Software Notes:
A Collection of Mathematical Software
Reference Papers
Volume 1, June 1984

\[ G(z) = e^{\ln G(z)} = \exp \left( \sum_{k \geq 1} \frac{S_k z^k}{k} \right) = \prod_{k \geq 1} e^{S_k z^{k/k}} \]

\[ = \left( 1 + S_1 z + \frac{S_1^2 z^2}{2!} + \cdots \right) \left( 1 + \frac{S_2 z^2}{2} + \frac{S_2^2 z^4}{2! \cdot 2!} + \cdots \right) \cdots \]

\[ = \sum_{m \geq 0} \left[ \sum_{k_1, k_2, \ldots, k_m \geq 0} \frac{S_1^{k_1}}{k_1!} \frac{S_2^{k_2}}{2^{k_2} k_2!} \cdots \frac{S_m^{k_m}}{m^{k_m} k_m!} \right] z^m \]

\[ \int \frac{dx}{ae^{mx} - be^{-mx}} = \begin{cases} 
\frac{1}{2m\sqrt{ab}} \log \frac{\sqrt{ae^{mx}} - \sqrt{b}}{\sqrt{ae^{mx}} + \sqrt{b}} & (a > 0, b > 0) \\
\frac{1}{m\sqrt{ab}} \tanh^{-1} \left( \frac{\sqrt{a}}{\sqrt{b}} e^{mx} \right) & (a > 0, b > 0) \\
\frac{-1}{m\sqrt{ab}} \coth^{-1} \left( \frac{\sqrt{a}}{\sqrt{b}} e^{mx} \right) & (a > 0, b > 0)
\end{cases} \]
ACKNOWLEDGEMENTS

The following people contributed articles to this collection: Russ Rew, James Curry, Richard Valent, John Adams, and Michael Pernice. Thanks are due them for their original work and for the time they took to update their respective articles and review editorial changes. Michael Pernice was the technical editor and the principal author of this work. Robert Nicol was the principal editor; Greg McArthur was the assisting editor. David Maxey provided preliminary editorial assistance.
### Table of Contents

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>&quot;Bessel Functions&quot; by Russ Rew.</td>
<td>2</td>
</tr>
<tr>
<td>Originally published in Computing Facility Notes #59 (July 1979)</td>
<td></td>
</tr>
<tr>
<td>&quot;Software Notes Update: Bessel Functions&quot; by James Curry</td>
<td>5</td>
</tr>
<tr>
<td>Originally published in the April, 1983 issue of <em>The Record</em></td>
<td></td>
</tr>
<tr>
<td>&quot;New Constrained Optimization Package Available&quot; by Michael Pernice</td>
<td>9</td>
</tr>
<tr>
<td>Originally published in the September, 1983 issue of <em>The Record</em></td>
<td></td>
</tr>
<tr>
<td>&quot;New Software for Elliptic PDEs&quot; by John C. Adams</td>
<td>14</td>
</tr>
<tr>
<td>Originally published in Computing Facility Notes #55 (Nov. 1979)</td>
<td></td>
</tr>
<tr>
<td>&quot;Fast Fourier Transforms&quot; by James Curry and Richard Valent</td>
<td>16</td>
</tr>
<tr>
<td>Originally published in the February, 1982 issue of <em>The Record</em></td>
<td></td>
</tr>
<tr>
<td>&quot;New FFT Available for the CRAY-1 Machines&quot; by Michael Pernice</td>
<td>20</td>
</tr>
<tr>
<td>Originally published in the July, 1983 issue of <em>The Record</em></td>
<td></td>
</tr>
<tr>
<td>&quot;General Software for PDEs&quot; by Michael Pernice</td>
<td>22</td>
</tr>
<tr>
<td>Previously unpublished</td>
<td></td>
</tr>
<tr>
<td>&quot;Tridiagonal Matrix Solvers at NCAR&quot; by Michael Pernice</td>
<td>36</td>
</tr>
<tr>
<td>Originally published in the October, 1983 issue of <em>The Record</em></td>
<td></td>
</tr>
</tbody>
</table>
Introduction

The past several years have seen rapid growth in the amount of public library software which is available on the Scientific Computing Division's computers at NCAR. At present the Software and Libraries Group in SCD maintains 20 separate libraries which total more than one million lines of source code, a fourfold increase over the last 12 years. Both individual software packages and entire new libraries have contributed to this growth. The on-line Software Catalog, available to users since September 1980, is continually updated to reflect software additions, modifications, and deletions.

The release of new software is generally made known to users in the Daily Bulletin and in the SCD monthly newsletter, The Record, formerly known as the Computing Facility Notes. These Software Notes articles provide information such as access procedures, comparison of algorithms, and location of documentation. Performance evaluations involving timing and accuracy tests are also included and, whenever possible, specific recommendations are made as to which software to use. Past topics have included software for stiff ordinary differential equations, fast Fourier transforms, and software for full and sparse systems of linear equations. Since the Software Catalog is limited to short descriptions, the Software Notes articles have supplied information that is helpful in deciding what software is appropriate for a given application. Unfortunately, locating a useful article is difficult, since the articles are scattered over many back issues of The Record and the Computing Facility Notes. The purpose of this collection is to provide an easily accessible source of information on various types of software available at NCAR.

This technical note is a compilation of several Software Notes articles which have appeared in past issues of the Computing Facility Notes and The Record. The articles have been updated to include software which has been acquired since their first publication. In addition, the article "General Software for Partial Differential Equations" is published here for the first time. By collecting and updating information which previously had been scattered over several issues of The Record and the Computing Facility Notes, this technical note will serve as a centralized source of information about public library software which is available at NCAR. Subsequent volumes of the NCAR Software Notes Collection will contain updates to this volume, as well as additional articles of interest.

Editor's Note: At the time of original publication of these articles, all of the authors were full or part-time employees at NCAR.
Bessel Functions
by Russ Rew

The topic of software available at NCAR for special functions divides naturally into two subtopics: Bessel functions and all other special functions.

Bessel functions are among the most commonly encountered special functions in applied mathematics. They occur frequently in the solution of ordinary differential equations, in the evaluation of definite integrals, and in the solution of partial differential equations expressed in cylindrical or spherical coordinates. A wide variety of library software is available at NCAR for evaluating Bessel functions.

The four most common Bessel functions are denoted by the letters I, J, K, and Y; for their definitions and elementary properties, see, for example, the Handbook of Mathematical Functions edited by Abramowitz and Stegun, National Bureau of Standards, 1964. J and Y are the Bessel functions of the first and second kind. I and K are the modified Bessel functions of the first and second kind. Each of these is really a function of two parameters: the order (denoted by a subscript) and the argument. Both the order and argument may, in general, be complex numbers. Complex arguments do occur in the solution of real physical problems and some software is available for this case. More frequently the evaluation of Bessel functions with integer order and real argument suffices; this is the problem addressed by most of the available library Bessel functions.

Real Argument, Integer Order

For order 0 or 1 and real argument, excellent software is available from FUNPACK, IMSL, and AMOSLIB. The first two sources offer essentially identical software, because IMSL (International Mathematical and Statistical Libraries, Inc.) has taken the original FUNPACK routines and modified them to conform to IMSL library conventions. The FUNPACK routine names are BESI0, BESI1, BESJ0, BESJ1, BESK0, and BESK1. Corresponding IMSL routines are MMBSI0, MMBSI1, MMBSJ0, MMBSJ1, MMBSK0, and MMBSK1. In each case, the last two characters of the name identify the function and order. The four AMOSLIB packages BESI01, BESJ01, BESK01, and BESY01 combine capabilities for order 0 or 1 in a single package; different approximations for the functions are used in the AMOSLIB software.

The FUNPACK routines were originally developed as part of a national effort to test high-quality mathematical software. Like most current special function software, these routines are highly machine-dependent; the approximations in the current version of FUNPACK at NCAR were modified specifically to achieve optimal accuracy for CRAY-1 computers. AMOSLIB routines were originally developed for Control Data computers. AMOSLIB software packages have been made available for the CRAY-1 computers by making the minimal changes necessary to the FORTRAN source to get it to successfully compile and run on the CRAY-1. Users of these AMOSLIB packages on the CRAY-1 should be aware that this software has not been as extensively tested or certified on the CRAY-1 as it has on Control Data computers. In particular, tests for valid ranges of
arguments, underflow, and overflow are likely to restrict the domain of these functions more than is necessary on the CRAY-1.

All of the FUNPACK routines and their IMSL counterparts deliver accuracy close to machine precision. The AMOSLIB routines provide somewhat less accuracy for arguments in their domain, but are still comparable in accuracy to the system SIN, COS, and EXP functions. In each of the packages that compute I or K functions, the domain of the functions is extended to avoid overflow by the provision of an option to scale the function values by an exponential.

A sequence of Bessel function values for consecutive integer orders of a single argument is needed for some problems. Such a sequence may be required, for example, in the evaluation of a series that includes Bessel function coefficients in each term. For real arguments, the recommended routines supporting this capability are BESR in ULIB (for I and J functions), BESYN in AMOSLIB (for Y functions), and BESKN in AMOSLIB (for K functions). BESR was adapted from a package that is unusual in that it is relatively portable; it can be adapted to run on a different machine by changing only a few well-isolated machine-dependent constants. It is also very accurate over its input domain. A double precision version of BESR is available as BESLRI in XLIB. A version of BESR that requires no workspace arguments or error flags is available in the PORT library in the functions BESRI and BESRJ, which have double precision counterparts DBESRI and DBESRJ, respectively.

Real Argument, Real Order

For the more general capability of handling arbitrary real order and real argument, library coverage is not quite so complete. The need for these Bessel functions arises from series whose terms involve a sequence of Bessel functions with increasing real orders or from other problems that lead to Bessel functions of fractional orders. For example, solutions to the wave equations on a sphere may involve Bessel functions of order 1/3 and 2/3.

The FUNPACK routine BESYN and the analogous IMSL routine MMBSYN provide values of Y at a sequence of increasing positive real orders for a positive real argument. The AMOSLIB packages BESI and BESJ provide a similar capability for I and J functions. The AMOSLIB routines SBESI provides a sequence of scaled I Bessel functions of a real argument, where the scaling may be specified by the user. The IMSL function MMBSKR will evaluate K Bessel functions of real order and real argument. Bessel functions of fractional order 1/3 and 2/3 may be computed using the Airy functions Ai(x) and Bi(x) and their derivatives Ai'(x) and Bi'(x) (the appropriate identities are given in Abramowitz and Stegun). Four packages for calculating the Airy functions and their derivatives for arbitrary real arguments are the AMOSLIB routines AIRY, BAIRY, DAIRY, and DBAIRY.

