interval

$$\sum_{j=1}^m -((W_{mj}^I)^5/2880)\cdot(f^{(4)})^I(P_{mj}^I)$$

which can be computed with RIA.

These repeated rules are usually expressed in the form $\sum_{j=1}^{n} w_{nj} f(t_{nj})$ where the w_{nj} are called the quadrature weights and the t_{nj} the quadrature points. In the repeated midpoint rule $w_{nj} = w(P_{nj}^I)$, $t_{nj} = c(P_{nj}^I)$ and $n = m$. For repeated Simpson's rule a calculation is necessary to get the points and weights and $n = 2m + 1$. In either case, RIA is used to find the w_{nj} and the t_{nj} and hence intervals of narrow but not necessarily zero width are found. The notation w_{nj}^I and t_{nj}^I will be used for the machine computed intervals which contain the exact points and weights.

In the applications, quadrature errors will be encountered in functions having the forms

$$G(s) = \int_a^b g(s,t)dt - \sum_{j=1}^{n} w_{nj} g(s,t_{nj}) \qquad (2.5\text{-}1)$$

and

$$H(s,u) = \int_a^b h(s,t,u)dt - \sum_{j=1}^{n} w_{nj} h(s,t_{nj},u) \qquad (2.5\text{-}2)$$

where s and u lie in [a,b]. The problem will be to obtain interval extensions for G and H.

First consider the problem for G when the repeated midpoint rule is being used as a first order method. The Lipschitz constant becomes a function of s and hence an interval valued function, say L_2^I, is needed which has the following property: If s^I, $t^I \subset [a,b]$, then for any real number $s \in s^I$ the interval $L_2^I(s^I,t^I)$ contains an interval $[-L_s,L_s]$ where L_s is a Lipschitz constant for g(s,t) for t in t^I.

If the second or fourth order method is being used, then partial derivatives enter the picture. If $f(x_1,x_2,\ldots,x_n)$ is a function

of n variables, then the notation $D_j f$ will be used for the partial derivative of f with respect to the j^{th} argument, x_j. Hence, the function $D_i D_j f$ would be the second partial derivative of f, first with respect to x_j and then with respect to x_i. In the case of the second order method, an interval extension of G can be obtained with an interval extension of $D_2 D_2 g$. An interval extension for $D_2 D_2 D_2 D_2 g$ is needed when the fourth order method is being used.

The problem of finding an interval extension for H is similar to the situation for G. With the first order method, an interval valued function, say L_3^I, is needed which has the property: If s^I, t^I, $u^I \subset [a,b]$, then for any real numbers $s \in s^I$ and $u \in u^I$, the interval $L_3^I(s^I,t^I,u^I)$ contains an interval $[-L_{s,u}, L_{s,u}]$ where $L_{s,u}$ is a Lipschitz constant for h(s,t,u) for t in t^I. The second and fourth order methods require interval extensions for $D_2 D_2 h$ and $D_2 D_2 D_2 D_2 h$, respectively.

CHAPTER 3

THE LINEAR PROBLEM

3.1 Background Concepts, Examples and Results

The integral equations and related equations of this and the next chapter will be viewed as operator equations on Banach spaces. The operators of this chapter are linear while those of the next chapter are nonlinear. An operator equation has the form $Tx = y$ where T and y are assumed known and the problem is to solve for x. An operator T is said to be invertible if there exists an operator T^{-1} such that $T^{-1}(Tx) = T(T^{-1}x) = x$ for all x. Hence, the existence of a unique solution for $Tx = y$ is equivalent to the existence of T^{-1}, in which case $x = T^{-1}y$. This sketch of the setting will now be elaborated upon with some examples of Banach spaces and operators which will be used later. For proofs and more information see Taylor [10]. The scalar field for all linear spaces is assumed to be the real numbers.

Example 3.1.1. Let $C[a,b]$ be the set of continuous real valued functions defined on the interval $[a,b]$. If $f, g \in C[a,b]$, then addition and scalar multiplication are defined by $(f + g)(t) = f(t) + g(t)$ and $(\alpha f)(t) = \alpha f(t)$. The norm will be $\|f\| = \max_{a \le t \le b} |f(t)|$ which makes $C[a,b]$ a Banach space.

Example 3.1.2. Let $\ell^\infty(n)$ be the set of all n-dimensional real vectors. For $\bar{x} = (x_1,x_2,\ldots,x_n)$ and $\bar{y} = (y_1,y_2,\ldots,y_n)$ in $\ell^\infty(n)$ define addition and scalar multiplication by $\bar{x} + \bar{y} = (x_1+y_2, x_2+y_2, \ldots, x_n+y_n)$ and $\alpha\bar{x} = (\alpha x_1, \alpha x_2, \ldots, \alpha x_n)$. The norm on $\ell^\infty(n)$ is $\|\bar{x}\| = \max_{1\le i\le n} |x_i|$ which makes $\ell^\infty(n)$ a Banach space.

Let X and Y represent Banach spaces. An operator $T : X \to Y$ is a linear operator from X to Y if $T(x + y) = T(x) + T(y)$ and $T(\alpha x) = \alpha T(x)$ for all $x, y \in X$ and all scalars α. The set of linear operators from X to Y forms a linear space with the same definition of addition and scalar multiplication as used for functions in C[a,b]. A linear operator $T : X \to Y$ is called bounded if $\sup_{\|x\|\le 1} \|Tx\| < \infty$.

Proposition 3.1.1. The space of bounded linear operators from X to Y is a Banach space with norm defined by

$$\|T\| = \sup_{\|x\|\le 1} \|Tx\| .$$

The notation [X,Y] will be used for the space of bounded linear operators from X to Y. If Y = X, then the notation is shortened to [X]. The next proposition gives some special properties of the operator norm.

Proposition 3.1.2. If X, Y and Z are Banach spaces, then

(a) $\|Tx\| \le \|T\| \cdot \|x\|$ where $T \in [X,Y]$ and $x \in X$.

(b) $\|TS\| \le \|T\| \cdot \|S\|$ where $T \in [Y,Z]$ and $S \in [X,Y]$.

(c) $\|I\| = 1$ where I is the identity operator on any Banach space.

Example 3.1.3. Let $S = \{t_1, t_2, \ldots, t_n\}$ be any set of n points in [a,b]. Then $P_n : C[a,b] \to \ell^\infty(n)$ defined by $P_n(f) = (f(t_1), f(t_2), \ldots, f(t_n))$ is a bounded linear operator with norm one.

Example 3.1.4. The space $[\ell^\infty(n)]$ consists of all linear operators T on $\ell^\infty(n)$. Each such T has a representation as an n x n matrix, $(a_{ij})_{i,j=1}^{n}$, such that Tx can be computed by matrix multiplication. Also, $\|T\|$ can be computed by the formula $\|T\| = \max_{1 \le i \le n} \sum_{j=1}^{n} |a_{ij}|$. The problem of finding T^{-1} is equivalent to the problem of inverting the matrix which represents T.

The next proposition is frequently used to prove the existence of an inverse operator.

Proposition 3.1.3. If X is a Banach space, $T \in [X]$ and $\|T\| < 1$, then $(I-T)^{-1}$ exists, $(I-T)^{-1} \in [X]$ and $\|(I-T)^{-1}\| \le 1/(1 - \|T\|)$.

An operator $T \in [X]$ is called compact if T maps bounded sets into relatively compact sets. If T is a compact operator, then (I-T) is one-to-one if and only if (I-T) is onto. This important property of compact operators is known as the Fredholm alternative. The next proposition is tailor-made for later use.

Proposition 3.1.4. Let $T, B \in [X]$ where X is a Banach space and T is compact. If $B(I-T) = I-A$ with $\|A\| < 1$, then $(I-T)^{-1}$ exists and $\|(I-T)^{-1}\| \le \|B\|/(1 - \|A\|)$.

Proof: The operator $(I-A)^{-1}$ exists by Proposition 3.1.3. Hence, $(I-A)^{-1} B(I-T) = I$ which means that (I-T) has a left inverse or,

equivalently, that (I-T) is one-to-one. By the Fredholm alternative, (I-T) is also onto which proves that $(I-T)^{-1}$ exists. It follows that $(I-T)^{-1} = (I-A)^{-1}B$ and then, using Propositions 3.1.2 and 3.1.3, that $\| (I-T)^{-1} \| = \| (I-A)^{-1}B \| \leq \| (I-A)^{-1} \| \cdot \| B \| \leq \| B \| / (1 - \| A \|)$.

3.2 Statement of Problem

The problem under consideration in this chapter is the Fredholm integral equation of the form:

$$x(s) - \int_a^b k(s,t)x(t)dt = y(s) \ , \ s\in[a,b]. \qquad (3.2\text{-}1)$$

The kernel, k, and the right hand side y are given. It is assumed that k is continuous on [a,b] x [a,b] and that y is continuous on [a,b]. The problem is to find the function x which satisfies (3.2-1).

