SLAC VM NOTEBOOK

Module 19: Numerical Analysis Program Library User's Guide

—— N A P L U G ——

J. H. Bolstad
T. F. Chan
W. M. Coughran, Jr.
E. H. Grosse
M. T. Heath
F. T. Luk

M. J. Berger
W. D. Gropp
R. J. LeVeque
S. G. Nash
L. N. Trefethen

Revision 0
November 17, 1981

Module 19: This guide describes the numerical subroutine library supported by SLAC Computing Services for solving mathematical problems numerically. The "core" library described herein consists of about 150 routines, most written in FORTRAN. Overviews of various problem areas are given to facilitate code selection.

SLAC Computing Services (SCS)
Stanford Linear Accelerator Center (SLAC)
Menlo Park, California 94025

Publishing History for Module 19

Revision 0..................November 17, 1981

CONTENTS FOR MODULE 19

N A P L   U S A G E   S U M M A R Y

Online assistance:
```
     HELP NAPL
```

To choos a routine:
```
     read the appropriate section of this guide
```

To get documentation for a routine:
```
     HELP NAPL <routine> (ALL
     HELPRINT NAPL <routine> (ALL
```

To get a usage example:
```
     NATEST <routine>
```

To call the routine:
```
     GLOBAL TXTLIB NAPL FORTSLAC FORTMOD2
```

To generate this guide:
```
     PRINTDOC NOTEBOOK MOD19
```

To retrieve source code (usually unnecessary):
```
     NASOURCE <routine>
```

Questions? See a Numerical Analysis Consultant

1. INTRODUCTION

The Numerical Analysis Program Library is SLAC's central collection of subroutines for solving mathematical problems numerically. The library is maintained by the Numerical Analysis group of the Stanford University Computer Science Department, and was established by John Bolstad, William Coughran, Eric Grosse, and others in 1975-77. It consists of roughly 1500 routines, of which some 150 make up the "core" library described here. These routines are all in Fortran (except for the random number generators, which are written in assembly language) and almost all are in double precision. This guide describes the contents of the NAPL and also gives references to further information.

1.1 PURPOSE OF THE NAPL

The aim of the NAPL is to provide state-of-the-art routines for commonly occurring numerical problems that are as efficient and reliable as any available. Rather than give many routines for each problem, which can face the user with a bewildering choice, our policy is to select one c two routines that should handle most needs. For more on the philosophy and implementation of the NAPL, see [2,3,5,7].

We strongly recommend use of this library whenever possible. The time is long past when the applications programmer could write numerical routines as efficient and reliable as those that have been developed by the numerical analysis community over the past two decades. If you do not believe this, test your favorite homemade product against QAGS (integration), ODE (ordinary differential equations), RG (eigenvalues of a matrix), or PWSCRT (Poisson/Helmholtz eqs.)!

1.2 ORIGIN OF NAPL ROUTINES

NAPL routines come from a variety of sources. Most are heavily tested programs that are in wide use around the world. The most important sources are these:

Argonne National Laboratory
IMSL (Internat'l Math. and Stat. Libraries, Inc.) [9]
National Center for Atmospheric Research
National Physical Laboratory, England
Sandia National Laboratories
Stanford University Computer Science Department

The entire IMSL library, a large collection of numerical analysis routines for various purposes, is installed as part of the NAPL in (with a few exceptions) double precision [9]. Also part of the NAPL are the FUNPACK, EISPACK, LINPACK, and MINPACK libraries from Argonne National Laboratory, and the NPL Optimization Library from the National Physical Laboratory in England. The following numerical software libraries are NOT available at SLAC: NAG, HARWELL, SSP. However, information on some of these may be obtained from a Numerical Analysis Constultant. Certain routines from the CERN library (European Center for Nuclear Research, Geneva) are also available; see an NA Consultant. For statistical computations beyond those covered by the linear systems, data fitting, and random numbers chapters here, see documentation for IMSL or SAS.

Numerical analysis is a rapidly changing field. Good introductions may be found in [1,4,6,8,10]. Those who wish to examine the current literature should be aware of the following journals:

<u>ACM Transactions on Mathematical Software</u>
<u>Journal of Computational Physics</u>

<u>Mathematics of Computation</u>
<u>Numerische Mathematik</u>
<u>SIAM Journal on Numerical Analysis</u>
<u>SIAM Review</u>
<u>SIAM Journal on Scientific and Statistical Computing</u>

1.3 HOW TO USE THE NAPL

This guide is Module 19 of the SLAC VM Notebook, and can be printed with the command

```
PRINTDOC NOTEBOOK MOD19
```

To find out what routine to use, read the appropriate section here. For complete documentation on the routine selected, enter

```
HELP NAPL <routine> [( FORM | DESC | PARM | ALL ]
```

or to print the same HELP file on the line printer,

```
HELPRINT NAPL <routine>
```

NAPL routines are stored in the TXTLIBs FORTSLAC and NAPL. Before running a program that calls one of them, therefore, one must enter a command such as

```
GLOBAL TXTLIB NAPL FORTSLAC FORTMOD2
```

All NAPL routines will then be accessed automatically without further effort.

1.4 SAMPLE PROGRAMS AND SOURCE CODE

For most NAPL routines, one or more sample driver programs are available. To print or run them, enter

```
NATEST <routine>
```

Usually there should be no need to obtain the original Fortran source code for a routine. However, source code can be obtained with the command

```
NASOURCE <routine>
```

Most NAPL routines are in the public domain, so their source can be freely moved to other computers. However, some are proprietary and subject to restrictions. An IMSL routine may be taken from SLAC only if it constitutes a necessary part of a working program. NPL routines may not be taken under any circumstances, and NASOURCE will not provide source for them.

During most of the year, Numerical Analysis Consultants from the Stanford Computer Science Department are on duty during posted hours in the SLAC Computer Building. We will be happy to discuss questions related to t e NAPL and give advice on numerical problems not handled by these routines.

1.5 REFERENCES FOR CHAPTER 1

[1] K. E. Atkinson, An Introduction to Numerical Analysis, Wiley, 1978.

[2] J. Bolstad, et al., "Numerical Analysis Program Library User's Guide," SCS User Note 82, 1979*.

[3] T. F. Chan, et al., "A numerical library and its support," ACM Trans. on Math. Software 6 (1980), 135-45*.

[4] S. D. Conte and C. de Boor, Elementary Numerical Analysis: An Algorithmic Approach, 3rd ed., McGraw-Hill, 1980.

[5] W. M. Coughran, Jr., "A note concerning the construction of a numerical analysis program library," SCS Tech. Memo 107, 1977*.

[6] G. Dahlquist, A. Bjorck, and N. Anderson, Numerical Methods, Prentice-Hall, 1974*.

[7] J. Ehrman, "Program library maintenance and monitoring," SCS Tech. Memo. 103, 1977*.

[8] G. Forsythe, M. Malcolm, and C. Moler, Computer Methods for Mathematical Computations, Prentice-Hall, 1977*.

[9] International Mathematical and Statistical Libraries, Inc., IMSL 8 Reference Manual, 1980*.

[10] E. Isaacson and H. B. Keller, Analysis of Numerical Methods, Wiley, 1966.

* Available at the Service Desk in the front lobby area of the SLAC Computer Building (CGB).

2. SYSTEMS OF LINEAR EQUATIONS

Type of matrix	Full	Band
General real	DGEFCS	DGBFCS
Symmetric positive definite	DPPFCS	DPBFCS
Symmetric ind finite	DSPFCS	
General complex	ZGEFCS	

Other special structure... see LINPACK Users' Guide [2]
Inverse or determinant... DGEFDI (see ** below)
Large and sparse... see a Numerical Analysis Consultant
Interactive matrix laboratory... MATLAB

2.1 INTRODUCTION

A system of linear equations may be written

$$Ax = b$$

where A is a known n-by-n matrix of coefficients, b is a known n-vector, and x is an unknown vector that is to be determined. Solving such a system is an extremely common problem in numerical computation. For general matrices the best method is Gaussian elimination with partial pivoting, a process that takes approximately n**3/3 floating point operations.