Complex Argument

Two AMOSLIB packages provide Bessel functions of orders 0 and 1 for complex arguments. CJ01BS in AMOSLIB evaluates J_0 and J_1 for arbitrary complex arguments. CJYHBS in AMOSLIB computes J_n, Y_n, or H_n, where n=0 or n=1 for arbitrary complex arguments; H is the Struve function.
For arbitrary integer order, the NSSL package BESC in ULIB computes sequences of I and J functions of complex arguments. BESC shares with the NSSL package BESR the attributes of portability and high accuracy. A double precision version of BESC is available in BESLCI in XLIB. The subroutines BESCI and BESCJ in the PORT library provide the same approximations and a simpler argument list than BESC. Their double precision counterparts are DBESCI and DBESCJ respectively.

The BESSKA routine in XLIB is available for computing the K function of arbitrary real order and complex argument. No capabilities are currently supported for evaluating I, J, or Y functions of arbitrary real order and complex argument, or for evaluating any functions of complex order.

Related Functions

The Airy functions, Ai and Bi, and Kelvin functions, ber, bei, ker, and kei, are closely related to Bessel functions. It may be possible to transform a calculation involving Bessel functions into a simpler evaluation of Airy or Kelvin functions using the identities that define these functions. Airy functions are associated with Bessel functions of order 1/3 and 2/3. In addition to the previously mentioned AMOSLIB files AIRY, BAIRY, DAIRY, and DBAIRY for calculating Airy functions of real arguments, an XLIB package, CAIRY, is available for evaluating the Airy function Ai(z) and its derivative Ai'(z) for complex arguments.

Kelvin functions are associated with Bessel functions with arguments that are real multiples of l+i or -l+i. MMKEL0 and MMKEL1 in IMSL provide values for ber, bei, ker, and kei functions of order 0 or 1 for real arguments. KRKI in AMOSLIB provides different approximations for ker and kei functions of order 0 and positive real argument. Derivatives of order 0 Kelvin functions are computed by the IMSL package MMKELD and the AMOSLIB package DKRKI. BERBEI in AMOSLIB returns a sequence of real orders of ber or bei functions for a real argument.

The only supported software for evaluating integrals of Bessel functions is FINTKO in AMOSLIB, which calculates the integral of K0 on the intervals (0,x] or [x,\infty) for positive x. JROOTS in AMOSLIB returns the positive zeros of the J function for any non-negative real order.

Documentation

Documentation for almost all of the programs mentioned above is available by through use of the GETDOC facility. The only exception is the PORT library; only binary versions of PORT routines are accessible by the user. Manuals for PORT and AMOSLIB are available for reference in the Consulting Office library. PORT documentation in the form of individual chapters organized by subject is also available to users.
Software Notes Update: Bessel Functions
by James H. Curry

Introduction

This article is an update of the Bessel functions paper by Russ Rew (originally published in Computing Facility Notes No. 59), which appears on page 2 in this Collection. Since the first article was published, NCAR has acquired the NAG and SLATEC libraries and new software devoted to Bessel functions, such as the COBY routine in XLIB. Further, this article also contains the results of extensive timing tests to aid the user in choosing among the available routines.

The earliest systematic studies of the cylindrical functions now known as the Bessel functions can be traced to F. Bessel in 1824. Apparently what are known as the integer Bessel functions \( J_0 \) appeared in the memoirs of L. Euler (1764) where he considered the problem of a stretched circular membrane. They were also studied by J. Lagrange while considering problems in celestial mechanics (1769), and by the Bernoulli brothers (James and Daniel) during the period 1694-1732.

Bessel functions are solutions to the second order ordinary differential equation:

\[
\frac{d^2 z}{dx^2} + \frac{1}{x} \frac{dz}{dx} + \left(1 - \frac{f^2}{x^2}\right) z = 0
\]

which is called the Bessel function of order \( f \), and \( z \) is a function of \( x \).

The simplest Bessel functions are of nonnegative integer order in which \( f \) is an integer \( n \) \((n=0, 1, 2,\ldots)\). These are called Bessel functions of the first kind of integer order. If the sign preceding \( f \) is changed in the equation above, the solutions are the modified Bessel functions.

Since the defining relation for Bessel functions is a second order differential equation it is not a surprise that there are solutions which are linearly independent of \( J_n \). These functions are denoted by \( Y_n \) and are called Bessel functions of the second kind. Bessel functions of the second kind have the following defining relation:

\[
Y_n(z) = \frac{J_n(z) \cos(n \pi) - J_{-n}(z)}{\sin(n \pi)}
\]

and the associated modified Bessel functions of the second kind are denoted by \( K_n \).
Bessel functions of the first and second kind satisfy the simple recurrence relationships:

\[ J_{n-1}(z) + J_{n+1}(z) = \frac{2n}{z} J_n(z) \]

\[ J_{n-1}(z) - J_{n+1}(z) = 2 \frac{d}{dz} J_n(z) \]

hence Bessel functions of the first kind are all expressible in terms of \( J_0 \) and \( J_1 \). Similar recurrence relations hold for the modified Bessel functions of the first kind \( I_0 \) and \( I_1 \).

**NCAR Software**

NCAR currently has five dozen routines available in six mathematical libraries which can aid in various aspects of computing when Bessel functions are required.

Most of the libraries or packages have a full complement of Bessel function software. In particular, most have code for computing Bessel functions of the first and second kind as well as the modified Bessel functions. Further, several of the routines implement an algorithm due to Sookne [4] while others trace their origin to the National Activity to Test Software [2],[3]. The fundamental reference on Bessel functions is [5].

It should be noted that while there are some intersections among the various package and library routines each has its own subtleties; for example, IMSL routines must be typed either real single or double precision, otherwise incorrect results may be obtained. It is therefore strongly recommended that a potential user of any of the special function software read the documentation carefully (see "Bessel Functions" on page 2 for more details).

**Recommendations**

Since NCAR has such an extensive collection of Bessel function software, the following suggestions are offered for choosing among them:

1. If the application only requires the computation of the Bessel functions \( J_{0,1} \) or \( I_{0,1} \) it is recommended that function versions be used.

2. If the user’s application requires Bessel functions of order greater than one it is recommended that subroutine versions be used.

The above suggestions are primarily based on the empirical observation that the function evaluation software is approximately five times faster than the analogous subroutine callable software where the evaluation of Bessel functions are concerned. Further, the FORTRAN-callable subroutines were designed to compute more than a single Bessel function and should be expected to require more time for such things as initialization. There is a natural division of the routines into two groups: callable subroutines and function evaluations. The following table contains the timing test results (by library) of running the integer order Bessel function software available at
NCAR on the CRAY-1 computers. (The figures for AMOSLIB also apply to the SLATEC Bessel function routines).

### TABLE I

<table>
<thead>
<tr>
<th>SUBROUTINE CALLABLE $J_n$</th>
<th>SUBROUTINE CALLABLE $I_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCARLB</td>
<td>NCARLB</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>AMOSLIB</td>
<td>AMOSLIB</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1.23</td>
<td>1.23</td>
</tr>
<tr>
<td>PORT</td>
<td>PORT</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1.43</td>
<td>1.45</td>
</tr>
</tbody>
</table>

### FUNCTION EVALUATION $J_{0,1}$

<table>
<thead>
<tr>
<th>FUNCTION EVALUATION $J_{0,1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMSL</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1.00</td>
</tr>
<tr>
<td>NAG</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1.06</td>
</tr>
<tr>
<td>AMOSLIB</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>1.93</td>
</tr>
<tr>
<td>FUNPACK</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>2.12</td>
</tr>
</tbody>
</table>

The entries in TABLE I are divided by category and ranked by speed of execution. The fastest entry in each category is used as a base value of 1.00. For example, of the available software for computing $I_0$ or $I_1$ using function evaluations, the routines in the NAG library were 0.54% faster than the corresponding IMSL routine in the same subgroup.

A concise tabulation of types and location of the various software is presented in Table II.

### TABLE II

<table>
<thead>
<tr>
<th>J</th>
<th>I</th>
<th>Y</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMOSLIB</td>
<td>CIFS</td>
<td>CIFS</td>
<td>I</td>
</tr>
<tr>
<td>FUNPACK</td>
<td>I</td>
<td>IS</td>
<td>IF</td>
</tr>
<tr>
<td>IMSL</td>
<td>CIF</td>
<td>CIFS</td>
<td>IF</td>
</tr>
<tr>
<td>NAG</td>
<td>I</td>
<td>IS</td>
<td>I</td>
</tr>
<tr>
<td>NCARLB</td>
<td>CI</td>
<td>CIS</td>
<td>N</td>
</tr>
<tr>
<td>PORT</td>
<td>CI</td>
<td>CI</td>
<td>N</td>
</tr>
<tr>
<td>ULIB</td>
<td>CI</td>
<td>CI</td>
<td>N</td>
</tr>
<tr>
<td>XLIB</td>
<td>N</td>
<td>FS</td>
<td>N</td>
</tr>
</tbody>
</table>

June 1984 -7- Software Notes
Notes:

The SLATEC Bessel function routines are the same as those in AMOSLIB.

The various columns of the table indicate whether:

A library has software for the evaluation of Bessel functions of a complex argument (C), integer (I) or fractional (F) order, or exponentially scaled (S) Bessel functions of various kinds, while (N) indicates that software of a specified type is not available in a corresponding library. Finally, though not indicated in TABLE II, some libraries do provide double precision software and the user should refer to the NCAR Software Catalog for more details.

Catalog Documentation

A short description of available Bessel function routines may be obtained by use of the CRAY-1 utility GETDOC:

GETDOC,LIB=DOCLIB,DOC=B5I.

References


New Constrained Optimization Package Available
by Michael Pernice

A fundamental problem frequently encountered in mathematical modelling is that of optimization: finding the minimum or maximum value of a real-valued function of one or more variables. Without loss of generality we may consider only minimization problems. When only one independent variable is involved, and the function is continuously differentiable, the problem is straightforward (recall Calculus I). The numerical implementation is also straightforward, requiring only some root-finding technique.

In higher dimensions the same ideas can be applied. The derivative is replaced by the gradient. The second derivative, which provides information about the shape (more precisely, the concavity) of a function, is replaced by the matrix of second partial derivatives, the Hessian. The requirement that the second derivative be positive for a minimum to occur is replaced by the requirement that the Hessian be a positive definite matrix at the point in question. The theory is fairly complete, but numerical implementation can be difficult and expensive. The essence of the numerical technique is to choose a search direction and then solve the one-dimensional problem that results from restricting to that direction. This problem is referred to as unconstrained optimization. See [2] for a brief but elegant treatment of the theory of optimization in higher dimensions.

The complexity of the problem is greatly increased by restricting the domain of the function to be minimized. For example, one might seek to find the minimum value of a function subject to the restriction that all of its independent variables must be non-negative. This problem is referred to as constrained optimization.