The solution follows the method proposed in Chapter 2 of [1]. The original problem, (3.2-1), is not solved directly, instead the solution proceeds as follows. First, a problem which approximates the original problem is solved producing an approximate solution to the original problem. Second, the error between this approximate solution and an exact solution is bounded thus producing a rigorous solution.

These problems will now be given operator formulations. Define the linear operator $K : C[a,b] \to C[a,b]$ by $(Kx)(s) = \int_a^b k(s,t)x(t)dt$. It is known that K is bounded and compact. If I is the identity operator on C[a,b], then equation (3.2-1) can be written as $(I-K)x = y$. The operator K is approximated by using a quadrature rule in place of

the integral. Using n to indicate the number of quadrature points, the approximation K_n to K is defined by $(K_n x)(s) = \sum_{j=1}^{n} w_{nj} k(s,t_{nj}) x(t_{nj})$. The approximate problem is $(I-K_n)x_n = y$, where x_n indicates the solution. The functions x_n and x represent, respectively, approximate and exact solutions for (3.2-1). Section 3.3 deals with the computation of x_n while Section 3.4 deals with a bound for $\|x - x_n\|$. The quadrature rule is assumed to be either the repeated midpoint or repeated Simpson's rule as discussed in Section 2.5.

3.3 Computing the Approximate Solution

This section deals with the rigorous solution of $(I-K_n)x_n = y$. Using the definition of K_n, this equation can be written as

$$x_n(s) - \sum_{j=1}^{n} w_{nj} k(s,t_{nj}) x_n(t_{nj}) = y(s). \qquad (3.3\text{-}1)$$

By successively setting $s = t_{n1}, t_{n2}, \ldots, t_{nn}$ a system of n linear equations in the n unknowns $x_n(t_{n1}), x_n(t_{n2}), \ldots, x_n(t_{nn})$ is obtained. The matrix of this system is $(\overline{I-K_n}) = (\delta_{ij} - w_{nj} k(t_{ni},t_{nj}))_{i,j=1}^{n}$ and the right hand side is $\bar{y} = (y(t_{n1}), y(t_{n2}), \ldots, y(t_{nn}))$. Assuming that $(\overline{I-K_n})$ is nonsingular, the solution vector $\bar{x}_n = (x_n(t_{n1}), x_n(t_{n2}), \ldots, x_n(t_{nn}))$ exists and hence (3.3-1) can be solved for x_n giving

$$x_n(s) = y(s) + \sum_{j=1}^{n} w_{nj} k(s,t_{nj}) x_n(t_{nj}). \qquad (3.3\text{-}2)$$

By using the operator P_n of Example 3.1.3 with $S = \{t_{n1}, t_{n2}, \ldots, t_{nn}\}$ and by defining the operator $\hat{K}_n : \ell^{\infty}(n) \to C[a,b]$

by $(\hat{K}_n\bar{v})(s) = \sum_{j=1}^{n} w_{nj}k(s,t_{nj})v_j$ where $\bar{v} = (v_1,v_2,\ldots,v_n) \in \ell^\infty(n)$, equation (3.3-2) can be expressed in operator form as follows:

$$x_n = (I-K_n)^{-1}y = (I + \hat{K}_n(\overline{I-K}_n)^{-1}P_n)y. \qquad (3.3\text{-}3)$$

Notice that the existence of $(I-K_n)^{-1}$ can be established by showing that the matrix $(\overline{I-K}_n)^{-1}$ exists.

On the computer, an interval extension $((I-K_n)^{-1})^I$ of $(I-K_n)^{-1}$ will be found by the formula $((I-K_n)^{-1})^I = (I^I + \hat{K}_n^I((\overline{I-K}_n)^I)^{-1}P_n^I)$, where the interval operators P_n^I, $((\overline{I-K}_n)^I)^{-1}$, and $\hat{K}_n^I$ are discussed below. Recall that the machine computes intervals w_{nj}^I and t_{nj}^I, $j = 1,2,\ldots,n$, which contain the exact quadrature weights and points.

The operator P_n maps $C[a,b]$ into $\ell^\infty(n)$. Let $z\varepsilon C[a,b]$ and let z^I be an interval extension of z. Then $P_nz = (z(t_{n1}),z(t_{n2}),\ldots, z(t_{nn}))$ which is contained in $(z^I(t_{n1}^I),z^I(t_{n2}^I),\ldots,z^I(t_{nn}^I))$. Thus, if P_n^I is defined by $P_n^I(z^I) = (z^I(t_{n1}^I),z^I(t_{n2}^I),\ldots,z^I(t_{nn}^I))$, then P_n^I is an interval extension of P_n.

The matrix $(\overline{I-K}_n)^I$ will be an interval matrix containing $(\overline{I-K}_n)$. The componentwise containment is given by $\delta_{ij} - w_{nj}k(t_{ni},t_{nj}) \varepsilon \delta_{ij} - w_{nj}^I k^I(t_{ni}^I,t_{nj}^I)$, where k^I is an interval extension of k. The method of Section 2.4 is used to prove that $(\overline{I-K}_n)^{-1}$ exists and to find the interval matrix $((\overline{I-K}_n)^I)^{-1}$ which is guaranteed to contain $(\overline{I-K}_n)^{-1}$. As remarked in that section, the operator $((\overline{I-K}_n)^I)^{-1}$ is an interval extension of $(\overline{I-K}_n)^{-1}$.

Let $\bar{v} = (v_1,v_2,\ldots,v_n) \in \ell^\infty(n)$ and let $\bar{v}^I = (v_1^I,v_2^I,\ldots,v_n^I)$ be an interval vector containing $\bar{v}$ and define $\hat{K}_n^I$ by $(\hat{K}_n^I\bar{v}^I)(s^I) =$

$\sum_{j=1}^{n} w_{nj}^I k^I(s^I, t_{nj}^I) v_j^I$, where k^I is an interval extension of k. Using RIA to compute $\hat{K}_n^I$ it follows that $(\hat{K}_n \bar{v})(s) \subset (\hat{K}_n^I \bar{v}^I)(s^I)$ for all $s \subset s^I$ and hence that $\hat{K}_n^I$ is an interval extension of $\hat{K}_n$.

Propositions 2.3.2 and 2.3.3 show that $((I-K_n)^{-1})^I$ is an interval extension of $(I-K_n)^{-1}$. If y^I is an interval extension of y, then the interval function $x_n^I = ((I-K_n)^{-1})^I y^I$ extends x_n and can be computed automatically. In the process, the computer will have proved that $(I-K_n)^{-1}$ exists and will have bounded the round-off error in the computation of the approximate solution.

3.4 Computing an Error Bound

This section deals with the problems of proving that $(I-K)^{-1}$ exists and of bounding the error, $\|x - x_n\|$, in the approximate solution. The mathematical theory, with minor rearrangement, comes from Chapter 1 of [1]. Let K and K_n be as defined in Section 3.2.

Theorem 3.4.1. If $(I-K_n)^{-1}$ exists and $\|A\| = \Delta_n < 1$ where $A = (I-K_n)^{-1}(K-K_n)K$, then $(I-K)^{-1}$ exists and

(a) $\|(I-K)^{-1}\| \leq \sigma_n/(1 - \Delta_n)$

where $\sigma_n = \|I + (I-K_n)^{-1}K\|$,

and (b) $\|x - x_n\| \leq b_n = \rho_n/(1 - \Delta_n)$

where $\rho_n = \|(I-K_n)^{-1}((K-K_n)y + (K-K_n)K(I-K_n)^{-1}y)\|$.

Proof: If $B = I + (I-K_n)^{-1}K$, then it follows from a calculation that $B(I-K) = I-A$. The hypothesis guarantees that $\|A\| < 1$ so

that the existence of $(I-K)^{-1}$ and conclusion (a) follow from Proposition 3.1.4. Another calculation shows that $x - x_n = ((I-K)^{-1} - (I-K_n)^{-1})y = (I-A)^{-1}(I-K_n)^{-1}((K-K_n) + (K-K_n)K(I-K_n)^{-1})y$, where $(I-A)^{-1}$ exists by Proposition 3.1.3 since $\|A\| < 1$. Hence, by Proposition 3.1.2, $\|x - x_n\| \leq \|(I-A)^{-1}\| \cdot \|(I-K_n)^{-1}((K-K_n)y + (K-K_n)K(I-K_n)^{-1}y)\|$ and then using Proposition 3.1.3 again gives $\|x - x_n\| \leq \rho_n/(1 - \Delta_n)$ which completes the proof.

Before discussing a computer implementation of these results, a convergence theorem from Anselone [1] will be stated, without proof, and some remarks will be made. Think of $\{K_n : n = 1,2,3...\}$ as a family of operators formed by using either the repeated midpoint or repeated Simpson quadrature rules. (Repeated Simpson's rule is only defined for odd n.)