Here are some basic matrix properties that are relevant to the solution of Ax = b:

(1) Size - The matrix is small if it can be stored in main storage (at SLAC, say, n<=400). Otherwise A is large and auxiliary storage may be needed.
(2) Sparseness - If the percentage of non-zero elements in A is relatively large (>5-10%) then A is dense. Otherwise A is sparse.
(3) Real or Complex - If all the elements of A are real then A is real. Otherwise A is complex.
(4) Symmetry - A is symmetric if it is real and A(i,j)=A(j,i) for all i,j . A is hermitian if A(i,j) equals the complex conjugate of A(j,i) for all i,j.
(5) Banded - A is banded if there exists m<<n such that A(i,j)=0 for |i-j|>m; the band width is 2*m+1. If m=1, then A is tridiagonal.
(6) Positive definite - A is symmetric positive definite if it is symmetric and all of its eigenvalues are positive.
(7) Condition number - The condition number of A is defined by

$$\text{cond}(A) = ||A|| * ||A^{-1}||$$

where $||A||$ is some norm of A. Nearly singular matrices have large condition numbers. The solution x will normally have relative accuracy approximately equal to machine precision times cond(A).

2.2 LINPACK

At SLAC the basic package for numerical linear algebra is LINPACK, an advanced and widely-distributed collection of routines from the Argonne National Laboratory [2]. The routines listed above are all drivers written at SLAC that call LINPACK routines. In cases where A has special structure, such as symmetry, the use of the corresponding special driver will increase both speed and accuracy. Additional routines for specialized computations other than those listed above are described in [2]. LINPACK also contains routines for computing QR decompositions.

LINPACK routines do not use the method of iterative improvement to attempt to achieve higher accuracy in solving linear systems. It is now felt that in general the advantages of iterative improvement do not outweigh the disadvantages. See [2] for a discussion of this.

2.3 OVER- AND UNDERDETERMINED SYSTEMS

In all cases it is assumed that A is square and nonsingular. If the problem is overdetermined (i.e., if A has more rows than columns), then a linear least squares solution is probably what is needed; see the data fitting chapter of this guide or the discussion of QR and singular value decompositions in [2] and [5]. If the problem is underdetermined (A has more columns than rows), then there is serious question as to what one might mean by a solution. A pseudoinverse or linear programming solution might make sense, but the user should rethink his problem and make sure he has used all the information at hand. Even if the matrix is square it may still be singular. In this case a solution is not uniquely determined and may not exist at all. Even if the matrix is theoretically nonsingular, it may be nearly singular computationally so that for practical purposes only limited accuracy can be obtained. The routines listed above return an estimate of cond(A), if the user requests it, and this can be used to detect near-singularity.

2.4 **MATRIX INVERSES

Explicit calculation of the inverse of a matrix is almost never needed and should generally be avoided for reasons of efficiency and accuracy. In solving linear equations, the triangular (LU) decomposition resulting from Gaussian elimination is all that is needed. Computing this d~compositon requires roughly three times fewer floating-point operations than computing the inverse (n**3/3 vs. n**3). Furthermore, the LU decomposition is more accurate, and it is just as easy to use as the inverse, even if numerous right hand sides b are to be used with the matrix A. The documentation for the routines above should make these details clear.

2.5 LARGE SPARSE MATRICES

For large, sparse linear systems (as arise for example in the discretization of partial differential equations), straightforward Gaussian elimination may be impossible. An experimental code called YALEPACK is available for such problems that takes advantage of the sparsity; see a Numerical Analysis Consultant. Like the LINPACK routines above, YALEPACK is based on so-called direct methods related to Gaussian elimination [4]. For some large sparse problems, however, it may be advantageous to use iterative methods instead, such as conjugate gradients [1] or Jacobi, Gauss-Seidel, or SOR [6, esp. Sec. 3.1; 8].

2.6 MATLAB -- INTERACTIVE MATRIX LABORATORY

See the Matrix Eigenvalue Analysis section for a discussion of this facility.

2.7 REFERENCES FOR CHAPTER 2

An extensive discussion of almost every aspect of solving linear equations numerically is found in [3]. Both algorithms and theory are also presented in [5]. For a helpful handbook with many examples, see [7].

[1] P. Concus, G. Golub, and D. O'Leary, "A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations," in J. Bunch and D. Rose (eds.), Sparse Matrix Computations, Academic Press, 1976.

[2] J. J. Dongarra, et al., LINPACK Users' Guide, SIAM, 1979*.

[3] G. E. Forsythe and C. B. Moler, _Computer Solution of Linear Algebraic Systems_, Prentice-Hall, 1967*.

[4] A. George and J. Liu, _Computer Solution of Large Sparse Positive Definite Systems_, Prentice-Hall, 1981.

[5] G. W. Stewart, _Introduction to Matrix Computations_, Academic Press, 1973.

[6] R. S. Varga, _Matrix Iterative Analysis_, Prentice-Hall, 1962.

[7] J. R. Westlake, _A Handbook of Numerical Matrix Inversion and Solution of Linear Equations_, Wiley, 1968.

[8] D. M. Young, _Iterative Solution of Large Linear Systems_, Academic Press, 1971.

* Available at the Service Desk in the front lobby area of the SLAC Computer Building (CGB).

3. MATRIX EIGENVALUE PROBLEMS AND SVD

Type of Matrix	Routine
complex general,Hermitian	CG,CH
real general,symmetric	RG,RS
real tridiagonal	RT,RST
real symmetric banded	RSB
real symmetric (packed storage)	RSP
generalized problem (Ax=sBx)	RGG,RSG
singular value decomposition	DSVDDR
interactive matrix laboratory	MATLAB

Large and sparse -- see an N. A. Consultant

3.1 INTRODUCTION

In the classical eigenvalue problem, an n-by-n matrix A is given and one seeks one or more of its eigenvalues s (scalars) and/or eigenvectors x (n-vectors), satisfying

$$Ax = sx .$$

Every n-by-n matrix A has n eigenvalues (counted with multiplicity), which may be complex even if A if real. If A has distinct eigenvalues then it will have n linearly independent eigenvectors; otherwise it may not. A matrix with an incomplete set of eigenvectors is called defective. In the special case in which A is real symmetric or complex Hermitian (A equals its complex conjugate transpose), A always has n real eigenvalues and a complete set of orthonormal eigenvectors.

The eigenvalue problem for Hermitian matrices is well conditioned, and the routines above will normally return eigenvalues accurate to machine precision relative to the norm of A. In the non-Hermitian case some eigenvalues will be ill-conditioned when A is nearly defective. The eigenvector problem is in general ill-conditioned whether A is Hermitian or not. In particular, eigenvectors associated with close or multiple eigenvalues will often change radically if A is perturbed slightly. A process called balancing is applied in the above routines to improve the condition of the non-Hermitian eigenvalue problem.

3.2 EISPACK

All of the recommended routines except DSVDDR and MATLAB come from EISPACK, an EIgenSystem PACKage from the Argonne National Laboratory [2,5]. EISPACK is one of the most successful achievements of applied numerical analysis, and is extremely reliable and efficient. Each program listed above is a driver that calls a sequence of more specialized EISPACK routines. The drivers permit computation of eigenvalues and eigenvectors or of eigenvalues alone. Various other specialized problems besides those in the list can also be solved efficiently by EISPACK. For example, routines are available to compute a small number of eigenvalues rather than all or none of them. See [2,5] for further information.