The most general formulation of a constrained optimization problem is:

\[
\begin{align*}
\text{minimize:} & \quad f(x_1, \ldots, x_n) \\
\text{subject to:} & \quad g_j(x_1, \ldots, x_n) = 0 \quad j=1,\ldots,m_{\text{eq}} \\
& \quad g_j(x_1, \ldots, x_n) \geq 0 \quad j=m_{\text{eq}}+1,\ldots,m.
\end{align*}
\]

We call \( f \) the objective function, \( g_1, \ldots, g_{m_{\text{eq}}} \) the equality constraints, and \( g_{m_{\text{eq}}+1}, \ldots, g_m \) the inequality constraints. The feasible region consists of those vectors which satisfy the constraints. The use of constraints makes the problem of
Optimization more difficult because the solution usually lies on the boundary of the feasible region; the old rule of seeking where the gradient vanishes is no longer necessarily applicable. For example, consider the one-dimensional constrained optimization problem:

minimize: \[ f(x) = x^2 \]
subject to: \[ x-1 \geq 0 \]
\[ 2-x \geq 0. \]

In principle, it is possible to implicitly solve for some of the independent variables in the equality constraints in terms of the rest of the variables. This will reduce the dimension of the problem and eliminate the equality constraints. In practice, this is usually quite difficult to do. What is often done instead is to somehow incorporate all of the constraints into the objective function and then apply some technique of unconstrained optimization to the "augmented" objective function.

There are several ways of accomplishing this. One way is to create a "penalty function" which is based on the constraints and which has the effect of increasing the value of the objective function if a constraint is violated. A major problem in this approach is constructing a penalty function which is well-behaved and at the same time performs the intended task in a numerically efficient manner. Another technique is based on the classical approach of Lagrange multipliers. See [1] for a discussion of the use of Lagrange multipliers in constrained optimization problems. In addition, [6] contains an informal discussion of several algorithms and a comparison of their performance under various implementations.

**Constrained Optimization at NCAR**

There are several library subroutines at NCAR which address the constrained optimization problem. One is a new acquisition, VMCON, which will be discussed shortly. The rest are found in the NAG program library. The techniques which are applied involve a modified Newton method, a conjugate gradient method, or a quasi-Newton method (sometimes referred to as a variable metric technique). All of these techniques are discussed in [1].

Recently, a new algorithm for dealing with constrained optimization has been developed by M. J. D. Powell [4,5] which combines the best of several techniques and which features quadratic convergence under certain assumptions [4]. Powell's algorithm utilizes a variable metric technique, and in general his algorithm requires fewer function evaluations. This is an important property of an optimization algorithm, for it reflects the efficiency of the algorithm's choice of search directions. An implementation of Powell's algorithm is compared to several other algorithms in [6], where it received high marks for its performance. NCAR has acquired an implementation of the Powell algorithm called VMCON, written by Roger L. Crane, Kenneth E. Hillstrom and Michael Minkhoff of the National Energy Software Center at Argonne National Laboratory.
Extensive timing and accuracy tests which compare VMCON and NAG routines E04VAF and E04WAF were developed by the author and performed on NCAR’s CRAY-1 computers. The choice of E04VAF and E04WAF from among the 17 constrained optimization routines in the NAG library was based on the fact that they implement a quasi-Newton algorithm. Following is a table which summarizes the results which were obtained. A total of 19 tests were run, using seven test functions and fifteen sets of constraints.

<table>
<thead>
<tr>
<th></th>
<th>report success</th>
<th>actual success</th>
<th>average function evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMCON</td>
<td>13</td>
<td>8</td>
<td>33</td>
</tr>
<tr>
<td>E04VAF</td>
<td>9</td>
<td>9</td>
<td>95</td>
</tr>
<tr>
<td>E04WAF</td>
<td>5</td>
<td>5</td>
<td>187</td>
</tr>
</tbody>
</table>

This table indicates that VMCON failed to find a "true" solution, corresponding to the global minimum, in nearly 40% of the test problems on which it reported a successful return. However one can only conclude that a local minimum, satisfying the stopping criteria, had been found, and not that the algorithm itself failed.

On those test problems where VMCON did not report a successful return, the error return code provided some information as to why the program was failing. On the other hand when the NAG routines did not report successful completion, the error return code indicated only that the maximum number of function evaluations (a parameter set by the user) had been exceeded, or that too many cycles of the algorithm had been completed without making progress towards a solution. In some of these cases the NAG routines had actually found a solution which would have been acceptable to VMCON. The following table summarizes these results for the NAG routines.

<table>
<thead>
<tr>
<th></th>
<th>function evaluation limit reached</th>
<th>acceptable solution actually found</th>
</tr>
</thead>
<tbody>
<tr>
<td>E04VAF</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>E04WAF</td>
<td>14</td>
<td>4</td>
</tr>
</tbody>
</table>

It is difficult to ascertain whether the NAG routines were actually making progress towards a solution in those cases where an acceptable solution had not yet been found. The error parameter for the NAG routines contains little useful information. A user must resort to using a monitoring routine and a reasonable amount of expertise in reading its output in order to establish how the optimization algorithm is behaving. On the other hand, VMCON has somewhat better error return codes which indicate several different difficulties that may be encountered while solving the problem, such as an inefficient line search, calculation of an uphill search direction, or the presence of a singular matrix in the updating formula.
Evaluation and Recommendations

Experience with VMCON has shown it to be quite simple to use. There are only two parameters which the user must set. The user must also provide a subroutine which evaluates the function and the constraints, and their gradients. VMCON also consistently required the fewest number of function evaluations.

The use of NAG constrained optimization routines requires the user to supply at least three subroutines: a routine for evaluating the objective function and its gradient, a routine for evaluating the constraints and their gradients, and a routine for monitoring the progress of the algorithm by requesting the printing of internal variables, such as the number of iterations performed, or the current values of the Lagrange multipliers. (By contrast VMCON can provide this information by simply setting an integer flag). A programmer may choose between an implementation of an algorithm that numerically approximates the Hessians and an implementation which calls user-provided subroutines that evaluate the Hessians. The latter option, however, can be quite time-consuming to implement, especially when there are many variables and/or many constraints. In using a NAG constrained optimization routine the user must supply several parameters in addition to those determining the stopping criterion and the maximum number of function evaluations. See [3] for a description of these.

In summary, the NAG routine E04VAF was much more successful at finding a global minimum than either VMCON or E04WAF. In the above tests E04WAF obtained values for the Hessians from user-supplied routines and E04VAF approximated the Hessians internally. As can be seen from the above tables, E04VAF actually performed better than E04WAF, requiring fewer function evaluations than E04WAF and reporting success more frequently. In contrast, VMCON was much simpler to implement and interpret and may serve well as an initial tool for preliminary study of newly-posed problems.

Implementation Notes

Documentation for VMCON is available on-line by using:

GETDOC(LIB=XLIB, DOC=VMCON)

There is also documentation on VMCON in the Desperate Users Catalog, which may be found in the Consulting Library.

Before using VMCON, one must first obtain and compile the source code by including the following statements:

GETSRC (LIB=XLIB, FILE=VMCON, L=VMCON)
CFT,I=VMCON.

Software Notes -12- June 1984
The NAG library can be accessed by specifying LIB=NAG on the LDR card. Online documentation is also available via the GETDOC utility. For example, including:

GETDOC,LIB=NAG,DOC=E04VAF.

in a job submitted to either of the CRAY-1 computers will put the documentation for E04VAF in $OUT.

References


June 1984

Software Notes
CROSEL and LIPTIC are two routines which solve linear two-dimensional non-separable elliptic partial differential equations on a rectangle. The general form of the equation is:

\[(i) \quad c_{xx}(x,y)U_{xx} + c_{xy}(x,y)U_{xy} + c_{yy}(x,y)U_{yy} +
+ c_x(x,y)U_x + c_y(x,y)U_y + c_f(x,y)U(x,y) = f(x,y)\]

where \(A < x < B\) and \(C < y < D\).

LIPTIC is designed to solve \(i\) when \(c_{xy}(x,y) = 0\) for all \(x,y\) (i.e., no cross derivative is present). The tasks these routines perform include the following:

1. Automatic discretization of \(i\) incorporating boundary conditions which can be periodic, specified, or mixed.

2. Computation of the solution of the matrix equation arising from the discretization of \(i\) directly (these are not iterative methods).

3. The option of calculating a fourth-order approximation in addition to the usual second-order finite difference approximation.

The approximation is generated on a uniform grid with no restriction on size in either direction. The boundary conditions for CROSEL along \(x = A\) can be of the form:

1. \(U(x + B - A, y) = U(x,y)\) for all \(x,y\) periodic

2. \(U(A,y)\) is specified for all \(y\) Dirichlet

3. \(a(y)\delta U/\delta x + b(y)\delta U/\delta y + c(y)U(A,y) = g_A(y)\) mixed

\(b(y)\) must be zero in (3) for LIPTIC, which does not allow oblique derivative boundary conditions. The other boundary conditions are the same with the exception that periodic boundary conditions in the \(y\) direction are not allowed. The automatic discretization of \(i\) yields a block tri-diagonal matrix equation. While generating this matrix, both CROSEL and LIPTIC do an LU-decomposition using matrix recursion formulas. Vectorized CAL subroutines are used to efficiently solve these equations on the CRAY-1. The codes are especially fast on non-initialization calls (i.e., calls which do not require LU-decomposition) with new right-hand sides only. The matrix equation is solved either in core or out of core depending on the available CRAY-1 work space that the user can provide. In the out of core case an attempt is made to cover buffering with computation. This may or may not be efficient (I/O bound) depending on the length of the work space provided.

Software Notes -14- June 1984
If \( C_{xy}(x,y) = 0 \) and if the coefficients \( c_{xx}, c_x \) depend only on \( x \); \( c_{yy}, c_y \) depend only on \( y \); and if \( c_f(x,y) = c_1(x) + c_2(y) \) then (i) is separable. In this case, one of the two routines (SEPELI or SEPX4) in FISHPAK should be used instead of CROSEL or LIPTIC. They utilize the cyclic reduction algorithm and are much faster and require far less storage.

Both CROSEL and LIPTIC compute fourth-order approximations using the technique of difference corrections. The second-order approximation to (i) is used to compute a second-order approximation to the truncation error. After adding this "correction" to the original right-hand side, a new solution of the block tri-diagonal matrix equation yields a fourth-order approximation to (i). If the cost of computation is measured as a function of allowable error then the advantage of the fourth-order option in reduction of both storage and computation is substantial.