Theorem 3.4.2. If $(I-K_n)^{-1}$ exists and $\|(I-K_n)^{-1}\| \cdot \|(K-K_n)K\| < 1$, then $(I-K)^{-1}$ exists and $\|x - x_n\| \leq B_n$ where

$$B_n = \frac{\|(I-K_n)^{-1}\| \left(\|(K-K_n)y\| + \|(K-K_n)K\| \cdot \|(I-K_n)^{-1}\| \cdot \|y\| \right)}{1 - \|(I-K_n)^{-1}\| \cdot \|(K-K_n)K\|}$$

Also, the operators $(I-K_n)^{-1}$ do exist for all n sufficiently large, $\|(I-K_n)^{-1}\| \cdot \|(K-K_n)K\| \to 0$, and $B_n \to 0$ as $n \to \infty$.

The expression for B_n has both theoretical and practical importance. Its theoretical importance lies, in part, in the appearance of the term $\|(K-K_n)K\|$. The general approximation theory developed in [1] was used to show that $\|(K-K_n)K\| \to 0$ as $n \to \infty$.

It has practical importance because the individual norms which appear can be computed or bounded in some cases. As will be seen below, the function $(K-K_n)y$ and the operator $(K-K_n)K$ are closely related to quadrature errors. From equation (3.3-3) and the fact that $\|\hat{K}_n\| = \|K_n\|$ it follows that $\|(I-K_n)^{-1}\| \leq 1 + \|K_n\| \cdot \|(\overline{I-K_n})^{-1}\|$, where $\|(\overline{I-K_n})^{-1}\|$ is the matrix norm given in Example 3.1.4. Yungen [11] used B_n by calculating $\|(K-K_n)y\|$, $\|(K-K_n)K\|$, $\|K_n\|$ and $\|y\|$ by hand and using a rigorous computer technique to bound $\|(\overline{I-K_n})^{-1}\|$.

While experimenting with automating the computation of B_n it was found that B_n, computed in this way, may be too large to be useful. It was then found that the error bound b_n could be used instead and would improve the bound on $\|x - x_n\|$. Notice that the expression for B_n can be obtained from the expression for b_n by several applications of Proposition 3.1.2 and the triangle inequality. The values of both B_n and b_n are given for Test Problem 3 which is discussed in the next section.

The quantity ρ_n involves the norm of a function, say f, in C[a,b] and the quantity Δ_n involves the norm of an operator, say T, in [C[a,b]]. As will be seen below, the computer will generate interval extensions f^I of f and T^I of T. It will now be shown how f^I and T^I can be used to bound $\|f\|$ and $\|T\|$.

By the definition of an interval extension, $\{|f(t)| : t\in[a,b]\} \subset |f^I([a,b])|^I$. Thus, an upper bound for $\|f\|$ can be obtained by using the right endpoint of $|f^I([a,b])|^I$. As mentioned in

Section 2.3, the range of values given by an interval extension can be larger than the actual range of values. Improvement may be possible by subdividing [a,b] and using the interval extension on narrower intervals. Let [a,b] be subdivided, by the method given in Section 2.5, into m intervals, P^I_{m1}, P^I_{m2},...,P^I_{mm}. Then

$$\{|f(t)| : t\varepsilon[a,b]\} = \bigcup_{j=1}^{m} \{|f(t)| : t\varepsilon P^I_{mj}\}$$

$$\subset \bigcup_{j=1}^{m} |f^I(P^I_{mj})|^I \quad , \text{ and hence the}$$

maximum of the right endpoints of $|f^I(P^I_{mj})|^I$, j = 1,2,...,m, is an upper bound to $\|f\|$.

To find a bound on $\|T\|$, first let u^I be the constant interval valued function which has value [-1,1]. If $x\varepsilon C[a,b]$ and $\|x\| \leq 1$, then u^I is an interval extension of x and hence $T^I u^I$ is an interval extension of Tx. The method of the previous paragraph allows use of $T^I u^I$ to compute a bound on $\|Tx\|$ which will also be a bound on $\|T\|$.

In the problem of bounding $b_n = \rho_n/(1 - \Delta_n)$, the quantities are $\rho_n = \|f\|$ and $\Delta_n = \|T\|$ where $f = (I-K_n)^{-1}((K-K_n)y + (K-K_n)K(I-K_n)^{-1}y)$ and $T = (I-K_n)^{-1}(K-K_n)K$. An interval extension of the operator $(I-K_n)^{-1}$ was constructed in the previous section. In view of Propositions 2.3.1 and 2.3.3 it only remains to find interval extensions for the function $(K-K_n)y$ and for the operator $(K-K_n)K$.

Writing out the function $(K-K_n)y$ gives

$$((K-K_n)y)(s) = \int_a^b k(s,t)y(t)dt - \sum_{j=1}^{n} w_{nj}k(s,t_{nj})y(t_{nj}).$$

Hence, $((K-K_n)y)(s) = G(s)$ where G is from equation (2.5-1) with

$g(s,t) = k(s,t)y(t)$. An interval extension for G can be constructed as discussed in Section 2.5.

If $x\varepsilon C[a,b]$ and $s\varepsilon[a,b]$, then the operator $(K-K_n)K$ can be written as $((K-K_n)Kx)(s) = \int_a^b H(s,u)x(u)du$, where H is as in equation (2.5-2) with $h(s,t,u) = k(s,t)k(t,u)$. An interval extension of H can be found by the methods of Section 2.5. Finally, an interval extension for $(K-K_n)K$ will be found by using the Riemann sum method on the integral $\int_a^b H(s,u)x(u)du$.

Thus, with some user supplied interval Lipschitz constants or derivatives with respect to t for the functions $g(s,t) = k(s,t)y(t)$ and $h(s,t,u) = k(s,t)k(t,u)$, the computer can find bounds ρ_n^* and Δ_n^* on ρ_n and Δ_n. If $\Delta_n^* < 1$, then Theorem 3.4.1 applies so that $(I-K)^{-1}$ exists and $\|x - x_n\| \leq b_n^* = \rho_n^*/(1 - \Delta_n^*)$.

If x_n^I is the interval extension of x_n from Section 3.3, then the formula $x^I(s^I) = x_n^I(s^I) + [-b_n^*, b_n^*]$ defines an interval extension of $x = (I-K)^{-1}y$. So, if s is any point in [a,b], then $x^I([s,s])$ is an interval which is guaranteed to contain x(s).

3.5 Computer Program and Test Results

A computer program, FRED, was written which is capable of finding a rigorous interval extension for the solution to the linear integral equation defined in Section 3.2. The program follows the method outlined in Sections 3.3 and 3.4 and utilizes RIA which was defined in Section 2.2. This section is devoted to describing the input required by FRED, the output from FRED and the results obtained on four test

problems. For each problem, the user supplies four interval function subroutines written in FORTRAN. Appendix B contains input-output information for the user of FRED and, using Test Problem 1 (below) as an example, gives the required form for the interval function subroutines.

Input for FRED:

(1) Machine intervals, [AMINUS,APLUS] and [BMINUS,BPLUS], which contain a and b, respectively.

(2) An integer, M, which gives the number of subdivisions to use in determining bounds on the maximum norms which occur in the expressions for ρ_n and Δ_n. It is suggested to start with M = 1 and increase M gradually to see if an improved error bound results.

(3) Interval extensions of k and y.

(4) Choice of quadrature rule. The repeated midpoint rule is available as either a first or second order method (IQFLAG = 1 or IQFLAG = 2) and repeated Simpson's rule is available as a fourth order method (IQFLAG = 4). The computer generates interval extensions for G and H from equations (2.5-1) and (2.5-2) with g(s,t) = k(s,t)y(t) and h(s,t,u) = k(s,t)k(t,u). The user must supply the appropriate interval Lipschitz constants or partial derivatives as discussed in Section 2.5.

(5) An integer, L, which specifies the number of repetitions of the quadrature rule chosen above. If the midpoint rule is used, then N = L and if Simpson's rule is used, then N = 2L+1 where N is the number of quadrature points. As the program is currently written,

N cannot exceed 51. The Riemann sum type integration also uses L subdivisions as does the computation of a bound on $\|x\|$.

(6) Answer point information. The user supplies points or intervals, s^I = [SMINUS,SPLUS], contained in [a,b] where the value or range of values of the answer function is desired. The variable NAPTS is used for the number of these intervals and NAPTS cannot exceed 100. Also, at the option of the user, the value of the answer is given at the machine computed quadrature points. The variable IFLAG should be set to one if this option is desired and set to zero otherwise.

Output from FRED:

(1) A bound on $\|E^I\|^I$ as discussed in Section 2.4. If $\|E^I\|^I < 1$, then the computer has proven that $(\overline{I-K_n})^{-1}$ exists and hence that $(I-K_n)^{-1}$ exists. The program terminates if the bound is not less than one.

(2) The bound Δ_n^* on Δ_n. If $\Delta_n^* < 1$, then the computer has shown that $(I-K)^{-1}$ exists. If Δ_n^* is not less than one the program terminates.

(3) The quadrature error bound, b_n^*.

(4) A bound on $\|x\|$.