3.3 GENERALIZED EIGENVALUE PROBLEM

In the standard generalized eigenvalue problem, a matrix appears on both sides of the equation,

$$Ax = sBx \ .$$

If B is nonsingular one might reduce this to a simple eigenvalue problem by multiplying both sides by B-inverse, but this is usually not a good idea. The routines RGG and RSG above treat the cases of general or symmetric A and B, respectively. Additional EISPACK routines for other kinds of generalized eigenvalue problems are also available [2].

3.4 SINGULAR VALUE DECOMPOSITION

The singular value decomposition (SVD) of a real m-by-n matrix A is a factorization A = UDV', where U and V are orthogonal matrices, V' denotes the transpose of V, and D is a diagonal matrix with positive real entries, which are called the <u>singular values</u> of A. The singular values are equal to the non-negative square roots of the eigenvalues of A'A. The SVD is important in numerical analysis in many applications. The largest singular value of A equals the Euclidean 2-norm of A, and the ratio of the largest to the smallest singular values is the condition number of A in that norm. If m=n, then A is singular if and only if at least one singular value is zero, and in general the number of nonzero singular values equals the rank of A. The SVD is also useful in computing the pseudo-inverse of a matrix [3] and in solving ill-conditioned least squares problems [1,2]. The recommended routine DSVDDR is a driver written at SLAC for routines from LINPACK [1].

3.5 LARGE SPARSE EIGENVALUE PROBLEMS

If A is large and sparse, and only a few of its eigenvalues are needed, the direct methods employed by EISPACK may not be appropriate. For information on software for alternative methods, particularly the Lanczos algorithm [4], see a Numerical Analysis Consultant.

3.6 MATLAB — INTERACTIVE MATRIX LABORATORY

A facility called MATLAB is available for interactive matrix computations under VM, written by Cleve Moler at the University of New Mexico. MATLAB is based on routines from EISPACK and LINPACK, and is extremely easy to use. It provides a full range of linear algebra computations, including eigenvalue and singular value decompositions, LU and QR and Cholesky decompositions, ranks, norms, solution of systems of equations, determinants, and inverses. For information enter HELP NAPL MATLAB or see a Numerical Analysis Consultant.

3.7 REFERENCES FOR CHAPTER 3

The EISPACK library is well documented in [2] and [5]. A comprehensive treatise on both the theoretical and computational aspects of solving eigenvalue problems is found in [7]. For a very readable treatment of the symmetric eigenvalue problem, see [4].

[1] J. J. Dongarra, et al., LINPACK User's Guide, SIAM, 1979*.

[2] B. S. Garbow, et al., Matrix Eigensystem Routines - EISPACK Guide Extension, Springer-Verlag Lecture Notes in Computer Science 51, 1977*.

[3] G. H. Golub and W. Kahan, "Calculating the Singular Values and Pseudoinverse of a Matrix," SIAM J. Numer. Anal. 2 (1965), pp. 205-224.

[4] B. N. Parlett, The Symmetric Eigenvalue Problem, Prentice-Hall, 1980*.

[5] B. T. Smith, et al., Matrix Eigensystem Routines - EISPACK Guide, 2nd ed., Springer-Verlag Lecture Notes in Computer Science 6, 1976*.

* Available at the Service Desk in the front lobby area of the SLAC Computer Building (CGB).

[6] G. W. Stewart, Introduction to Matrix Computations, Academic Press, 1973.

[7] J. H. Wilkinson, The Algebraic Eigenvalue Problem, Oxford University Press, 1965.

4. NONLINEAR EQUATIONS AND OPTIMIZATION

<u>Zerofinding</u>

one variable real	ZEROIN [1,2]
one variable complex analytic	ZANLYT [5]
real, complex polynomial	PA07AD, PA06AD [7]
re l system (fast)	ZSYSTM [5]
real system (robust)	HYBRD/HYBRD1[8] or NS01A[12]
real system (analytic Jacobian)	HYBRJ/HYBRJ1 [8]

<u>Optimization</u>	<u>requires</u> <u>F</u>(<u>x</u>)	<u>requires</u> <u>F</u>(<u>x</u>),<u>g</u>(<u>x</u>)	<u>requires</u> <u>F</u>(<u>x</u>),<u>g</u>(<u>x</u>),<u>G</u>(<u>x</u>)	
one variable	FMIN [2]	UNIGRD [10]		
system	QNMDIF/UBNDQ1	QNMDER/UBFDQ2	MNA/UBSDN2	[10]
least-squares system	LMDIF/LMDIF1	LMDER/LMDER1		[8]
linearly constrained	LCQNDF	LCQNDR	LCMNA	[10]
nonlin. constrained	SALQDF	SALQDR	SALMNA	[10]
linear programming	LP [10]			

(xxx/yyy - yyy is an easy-to-use version of xxx)

/

4.1 INTRODUCTION

We are concerned here with the closely related problems of locating points where functions take on zero values or extreme (minimal or maximal) values. These problems are related in a theoretical sense in that extrema of functions correspond to zeros of their derivatives; conversely, a zero of a function F might be found by locating a minimum of F**2. Only minimization will be discussed here, since a maximum of F is a minimum of -F. Optimization is also closely related to questions of approximation and data fitting, which are discussed in a separate section.

Algorithms for either problem are based on iteratively approximating the given function locally by another simpler function whose zeros or extrema can be calculated more easily. All of them, however, are subject to a variety of pitfalls, and some of the most rapidly convergent methods in theory can also get into the most trouble if care is not taken in their application. Among the problems which may be encountered are slow convergence to multiple or close roots, loss of accuracy in using deflation to find more than one root, oscillation due to almost horizontal tangents, and convergence to the wrong zero or to the wrong local minimum.

4.2 LOCAL VS GLOBAL SOLUTIONS

In general it is impossible to determine whether a set of zeros found numerically includes all the zeros of the function, or whether a minimum found numerically is a global minimum or only local. One way to check this is to try several different (widely separated) starting estimates and see if the same solution results. The computational problem becomes much more difficult in several dimensions than in one, and more difficult again if constraints are present.

4.3 DERIVATIVE INFORMATION

Many methods for multivariate problems are based on Newton's method or generalizations thereof, and so usually require derivative information about the function. In solving nonlinear equations the Jacobian matrix (first partials) of the system may be required. For nonlinear function minimization the gradient vector g (first partials) or Hessian matrix G (second partials) of the objective function may be needed. In general, the more derivative information a method requires the faster its potential convergence rate, but at the same time such a method is more trouble to the user, may be less robust, and may require a better initial estimate than a method which uses less information about derivatives.

4.4 ZEROFINDING

The table above lists recommended routines for finding zeros of functions of one variable -- real, complex, or (a special case) polynomial. Polynomials have the advantage that one knows exactly how many zeros to look for. For finding a zero of a system of nonlinear equations several routines are available. For very rapid convergence from good initial estimates we recommend ZSYSTM. If some of the equations in the system are linear, then ZSYSTM can take special advantage of this. The remaining routines are more robust programs capable of handling much poorer initial estimates. HYBRD/HYBRD1 and HYBRJ/HYBRJ1 come from the MINPACK-1 library released in 1980 by the Argonne National Laboratory [8]. If it is is possible to compute a Jacobian matrix analytically, use HYBRJ/HYBRJ1 in preference to HYBRD/HYBRD1 or NS01A. In each case of two routines separated by a slash, the second routine listed is an easy-to-use but less flexible version of the first.