The table below summarizes some accuracy and timing results for LIPTIC on the CRAY-1 computer. CROSEL is 20 to 30\% slower than LIPTIC on the same size grids. (i) was solved on an \( N \) by \( N \) uniform grid covering \( 0 < x < 2\pi \), \( 0 < y < 1 \). The exact solution is taken to be \( U(x,y) = (\sin(x) + 1)(y^3 - y + 1) \). Periodic boundary conditions in the \( x \)-direction and specified (Dirichlet) boundaries at \( y = 0,1 \) are used. INTL - O(1) corresponds to initialization (non-initialization) calls. IORDER = 2(4) flags second- (fourth-) order approximations. Times are given in seconds. The coefficients are:

\[
\begin{align*}
    c_{xx} &= \frac{(y + 1)^2}{4\pi^2}, \\
    c_x &= \frac{2(y + 1)}{4\pi^2}, \\
    c_{yy} &= (x + 1)^2, \\
    c_y &= 2(x + 1), \text{ and} \\
    c_f &= -(x + y).
\end{align*}
\]

<table>
<thead>
<tr>
<th>N</th>
<th>INTL</th>
<th>IORDER</th>
<th>TIME</th>
<th>ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>0</td>
<td>2</td>
<td>2.7E-1</td>
<td>2.4E-3</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>2</td>
<td>3.0E-2</td>
<td>2.4E-3</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>4</td>
<td>8.3E-2</td>
<td>1.0E-6</td>
</tr>
<tr>
<td>64</td>
<td>0</td>
<td>2</td>
<td>2.5E0</td>
<td>5.8E-5</td>
</tr>
<tr>
<td>64</td>
<td>1</td>
<td>2</td>
<td>2.2E-1</td>
<td>5.8E-5</td>
</tr>
<tr>
<td>64</td>
<td>1</td>
<td>4</td>
<td>5.4E-1</td>
<td>2.9E-7</td>
</tr>
<tr>
<td>128</td>
<td>0</td>
<td>2</td>
<td>2.6E+1</td>
<td>1.4E-5</td>
</tr>
<tr>
<td>128</td>
<td>1</td>
<td>2</td>
<td>1.7E0</td>
<td>1.4E-5</td>
</tr>
<tr>
<td>128</td>
<td>1</td>
<td>4</td>
<td>3.8E0</td>
<td>5.5E-8</td>
</tr>
</tbody>
</table>

CROSEL and LIPTIC are available as FORTRAN source in CRAYLIB and as object code in CRAY binary $NCARLB$. The documentation, giving details on how to use them, may be obtained by including the following statements in a job submitted to either of NCAR's CRAY-1 computers:

GETDOC, LIB=CRAYLIB, DOC=CROSEL.
Fast Fourier Transforms
by James H. Curry
and Richard Valent

Introduction

The term "Fast Fourier Transform" refers to certain algorithms for computing the discrete Fourier transform of a set of data. The adjective "Fast" refers to the fact that the conventional Fourier Transform algorithm, e.g. using Horner's scheme to evaluate the Fourier components, requires on the order of N-squared operations while the Fast Fourier Transform requires on the order of N log N operations, where N is the length of the data vector to be transformed, and an operation is defined to be a complex multiplication followed by a complex addition.

Versions of the Fast Fourier Transform (FFT) algorithm have been rediscovered several times. An algorithm was first discovered by K. Runge [4] and subsequently described in an article published in 1924 by Runge and König [5]. It seems that an FFT algorithm was also discovered by G. Danielson and C. Lanczos in their work on X-ray scattering in 1942; however, their paper went unnoticed until the mid-1960's. See [1] for a brief history of FFTs.

Since its reintroduction into the scientific community in the fundamental article of Cooley and Tukey [2], the Fast Fourier Transform has become an indispensable tool in many applications of scientific computing. Perhaps the most widely known application is in spectral prediction models, such as the COMMUNITY CLIMATE MODEL currently being implemented at NCAR (see [3] for an introduction to spectral methods in scientific computing). Another application is the smoothing of data where one generates the Fourier components, filters these coefficients, and then reconstructs a smoother version of the original data.

The Discrete Complex Fourier Transform

A general introduction to Fast Fourier Transforms can be found in the book The Fast Fourier Transform by E. Oran Brigham, (1974 Prentice-Hall, Inc., Englewood Cliffs, N.J). In particular, sections 6.2 through 6.4 give an account of how the discrete Fourier Transform can be derived from the continuous Fourier Transform.

Given a complex sequence $x_j$, $j=0,1,2,\ldots,N-1$ the complex periodic (forward) Discrete Fourier Transform is defined as follows:

$$X_k = \frac{1}{N} \sum_{j=0}^{N-1} x_j e^{-\frac{2\pi i j k}{N}}$$
and the inverse (backward) Discrete Fourier Transform is given by:

\[ x_j = \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i j k}{N}} \]

where \( i = \sqrt{-1} \). The above definitions are fairly standard but by no means universal; it is therefore strongly suggested that the users familiarize themselves with the definition used by the software of their interest.

NCAR Software

NCAR currently has three dozen routines available in six mathematical libraries which can aid in various aspects of computing and/or data analysis when an FFT is called for.

It should be noted that all of the highly supported library software routines for performing FFTs which are available at NCAR contain a full complement of transforms. Hence in particular each library contains routines for complex-complex, complex-real, and real-complex transforms. Further, each library contains a special routine for transforming data whose length is a power of two, and almost all routines require some initialization.

Just as there are similarities among the available routines there are also differences; however, most of these differences are invisible to the user and are mainly concerned with which of the several FFT algorithms are used. For example, both IMSL and PORT use versions of algorithms due to R.C. Singleton ([6], [7]) while the FFT routines of ULIB are due to Cooley-Tukey, Swartztrauber, and Temperton [9], [10], [11] (see [8] for an excellent description of the various FFT algorithms).

Recommendations

Since NCAR has available such a large number of FFT routines, the following two suggestions for choosing among them are offered:

1. If the user's application requires real or complex multiple FFTs on the CRAY-1, then it is strongly recommended that the CAL version of Temperton's FFT algorithm (FFT99 in ULIB) be used. This routine requires that \( N \) must have no prime factors greater than five if the data to be transformed has length \( N \). Further, if either a real-complex or complex-real transform is desired then \( N \) must be even. Software for multiple FFTs is also available in the IMSL and PORT libraries. The NAG (Numerical Algorithms Group) library also contains such software.

2. If the application does not require real or complex multiple FFTs on the CRAY-1, it is suggested that FFFPACK (available in ULIB) be used. It should be noted that the real-complex and complex-real transforms available in FFFPACK do not require that \( N \) be even.

These suggestions are made primarily because the indicated software is among the best documented, and the FFT mentioned in 1) is currently the fastest.
available at NCAR and was specifically designed to take full advantage of the architecture of the CRAY-1. (See the following article entitled "New Fast Fourier Transform Available for the CRAY-1 Machines" for a description of a new version of the Temperton FFT and timing comparisons with the old version).

Because of the variety of scientific applications which use FFT software, the two "global" suggestions made above may not be consistent with all possible applications; consequently the following table is provided to aid potential users in selecting FFT software suited to their needs.

<table>
<thead>
<tr>
<th></th>
<th>R-ODD</th>
<th>PF</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRAYLIB</td>
<td>N</td>
<td>5</td>
<td>E</td>
</tr>
<tr>
<td>SCILIB</td>
<td>N</td>
<td>2</td>
<td>E</td>
</tr>
<tr>
<td>ULIB</td>
<td>Y</td>
<td>A</td>
<td>I</td>
</tr>
<tr>
<td>IML</td>
<td>N</td>
<td>A</td>
<td>E</td>
</tr>
<tr>
<td>NAG</td>
<td>Y</td>
<td>*</td>
<td>E</td>
</tr>
<tr>
<td>PORT</td>
<td>N</td>
<td>23</td>
<td>E</td>
</tr>
<tr>
<td>XLIB</td>
<td>Y</td>
<td>A</td>
<td>I</td>
</tr>
</tbody>
</table>

Notes:

The various columns of the table indicate whether or not:

a. A library has available software which will perform a real transform on data of odd length (R-ODD).

b. A library has specific requirements on the prime factorization of N (data length), eg., a PF value of 5 indicates prime factors less than or equal to five, and a PF value of "A" indicates arbitrary factorization. The NAG PF entry (*) is 19 for the real-complex case, and arbitrary (A) for the complex-complex case.

c. A library is supported externally (E) or internally (I).

Catalog Documentation

To obtain a current list of NCAR’s available FFT software and a short description about each, include the following statement in a job submitted to either of NCAR’s CRAY-1 machines:

GETDOC,LIB=DOCLIB,DOC=D6.
References


New Fast Fourier Transform Available for the CRAY-1 Machines

by Michael Pernice

Introduction

NCAR has recently obtained a new implementation of the Temperton Fast Fourier Transform algorithm from the European Centre for Medium Range Weather Forecasts. The present version is called FFT99 and resides in CRAYLIB. The new version contains the same entry names with the same calling sequences; however, it resides in a different library, ECMFFT. In this way, you will be able to use the new version simply by ACQUIREing the new library and then directing your loader to search the ECMFFT library; there will be no need to change your code. The new version still requires the same initialization calls as the CRAYLIB version. The new version also demands that the only factors in the length of the transforms be 2, 3 and 5, but it is no longer required that the length be even. However, if you implement FFT99 after changing the code (e.g., by using the EDITOR prior to compilation) and deem it absolutely necessary to continue doing so, you must continue using the old version: the source code for the new version will not be made available due to proprietary restrictions, and a modpack designed to make changes in the old version will not have the desired results on the new version.

Timing Tests

Actually NCAR has several implementations of the Temperton FFT. Two of these are in the XLIB library, and were written to run on the CDC 7600. These are FFT99F, which is an all-FORTRAN version, and CFFT99, which is a FORTRAN version adapted to transform complex vectors. The version that resides in CRAYLIB is written partly in CAL, which accounts for the superior performance of the CRAYLIB version over the XLIB versions. In another article in this Collection ("Fast Fourier Transforms"), FFT99 (in CRAYLIB) is reported as being the fastest FFT then available at NCAR. The new version is even faster, running 20-55% faster (depending on the length of the vector being transformed) than the old version and averaging 37% faster in the tests that were performed. Several timing tests were performed on all of these packages, and the new version of FFT99 showed the best performance, as indicated by the following table:
NOTE: These timing tests included one initialization call for each call of the FFT, so that users who implement several calls to FFT99 with only one initialization should expect slightly less relative overall improvement over the old version. Also, these timing tests apply solely to the FFT and its initialization calls, and do not accurately reflect the overall savings in execution time that a user should expect to see.

As one can see from this table, the new version runs much faster than the old version and is more than twice as fast as FFT99F. The poor performance shown by CFFT99 may be explained by noting that CFFT99 transforms vectors of complex numbers, and so must process twice as many real numbers.