(5) At each answer point (interval), s^I, the following information is given:

(a) The interval $a^I = x^I(s^I)$.

(b) The center of a^I.

(c) The half width of a^I. If s^I has zero width, then this is the overall error bound.

(d) The half width of $x_n^I(s^I)$. If s^I has zero width, then this is the bound on round-off error for the solution of the approximate problem.

Program FRED was run with four test problems. Problems one and two are reported by Yungen [11] and problem three is from Appendix 2 of [1]. These problems are discussed briefly below followed by tables containing the results. The results printed in the tables include the interval matrix norm, $\|E^I\|^I$, the quadrature error bound, b_n^*, the bound on $\|x\|$ and the execution time. The answer function was sampled on a grid with spacing 0.1. Included in the tables is the maximum round-off error bound that was encountered on this grid. Problems one, two and three have known exact solutions and also included in the tables is the maximum actual error made by the center on the grid. All runs were made with M = 1.

Test Problem 1.

$$x(s) - \frac{1}{2}\int_0^1 e^{s-t}x(t)dt = 1$$

The exact solution is $x(s) = 1 + e^s - e^{s-1}$ which has norm equal to e. Table 1 contains the results when Simpson's rule is used. In this problem b_n^* is approximately $0.01/(n-1)^4$.

Test Problem 2.

$$x(s) - \int_0^1 k(s,t)x(t)dt = (1/2)s(1-s)$$

where $k(s,t) = \begin{cases} t(1-s) & 0 \le t \le s \le 1, \\ s(1-t) & 0 \le s \le t \le 1. \end{cases}$

The exact solution is $x(s) = (\tan \frac{1}{2})\sin(s) + \cos(s) - 1$ which has norm approximately equal to 0.14. The first partial derivatives of k are discontinuous and hence only the first order method is applicable. Table 2 contains the results where it is found that b_n^* is approximately 0.05/n.

Test Problem 3.

$$x(s) - 10 \int_0^1 e^{st} x(t)dt = s - 10((s-1)e^s+1)/s^2$$

The exact solution is $x(s) = s$ which has norm equal to one. For this problem, the bound B_n from Theorem 3.4.2 was found to be too large to be useful. For example, when Simpson's rule is used $B_{11} = 9559.4$. However, again using Simpson's rule, the program finds $b_{11}^* = 0.1537$. Table 3A contains the results from FRED and Table 3B contains the individual norms and the value of B_n. Notice that the execution times for computing b_n^* are longer than for computing B_n. This is because operators which appear separated in the expression for B_n are composed when b_n^* is computed.

Test Problem 4.

$$x(s) - \frac{1}{\pi} \int_{-1}^{1} \frac{1}{1 + (s-t)^2} x(t)dt = 1$$

The results from runs with Simpson's rules are given in Table 4.

TABLE 1

Problem:

$$x(s) - \frac{1}{2} \int_0^1 e^{s-t} x(t)dt = 1$$

Quadrature Option: Repeated Simpson's Rule (IQFLAG = 4)

Results:

	N = 11	N = 21	N = 41
$\|\|E^I\|\|^I$	1.5×10^{-13}	2.2×10^{-13}	3.3×10^{-13}
$\|\|x\|\|$	2.718283836	2.718281951	2.718281837
Round-off Error Bound	3.0×10^{-13}	4.2×10^{-13}	6.5×10^{-13}
Quadrature Error Bound	1.054×10^{-6}	6.270×10^{-8}	3.823×10^{-9}
Actual Error	9.535×10^{-7}	5.965×10^{-8}	3.729×10^{-9}
Execution Time	1.171 sec.	6.225 sec.	42.122 sec.

TABLE 2

Problem:

$$x(s) - \int_0^1 k(s,t)x(t)dt = (1/2)s(1-s)$$

$$\text{where } k(s,t) = \begin{cases} t(1-s) & 0 \le t \le s \le 1, \\ s(1-t) & 0 < s < t < 1. \end{cases}$$

Quadrature Option: Repeated midpoint rule as a first order method (IQFLAG = 1)

Results:

	N = 10	N = 20	N = 40
$\|E^I\|^I$	5.8×10^{-14}	9.7×10^{-14}	1.81×10^{-13}
$\|x\|$	0.14611	0.14264	0.14103
Round-off Error Bound	3.6×10^{-15}	4.5×10^{-15}	6.3×10^{-15}
Quadrature Error Bound	5.525×10^{-3}	2.511×10^{-3}	1.192×10^{-3}
Actual Error	4.466×10^{-5}	1.117×10^{-5}	2.793×10^{-6}
Execution Time	1.335 sec.	7.562 sec.	52.653 sec.

TABLE 3A

Problem:

$$x(s) - 10 \int_0^1 e^{st} x(t)dt = s - 10((s-1)e^s+1)/s^2$$

Quadrature Option: Repeated Simpson's Rule (IQFLAG = 4)

Results:

	N = 11	N = 21	N = 41
$\|E^I\|^I$	1.2×10^{-11}	1.6×10^{-11}	2.4×10^{-11}
$\|x\|$	2.3	1.6	1.31
Round-off Error Bound	1.4×10^{-8}	2.0×10^{-8}	3.2×10^{-8}
Quadrature Error Bound	1.537×10^{-1}	4.436×10^{-3}	1.775×10^{-4}
Actual Error	2.8×10^{-4}	1.75×10^{-5}	1.1×10^{-6}
Execution Time	1.566 sec.	8.219 sec.	54.265 sec.

TABLE 3B

Problem: (Same as in Table 3A)

Quadrature Option: (Same as in Table 3A)

Results:

	N = 11	N = 21	N = 41
$\|\|(\overline{I-K_n})^{-1}\|\|$	27.72	27.69	28.60
$\|\|K_n\|\|$	17.183	17.183	17.183
$\|\|(K-K_n)K\|\|$	1.446×10^{-3}	7.215×10^{-5}	4.018×10^{-6}
$\|\|y\|\|$	9.0	9.0	9.0
$\|\|(K-K_n)y\|\|$	9.470×10^{-4}	5.622×10^{-5}	3.424×10^{-6}
B_n	9559.4	152.8	8.8
Execution Time	1.02 sec.	3.64 sec.	20.80 sec.

TABLE 4

Problem:

$$x(s) - \frac{1}{\pi} \int_{-1}^{1} \frac{1}{1 + (s-t)^2} x(t)dt = 1$$

Quadrature Option: Repeated Simpson's Rule (IQFLAG = 4)

Results:

	N = 11	N = 21	N = 41
$\|\|E^I\|\|^I$	1.02×10^{-13}	1.56×10^{-13}	2.75×10^{-13}
$\|\|x\|\|$	2.04895	1.97110	1.94622
Round-off Error Bound	1.3×10^{-13}	1.8×10^{-13}	2.8×10^{-13}
Quadrature Error Bound	2.811×10^{-2}	1.164×10^{-3}	5.987×10^{-5}
Execution Time	2.349 sec.	13.958 sec.	98.440 sec.

CHAPTER 4

THE NONLINEAR PROBLEM

4.1 Background Concepts, Examples and Results

The operators encountered in the solution of the Urysohn equation are nonlinear and hence the solution will have a different theoretical basis from the linear theory of Chapter 3. Underlying the solution is the concept of Newton's method which is an iterative technique for finding roots of an equation. For the equation $f(x) = 0$, where f is a real function of a real variable, the Newton iteration is defined by:

$$x_{n+1} = x_n - (1/f'(x_n))f(x_n) \quad , \quad n = 0,1,2,\ldots \; . \tag{4.1-1}$$

In order to interpret this iteration when f is an operator on a Banach space, the concept of derivative must be extended to such operators. The Fréchet derivative generalizes the usual derivative and will now be defined. Let X and Y be Banach spaces and let S be an open subset of X. A continuous function f of S into Y is said to be Fréchet differentiable at $x_0 \in S$ with derivative $f'(x_0)$ if $f'(x_0)$ is a linear operator from X to Y and

$$\lim_{\substack{h \to 0 \\ h \neq 0}} \frac{\| f(x_0+h) - f(x_0) - f'(x_0)h \|}{\| h \|} = 0 \; .$$

The next proposition states important elementary properties of the Fréchet derivative. Proofs of these and other properties summarized

in this section can be found in Chapter VIII of [4].

Proposition 4.1.1. The Fréchet derivative or, for short, just derivative, has the following properties:

(a) The derivative, when it exists, is unique.

(b) The derivative is a bounded linear operator.

(c) A constant function on S is differentiable at every point of S and its derivative is the zero linear operator.

(d) If $T \in [X,Y]$, then $T'(x)$ exists at all points x in X and $T'(x) = T$.

(e) If f and g are differentiable at $x_0 \in S$, then $f + g$ and αf are differentiable at x_0 with $(f+g)'(x_0) = f'(x_0) + g'(x_0)$ and $(\alpha f)'(x_0) = \alpha f'(x_0)$.