4.5 OPTIMIZATION

Many optimization problems, especially those involving minimizing a sum of squares, come from data fitting problems. See the Approximation and Data Fitting section for more information on these. For general optimization problems of all kinds, our main source of routines is the Numerical Optimization Software Library recently released by the National Physical Laboratory (NPL) in Teddington, England. This library is the most comprehensive and sophisticated collection of optimization routines currently available, and it should handle most optimization problems effectively. Unfortunately, these routines are for the most part quite difficult to use. We recommend that the user with a reasonably straightforward problem begin with an easy-to-use routine, then move to a standard routine if he has trouble finding a solution, if efficiency is a problem, or if he intends to solve a similar problem over and over again. The three columns in the table show what function information must be supplied to each routine: F(x) = function; g(x) = gradient vector; G(x) = Hessian matrix.

4.6 CONSTRAINED OPTIMIZATION

The NPL library contains powerful routines for constrained optimization problems, both linear and nonlinear. Both equality and inequality constraints are permitted, and they may be mixed. The special case of a linear objective function subject to linear constraints is called a linear programming problem and should be solved by a routine such as LP, which is specifically designed for this problem.

The table above is far from a complete list of routines in the NPL optimization library. Additional NPL routines perform the following functions, among others:

- Check user-supplied first or second derivatives numerically for consistency with the given function (highly recommended!)

- Solve problems efficiently which involve simple bound (rather than general linear) constraints

- Solve minimization problems involving unsmooth (possibly discontinuous) objective functions

The user with a special problem should refer to [9,10] or see a Numerical Analysis Consultant for further information.

4.7 LARGE SPARSE OPTIMIZATION PROBLEMS

A powerful system called MINOS is available to deal with large optimization problems, linear or nonlinear, unconstrained or constrained. See a Numerical Analysis Consultant for information.

4.8 REFERENCES FOR CHAPTER 4

For a linear least-squares minimization problem, see the chapter on data fitting for further recommendations. See [13] and [11], respectively, for solution of nonlinear equations in one and several variables. The booklet [8] gives a simple guide to the MINPACK collection of routines. There are many books on constrained optimization, of which [4] and [6] are particularly useful for applied problems. Users of the NPL routines should refer to [9] and [10] for much more advice on code use and selection; further information on the NPL routines as well as on other topics in optimization is given in [3].

[1] R. Brent, Algorithms for Minimization without Derivatives, Prentice-Hall, 1973*.

[2] G. Forsythe, M. Malcolm, and C. Moler, Computer Methods for Mathematical Computations Prentice-Hall, 1977*.

[3] P. Gill, W. Murray, and M. Wright, Practical Optimization, Academic Press, 1981.

[4] Himmelblau, D. M., Applied Nonlinear Programming, McGraw-Hill, 1972.

[5] International Mathematical and Statistical Libraries, IMSL 8 Reference Manual, 1980*.

[6] D. G. Luenberger, Introduction to Linear and Nonlinear Programming, Addison-Wesley, 1973

[7] K. Madsen and J. Reid, "Fortran Subroutines for Finding Polynomial Zeros," Report R.7986, AERE, Harwell, England, 1975.

[8] J. More, B. Garbow, and K. Hillstrom, User Guide for MINPACK-1, Argonne National Laboratory Report ANL-80-74, 1980*.

[9] National Physical Laboratory, "A Brief Guide to the NPL Numerical Optimization Software Library," 1978*.

* Available at the Service Desk in the front lobby area of the SLAC Computer Building (CGB).

[10] National Physical Laboratory, Introduction to the NPL Numerical Optimization Software Library, Vols. I & II, 1978*.

[11] J. Ortega and W. Rheinboldt, Iterative Solution of Nonlinear Equations in Several Variables, Academic Press, 1970.

[12] M. J. Powel , "A FORTRAN Subroutine for Solving Systems of Nonlinear Equations," Report R.5947, AERE, Harwell, England, 1968.

[13] J. F. Traub, Iterative Methods for the Solution of Equations, Prentice-Hall, 1964.

5. APPROXIMATION AND DATA FITTING

Cubic spline interpolation	CSPLIN/CSEVAL
Least-squares cubic spline	ICSFKU/ICSEVU
General linear least-squares	LINLSQ
General nonlinear least-squares	VARPRO
Rational Chebyshev approximation	IRATCU

5.1 INTRODUCTION

When fitting a model to observed data or approximating a complicated function with a simpler one, there are two major choices to make: the functional form of the approximation, and the method or norm used to define it. Common approximation functions include polynomials, rational functions, sums of exponentials, and splines. The simplest approximation method is interpolation, and the other main possibility is to minimize or nearly minimize some norm of the error between the data and the approximation. The Euclidean 2-norm measures the sum or the integral of the squares of the error at each data point. The maximum (also called supremum, minimax, or Chebyshev) norm gives the maximum pointwise error.

In general least-squares fits are easier to compute than minimax fits, and for most purposes they are nearly as good. They are particularly appropriate for fitting data points that show overall trends but may contain random noise. Interpolatory approximations are also easy to compute, but are more hazardous (see below).

5.2 INTERPOLATION AND LEAST-SQUARES FITTING BY CUBIC SPLINES

A cubic spline is a piecewise cubic polynomial with continuous first and second derivatives at the breakpoints, or knots. This is the most reliable general-purpose form of approximation. For cubic spline interpolation, use CSPLIN and CSEVAL. For a least-squares spline fit, which is usually preferable, use the IMSL routines ICSFKU and ICSEVU. For this it will be necessary to choose breakpoints for the spline. The general idea is to put more breakpoints where the function varies rapidly, but some experimentation may be necessary. (The IMSL library also offers a variable-knot spline routine to select breakpoints automatically, but this is an expensive and unreliable process, and we do not recommend it for most applications.)

The IMSL library further offers a smoothing spline routine, and bicubic splines for two-dimensional data fitting; see [2]. A collection of the subroutines from the excellent text by de Boor [1] is also available from a Numerical Analysis Consultant. These programs permit a wide variety of spline computations not handled by the routines above, including manipulation of splines of arbitrary order.

5.3 GENERAL LINEAR AND NONLINEAR LEAST-SQUARES

A linear least squares problem is one that is linear in the unknown parameters even though it is usually nonlinear in the independent variable. For example, quadratic regression uses

$$f(t_i) = a_0 + a_1*t_i + a_2*t_i**2.$$

For regression problems of small degree, say n < 5 , it is common to solve this problem by means of the so-called normal equations [3]. This procedure is in general ill-conditioned, and we recommend the routine LINLSQ instead even if n is small and especially if n is at all large. LINLSQ is based on a QR decomposition [3]. For nonlinear least squares problems, use the routine VARPRO. This routine takes advantage of the common occurrence that some of the parameters enter linearly, such as the a's in the exponential sum

$$f(t_i) = a_0 + a_1*exp(-b_1*t_i) + a_2*exp(-b_2*t_i) ,$$

making it much faster than more general nonlinear routines in these cases.

Both LINLSQ and VARPRO make it easy to perform fits by much more complicated sets of functions than polynomials or exponential sums. For example, in some problem it might be appropriate to fit a set of data on [0,infinity) by a continuous function made up of a polynomial on [0,1] connected to a decaying exponential on [1,infinity). Try to find a functional form that is physically and graphically reasonable.

For constrained least squares, see a Numerical Analysis Consultant about the availability of experimental codes. In general, approximation in least squares or other norms is closely related to optimization problems, which are discussed in the Nonlinear Equations and Optimization section of this guide.

5.4 POLYNOMIAL INTERPOLATION

Interpolation by polynomials (as opposed to piecewise polynomials) is a dangerous practice, particularly at high degree, because it can lead to large unwanted oscillations (the "Runge phenomenon"). To alleviate this problem one should interpolate not at equally spaced points but at Chebyshev points, that is,

$$x(j) = cos ((2j+1) pi/(2n+2)), \quad j = 0,...,n$$

for the interval [-1,1]. One should also use a well-conditioned basis of orthogonal polynomials, rather than the monomials x**n .