Implementation

In order to use the new version of FFT99, all you need do is alter your JCL by directing the loader to search the binary library where the new version resides rather than the default library $NCARLB:

<table>
<thead>
<tr>
<th>LENGTH</th>
<th>FFT99F XLIB</th>
<th>CFFT99 XLIB</th>
<th>FFT99 CRAYLIB</th>
<th>FFT99 ECMFFFT</th>
<th>ACTUAL EXECUTION TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>2.208</td>
<td>6.928</td>
<td>1.319</td>
<td>1.000</td>
<td>.0004817</td>
</tr>
<tr>
<td>96</td>
<td>2.206</td>
<td>7.450</td>
<td>1.290</td>
<td>1.000</td>
<td>.0007833</td>
</tr>
<tr>
<td>128</td>
<td>2.172</td>
<td>7.401</td>
<td>1.284</td>
<td>1.000</td>
<td>.0010541</td>
</tr>
<tr>
<td>180</td>
<td>3.544</td>
<td>5.678</td>
<td>1.548</td>
<td>1.000</td>
<td>.0014013</td>
</tr>
<tr>
<td>450</td>
<td>3.710</td>
<td>7.435</td>
<td>1.324</td>
<td>1.000</td>
<td>.0011956</td>
</tr>
<tr>
<td>540</td>
<td>3.520</td>
<td>6.041</td>
<td>1.504</td>
<td>1.000</td>
<td>.0013922</td>
</tr>
<tr>
<td>900</td>
<td>3.920</td>
<td>6.922</td>
<td>1.409</td>
<td>1.000</td>
<td>.0025774</td>
</tr>
<tr>
<td>1024</td>
<td>2.117</td>
<td>6.702</td>
<td>1.203</td>
<td>1.000</td>
<td>.0026891</td>
</tr>
<tr>
<td>2400</td>
<td>3.286</td>
<td>7.183</td>
<td>1.226</td>
<td>1.000</td>
<td>.0078005</td>
</tr>
<tr>
<td>4096</td>
<td>3.124</td>
<td>5.946</td>
<td>1.321</td>
<td>1.000</td>
<td>.0124636</td>
</tr>
</tbody>
</table>

The on-line documentation of ECMFFFT may be obtained by including the following statement in a job submitted to either of NCAR’s CRAY-1 computers:

GETDOC,LIB=DOCLIB,DOC=ECMFFFT.

June 1984 -21- Software Notes
General Software for Partial Differential Equations
by Michael Pernice

Introduction

One of the most important branches of numerical analysis is the numerical treatment of partial differential equations (abbreviated PDEs.) PDEs are fundamental in the formulation of mathematical models and appear in many contexts. This wide variety is reflected in the diversity of the kinds of PDEs that arise. PDEs are generally of three types: hyperbolic, parabolic, and elliptic. However, most equations may be of mixed type. Finding the solution to each of these types of PDEs presents its own set of difficulties. For example, numerical instability in solving a hyperbolic PDE will occur unless boundary conditions are expressed in terms of characteristic variables, and consequently general software for hyperbolic PDEs is not common. On the other hand, solving an elliptic PDE on a bounded domain can be done with relative ease, and software that solves elliptic PDEs abounds.

As with the case of ordinary differential equations (abbreviated ODEs), further conditions besides the PDE must be specified in order to guarantee a unique solution. These may be either boundary or initial conditions. Boundary conditions may take several forms and may involve specifying the solution at the boundary (Dirichlet conditions) or derivatives of the solution at the boundary (Neumann conditions). Boundary conditions can appear in problems involving all three types of PDEs. Initial conditions can occur only with hyperbolic or parabolic PDEs; these generally appear in models of evolving physical systems. The initial condition may represent an initial configuration of the system, and the solution of the PDE at a given time represents the configuration of the system at that time. One problem in designing general purpose PDE software is providing the capability to deal with as wide a scope of boundary and/or initial conditions as possible.

Associated with the specification of boundary and/or initial conditions is the notion of a well-posed problem. No general procedure exists for determining if a problem is well-posed. Mathematically speaking, a problem is well-posed if the solution exists, is unique, and depends continuously on the boundary data. Practically speaking, the complexity of most applications makes existence and uniqueness quite difficult to determine. Moreover, large changes in the solution caused by small changes in boundary conditions may be an important feature of a model. General PDE software cannot detect whether or not a problem is well-posed. Some algorithms may simply fail as a result of an ill-posed problem, but others may produce "results" which are of no value.

Methods

A frequently encountered method for numerically solving a PDE involves the use of finite differences. This technique has had its greatest success with elliptic PDEs. Every instance of a derivative expression in the PDE of interest is replaced by an appropriate finite difference approximation and an approximate solution whose values are known only at finitely many points (the meshpoints) is sought, rather than seeking a solution which is defined...
everywhere on the domain. This process, known as discretization, results in reducing the problem of solving a PDE to the problem of solving a system of linear equations. Values of the solution at points other than the meshpoints can then be found by interpolation.

Variations of this procedure can be distinguished by the manner in which the linear system which results from the discretization is solved. The linear system may be solved iteratively using techniques such as successive over-relaxation or pre-conditioned conjugate gradient methods. A direct method usually employs a variant of the L-U decomposition, such as cyclic reduction. Research in this direction has been greatly stimulated by the development of vector and parallel processors. One of the products of the latter approach is the FISHPAK software which was developed at NCAR during the mid-1970's and which is discussed below.

An entirely different approach to solving a PDE is based on the idea of replacing the PDE with several ODEs, and then using existing software to solve the ODEs. One such method that is frequently applied to parabolic PDEs is the method of lines. In this technique only one variable is discretized, resulting in a coupled system of first-order ODEs which may then be solved by standard integration techniques. Variants of the method of lines include collocation techniques, which can generate a system of ODEs from a parabolic PDE by using a piecewise polynomial fit instead of discretization.

Many other techniques exist which are too numerous to describe in this article. Among them are the Galerkin method (which includes spectral methods), variational methods, and finite element methods. These techniques (with the exception of spectral methods) generally are not used as widely as finite difference methods in applications to geophysical fluid flow problems. Each of these methods has an important class of applications; recent successes with finite element methods show that these techniques are useful for resolving the behavior of a solution near an irregular boundary.

**PDE Software at NCAR**

NCAR has a wide variety of software which can be used to solve PDEs. There is software designed to solve each of the three types of PDEs described above. The hyperbolic package (RKFPDE in XLIB) allows boundary conditions in terms of characteristic variables and can solve a wide range of problems. The parabolic solvers generally use the method of lines combined with some integration technique. The elliptic solvers which are available range from software designed to solve non-separable equations to fast, direct methods utilizing block cyclic reduction. Much of this latter software was developed at NCAR and is portable.

The major product of PDE software development at NCAR is a package of routines called FISHPAK, developed by Paul Swarztrauber, Roland Sweet and John Adams. This collection of software can be applied to a subclass of elliptic problems (the Helmholtz equation) in various coordinate systems. Included with the PDE solvers are auxiliary routines, including Fast Fourier Transforms, which may be used along with other FISHPAK routines to solve three-dimensional PDEs, and linear algebra routines which solve block tridiagonal systems using generalized cyclic reduction. FISHPAK specifically provides different routines for...
different coordinate systems, so the user interface with the routines in FISH-PAK is greatly simplified.

Since new software is always being added to the public libraries at NCAR, the Software Catalog should be checked for an up-to-date summary of available software. The PDE software at NCAR is briefly summarized in Section D3. The catalog is available both in the consulting library and on-line through use of the CRAY-1 utility GETDOC:

```
GETDOC,LIB=DOCLIB,DOC=D3.
```

The remainder of this article contains:

- a table describing PDE software which is available at NCAR;
- descriptions of the PDE software listed in the table;
- a table describing linear algebra software useful in solving PDEs numerically which is available at NCAR;
- descriptions of these linear algebra routines.

Several survey articles on numerical PDE software have appeared in recent years; see [2], [5].
Table I
PDE Software at NCAR

<table>
<thead>
<tr>
<th>Software</th>
<th>Type</th>
<th>Cond</th>
<th>Domain</th>
<th>Ref</th>
<th>Port</th>
<th>Demo</th>
</tr>
</thead>
<tbody>
<tr>
<td>RKFPDE (XLIB)</td>
<td>H-NL</td>
<td>BC</td>
<td>Rectangular</td>
<td>2</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>system</td>
<td>IC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DPDES (IMSL)</td>
<td>P-NL</td>
<td>BC</td>
<td>Interval</td>
<td>4</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>system</td>
<td>IC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDECOL (XLIB)</td>
<td>P-NL</td>
<td>BC</td>
<td>Interval</td>
<td>6,7</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>system</td>
<td>IC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D03PAF (NAG)</td>
<td>P-NL</td>
<td>BC</td>
<td>Interval</td>
<td>8,9</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>system</td>
<td>IC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D03PBF (NAG)</td>
<td>P-NL</td>
<td>BC</td>
<td>Interval</td>
<td>8,9</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>system</td>
<td>IC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D03PGF (NAG)</td>
<td>P-NL</td>
<td>BC</td>
<td>Interval</td>
<td>8,9</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>system</td>
<td>IC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D03EAF (NAG)</td>
<td>Laplacian</td>
<td>BC</td>
<td>Plane Region</td>
<td>8</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td></td>
<td>in 2-d</td>
<td></td>
<td>Smooth bdry</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CROSEL (CRAYLIB, $NCARLB)</td>
<td>EX-L</td>
<td>BC</td>
<td>Rectangular</td>
<td>*</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>LIPTIC (CRAYLIB, $NCARLB)</td>
<td>NSE-L</td>
<td>BC</td>
<td>Rectangular</td>
<td>1,*</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>SEPEL (ULIB, $NCARLB)</td>
<td>SE-L</td>
<td>BC</td>
<td>Rectangular</td>
<td>1</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SEPX4 (ULIB, $NCARLB)</td>
<td>SE-L</td>
<td>BC</td>
<td>Rectangular</td>
<td>1</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>HW3CRT (ULIB, $NCARLB)</td>
<td>Helmholtz in 3-d</td>
<td>BC</td>
<td>Rectangular</td>
<td>-</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>HWSCRT (ULIB, $NCARLB)</td>
<td>Helmholtz</td>
<td>BC</td>
<td>Rectangular</td>
<td>12,13</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>HW3PLR (ULIB, $NCARLB)</td>
<td>Helmholtz</td>
<td>BC</td>
<td>Polar</td>
<td>12,13</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>HWSCYL (ULIB, $NCARLB)</td>
<td>Helmholtz</td>
<td>BC</td>
<td>Cylinder</td>
<td>12,13</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>HWSCSP (ULIB, $NCARLB)</td>
<td>Helmholtz</td>
<td>BC</td>
<td>Axisymmetric Spherical</td>
<td>12,13</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>HWSSSP (ULIB, $NCARLB)</td>
<td>Helmholtz</td>
<td>BC</td>
<td>Unit Sphere</td>
<td>12,13</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

* see "New Software for Elliptic PDEs", on page 15 in this collection.
Notes

1) Explanation of abbreviations:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>Hyperbolic</td>
</tr>
<tr>
<td>P</td>
<td>Parabolic</td>
</tr>
<tr>
<td>E</td>
<td>Elliptic</td>
</tr>
<tr>
<td>NL</td>
<td>Nonlinear</td>
</tr>
<tr>
<td>BC</td>
<td>Boundary Conditions</td>
</tr>
<tr>
<td>EX</td>
<td>Elliptic with cross-term</td>
</tr>
<tr>
<td>NSE</td>
<td>Non-separable Elliptic</td>
</tr>
<tr>
<td>SE</td>
<td>Separable Elliptic</td>
</tr>
<tr>
<td>L</td>
<td>Linear</td>
</tr>
<tr>
<td>IC</td>
<td>Initial Conditions</td>
</tr>
</tbody>
</table>

2) Explanation of column headers:

- **Cond**: type of conditions which are required
- **Port**: portability of the software
- **Demo**: availability of on-line demonstration drivers through the "TEST" utility (see the December 1983 issue of The Record.)