Example 4.1.1. Let I be the identity operator on X and define $P(x) = x$. By property (d) above, $P'(x) = I$ for all $x \in X$.

Let f be a differentiable mapping of S into Y so f' maps S into [X,Y]. If f' is continuous, then f is said to be continuously differentiable on S.

Example 4.1.2. Let $X = Y = \ell^{\infty}(n)$ and let $f = (f_1, f_2, \ldots, f_n)$ be an operator on $S \subset X$ where $f_1, f_2, \ldots, f_n$ are real functions such that if $\bar{x} \in S$, then $f(\bar{x}) = (f_1(\bar{x}), f_2(\bar{x}), \ldots, f_n(\bar{x}))$. It is known that f is continuously differentiable on S if and only if each of the partial derivatives $D_j f_i$ exists and is continuous on S. Suppose

that this is the case and that $\bar{x}_0 \in S$, then $f'(\bar{x}_0) \in [X]$ is represented by the matrix $(D_j f_i(\bar{x}_0))_{i,j=1}^{n}$ which is called the Jacobian matrix of f at $\bar{x}_0$.

If f is continuously differentiable on S and the mapping $x \to f'(x)$ is differentiable at $x_0 \in S$, then f is said to be twice differentiable at x_0. The notation $f''(x_0)$ will be used for the second derivative which is an element of $[X,[X,Y]]$.

Example 4.1.3. Let P be the operator of Example 4.1.1. Since $P'(x) = I$, Property (c) of Proposition 4.1.1 implies that $P''(x) = 0$.

This concludes the summary of definitions and properties relating to the Fréchet derivative.

Now, if P is an operator on a Banach space, then the analogue of (4.1-1) for solving $P(x) = 0$ is $x_{n+1} = x_n - (P'(x_n))^{-1}P(x_n)$, $n = 0,1,2,\ldots$. Starting from an arbitrary initial guess, this iteration may or may not converge (may even fail to exist!) to a solution of $P(x) = 0$. Suppose that the iteration appears to converge to an element $x_0 \in X$ so that x_0 can be thought of as an approximate solution for $P(x) = 0$. At this point, the following theorem of Kantorovich, which can be found in [7], can be used to give information about exact solutions for $P(x) = 0$. First some convenient notation.

If X is a Banach space, let $B(x,r)$ be the open ball of radius r centered at x and let $\overline{B(x,r)}$ denote its closure. Also, define the functions ω and ω^+ by

$$\omega(h) = \begin{cases} (1 - \sqrt{1-2h})/h & 0 < h \leq \frac{1}{2}, \\ 1 & h = 0, \end{cases}$$

and

$$\omega^{+}(h) = (1 + \sqrt{1-2h})/h \qquad 0 < h \leq \tfrac{1}{2}.$$

Theorem 4.1.1. Assume that P is a twice differentiable operator on a Banach space X, $x_0 \in X$, $r > 0$ and

(a) $(P'(x_0))^{-1}$ exists in [X] with $\|(P'(x_0))^{-1}\| \leq \alpha$,

(b) $\|P(x_0)\| \leq \beta$,

(c) $\|P''(x)\| \leq \gamma$ for all x in $B(x_0,r)$,

(d) $h = \alpha^2\beta\gamma \leq \frac{1}{2}$,

(e) $r \geq r_0 = \alpha\beta\omega(h)$.

Then there exists a unique $x^* \in \overline{B(x_0,r_0)}$ such that $P(x^*) = 0$. And, if $h < \frac{1}{2}$, then x^* is the unique zero of P in $B(x_0,r) \cap B(x_0,r_1)$ where $r_1 = \alpha\beta\omega^{+}(h)$.

It is possible to weaken the differentiability assumption on P and replace condition (c) with the condition that $\|P'(x) - P'(y)\| \leq \gamma \|x - y\|$ for all x , y in $B(x_0,r)$.

4.2 Statement of Problem

The problem considered in this chapter is the Urysohn integral equation which has the form:

$$x(s) - \int_a^b k(s,t,x(t))dt = y(s) \ , \ s\in[a,b]. \qquad (4.2\text{-}1)$$

It is assumed that k is continuous in s and t and twice continuously

differentiable in x and that y is continuous. The Urysohn integral operator K on C[a,b] will be defined by $(K(x))(s) = \int_a^b k(s,t,x(t))dt$, allowing equation (4.2-1) to be written as $(I-K)(x) = y$. By defining the operator P on C[a,b] by $P(x) = (I-K)(x) - y$, problem (4.2-1) becomes equivalent to the problem of solving $P(x) = 0$.

The program for solving $P(x) = 0$ involves two stages. First, an attempt is made to find an approximate solution x_0. Second, Theorem 4.1.1 is used with the hope of finding a "small" r_0 and hence proving that x_0 is actually close to an exact solution. The approximate solution will come from solving $P_n(x) = (I-K_n)(x) - y = 0$ where K_n is defined by $(K_n(x))(s) = \sum_{j=1}^{n} w_{nj}k(s,t_{nj},x(t_{nj}))$. The problem of finding x_0 is discussed in Section 4.3 and the problem of applying Theorem 4.1.1 is considered in Section 4.4.

The following Fréchet derivatives of K and K_n will be needed later:

$$(K'(x)u)(s) = \int_a^b D_3k(s,t,x(t))u(t)dt,$$

$$(K''(x)uv)(s) = \int_a^b D_3D_3k(s,t,x(t))u(t)v(t)dt,$$

and

$$(K_n'(x)u)(s) = \sum_{j=1}^{n} w_{nj}D_3k(s,t_{nj},x(t_{nj}))u(t_{nj}).$$

Notice that, for each x , $K'(x)$ is a linear integral operator and that $K_n'(x)$ is the quadrature approximation to $K'(x)$ in the sense

of Chapter 3. Thus, $K_n'(x)$ can be viewed as either the quadrature approximation to $K'(x)$ or the derivative of the quadrature approximation K_n to K.

4.3 Trying for an Approximate Solution

From the formula $P_n(x) = x - K_n(x) - y$, it follows that a solution x_n^* to $P_n(x) = 0$ has the form $x_n^*(s) = y(s) + \sum_{j=1}^{n} w_{nj} k(s,t_{nj},x_n^*(t_{nj}))$. Thus, by successively setting $s = t_{n1},t_{n2},\ldots,t_{nn}$ it can be seen that the problem of finding x_n^* is equivalent to solving a system of n nonlinear equations for the n unknowns $x_n^*(t_{n1}),x_n^*(t_{n2}),\ldots,x_n^*(t_{nn})$. Let $\bar{v} = (v_1,v_2,\ldots,v_n)\varepsilon\ell^\infty(n)$ and define the operator $\bar{K}_n$ on $\ell^\infty(n)$ by

$$\bar{K}_n(\bar{v}) = \left(\sum_{j=1}^{n} w_{nj} k(t_{n1},t_{nj},v_j),\ldots,\sum_{j=1}^{n} w_{nj} k(t_{nn},t_{nj},v_j)\right).$$

Solving the nonlinear system mentioned above is equivalent to solving $\bar{P}_n(\bar{x}) = (\bar{I}-\bar{K}_n)(\bar{x}) - \bar{y} = 0$ where $\bar{y} = (y(t_{n1}),y(t_{n2}),\ldots,y(t_{nn}))$.

There is no general technique for solving $\bar{P}_n(\bar{x}) = 0$; however, the following procedure may produce a good approximate solution. Let $\bar{x}_n^0 \varepsilon \ell^\infty(n)$ be a guess for a solution and use $\bar{x}_n^0$ as a starting point in the Newton iteration: $\bar{x}_n^{m+1} = \bar{x}_n^m - (\bar{P}_n'(\bar{x}_n^m))^{-1}\bar{P}_n(\bar{x}_n^m)$, $m = 0,1,2,\ldots$. If the iterates appear to converge, then the iteration is stopped. Specifically, the iteration is stopped when $\|\bar{x}_n^{m+1} - \bar{x}_n^m\| / \|\bar{x}_n^m\| \leq \varepsilon$ where ε is a user supplied tolerance or when the number of iterations has exceeded a user supplied limit. Suppose that the convergence criterion has been met. Then label the last iterate as

$\bar{z}_n = (z_{n1}, z_{n2}, \ldots, z_{nn})$ so that $\bar{z}_n$ is an approximate solution to $\bar{P}_n(\bar{x}) = 0$.

Now define x_0 by $x_0(s) = y(s) + \sum_{j=1}^{n} w_{nj} k(s, t_{nj}, z_{nj})$. Note that $\bar{z}_n$ is only an approximate solution to $\bar{P}_n(\bar{x}) = 0$ and hence x_0 will be only an approximate solution to $P_n(x) = 0$. On the computer, an interval extension of x_0 will be used. The components of $\bar{z}_n$ are considered to be exact so x_0^I is defined by $x_0^I(s^I) = y^I(s^I) + \sum_{j=1}^{n} w_{nj}^I k^I(s^I, t_{nj}^I, z_{nj}^I)$ where y^I and k^I are interval extensions for y and k and $z_{nj}^I = [z_{nj}, z_{nj}]$, $j = 1,2,\ldots,n$.