5.5 TRIGONOMETRIC INTERPOLATION AND LEAST-SQUARES FITS

These are appropriate for periodic functions or data. See the section on the Fast Fourier Transform.

5.6 RATIONAL MINIMAX APPROXIMATION

The routine IRATCU applies the Remes algorithm to find the high-accuracy rational minimax approximation to a known function [2]. That is, it finds the quotient of polynomials of specified degree that minimizes the maximum absolute error over an interval.

5.7 REFERENCES FOR CHAPTER 5

The book by de Boor [1] is a comprehensive and readable reference on splines. For an introduction to regression analysis see [4]. See [5] for a short but thorough discussion of modern methods for solving least squares problems. Lawson and Hanson [3] is the best general reference on least squares and constrained least squares problems.

[1] C. de Boor, A Practical Guide To Splines, Springer-Verlag, 1978*.

[2] International Mathematical and Statistical Libraries, IMSL 8 Reference Manual, 1981*.

[3] C. L. Lawson and R. J. Hanson, Solving Least Squares Problems, Prentice-Hall, 1974*.

[4] G. A. F. Seber, Linear Regression Analysis, Wiley, 1977.

[5] C. Van Loan, "Lectures in Least Squares," TR 76-279, Department of Computer Science, Cornell University, 1976*.

* Available at the Service Desk in the front lobby area of the SLAC Computer Building (CGB).

6. FAST FOURIER TRANSFORM

Real transform	RFFTI, RFFTF, RFFTB	
Complex transform	CFFTI, CFFTF, CFFTB	all
Sine transform	SINTI, SINT	routines
Cosine transform	COSTI, COST	single
1/4-wave si e transform	SINQI, SINQF, SINQB	precision
1/4-wave cosine transform	COSQI, COSQF, COSQB	

6.1 INTRODUCTION

The usual continuous Fourier integral transform (FT) is well known and used throughout physics and engineering wherever phenomena may be broken down into frequency components. Discrete Fourier transforms (DFTs) arise both as approximations to FTs and in their own right in a wide variety of applications. For a comparison of the FT and the DFT, see [2] and the remarks on trigonometric least-squares fitting, below. The Fast Fourier Transform (FFT) is the name of a class of extremely rapid algorithms for computing DFTs. To compute an FFT of length N requires on the order of N*log(N) floating-point operations.

Let X(j) be a complex vector of length N. The unnormalized forward complex DFT of X(j) is another complex N-vector defined by the formula

```
                    N
        Y(k)  =   SUM X(j) * EXP(-2i*j*k*PI/N)
                  j=1
```

where i = SQRT(-1) . Y(k) may be interpreted as N times the component of the vector X in the "direction" of the exponential EXP(2i*k*PI/N). The inverse complex DFT is defined by

```
                    N
        Z(j)  =   SUM Y(k) * EXP(+2i*j*k*PI/N)
                  k=1
```

This pair of formulas is unnormalized in the sense that if X is transformed to Y and then Y is inverse transformed to Z, then Z will be equal to X multiplied by the factor N.

6.2 SWARZTRAUBER FFT PACKAGE

The routines above make up a package for FFT computations recently released by Paul Swarztrauber of the National Center for Atmospheric Research in Boulder, Colorado. These programs are efficient and very easy to use, and are all in single precision (computing the FFT is numerically a v ry stable process). For a general complex DFT, the appropriate routines are

CFFTI -- initialization
CFFTF -- forward transform
CFFTB -- unnormalized inverse transform

These routines can handle arbitrary dimensions N, but are much more efficient if N is a product of small prime numbers such as 2, 3, 5. (However, padding a sequence of data with zeros to make N a round number is not recommended, because it can lead to unwanted oscillations; this is known as the Gibbs phenomenon.) The remaining routines are designed for special cases of complex DFTs, where the speed can be increased and the storage requirement reduced by taking advantage of the special structure. If X is real, use RFFTI, RFFTF, RFFTB. If X is real and even, use COSTI and COST. If it is real and odd, use SINTI and SINT. (Sine and cosine transform routines come in pairs rather than triplets because the forward transform turns out to be its own unnormalized inverse.) If it is real and even and contains only odd wave numbers (e.g. COS(T) + COS(3T)), use COSQI, COSQF, COSQB. If it is real and odd and contains only odd wave numbers, use SINQI, SINQF, SINQB.

6.3 TRIGONOMETRIC INTERPOLATION AND LEAST-SQUARES FITTING

Suppose X(j) above represents the values at x = j of a continuous function X(x) defined for all x and having period N . Then Z(j) also extends to a continuous function Z(x), and this equals N times the trigonometric sum of degree N that interpolates X(x) at the integers j . If the sum defining Z(x) is truncated at some M < N , then one obtains N times the trigonometric polynomial which is the best least-squares approximation to X(x) of degree M with respect to the integers x = j . If N is large relative to the smoothness of X(x) , this will be very close to the best least-squares fit on the continuum of all x values, which is the partial sum of degree M of the continuous FT of X(x).

6.4 REFERENCES FOR CHAPTER 6

Hamming [3] provides a comprehensive treatment of the underlying theory of Fourier analysis, both continuous and discrete. Almost one third of the book deals with Fourier approximation. Chapter 9 of [1] also discusses Fourier methods and on page 552 there is a list of algorithms for the FFT published during the period 1960-70. For other presentations of the FFT together with interesting applications, see [4] and [5].

[1] G. Dahlquist, A. Bjorck, and N. Anderson, Numerical Methods, Prentice-Hall, 1974*.

[2] W. Gentleman and G. Sande, "FFT for Fun and Profit," Proceedings of the Fall Joint Computer Conference, 1966, 563-578.

[3] R. W. Hamming, Numerical Methods for Scientists and Engineers, 2nd ed., McGraw-Hill, 1973.

[4] P. Henrici, "Fast Fourier methods in complex analysis," SIAM Review 21 (1979), 481-527.

[5] L. R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing, Prentice-Hall, 1975.

* Available at the Service Desk in the front lobby area of the SLAC Computer Building (CGB).

7. NUMERICAL INTEGRATION

Adaptive	QAGS [3] or DCADRE [6]
Gaussian	GAUSSQ/QUADS3 [5]
tabulated data	DCSQDU [6]
multiple integrals	DBLINT [6] or GAUSSQ/QUADS3 [5]

7.1 ADAPTIVE QUADRATURE

Evaluating a single one-dimensional integral has become an easy task through the invention of powerful adaptive quadrature routines such as QAGS and DCADRE. ("Quadrature" = 1D numerical integration.) Rather than applying a fixed formula, these routines examine the behavior of the integrand as they proceed, calling for few function evaluations where the integrand is smooth and many where it is ill-behaved. This enables them to treat irregular integrands successfully and efficiently, and more important, to return a result that is accurate (almost always) to within a tolerance specified by the user. For one-shot integration, even of very ill-behaved functions, they are completely satisfactory. We recommend QAGS, which is somewhat faster than DCADRE and also in the public domain. For large numbers of integrals, on the other hand, adaptive quadrature may be much too time-consuming, and one should consider Gaussian quadrature or another method.

7.2 GAUSSIAN QUADRATURE

Gaussian quadrature, by contrast, consists of applying a fixed rule of the form

$$I(f) = \sum_{i=1}^{K} w_i * f(x_i) ,$$

to estimate the integral. Here the values x(i) are an optimally chosen set of K "nodes" and the coefficients w(i) are K "weights". Rather than choosing an error tolerance, the user must now choose K. This calls for some experimentation (try, for example, K = 4,8,16). The advantage of Gaussian rules is that they are extraordinarily accurate if the integrand is smooth. (The K-point Gaussian rule integrates all polynomials of degree 2*K-1 exactly.) Often a Gaussian formula will achieve as high accuracy as an adaptive program with an order of magnitude fewer integrand evaluations. Use routines GAUSSQ (for computing nodes and weights) and QUADS3 (for summing the formula once the nodes and weights are known).