3) The word 'system' indicates that the software is capable of solving a system of more than one PDE. Otherwise it should be assumed that only a single equation is solved.

4) The routines which begin with 'HWS' use a five-point centered finite difference approximation for the Helmholtz equation. There are versions of these programs which use a staggered grid approximation and begin with an 'HST' string; see [12,13]. All other information holds for these routines except that demonstration drivers are not yet available.

Description of Routines in Table I

RKFPDE was written by John Gary for the CDC 7600 and has since been modified to run on the CRAY-1. The package has also been referred to in the literature as HYPACK. The problem must be of the form

\[
\begin{align*}
\frac{\partial u}{\partial t} &= \frac{\partial f_n}{\partial t} + \frac{\partial g_n}{\partial t} + \text{hun}, \\
\end{align*}
\]

where \( u = (u_1, \ldots, u_n) \) is the vector of unknowns. Equation (1a) may be coupled with the elliptic equation

\[
\begin{align*}
\frac{\partial^2 p}{\partial x^2} + \frac{\partial p}{\partial x} c(x) + \frac{\partial^2 p}{\partial y^2} + \frac{\partial p}{\partial y} e(y) + f(y) p &= \text{hpn,}
\end{align*}
\]

in which the coefficients must be real-valued. The functions \( f_n, g_n, \text{hun} \) and \( \text{hpn} \) are also \( n \)-vector valued functions where
and \( w = (u_1, \ldots, u_n) \). The boundary conditions which are allowed by RKFPDE are

1) Periodic
2) Symmetry with respect to the boundary
3) Skew symmetry with respect to the boundary
4) Mixed
5) Extrapolation of characteristic variables.

Algorithm:

1) The elliptic equation is solved by SEPELI (see discussion below.)
2) The spatial variable is discretized, generating a system of ODEs.
3) The system is then solved from some initial time to an output time (provided by the user) by integrating the system of ODEs using a Runge-Kutta-Fehlberg 3-4 scheme.

Although it often suffices to use only the function \( \text{hun} \) to define the problem, inclusion of the \( \text{fn} \) and \( \text{gn} \) terms allows the user to pose the problem in 'conservation form', which may help to guarantee a numerically stable solution. This implementation is intended primarily for use on hyperbolic problems; see [3].

DPDES is an IMSL routine which may be used to solve a parabolic PDE of the form

\[
\frac{\partial u}{\partial t} = f(x, t, u, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2})
\]

where \( u = (u_1, \ldots, u_n) \) is the vector of unknowns. Boundary conditions must be of the form...
\[ (2b) \quad \alpha_i u_i + \beta_i \frac{\partial u_i}{\partial x} = y_i(t), \quad 1 \leq i \leq n, \]

where \( y_1, \ldots, y_n \) are continuous and \( \alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n \) are real constants. The variable \( x \) must lie in an interval and the conditions in (2b) must hold at the endpoints. Initial conditions must also be provided, and must be consistent with the boundary conditions at the initial time.

**Algorithm:**

1) The space variable is discretized using meshpoints supplied by the user.

2) The right-hand side of the PDE is approximated by collocation using piecewise Hermite polynomials of degree three. This provides a fourth order spatial discretization error for smooth solutions. The boundary conditions are incorporated in the polynomial fit.

3) The resulting system of ODEs is integrated using an IMSL variation of Gear's method.

The user is able to control the time discretization error through use of a parameter, but the spatial discretization is fixed by the degree of the polynomials used in the fit. For further details see [4].

PDECOL is a package written by N.K. Madsen and R.F. Sincovec [7] which consists of a driver named PDECOL and a collection of routines. PDECOL solves a parabolic PDE of the form

\[ (3a) \quad \frac{\partial u}{\partial t} = f(t, x, u, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}) \]

where \( u \) and \( f \) are n-vector-valued functions. The domain in \( t \) and \( x \) is assumed to be bounded, \( t_o \leq t \leq t_1 \) and \( x_L \leq x \leq x_R \). The initial conditions must have the form

\[ (3b) \quad u(t_o, x) = \phi(x), \quad x_L \leq x \leq x_R \]

Where \( \phi \) is n-vector-valued and \( t_o \) is the initial time. The boundary conditions may be specified at either \( x_L \) or \( x_R \) or both and must be of the form

\[ (3c) \quad b(u, \frac{\partial u}{\partial x}) = z(t) \]

where \( b \) and \( z \) are n-vector-valued functions. The initial and boundary conditions must be consistent with the boundary conditions at the initial time.
conditions must be consistent in the sense that \( z(t_o) \) must equal \( \phi \) wherever the boundary conditions are specified.

Algorithm:

1) The space variable is discretized.

2) A finite element collocation procedure is performed. This assumes that at each time step the solution may be represented as a linear combination of piecewise polynomial functions, and results in a system of ODEs. The boundary conditions, if present, are incorporated into the polynomial fit.

3) The resulting first-order system of ODEs, together with the initial conditions, now forms an initial value problem, which is then solved by a modification of Gear's method.

The integration procedure controls the temporal discretization error as it solves the first-order system of ODEs that results from the collocation procedure. The spatial discretization error is controlled by the user's choice of the degree and the smoothness of the piecewise polynomial fit. In general, increasing the degree of the polynomials used in the collocation procedure will decrease the spatial discretization error, though this is done at the cost of increased computing overhead. The authors provide a description of the package's features and some examples in [6].

D03PAF, D03PBF, and D03PGF are the parabolic solvers in the NAG library. Each of these routines uses basically the same method (see [9]). Each requires initial conditions by specifying the solution values at some initial time.

D03PAF approximates the solution to a parabolic equation in one space and one time variable of the form

\[
\frac{\partial u}{\partial t} = x^{-m} \frac{\partial}{\partial x} \left( x^{m} g(x,t,u) \frac{\partial u}{\partial x} \right) + f(x,t,u,\frac{\partial u}{\partial x})
\]

subject to boundary conditions of the form

\[
p(t)u + q(t)\frac{\partial u}{\partial x} = r(t,u).
\]

D03PBF approximates the solution to a system of parabolic PDEs of the form.
\[ c_i \frac{\partial u_i}{\partial t} = x^{-m} \sum_{j=1}^{n} \frac{\partial}{\partial x} \left( x^m g_{ij}(x,t,u) \frac{\partial u_i}{\partial x} \right) \]
\[ + f_i(x,t,u, \frac{\partial u_i}{\partial x}), \quad 1 \leq i \leq n \]

where

\[ c_i = c_i(x,t,u, \frac{\partial u_i}{\partial x}), \quad 1 \leq i \leq n. \]

The boundary conditions must have the form

\[ p_i(t)u + q_i(t) \frac{\partial u_i}{\partial x} = r_i(t,u), \quad 1 \leq i \leq n \]

where \( u = (u_1, \ldots, u_n) \) is the vector of unknowns and \( p_i, q_i, \) and \( r_i \) are continuous real-valued functions for \( 1 \leq i \leq n. \)

D03PGF approximates the solution to a system of parabolic PDEs of the form

\[ c_i \frac{\partial u_i}{\partial t} = x^{-m} \sum_{j=1}^{n} \frac{\partial}{\partial x} \left( x^m g_{ij}(x,t,u) \frac{\partial u_i}{\partial x} \right) \]
\[ + f_i(x,t,u, \frac{\partial u_i}{\partial x}), \quad 1 \leq i \leq n \]

where \( c_i \) also has the form (5b). D03PGF also requires that boundary conditions are of the form (5c).

Algorithm:

1) The space variable is discretized, generating a system of ODEs.

2) The resulting system of ODEs is solved using Gear's method of integration.

There are several restrictions which must be observed for each type of PDE; for full details see [8].

D03EAF is a NAG routine that approximates the solution to Laplace's equation in two dimensions for an arbitrary domain. Dirichlet and Neumann boundary conditions are allowed. The routine integrates a Green's function for the solution and must be called once for each point in the domain at which a solution value is desired; again, see [8].

Software Notes -30- June 1984
CROSEL was written by John Adams. The equation must be of the form

\[
(7) \quad c_{xx} \frac{\partial^2 u}{\partial x^2} + c_{xy} \frac{\partial^2 u}{\partial x \partial y} + c_{yy} \frac{\partial^2 u}{\partial y^2} + c_x \frac{\partial u}{\partial x} + c_y \frac{\partial u}{\partial y} + c_u = f(x,y)
\]

where the coefficient functions must be real-valued and may depend on \( x \) and \( y \). Allowable boundary conditions are:

1) Periodic (in \( x \)-direction only)
2) Dirichlet
3) Mixed (tangential derivatives at the boundary not allowed)

and may be specified independently on each of the sides of the solution domain.

Algorithm:

1) The PDE is discretized, using a second order finite difference approximation. Incorporating the boundary conditions generates a system of linear equations.
2) An L-U decomposition is performed.
3) The resulting linear system is solved.
4) An optional fourth order approximation is obtained by the use of deferred corrections.

The routine is capable of skipping steps 1) and 2) if it is used to solve the same PDE several times. The user may elect to use either an in-core or out-of-core version. When the out-of-core version is used, CROSEL attempts to optimize execution time by performing computations while buffering. If the PDE is not elliptic or is of mixed type CROSEL will still attempt a solution, which will be of uncertain quality since the linear system generated by the discretization is likely to be ill-conditioned or possibly even singular.

LIPTIC was also written by John Adams and solves (7) with \( c_{xy} = 0 \); see [1]. See also "New Software for Elliptic PDEs", on page 15 in this collection.