The iteration for $\bar{z}_n$ is carried out using single precision arithmetic. At each step, Gauss elimination is used to solve the linear system, $\bar{P}_n'(\bar{x}_n^m)(\bar{x}_n^m - \bar{x}_n^{m+1}) = \bar{P}_n(\bar{x}_n^m)$, for $(\bar{x}_n^m - \bar{x}_n^{m+1})$. The derivative, $\bar{P}_n'(\bar{x}_n^m)$, is of the type discussed in Example 4.1.2. In the notation of that example, the function f_i for the operator $\bar{P}_n$ is given by $f_i(x_1, x_2, \ldots, x_n) = x_i - \sum_{j=1}^{n} w_{nj} k(t_{ni}, t_{nj}, x_j) - y(t_{ni})$. Hence, the Jacobian matrix of $\bar{P}_n'$ at $(x_1, x_2, \ldots, x_n)$ has its ij^{th} element equal to $D_j f_i(x_1, x_2, \ldots, x_n) = \delta_{ij} - w_{nj} D_3 k(t_{ni}, t_{nj}, x_j)$.

4.4 Computing Existence and Uniqueness Radii

Given the function x_0 as computed in the previous section, the problem of this section is to verify the hypothesis of Theorem 4.1.1 with $P(x) = (I-K)(x) - y$. It follows from Proposition 4.1.1 and Examples 4.1.1 and 4.1.3 that $P'(x) = I - K'(x)$ and $P''(x) = -K''(x)$. Hence, conditions (a), (b) and (c) of Theorem 4.1.1 take

the form:

(a') $(I-K'(x_0))^{-1}$ exists in [C[a,b]] with

$$\|(I-K'(x_0))^{-1}\| \leq \alpha,$$

(b') $$\|(I-K)(x_0) - y\| \leq \beta,$$

and

(c') $$\|K''(x)\| \leq \gamma \text{ for all } x\varepsilon B(x_0,r).$$

First consider condition (a'). As noted in Section 4.2, the derivative $K'(x_0)$ is a linear integral operator and $K_n'(x_0)$ is its quadrature approximation in the sense of Chapter 3. Hence, Theorem 3.4.1 is applicable which means that the existence of $(I-K'(x_0))^{-1}$ ε[C[a,b]] can be inferred from the existence of $(I-K_n'(x_0))^{-1}$ together with the fact that $\Delta_n = \|(I-K_n'(x_0))^{-1}(K'(x_0) - K_n'(x_0))K'(x_0)\| < 1$. Also, $\|(I-K'(x_0))^{-1}\| \leq \sigma_n/(1 - \Delta_n)$ where $\sigma_n = \|I + (I-K_n'(x_0))^{-1}K'(x_0)\|$. The problem of finding a bound, Δ_n^*, on Δ_n was solved in Sections 3.3 and 3.4 which includes proof of the existence of $(I-K_n'(x_0))^{-1}$. Finding a bound, σ_n^*, on σ_n can be done by similar methods with the only additional requirement being an interval extension for the operator $K'(x_0)$. Since $(K'(x_0)u)(s) = \int_a^b D_3k(s,t,x_0(t))u(t)dt$, an interval extension is found by using interval extensions for D_3k and x_0 and using the Riemann sum method to compute the integral. Assuming that $\Delta_n^* < 1$, α can be taken as $\sigma_n^*/(1 - \Delta_n^*)$.

Next, following Bryan [2], consider condition (b'). The function whose norm is desired is

$$((I-K)(x_0) - y)(s) = \sum_{j=1}^{n} w_{nj}k(s,t_{nj},z_{nj}) - \int_a^b k(s,t,x_0(t))dt$$

$$= f_1(s) + f_2(s)$$

where

$$f_1(s) = \sum_{j=1}^{n} w_{nj}k(s,t_{nj},z_{nj}) - \sum_{j=1}^{n} w_{nj}k(s,t_{nj},x_0(t_{nj}))$$

and

$$f_2(s) = \sum_{j=1}^{n} w_{nj}k(s,t_{nj},x_0(t_{nj})) - \int_a^b k(s,t,x_0(t))dt.$$

Using the Mean Value Theorem, the function f_1 can be written as $f_1(s) = \sum_{j=1}^{n} w_{nj}(z_{nj} - x_0(t_{nj}))D_3k(s,t_{nj},\theta_j)$ where θ_j lies between z_{nj} and $x_0(t_{nj})$ for $j = 1,2,\ldots,n$. An interval extension, f_1^I, for f_1 can be found by replacing θ_j with the interval from z_{nj} to $x_0(t_{nj})$ and using an interval extension for the function D_3k. The function f_2 involves a quadrature error and so an interval extension, f_2^I, for f_2 can be found by previously developed techniques. Using f_1^I and f_2^I the computer can find bounds on $\|f_1\|$ and $\|f_2\|$. Since $\|(I-K)(x_0) - y\| = \|f_1 + f_2\| \leq \|f_1\| + \|f_2\|$ a value for β can be found by the computer.

Finally consider condition (c'). A value for r must be chosen at this point. The value $2\alpha\beta$ will be used for the following reasons. Since $\omega(h) \leq 2$, $r = 2\alpha\beta$ is the smallest value of r for which condition (e) will always be satisfied. And, the smaller the value of r the better from the standpoint of getting a small γ and hence of satisfying condition (d). It can be shown that $\|K''(x)\| = \|f\|$ where

$f(s) = \int_a^b D_3D_3k(s,t,x(t))dt$ so that a bound on $\|K''(x)\|$ can be computed if an interval extension for f can be found. Such an interval extension can be obtained using interval extensions for the functions D_3D_3k and x and the Riemann sum method to carry out the integration. Suppose that x^I is defined by $x^I(s^I) = x_0^I(s^I) + [-r,r]$, then x^I is an interval extension for all functions in $B(x_0,r)$. Using x^I, an interval extension of f is found which holds for all x in $B(x_0,r)$ and hence a bound, γ, on $\|K''(x)\|$ can be found which holds for all x in $B(x_0,r)$ as desired.

Assume that α, β and γ have been computed by the above methods and that $h = \alpha^2\beta\gamma \leq \frac{1}{2}$ so that the hypothesis of Theorem 4.1.1 is satisfied. Then there exists a unique solution x to $(I-K)(x) = y$ such that $x\varepsilon\overline{B(x_0,r_0)}$. The computer produces the interval function x^I defined by $x^I(s^I) = x_0^I(s^I) + [-r_0,r_0]$ which is an interval extension of x.

The value r_0 gives the radius of a ball centered at x_0 in which existence and uniqueness of an exact solution are guaranteed. Theorem 4.1.1 also contains the possibility of finding a larger ball in which uniqueness of the exact solution is still guaranteed. Let α and β be computed as above and input a value for r which is larger than the automatic value of $2\alpha\beta$. Then compute the corresponding γ, h and r_0. If $h < \frac{1}{2}$ and $r_0 \leq r$ then the further uniqueness result is available.

4.5 Computer Program and Test Results

A computer program, URYSOHN, was written which implements the methods of Sections 4.3 and 4.4 in an attempt to find a rigorous interval extension for an exact solution to the equation defined in Section 4.2. The success of the program depends on finding convergence of Newton's method and on satisfying the hypothesis of Theorem 4.1.1. The input required by URYSOHN, the output from URYSOHN and the results obtained on two test problems are described in this section. For each problem, the user supplies three real function subroutines and six interval function subroutines written in FORTRAN. Appendix C contains input-output information for the user of URYSOHN and gives the required form for the function subroutines using Test Problem 5 (below) as an example.

Input for URYSOHN;

(1) Same as input (1) for FRED. (see page 37)

(2) Same as input (2) for FRED except that the maximum norms occur in the expressions for σ_n, Δ_n, β and γ.

(3) Real function subroutines which give k, D_3k and y to single precision accuracy. An integer, ITRLIM, which sets a limit to the number of iterations in Newton's method, a value of EPSILON which gives the stopping criterion and an N-dimensional vector which serves as the initial guess for Newton's method.

(4) Interval extensions for k, D_3k, D_3D_3k and y.

(5) Same as input (4) for FRED except that now

$g(s,t) = k(s,t,x_0(t))$ and $h(s,t,u) = k(s,t,x_0(t)) \cdot k(t,u,x_0(u))$.

(6) Same as input (5) for FRED.

(7) Same as input (6) for FRED.

(8) The number, NUVALS, of values of R for which uniqueness information is desired and the values of R themselves (if NUVALS > 0).

Output from URYSOHN:

(1) At each step of the Newton iteration the value of the relative correction, $\|\bar{x}_n^{m+1} - \bar{x}_n^m\| / \|\bar{x}_n^m\|$, is given.