7.3 MULTIPLE INTEGRALS

As the dimension N of an integral increases from 1 to 2, 3, or more, numerical integration rapidly becomes difficult, then almost intractable. On top of this, the penalty for using adaptive rather than Gaussian routines may increase geometrically. For N = 2 one may try the IMSL routine DBLINT, an adaptive program based on DCADRE. But for evaluating many double integrals, or integrals in three or more dimensions, DBLINT may be extremely expensive. First, make certain that is is not possible to eliminate one or more of the dimensions of integration analytically. If it is not, use nested Gaussian quadrature if the integrand is reasonably smooth. Since Fortran does not permit recursion, this will necessitate acquiring N-1 copies of the source for QUADS3 and renaming them. Alternately, as QUADS3 does nothing more that evaluate a sum of K products, the sum can be coded in a few lines by the user.

For truly ill-behaved integrands with N = 3 or more, an experimental adaptive code can be obtained from a Numerical Analysis Consultant. Many users may also be familiar with Monte-Carlo integration, but this is an extremely time-consuming process and should only be considered as a last resort.

7.4 SINGULARITIES AND DISCONTINUITIES

Special Gaussian formulas are available to handle singularities at endpoints of type x**a (a > -1); see the documentation for GAUSSQ on Gauss-Jacobi and Gauss-Chebyshev quadrature. For singularities at endpoints of unknown type, QAGS is usually effective. If the integrand has singularities or discontinuities at known points in the interior, it is best to split the integral into subintervals so that these occur only at endpoints. This advice holds even for discontinuities in a higher derivative -- for example, in integrating a cubic spline. If there are internal singularities at unknown locations, use QAGS. See also [1], chapters 4 and 15.

7.5 INFINITE INTERVALS

For infinite intervals with exponential decay factors exp(-x) or exp(-x**2), Gauss-Hermite or Gauss-Laguerre quadrature may be applicable; see the documentation for GAUSSQ. Otherwise, if the integrand decays too slowly for the interval to be truncated at a finite point, it is probably best to change variables so as to reduce the problem to a finite interval with a singularity at one or both endpoints, and then use QAGS.

7.6 INTEGRATION OF TABULATED DATA

If the function to be integrated is defined by a set of tabulated values rather than analytically, there are two main approaches to consider. If the data is located at equally spaced points, one can apply a simple quadrature rule directly, such as the trapezoid rule or Simpson's rule [1,2]. For noisy data or irregular abscissae, one can fit the data by a cubic spline, then integrate the spline exactly. This is done by routine DCSQDU. See also [4], chapter 5.

7.7 REFERENCES FOR CHAPTER 7

For a general discussion of most aspects of numerical integration Davis and Rabinowitz [2] is a good reference. For more detail on Gaussian quadrature see [7] or [9]. For a discussion of the multidimensional case see [8].

[1] F. S. Acton, Numerical Methods that Work, Harper and Row, 1970.

[2] P. J. Davis and P. Rabinowitz, Methods of Numerical Integration, Academic Press, 1975*.

[3] E. de Doncker, (reference not yet available)

[4] G. Forsythe, M. Malcolm, and C. Moler, Computer Methods for Mathematical Computations, Prentice-Hall, 1977*.

[5] G. H. Golub and J. H. Welsch, "Calculation of Gaussian Quadrature Rules," Mathematics of Computation 23 (1969), 221-230.

[6] International Mathematical and Statistical Libraries, Inc., IMSL 8 Reference Manual, 1980.

[7] V. I. Krylov, Approximate Calculation of Integrals, Macmillan, 1962.

[8] A. H. Stroud, Approximate Calculation of Multiple Integrals, Prentice-Hall, 1971.

[9] A. H. Stroud and D. Secrest, Gaussian Quadrature Formulas, Prentice-Hall, 1966.

* Available at the Service Desk in the front lobby area of the SLAC Computer Building (CGB).

8. ORDINARY DIFFERENTIAL EQUATIONS

General	ODE	[11]
General, easier problems	RKF45	[3]
Implicit end condition	DEROOT	[3,5]
Stiff	EPSODE	[4,7]
Stiff, easier problems	GEAR	[4,8]
Boundary-value problems	COLSYS	[1]

8.1 INTRODUCTION

A first-order system of ordinary differential equations has the form

$$y'(t) = f(t,y(t)) ,$$

where y and f are n-vectors and y' denotes dy/dt . Most theory and software for ODEs (including all routines above except COLSYS) begins by eliminating higher-order derivatives, if present, so as to bring the system into this form. This is done by the addition of auxiliary variables. For example, the single second-order equation $y'' = f$ is equivalent to the system of two first-order equations $u' = v$, $v' = f$. See [11] for further examples.

To completely specify the problem boundary conditions are required. In an initial-value problem (IVP) all n conditions are imposed at one point,

$$y(a) = y0 ,$$

where $y0$ is an n-vector and one wishes to solve for $y(t)$ on an interval $a<x<=b$. Software for IVPs is highly advanced, and most of them can be solved rapidly to high accuracy. Boundary-value problems (BVPs) involve boundary conditions at two or more points and are more difficult.

8.2 INITIAL-VALUE PROBLEMS

For most IVPs we recommend the routine ODE [11]. This program is reliable and extremely easy to use. It is based on a variable-step, variable-order Adams method (explicit linear multistep method). For non-stiff problems (see below) with expensive derivative evaluations or a tight accuracy requirement, ODE is almost certainly the best routine. For easier problems in which evaluating the derivatives $f(t,y(t))$ is cheap, however, RKF45 may be faster. This routine is based on Runge-Kutta-Fehlberg formulas [3].

Sometimes it is desired to integrate an IVP from $t=a$ to $t=z$, where z is defined implicitly as the zero of a function $g(t,y,y')$. The routine DEROOT is an augmented version of ODE designed to do this efficiently.

8.3 STIFF IVP'S

A stiff ODE, roughly speaking, is one whose general solution involves both slowly varying components and rapidly varying components of a decaying nature. Stiff problems are numerically difficult becuase very small step sizes may be required in the solution process to avoid numerical instability. For discussions see [4,9,10]. Routines ODE and RKF45 are designed for non-stiff systems, and will work very inefficiently on a stiff problem. However, ODE can usually detect stiffness, and will return an error code when it does. For stiff problems the user should try EPSODE or GEAR, above. Both of these are adaptive codes based on so-called backward differentiation formulas. EPSODE is slower than GEAR, but can handle more difficult problems. An experimental code designed to efficiently handle IVPs arising from a method of lines semidiscretization of a PDE is also available; see a Numerical Analysis Consultant.

8.4 BOUNDARY-VALUE PROBLEMS

Users with boundary-value problems are encouraged to take a look at the brief monograph by Keller [8]. For simple problems, a shooting method may work -- see [8] or many other numerical analysis texts. In general more sophisticated methods are called for, such as those based on finite differences. The program COLSYS is based on a collocation method [1]. It is somewhat difficult to get started, but powerful, and users who will be solving many BVPs or difficult ones should try it. The article [2] describes many tricks to make COLSYS or a similar routine handle special problems: infinite intervals, eigenvalue boundary-value problems, nonstandard boundary conditions, singularities, etc. A special code is also available for solving Sturm-Liouville eigenvalue problems; see a Numerical Analysis Consultant.

8.5 REFERENCES FOR CHAPTER 8

Most numerical analysis books, including [3], contain a chapter on the solution of ODEs. A well-written survey of methods for the IVP is given in [10]. For a more substantial introduction see [9] for IVP's and [8] for BVP's.