SEPELI is a FISHPAK routine which solves an equation of the form

June 1984 -31- Software Notes
(8) \[\frac{\partial}{\partial x} \frac{\partial^2 u}{\partial x^2} + \frac{\partial}{\partial x} \frac{\partial u}{\partial x} + \frac{\partial^2}{\partial y^2} \frac{\partial u}{\partial y} + \frac{\partial}{\partial x} \frac{\partial u}{\partial x} + \frac{\partial}{\partial y} \frac{\partial u}{\partial y} + f(y)u = g(x,y)\]

where the coefficient functions are real-valued. Allowable boundary conditions are:

1) Periodic
2) Dirichlet
3) Mixed

and may be independently prescribed for each of the independent variables.

Algorithm:

1) The equation is discretized with a second order approximation and the boundary conditions are incorporated, generating a block tridiagonal linear system.

2) A possibly singular system is adjusted so as to guarantee a least squares solution (see below).

3) The linear system is solved using generalized cyclic reduction. This is done by a call to BLKTRI (see description below).

4) If specified by the user, a fourth order solution is generated by the use of deferred corrections.

Certain combinations of the boundary conditions may cause the PDE involved to have no solution. In this case SEPELI calculates a perturbation which, when subtracted from right-hand side of the linear system generated by the discretization, guarantees the existence of a least-squares solution to the linear system involved.

SEPX4 is another FISHPAK routine which solves (8) when \(df(y) = 1\) and \(ef(y) = ff(y) = 0\); see [1].

HWSSSP is a FISHPAK routine that solves the Helmholtz equation on a unit sphere:

(9) \[\frac{1}{\sin \theta} \frac{\partial}{\partial \theta} \left( \sin \theta \frac{\partial u}{\partial \theta} \right) + \frac{1}{\sin^2 \theta} \frac{\partial^2 u}{\partial \phi^2} + \lambda u = f(\theta, \phi).\]

Allowable boundary conditions are

Software Notes -32- June 1984
1) Dirichlet
2) Neumann
3) Mixed
4) Periodic (in longitude only).

Algorithm:

1) The equation is discretized and the boundary data is included, producing a linear system of equations.

2) A singular system is adjusted to guarantee a least-squares solution (see discussion below).

3) The resulting linear system is solved by calling GENBUN (see description below).

If the PDE in question reduces to a Poisson equation ($\Delta u = 0$) a solution may not exist for certain combinations of boundary conditions. In this case HWSSSP computes a perturbation term which will guarantee the existence of a least-squares solution when added to the RHS of the equation. A complete description of 1) may be found in [12,13].

Routines HWSCRT, HWSPLR, HWSCSP, HWSCYL, HSTCRT, HSTPLR, HSTCSP, HSTCYL, and HSTSSP each solve the Helmholtz equation in which the Laplacian is expressed in different coordinate systems (see Table I above). Otherwise the features of the software are quite similar to those described above for HWSSSP. See [12,13].

HW3CRT solves the three-dimensional Helmholtz equation where the Laplacian is expressed in rectangular coordinates.

<table>
<thead>
<tr>
<th>Software</th>
<th>Algorithm</th>
<th>Ref</th>
<th>Port</th>
<th>Demo</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENBUN</td>
<td>Block Cyclic Reduction</td>
<td>14</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>BLKTRI</td>
<td>Block Cyclic Reduction</td>
<td>11,12,13</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>D03EBF</td>
<td>Strongly Implicit</td>
<td>8,10</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>D03ECF</td>
<td>Strongly Implicit</td>
<td>8,10</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>
Descriptions of Routines in Table II

GENBUN is a FISHPAK routine which solves a block tridiagonal system of linear equations in which the off-diagonal blocks are negative identity matrices. GENBUN uses generalized cyclic reduction (see [14]) and is an updated version of the ULIB routine POIS. A version of GENBUN which can handle complex coefficients is also available.

BLKTRI is another FISHPAK routine which solves a block tridiagonal system of linear equations using generalized cyclic reduction. BLKTRI solves a more general block tridiagonal system than GENBUN in which the off-diagonal blocks may be different multiples of the identity matrix [11]. A detailed example of how to set up a problem for BLKTRI may be found in [12]. A version of BLKTRI which can handle complex coefficients is also available.

The NAG library provides two subroutines which implement the Strongly Implicit Procedure [10] and which can be used to solve systems of linear equations that arise from certain discretizations.

D03EBF solves the linear system of equations which is generated by discretizing an elliptic PDE in two variables with a 5-point stencil on a rectangular mesh. The linear system which D03EBF may solve must be of the form

\[(10a) \quad a_{ij}t_{i,j-1} + b_{ij}t_{i-1,j} + c_{ij}t_{ij} + d_{ij}t_{i+1,j} + e_{ij}t_{i,j+1} = q_{ij},\]

where \(1 \leq i \leq n_1\), \(1 \leq j \leq n_2\), and it is assumed that \(c_{ij} \neq 0\).

D03ECF solves the linear system of equations which is generated by discretizing an elliptic PDE in three variables with a 7-point stencil on a rectangular mesh. The linear system which D03ECF may solve is of the form

\[(10b) \quad a_{ijk}t_{i,j,k-1} + b_{ijk}t_{i,j-1,k} + c_{ijk}t_{i-1,j,k} + d_{ijk}t_{i,j,k} + e_{ijk}t_{i,j+1,k} + f_{ijk}t_{i,j,k+1} + g_{ijk}t_{i,j,k+1} = q_{ijk},\]

where \(1 \leq i \leq n_1\), \(1 \leq j \leq n_2\), \(1 \leq k \leq n_3\), and it is assumed that \(d_{ijk} \neq 0\).

REFERENCES


Tridiagonal Matrix Solvers at NCAR
by Michael Pernice

In a previous Software Notes article (CF Notes #62, October 1979, ‘Solving Sparse Square Linear Systems’), the general problem of solving sparse linear systems was addressed and a summary of applicable software at NCAR was given. This article specifically focuses on banded systems. In particular, tridiagonal systems are examined and a new collection of software that solves tridiagonal systems is evaluated.

Introduction

The computation of a numerical solution to a system of linear equations is facilitated by taking advantage of any special structure of the matrix of coefficients. An example of a specially-structured matrix which is frequently encountered is a symmetric matrix. This type of matrix can be efficiently stored since only the lower triangular portion of the matrix needs to be known. Also, no elaborate re-indexing scheme is needed to do this. If a symmetric matrix is also positive definite, a Cholesky decomposition can be performed. A distinct advantage of this method is that the Cholesky decomposition can be stored in half the space needed by an L-U decomposition.

Another special matrix structure that appears often in applications is a banded structure. This is the situation where all the matrix entries are zero except for those ‘close’ to the diagonal. To be precise, an NxN matrix

\[ A = \begin{bmatrix} a_{ij} \end{bmatrix} \]

is banded if there exists positive integers k,l such that:

\[ a_{ij} = 0 \quad \text{if} \quad i-j > k \quad \text{or} \quad j-i > l. \]

We call l the lower bandwidth and k the upper bandwidth. A banded matrix can be efficiently stored by saving only the elements which appear in the band; complicated re-indexing can be avoided by locating each matrix element relative to the main diagonal.

When the total bandwidth is small relative to the number N of equations, simple modifications of Gaussian elimination can lead to substantial savings in execution time. For example, an algorithm can compare the upper and lower bandwidths of a banded system, and, based on which is larger, decide whether it is more efficient to reduce to an upper or a lower triangular system. If the upper and lower bandwidths are equal, an algorithm can easily be constructed that eliminates the subdiagonals starting from the second row while eliminating the superdiagonals starting from the last row. This continues until the two elimination processes meet in the middle of the matrix. At this point the matrix has been decomposed into a lower triangular block and an upper triangular block. The two subsystems can then be solved via back-substitution and forward-substitution, respectively.

It must be pointed out that such modifications to Gaussian elimination are made without considering the vector and parallel processing capabilities of modern computers. In its standard formulation Gaussian elimination is a
recursive process, and so prima facie it is not vectorizable. However a slight modification leads to a vectorizable algorithm; see [4] for a discussion on vectorizing Gaussian elimination.

A banded matrix whose upper and lower bandwidth is one is called a tridiagonal matrix. Tridiagonal matrices appear naturally in several contexts, for example in discretized versions of elliptic partial differential equations. In fact this was the context in which the need for decreased execution time stimulated the search for an efficient approach to solving tridiagonal systems.

An entirely different approach to solving tridiagonal systems is based on the observation that in a tridiagonal system the odd-indexed variables can be removed from the even-indexed equations with relatively little effort. This decomposes the system into two smaller subsystems, each half the size of the original. This decomposition can be effected by permuting the rows and columns of the matrix (which amounts to a re-indexing of the variables and the equations) followed by a block L-U decomposition. The algorithm which is based on this idea is called cyclic reduction; the names 'odd-even reduction' and 'cyclic odd-even reduction' also appear in the literature. Because an L-U decomposition is used in the reduction process, cyclic reduction and L-U decomposition have the same stability properties.

R. W. Hockney is generally credited with the first use of cyclic reduction in his 1965 paper [2]. The algorithm was improved after G. H. Golub observed that cyclic reduction may be repeatedly applied in order to further reduce the size of the system [1]. During the 1970's Paul Swarztrauber and Roland Sweet applied cyclic reduction to block tridiagonal systems in developing efficient and accurate direct methods for solving elliptic partial differential equations [5].

The following figures illustrate cyclic reduction for the case N=6. The initial matrix is given by:

\[
\begin{array}{ccccccc}
D_1 & U_1 & & & & & \\
L_1 & D_2 & U_2 & & & & \\
L_2 & D_3 & U_3 & & & & \\
& L_3 & D_4 & U_4 & & & \\
& & L_4 & D_5 & U_5 & & \\
& & & L_5 & D_6. & & \\
\end{array}
\]

Rows and columns are then permuted to yield:

\[
\begin{array}{ccccccc}
D_1 & U_1 & & & & & \\
D_3 & L_2 & U_3 & & & & \\
D_5 & L_4 & U_5 & & & & \\
L_1 & U_2 & D_2 & & & & \\
L_3 & U_4 & D_4 & & & & \\
L_5 & D_6. & & & & & \\
\end{array}
\]

A block L-U decomposition is performed to yield:
For large systems the reduction is repeated on the lower right-hand tridiagonal subsystem. A solution is computed by backsolving the reduced system.

Software at NCAR

There are many subroutines in the 20 software libraries at NCAR which are specialized to deal with banded systems of linear equations. Among these subroutines are versions which are specialized further to deal with general and symmetric tridiagonal systems. The SLATEC routines are updated versions of LINPACK routines; these in turn have CRAY-optimized versions which reside in $SCILIB$. There are also banded system solvers in CRAYLIB, ULIB and IMSL; the ULIB routine deals specifically with tridiagonal systems and one of the CRAYLIB routines is written in CAL. See Tables Ia, Ib for a summary of software at NCAR that solves tridiagonal systems.