(2) A bound on $\|E^I\|^I$. If this bound is less than one, then $(I-K_n'(x_0))^{-1}$ is known to exist. The program terminates if this bound is not less than one.

(3) The bound Δ_n^* on Δ_n. If $\Delta_n^* < 1$, then the existence of $(I-K'(x_0))^{-1}$ has been established. The program terminates if $\Delta_n^* \geq 1$.

(4) The values of α, β, r, and γ.

(5) The value of $h = \alpha^2\beta\gamma$. If $h > \frac{1}{2}$, the program terminates.

(6) The radius, r_0, of an existence and uniqueness ball.

(7) A bound on the norm of an exact solution.

(8) At each answer point (interval), s^I, the following information is given:

(a) The interval $a^I = x^I(s^I)$.

(b) The center of a^I.

(c) The half width of a^I. If s^I has zero width, then this is the overall error bound.

(9) For each value of R for which uniqueness information is desired, the program gives the following:

(a) Corresponding values of γ and h. If $h \geq \frac{1}{2}$, then the program goes on to the next value of R.

(b) The value of r_0. The next value of R is taken if $r_0 > R$.

(c) The value of r_1 and the radius of the uniqueness ball which is the minimum of R and r_1.

Program URYSOHN was run on two test problems. The first is from Moore [9] and the second is the problem considered by Bryan [2]. These problems are discussed briefly below followed by the results in table form. The results include the quantities α, β, r, γ, h, and r_0 from Theorem 4.1.1. The answer function was sampled on a grid with spacing 0.1 and the maximum overall error bound on this grid is given in the tables. The first problem has a known exact solution and the maximum actual error made by the center is included in the table for this problem. Also included is the programs bound on $\|x\|$.

Test Problem 5.

$$x(s) - \int_0^1 stx^2(t)dt = (3/4)s$$

This problem has two exact solutions, $x(s) = s$ and $x(s) = 3s$. The convergence tolerance was set to 10^{-12} and the initial guess was the vector with each element equal to $\frac{1}{2}$. The quadrature errors encountered in this problem occur on cubic integrands; thus, Simpson's

rule commits no error. Table 5 contains the existence results obtained from using the repeated midpoint rule as a second order method and from using Simpson's rule. In both cases, the convergence criterion was met after six iterations and it was the solution $x(s) = s$ which was approached. In this problem, $K''(x)$ is independent of x and hence γ and r_1 are independent of r so the best uniqueness radius occurs with the value of r_1. In the case of the repeated midpoint rule with $N = 20$ the value of r_1 was 0.78186.

Test Problem 6.

$$x(s) - \int_0^1 e^{st-x^2(t)}\, dt = 1$$

The convergence tolerance was set to 10^{-12} and the vector with each element equal to one was used for the initial guess. Table 6A contains the existence results from the use of the repeated midpoint rule as a second order method. The convergence criterion was met after five iterations. In this problem, γ increases with increasing r so that r_1 decreases with increasing r. Table 6B contains the uniqueness output corresponding to $r = 0.1$, 0.5 and 1.0 obtained from the run with $N = 15$. Notice that $r_1 > r$ when $r = 0.1$ but that $r_1 < r$ when $r = 0.5$. The best uniqueness radius is found to be 0.1.

TABLE 5

Problem:

$$x(s) - \int_0^1 stx^2(t)dt = (3/4)s$$

Quadrature Option: Repeated Midpoint Rule (IQFLAG = 2) and

Repeated Simpson's Rule (IQFLAG = 4)

Results:

	Midpoint Rule N = 10	Midpoint Rule N = 20	Simpson's Rule N = 3
α	2.535	2.434	5.0
β	1.369×10^{-3}	3.277×10^{-4}	1.2×10^{-14}
r	6.94×10^{-3}	1.595×10^{-3}	1.2×10^{-13}
γ	1.1	1.05	2.0
h	9.374×10^{-3}	2.038×10^{-3}	5.92×10^{-13}
r_0	3.486×10^{-3}	7.984×10^{-4}	5.8×10^{-14}
Overall Error Bound	3.486×10^{-3}	7.984×10^{-4}	7.5×10^{-14}
Actual Error	2.48×10^{-3}	6.24×10^{-4}	1.0×10^{-14}
$\|\|x\|\|$	1.001002	1.000175	$1.0 + 10^{-13}$
Execution Time	5.139 sec.	56.086 sec.	0.294 sec.

TABLE 6A

Problem:

$$x(s) - \int_0^1 e^{st-x^2(t)}\, dt = 1$$

Quadrature Option: Repeated Midpoint Rule (IQFLAG = 2)

Results:

	N = 10	N = 15
α	2.75	2.73
β	8.15×10^{-5}	3.483×10^{-5}
r	4.482×10^{-4}	1.9×10^{-4}
γ	1.653	1.6054
h	1.02×10^{-3}	4.171×10^{-4}
r_0	2.242×10^{-4}	9.515×10^{-5}
Overall Error Bound	2.242×10^{-4}	9.515×10^{-5}
$\|x\|$	1.339593	1.339478
Execution Time	29.772 sec.	135.709 sec.

TABLE 6B

	r = 0.1	r = 0.5	r = 1.0
γ	2.528	10.449	30.97
h	6.57×10^{-4}	2.72×10^{-3}	8.047×10^{-3}
r_0	9.516×10^{-5}	9.525×10^{-5}	9.551×10^{-5}
r_1	0.2896	0.06998	0.0235

CHAPTER 5

SUMMARY AND CONCLUSION

Programs have been developed and tested which can produce rigorous solutions to Fredholm and Urysohn integral equations of the second kind. The main goal was to bound the error of the numerical methods and the round-off error automatically. A second goal was, of course, to produce useful error bounds.

The main goal has been accomplished to a large extent. Interval arithmetic proved to be an excellent tool in bounding round-off error. A nice addition in this area would be interval input-output routines which automatically compensate for error in decimal-binary conversions. Interval arithmetic also proved useful in bounding the error of the numerical methods which primarily took the form of quadrature errors. In this regard, the user supplied interval extensions for the functions which defined the problems and for certain Lipschitz constants or derivatives of related functions. Although this may be difficult in certain problems, it is almost certain to be easier and more reliably done than computing the error bounds by hand. Perhaps a symbolic differentiation scheme, which would enable the computer to find the derivatives, would improve the automatic character of the solutions.

Progress has also been made on the goal of useful error bounds for the linear problem. The use of the error bound b_n (Theorem

3.4.1) can represent a significant improvement over the bound B_n (Theorem 3.4.2) which has been used previously.

BIBLIOGRAPHY

1. Anselone, P. M., Collectively Compact Operator Approximation Theory, Prentice-Hall, Inc., New Jersey, 1971.

2. Bryan, C. A., Approximate Solutions to Nonlinear Integral Equations, SIAM J. Num. Anal., 5(1968), pp. 151-155.

3. Control Data Corporation, 6000 Series - Introduction and Peripheral Processors Training Manual, Second Edition, St. Paul, Minnesota, 1968. (Publication no. 60250400)

4. Dieudonné, J., Foundations of Modern Analysis, Academic Press, Inc., New York, 1960.

5. Forsythe, G. E. and C. B. Moler, Computer Solutions of Linear Algebraic Systems, Prentice-Hall, Inc., New Jersey, 1967.

6. Hansen, E., Interval Arithmetic in Matrix Computations, SIAM J. Num. Anal., series B, 2(1965), pp. 308-320.

7. Kantorovich, L. V. and G. P. Akilov, Functional Analysis in Normed Spaces, trans. by D. E. Brown, The Macmillan Company, New York, 1964.

8. Moore, R. E., Interval Analysis, Prentice-Hall, Inc., New Jersey, 1966.

9. Moore, R. H., Approximations to Nonlinear Operator Equations and Newton's Method, Numer. Math., 12(1968), pp. 23-34.

10. Taylor, A. E., Introduction to Functional Analysis, John Wiley and Sons, Inc., New York, 1958.

11. Yungen, W. A., Rigorous Computer Inversion of Some Linear Operators, M.S. Thesis, Oregon State University, Corvallis, 1968.

APPENDICES

APPENDIX A

This appendix contains information on the use of RIA, as described in Section 2.2, in FORTRAN programs. The endpoints of the machine intervals are normalized single precision floating-point numbers which are REAL constants in FORTRAN. In order to handle an interval as a single variable, the TYPE DOUBLE specification is used. This is useful because TYPE DOUBLE constants and variables occupy two computer words and hence can accommodate the two endpoints of an interval. However, caution must be exercised because the usual arithmetic operations between TYPE DOUBLE quantities treat them as double precision numbers.

Let A, B and C be TYPE DOUBLE and let X and Y be TYPE REAL. The following subroutines, which perform various operations on intervals, are provided:

(a) A function subroutine which forms an interval given the endpoints. The statement A = ISET(X,Y) sets A to be the interval [X,Y].

(b) A subroutine which recovers the endpoints. The statement CALL ENDPTS(A,X,Y) returns the left endpoint of A in X and the right endpoint of A in Y.