[1] U. Ascher, J. Christiansen, and R. D. Russell, "Collocation software for boundary-value ODEs," ACM Trans. Math. Software 7 (1981), 209-222*.

[2] U. Ascher and R. D. Russell, "Reformulation of boundary value problems into 'standard' form," SIAM Review 23 (1981), 238-254*.

[3] G. Forsythe, M. Malcolm, and C. Moler, _Computer Methods for Mathematical Computations_, Prentice-Hall, 1977.

[4] C. W. Gear, _Numerical Initial Value Problems in Ordinary Differential Equations_, Prentice-Hall, 1971*.

[5] M. K. Gordon, "Using DEROOT/STEP,INTRP to Solve Ordinary Differential Equations," SAND-75-0211, Sandia Laboratories, 1975.*

[6] A. C. Hindmarsh, "GEAR: Ordinary Differential Equation System Solver," UCID-30001, Rev. 3, Lawrence Livermore Laboratory, December 1974*.

[7] A. C. Hindmarsh and G. D. Bryne, "EPISODE: An Experimental Package for the Integration of Systems of Ordinary Differential Equations," UCID-30112, Lawrence Livermore Laboratory, May 1975*.

[8] H. B. Keller, _Numerical Solution of Two-Point Boundary Value Problems_, Society for Industrial and Applied Mathematics 24, 1976.

[9] J. D. Lambert, _Computational Methods in Ordinary Differential Equations_, Wiley, 1973.

[10] J. D. Lambert, "The Initial Value Problem for Ordinary Differential Equations: A Survey," Report No. 15, University of Dundee, Department of Mathematics, 1976*.

[11] L. F. Shampine and M. K. Gordon, _Computer Solution of Ordinary Differential Equations: The Initial Value Problem_ Freeman, 1975*.

* Available at the Service Desk in the front lobby area of the SLAC Computer Building (CGB).

9. PARTIAL DIFFERENTIAL EQUATIONS

<u>Modified Helmholtz equation in various coordinate systems</u> $u_{xx}+u_{yy}+su=g$

Cartesian - x and y	PWSCRT
polar - r and theta	PWSPLR
cylindrical - r and z	PWSCYL
surface sperical - theta and phi	PWSSSP
interior spherical - r and theta	PWSCSP

<u>General separable elliptic problems</u>

$au_{xx}+bu_{x}+cu+u_{yy}=g$	POIS
$au_{xx}+bu_{x}+cu+du_{yy}+eu_{y}+fu=g$	BLKTRI

u = unknown function of x,y s = scalar
a,b,c = functions of x d,e,f = functions of y
g = function of x,y

9.1 INTRODUCTION

Partial differential equations (PDEs) come in three broad classes: elliptic, parabolic, and hyperbolic. Generally, elliptic problems correspond to time-independent phenonema (e.g. Poisson's equation), parabolic problems to diffusion processes (e.g. the heat equation), and hyperbolic problems to translational phenonema (e.g. the wave equation). Both the behavior of the solutions and the methods of solution for each class are different. Even more difficult are the problems of mixed type; these are problems with features of two or three of the above classes. Courant and Hilbert [2] presents much of the theory behind PDEs with many phsycial examples.

There are various numerical methods available for PDEs, such as finite differences, finite elements, and the method of lines. Nevertheless, the numerical solution of PDEs is difficult because of the wide variety of boundary conditions, geometries, and physical phenomena involved and the large number of unknowns required in any discretization. As a result general-purpose software is far less developed for the solution of PDEs than for the other problems discussed in this guide.

9.2 ELLIPTIC PROBLEMS

The routines above come from the "FISHPACK" collection of "Fast Poisson solvers" for solving Laplace, Poisson, Helmholtz and related equations by finite differences on simple geometries. These routines are extremely fast and allow for a variety of boundary conditions; see their HELP file documentation and [10] for further information. It is also shown in [10] how these routines can be adapted to three-dimensional problems by using them in conjunction with the Fast Fourier Transform. Where possible, one should use one of the PWSxxx routines above in preference to the more general routines POIS and BLKTRI.

9.3 SPECIAL ELLIPTIC PROBLEMS

Several experimental codes are available from a Numerical Analysis Consultant for special elliptic problems. One package solves Laplace's equation on bounded or unbounded polygonal regions by means of the Schwarz-Christoffel conformal map. Another solves Helmholtz equations on arbitrary regions by capacitance matrix techniques. A third solves the biharmonic equation rapidly on a rectangle. The NAPL does not currently include any of the various large finite-element packages that are on the market.

9.4 PARABOLIC PROBLEMS

Currently we have no routines for solving parabolic PDEs. For simple problems (e.g., the heat equation on a slab), [6] and [1] are good references. Richtmyer and Morton [7] gives a more detailed and technical discussion of finite difference approximations; in particular, the second half of the book discusses some practical applications. For more complicated problems, see a Numerical Analysis Consultant.

9.5 HYPERBOLIC PROBLEMS

Hyperbolic problems are more difficult to solve numerically than elliptic or parabolic problems, and there are currently no routines in the NAPL for them. There is no good reference that covers all aspects of solving hyperbolic equations numerically; however, [1], [5], [6], and [7] discuss various parts of the problem. An experimental code called MR1D is available for solving nonlinear hyperbolic problems in one dimension by a finite-difference scheme with adaptive mesh refinement. For further information see a Numerical Analysis Consultant.

9.6 REFERENCES FOR CHAPTER 9

For a wealth of non-numerical information, as mentioned above, see [2]. A good introductory description the numerical solution of all three classes of PDEs is found in [8]. The books [3] and [11] contain fairly advanced treatments of finite difference methods for elliptic problems. For general elliptic problems, the finite element method is often the best choice; see [9] for theory and [4] for practice. For recent developments it may be worthwhile to consult some of the technical journals listed in the Introduction.

[1] W. F. Ames, Numerical Methods for Partial Differential Equations, 2nd ed., Academic Press, 1977.

[2] R. Courant and D. Hilbert, Methods of Mathematical Physics, v. 2, Wiley-Interscience 1962.

[3] G. E. Forsythe and W. R. Wasow, Finite Difference Methods for Partial Differential Equations, Wiley, 1960.

[4] K. H. Huebner, The Finite Element Method for Engineers, Wiley, 1975.

[5] H.-O. Kreiss and J. Oliger, Methods for the Approximate Solution of Time-Dependent Problems, Global Atmospheric Research Programme Publication No. 10, 1973.

[6] A. R. Mitchell and D. F. Griffiths, The Finite Difference Method in Partial Differential Equations, Wiley, 1980.

[7] R. D. Richtmyer and K. W. Morton, Difference Methods for Initial Value Problems, 2nd ed., Wiley-Interscience, 1967.

[8] G. D. Smith, Numerical Solution of Partial Differential Equations: Finite Difference Methods, 2nd ed., Clarendon Press, 1978.

[9] G. Strang and G. Fix, An Analysis of the Finite Element Method, Prentice-Hall, 1973.

[10] P. Swarztrauber and R. Sweet, "Efficient FORTRAN Programs for the Solution of Elliptic Partial Differential Equations," NCAR-TN/IA-109, National Center for Atmospheric Research, 1975*.

[11] R. Varga, Matrix Iterative Analysis, Prentice-Hall, 1962.

* Available at the Service Desk in the front lobby area of the SLAC Computer Building (CGB).