There is a new implementation of the cyclic reduction algorithm which has recently been made available to users at NCAR. Developed at Lawrence Livermore National Laboratory, it is designed specifically for the vector processing capabilities of the CRAY-1 and uses intrinsic CRAY-1 functions to perform the cyclic reduction decomposition. For a full description of this implementation see [3].

The package of cyclic reduction routines is organized into two parts: CYCRDLL (in $NCARLB$ and CRAYLIB) and CYCRDLLF (in CRAYLIB). Both contain four subroutines: two which perform cyclic reduction on general tridiagonal systems and symmetric tridiagonal systems (CRDEC and SCRDEC, respectively) and two routines which compute a solution based on this decomposition (CRSOL, SCRSOL). The routines in CYCRDLL are written in CAL and those in CYCRDLLF are in FORTRAN.

Since CYCRDLL is loaded by default, no LIB specification on the LDR card is necessary in order to access the CAL routines. The entry points and the particulars of how to use the routines may be obtained by using the CRAY-1 utility GETDOC. Including the statement

```
GETDOC,LIB=CRAYLIB,DOC=CYCRDLL.
```

in a job submitted to either of NCAR's CRAY-1 computers will put about four pages of documentation in $OUT$.

In order to access the source code for the cyclic reduction routines, including:

```
GETSRC,LIB=CRAYLIB,FILE=CYCRDLLF.
```

Software Notes

-38- June 1984
in a job submitted to either CRAY-1 computer will put the source code in $OUT$.

Timing and accuracy tests have been performed at NCAR by the author on the software which is listed in Tables Ia and Ib. All routines returned solutions having at least nine significant digits, so no distinction among them can be made on this basis. Some of the timing results are summarized in Tables IIa, IIb, IIIa and IIIb, where the FORTRAN routines are compared; Table IV summarizes some results for the CAL routines. Note the dramatic improvement in execution time which was achieved by cyclic reduction.

When comparing execution times the information provided in Tables Ia and Ib should be kept in mind. A routine that pivots will provide good results for more matrices than a routine that doesn’t pivot, but only at the cost of increased execution time. Of particular significance is the comparison between the cyclic reduction routines and the ULIB routine TRDI. These results provide a measure of the improvement of cyclic reduction over Gaussian elimination. For small systems, cyclic reduction takes more execution time than Gaussian elimination because of vector startup time; for larger systems (N > 20), cyclic reduction requires far less execution time.

Recommendations

Based on the results of the timing tests alone, it is clear that the cyclic reduction routines are preferred for solving tridiagonal systems, unless any of the additional features of the other routines is necessary. For example, a badly scaled tridiagonal system should be solved with the IMSL routine, since it scales the variables and pivots.

In solving symmetric systems when the matrix involved is not known to be positive definite, Cholesky decomposition should be used since it will return an error if the matrix is not positive definite, while cyclic reduction will still attempt a solution. On the other hand, for production runs involving matrices which are well understood, cyclic reduction will provide a solution much more quickly. Also note that among the symmetric tridiagonal solvers, only cyclic reduction has a CAL version, and that Tables IIb and IIIb compare only FORTRAN versions. The CAL version of cyclic reduction applied to symmetric systems averages 39% faster than the FORTRAN version.
<table>
<thead>
<tr>
<th>Routine (library)</th>
<th>Language</th>
<th>Algorithm</th>
<th>Special Conditions</th>
<th>Extra Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGLSLV (CRAYLIB, $NCARLB)</td>
<td>FORTRAN</td>
<td>L-U decomposition</td>
<td>diagonal dominant</td>
<td>CRAY optimized; can be adapted for iterative refinement; solves banded systems</td>
</tr>
<tr>
<td>LQUTLB (IMSL)</td>
<td>FORTRAN</td>
<td>L-U decomposition</td>
<td>none</td>
<td>equilibration; solves banded systems; pivots; solves systems with several right-hand sides</td>
</tr>
<tr>
<td>F01LBF, F04LDF (NAG)</td>
<td>FORTRAN</td>
<td>L-U decomposition</td>
<td>none</td>
<td>solves banded systems; pivots; does row scaling; solves systems with several right-hand sides</td>
</tr>
<tr>
<td>SGTSL ($SCILIB)</td>
<td>FORTRAN</td>
<td>Gaussian elimination</td>
<td>none</td>
<td>CRAY optimized; pivots</td>
</tr>
<tr>
<td>TRDI (ULIB, $NCARLB)</td>
<td>FORTRAN</td>
<td>L-U decomposition</td>
<td>diagonal dominant</td>
<td>can be adapted for iterative refinement</td>
</tr>
<tr>
<td>TRISLV (CRAYLIB, $NCARLB)</td>
<td>CAL</td>
<td>L-U decomposition</td>
<td>diagonal dominant</td>
<td>solves systems with several right-hand sides; pivots</td>
</tr>
<tr>
<td>CYCRDLL (CRAYLIB, $NCARLB)</td>
<td>CAL</td>
<td>cyclic reduction</td>
<td>diagonal dominant</td>
<td>can be adapted for iterative refinement</td>
</tr>
<tr>
<td>CYCRDLLF (CRAYLIB)</td>
<td>FORTRAN</td>
<td>cyclic reduction</td>
<td>diagonal dominant</td>
<td>can be adapted for iterative refinement</td>
</tr>
</tbody>
</table>
### Table Ib
Summary of Symmetric Tridiagonal Solvers

<table>
<thead>
<tr>
<th>Routine (library)</th>
<th>Language</th>
<th>Algorithm</th>
<th>Special Conditions</th>
<th>Extra Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSPSLV (CRAYLIB, $NCARLB$)</td>
<td>FORTRAN</td>
<td>Cholesky decomposition</td>
<td>positive definite</td>
<td>same as BGLSLV</td>
</tr>
<tr>
<td>LEQ1PB (IMSL)</td>
<td>FORTRAN</td>
<td>Cholesky decomposition</td>
<td>positive definite</td>
<td>solves systems with several right-hand sides; solves banded systems</td>
</tr>
<tr>
<td>F04ACF (NAG)</td>
<td>FORTRAN</td>
<td>Cholesky decomposition</td>
<td>positive definite</td>
<td>solves systems with several right-hand sides; solves banded systems</td>
</tr>
<tr>
<td>SPTSL (SCILIB)</td>
<td>FORTRAN</td>
<td>modified Gaussian elimination*</td>
<td>positive definite</td>
<td>Cray optimized</td>
</tr>
<tr>
<td>CYCRDLL (ULIB, $NCARLB$)</td>
<td>CAL</td>
<td>cyclic reduction</td>
<td>diagonal dominant</td>
<td>can be adapted for iterative refinement</td>
</tr>
<tr>
<td>CYCRDLLF (CRAYLIB)</td>
<td>FORTRAN</td>
<td>cyclic reduction</td>
<td>diagonal dominant</td>
<td>can be adapted for iterative refinement</td>
</tr>
</tbody>
</table>

*see LINPACK USER'S GUIDE.*

### Table IIa
Comparison of Nonsymmetric Solvers

<table>
<thead>
<tr>
<th>Actual Execution Times (milliseconds) (FORTRAN Routines)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIM</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>68</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>142</td>
</tr>
<tr>
<td>233</td>
</tr>
<tr>
<td>420</td>
</tr>
<tr>
<td>500</td>
</tr>
<tr>
<td>610</td>
</tr>
<tr>
<td>750</td>
</tr>
<tr>
<td>800</td>
</tr>
</tbody>
</table>

June 1984
**Table IIb**
Comparison of Symmetric Solvers

<table>
<thead>
<tr>
<th>Actual Execution Times (milliseconds)</th>
<th>(FORTRAN Routines)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DIM</strong></td>
<td><strong>BSPDSL</strong></td>
</tr>
<tr>
<td>12</td>
<td>0.104</td>
</tr>
<tr>
<td>68</td>
<td>0.419</td>
</tr>
<tr>
<td>100</td>
<td>0.610</td>
</tr>
<tr>
<td>142</td>
<td>0.861</td>
</tr>
<tr>
<td>233</td>
<td>1.40</td>
</tr>
<tr>
<td>420</td>
<td>2.52</td>
</tr>
<tr>
<td>500</td>
<td>3.00</td>
</tr>
<tr>
<td>610</td>
<td>3.65</td>
</tr>
<tr>
<td>750</td>
<td>4.49</td>
</tr>
<tr>
<td>800</td>
<td>4.79</td>
</tr>
</tbody>
</table>

**Table IIIa**
Comparison of Nonsymmetric Solvers

<table>
<thead>
<tr>
<th>Relative Execution Times (FORTRAN Routines)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DIM</strong></td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>68</td>
</tr>
<tr>
<td>100</td>
</tr>
<tr>
<td>142</td>
</tr>
<tr>
<td>233</td>
</tr>
<tr>
<td>420</td>
</tr>
<tr>
<td>500</td>
</tr>
<tr>
<td>610</td>
</tr>
<tr>
<td>750</td>
</tr>
<tr>
<td>800</td>
</tr>
</tbody>
</table>

**Software Notes**

June 1984
Table IV
Comparison of CAL Routines

<table>
<thead>
<tr>
<th>DIM</th>
<th>TRISLV</th>
<th>CYCRDLL</th>
<th>RATIO*</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0.0446</td>
<td>0.0259</td>
<td>1.722</td>
</tr>
<tr>
<td>68</td>
<td>0.232</td>
<td>0.0534</td>
<td>4.345</td>
</tr>
<tr>
<td>100</td>
<td>0.339</td>
<td>0.0654</td>
<td>5.183</td>
</tr>
<tr>
<td>142</td>
<td>0.480</td>
<td>0.0830</td>
<td>5.783</td>
</tr>
<tr>
<td>233</td>
<td>0.785</td>
<td>0.111</td>
<td>7.072</td>
</tr>
<tr>
<td>420</td>
<td>1.41</td>
<td>0.179</td>
<td>7.877</td>
</tr>
<tr>
<td>500</td>
<td>1.68</td>
<td>0.205</td>
<td>8.195</td>
</tr>
<tr>
<td>610</td>
<td>2.05</td>
<td>0.247</td>
<td>8.300</td>
</tr>
<tr>
<td>750</td>
<td>2.52</td>
<td>0.293</td>
<td>8.601</td>
</tr>
<tr>
<td>800</td>
<td>2.69</td>
<td>0.313</td>
<td>8.594</td>
</tr>
</tbody>
</table>

*ratio of TRISLV time to CYCRDLL time

References