(c) Four function subroutines which perform RIA.

(i) C=IADD(A,B) performs C=A+B

(ii) C=ISUB(A,B) performs C=A-B

(iii) C=IMUL(A,B) performs C=A·B

(iv) C=IDIV(A,B) performs C=A/B

The fatal errors which can occur with these operations are discussed in Section 2.2. If such an error occurs, then an appropriate message is printed on the user's DAYFILE and the program terminates.

(d) The assignment statement A=B will properly set the interval A equal to the interval B.

(3) Function subroutines which perform various utility operations.

(i) A=INEG(B) performs A=-B (exactly)

(ii) A=IABSV(B) performs $A=|B|^I$ (exactly)

(iii) X=SNORM(A) performs $X=\|A\|^I$ (exactly)

(iv) X=CENTER(A) performs X=c(A)

(v) X=WIDTH(A) performs X=w(A)

The center function uses single precision unrounded arithmetic and always returns a value contained in A. By rounding up if necessary, the width function returns a value which is an upper bound to the exact width of A.

(f) Interval extensions for the exponential and square root functions.

(i) A=IEXPF(B)

(ii) A=ISQRTF(B)

The computer subroutines EXP and SQRT have relative errors of less than 2^{-46} and both functions are monotonic. The interval extensions

IEXPF and ISQRTF are constructed by the method discussed in Section 2.3 using RIA to carry out the interval product that is involved.

APPENDIX B

This appendix contains programming information needed by the user of FRED. The numerical input to FRED is read in free format from cards and the output is written on the line printer. The following program should be used to call FRED:

```
PROGRAM DRIVER(INPUT=/80,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
CALL FRED
STOP
END
```

The numerical input, as discussed in Section 3.5, is read by FRED with the following statements:

```
READ(5,*) AMINUS,APLUS,BMINUS,BPLUS,M,IQFLAG,L

READ(5,*) NAPTS,IFLAG

READ(5,*) SMINUS,SPLUS    (executed NAPTS times)
```

The four required interval function subroutines will now be given using the functions of Test Problem 1 as an example. The names and the number and order of the arguments must be as in this example.

(a) Interval extension of k where $k(s,t) = \frac{1}{2}e^{s-t}$.

```
DOUBLE FUNCTION KERL(SS,TT)
DOUBLE SS,TT,ISET,ISUB,IMUL,TT1,TT2,IEXPF
TT1=ISET(0.5,0.5)
TT2=ISUB(SS,TT)
TT2=IEXPF(TT2)
KERL=IMUL(TT1,TT2)
RETURN
END
```

(b) Interval extension of y where $y(s) = 1$.

```
      DOUBLE FUNCTION YY(SS)
      DOUBLE SS,ISET
      YY=ISET(1.0,1.0)
      RETURN
      END
```

(c) Interval extension of $D_2D_2D_2D_2g$ where $g(s,t) = k(s,t)y(t) = \frac{1}{2}e^{s-t}$. The fourth partial derivative with respect to t is $\frac{1}{2}e^{s-t}$ which happens to equal $k(s,t)$.

```
      DOUBLE FUNCTION DDKY(SS,TT)
      DOUBLE SS,TT,KERL,TT1
      TT1=KERL(SS,TT)
      DDKY=TT1
      RETURN
      END
```

(d) Interval extension of $D_2D_2D_2D_2h$ where $h(s,t,u) = k(s,t)k(t,u) = \frac{1}{4}e^{s-u}$. Thus, the fourth partial derivative with respect to t is identically zero.

```
      DOUBLE FUNCTION DDKK(SS,TT,UU)
      DOUBLE SS,TT,UU,ISET
      DDKK=ISET(0.0,0.0)
      RETURN
      END
```

APPENDIX C

This appendix contains programming information needed by the user of URYSOHN. The numerical input to URYSOHN is read in free format from cards and the output is written on the line printer. The following program should be used to call URYSOHN:

```
PROGRAM DRIVER(INPUT=/80,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
CALL URYSOHN
STOP
END
```

The numerical input, as discussed in Section 4.5, is read by URYSOHN with the following statements:

```
READ(5,*) AMINUS,APLUS,BMINUS,BPLUS,M,IQFLAG,L

READ(5,*) ITRLIM,EPSILON

READ(5,*) T1    (executed N times - this is the initial
                 guess vector for Newton's method)

READ(5,*) NAPTS,IFLAG

READ(5,*) SMINUS,SPLUS    (executed NAPTS times)

READ(5,*) NUVALS

READ(5,*) R    (executed NUVALS times)
```

The nine required function subroutines will now be given using the functions of Test Problem 5 as an example. The names and the number and order of the arguments must be as in this example.

(a) Single precision subroutine for k where $k(s,t,x) = stx^2$.

```
FUNCTION RKER(S,T,X)
RKER=S*T*X*X
RETURN
END
```

(b) Single precision subroutine for D_3k which in this problem is 2stx.

```
FUNCTION RKER3(S,T,X)
RKER3=2.0*S*T*X
RETURN
END
```

(c) Single precision subroutine for y where y(s) = (3/4)s.

```
FUNCTION RY(S)
RY=(3.0*S)/4.0
RETURN
END
```

(d) Interval extension for k where $k(s,t,x) = stx^2$.

```
DOUBLE FUNCTION KER(SS,TT,XX)
DOUBLE SS,TT,XX,IMUL,TT1
TT1=IMUL(SS,TT)
TT1=IMUL(TT1,XX)
KER=IMUL(TT1,XX)
RETURN
END
```

(e) Interval extension for D_3k which in this problem is 2stx.

```
DOUBLE FUNCTION KER3(SS,TT,XX)
DOUBLE SS,TT,XX,IMUL,ISET,TT1
TT1=ISET(2.0,2.0)
TT1=IMUL(TT1,SS)
TT1=IMUL(TT1,TT)
KER3=IMUL(TT1,XX)
RETURN
END
```

(f) Interval extension for D_3D_3k which in this problem is 2st.

```
DOUBLE FUNCTION KER33(SS,TT,XX)
DOUBLE SS,TT,XX,ISET,IMUL,TT1
TT1=ISET(2.0,2.0)
TT1=IMUL(TT1,SS)
KER33=IMUL(TT1,TT)
RETURN
END
```

(g) Interval extension for y where y(s) = (3/4)s.

```
DOUBLE FUNCTION YY(SS)
DOUBLE SS,ISET,IMUL,TT1
TT1=ISET(0.75,0.75)
YY=IMUL(TT1,SS)
RETURN
END
```

(h) Interval extension for D_2D_2g where $g(s,t) = k(s,t,x_0(t)) = ((3/4) + \sum_{j=1}^{n} w_{nj}t_{nj}z_{nj}^2)^2st^3$. The second partial derivative is required since the second order quadrature option is being used. The value of n and the intervals containing the t_{nj}, w_{nj} and z_{nj}, j = 1,2,...,n, are stored in blank common. The user can access them as follows:

```
COMMON N,TI(51),WI(51),ZI(51)
DOUBLE TI,WI,ZI
```

Also, the interval valued function $x_0^I(s^I)$ is available from the program as XXZF(SI). In general, it will be necessary to use the t_{nj}, w_{nj} and z_{nj} from the program, but in this problem it is not necessary since the factor $((3/4) + \sum_{j=1}^{n} w_{nj}t_{nj}z_{nj}^2)$ is $x_0(1)$. The program for $D_2D_2g(s,t) = 6(x_0(1))^2st$ can be written as follows:

```
DOUBLE FUNCTION DDKER(SS,TT)
DOUBLE SS,TT,XXZF,ISET,IMUL,TT1,TT2
TT1=ISET(1.0,1.0)
TT1=XXZF(TT1)
TT1=IMUL(TT1,TT1)
TT1=IMUL(TT1,SS)
TT2=ISET(6.0,6.0)
TT1=IMUL(TT1,TT2)
DDKER=IMUL(TT1,TT)
RETURN
END
```

(i) Interval extension for D_2D_2h where $h(s,t,u) = k(s,t,x_0(t))k(t,u,x_0(u)) = 4((3/4) + \sum_{j=1}^{n} w_{nj}t_{nj}z_{nj}^2)^2 su^2t^3$. The program for $D_2D_2h(s,t,u) = 24(x_0(1))^2su^2t$ could be as follows:

```
      DOUBLE FUNCTION DDKK(SS,TT,UU)
      DOUBLE SS,TT,UU,XXZF,ISET,IMUL,TT1,TT2
      TT1=ISET(1.0,1.0)
      TT1=XXZF(TT1)
      TT1=IMUL(TT1,TT1)
      TT2=IMUL(UU,UU)
      TT1=IMUL(TT1,TT2)
      TT1=IMUL(TT1,TT)
      TT1=IMUL(TT1,SS)
      TT2=ISET(24.0,24.0)
      DDKK=IMUL(TT1,TT2)
      RETURN
      END
```