10. SPECIAL FUNCTIONS

Name	Function	Source
Trigonometric, hyperbolic, logarithmic, and exponential functions:		IBM FORTRAN
Exponential integral and related functions:		
DPEONE	E1	FUNPACK
DEI, DEXPEI	Ei, exp(-x)*Ei(x)	FUNPACK
SICIEI	Si, Ci, Ei, Shi, Chi	NBS
Gamma function and related functions:		
DGAMMA, DLGAMA	Gamma, log Gamma	IBM FORTRAN
DPSI	Psi	FUNPACK
Error function and related functions:		
DERF, DERFC	erf, erfc	IBM FORTRAN
MERFI, MERFCI	inverse erf,inverse erfc	IMSL
DDAW	Dawson's integral	FUNPACK
Bessel functions and related functions:		
BESI0, BESEI0	I0, exp(-x)*I0	FUNPACK
BESI1, BESEI1	I1, exp(-x)*I1	FUNPACK
BESJ0	J0	FUNPACK
BESJ1	J1	FUNPACK
BESK0, BESEK0	K0, exp(x)*K0	FUNPACK
BESK1, BESEK1	K1, exp(x)*K1	FUNPACK
DMQBFS	zJn'(z)/Jn(z),z=sqrt(x)	Bell Labs
DYNU,DBESY,MMKEL1	Y nu	FUNPACK,IMSL
BESSEL	Jn(z),Yn(z),In(z),Kn(z) (complex argument)	BRL
Elliptic integrals and related functions:		
DELIPK, DELIK1, DELIKM	K	FUNPACK
DELIPE, DELIE1, DELIEM	E	FUNPACK

10.1 INTRODUCTION

Special functions have long played an important role in applied mathematics and mathematical physics. When applicable they give not only an inexpensive numerical solution, but also analytic properties that often provide deeper insight. Use them with good numerical sense, however. If you find yourself summing 10,000 alternating terms in a series of Bessel functions, for example, the chances are good that there's a better way to solve the problem.

The primary special function routines available in the NAPL are listed above, following the organization and notation of the Handbook of Mathematical Functions by Abramowitz and Stegun [1]. This book contains a tremendous amount of information on special functions, and anyone making use of them should be familiar with it.

For guidance in computing functions not listed in the table, see [3,4,5]. The Collected Algorithms of the ACM Index may also be useful. Unfortunately, there are a vast number of special functions and the list above covers only a few of them. Certain additional routines, particularly for computing Bessel functions, may be available from a Numerical Analysis Consultant.

10.2 BESSEL FUNCTIONS AND FUNPACK

For computing Bessel functions, the table lists both a general routine BESSEL and a number of specialized FUNPACK routines. FUNPACK is a carefully constructed package of routines developed at the Argonne National Laboratory. It is designed to give accurate function values as rapidly as possible. Where applicable, therefore, it will be much faster than BESSEL. The disadvantages are that each FUNPACK routine is very narrow in function, and that they are highly tuned to a particular machine (hence not easily portable). A survey of the history and structure of FUNPACK may be found in [2].

10.3 FUNPACK ROUTINES WITH MULTIPLE ENTRIES

In the case of the elliptic integral functions K and E there are three routines for each that differ only in their arguments: DELIK1 and DELIE1 use x, DELIPK and DELIPE use x*x, and DELIKM and DELIEM use 1-x*x. For general arguments one may call DELIK1 and DELIE1, but to achieve maximum accuracy when $|x|$ is near 0 or 1, one should use DELIPK and DELIPE or DELIKM and DELIEM (see [2] or a Numerical Analysis Consultant for details). A similar situation holds with the secondary entries DEXPEI, BESEI0, BESEI1, BESEK0, BESEK1 listed above, which provide superior accuracy for large x .

10.4 REFERENCES FOR CHAPTER 10

[1] M. Abramowitz and I. Stegun, Handbook of Mathematical Functions, Dover, 1965*.

[2] W. J. Cody, "The FUNPACK Package of Special Function Subroutines," ACM Transactions on Mathematical Software 1 (1975), 13-25*.

[3] C. T. Fike, Computer Evaluation of Mathematical Functions, Prentice-Hall, 1968.

---

* Available at the Service Desk in the front lobby area of the SLAC Computer Building (CGB).

[4] W. Gautschi, "Computational Methods in Special Functions," Theory and Application of Special Functions, Academic Press, 1975, 1-98.

[5] J. Hart, et al., Computer Approximations, Wiley, 1968.

[6] NATS. "FUNPACK 2 User's Guide," 1976*.

11. RANDOM NUMBER GENERATION

initialize seed of random sequence	RAN11A
one random real number or integer	RAN11
array of random real numbers	ARAN11
array of random integers	IRAN11
random nos. from non-uniform distributions	IMSL routines

11.1 INTRODUCTION

It is impossible to generate sequences of truly random numbers, but methods have been devised to produce pseudo-random sequences which appear to be random and which exhibit many of the properties of a random sequence. The randomness properties required will depend completely on the application, however, and for this reason one should always be cautious in using a random number generator.

Most generators in common use are of the linear congruential type, constructed in integer arithmetic as follows:

```
          X(1) = initial "seed", an integer

          X(K+1) = (A*X(K)+C) (MOD M)                     (1)

          Kth random number: X(K)/M
```

The randomness properties of the generator, and sometimes its speed, depend critically on the choices of A, C, and M (C is often taken to be 0). Even though formula (1) is very simple, a well chosen set of parameters can produce a generator with very good performance. Also, the simplicity of the generator allows very rapid generation of random numbers, especially if M is chosen to correspond to the machine radix and the programming is done in assembly language. Because of the care taken in choosing values of A, C, and M, the user must resist the urge to tamper with the subroutine in order to further "randomize" it. Seemingly random manipulations can lead accidentally to very non-random results. Furthermore, the numbers in the random sequence should be treated as random reals (or integers), not as random bit patterns to be extracted by masking or shifting.

11.2 RAN11 SERIES

The recommended routines in the RAN11 series are assembler language programs adapted from a modified version (called SUPER-DUPER) of the linear congruential method described above [5]. As indicated in the table, the series contains routines for generating uniform random real numbers in (0,1) and uniform random integers in [1,2**31-1]. It is possible to produce one number at a time, or to fill a vector with a sequence of random numbers for greater efficiency.

For the initial seed, use any odd integer. If the same seed is used repeatedly, then the sequence of random numbers obtained will be duplicated exactly from one run to the next. If it is desired to have a different sequence each time the program is run, one can vary the initial seed according to whim, or use an internal machine clock to choose the seed, or store the final seed somewhere at the end of one run and read it in as input at the beginning of the next.

11.3 NON-UNIFORM DISTRIBUTIONS

In case random numbers from a distribution other than uniform are required, the IMSL library provides generators for normal, exponential, Poisson, and several other types of random deviates. Routines are also provided there to test for randomness in a given distribution of numbers. For further information see Section G of [2]. If you require a portable random number generator or a generator with special properties, speak to a Numerical Analysis Consultant.

11.4 REFERENCES FOR CHAPTER 11

For a very readable overview of random numbers and their use at SLAC, see [1]. The book by Knuth [4] gives a thorough discussion of the theoretical aspects of random number generation. Included are methods for testing random sequences: testing for uniformity, for filling out of n-space, for "runs" in the sequence, etc. The survey by James [3] discusses the common application of pseudorandom numbers to Monte-Carlo methods.

[1] J. Ehrman, "The care and feeding of random numbers," SLAC VM Notebook Module 18 (User Note 81), Stanford Linear Accelerator Center, 1981*.

[2] International Mathematical and Statistical Libraries, IMSL 8 Reference Manual, 1980*.

* Available at the Service Desk in the front lobby area of the SLAC Computer Building (CGB).

[3] F. James, "Monte Carlo Theory and Practice," CERN Report CERN-DD/80/6, 1980*. (Submitted to Reports on Progress in Physics.)

[4] D. Knuth, Seminumerical Algorithms, Addison Wesley, 1969.

[5] G. Marsaglia, et al., "SUPER-DUPER random number package," McGill University School of Computer Science, 1978.

INDEX TO MODULE 19