Contents

Appendix B
Fortran Programs

B.1 Introduction

All subprograms in this appendix are written in Fortran 77, and have been checked to some extent, but the author cannot guarantee their correctness. **Readers are welcome to use these subprograms at their own risk**. It should be noted that, these subprograms are not a substitute for standard mathematical software, but merely concrete (and straightforward) examples of implementing numerical algorithms described in this book. They are only suitable for solving exercises and simple problems encountered in scientific computations. These subprograms can be effectively used to understand the working as well as the limitations of various numerical algorithms, on different problems. Readers are expected to use these subprograms with 'reasonable' inputs. Passing on arbitrary or invalid input parameters may give upredictable results with or without any warning. Apart form the mathematical software libraries a number of software packages are also available which allow the mathematical problems to be specified in a convenient form for numerical solution. These software may also produce incorrect results without any warning. Thus it is advisable to use only those software where the user is aware of which technique is actually implemented, so that their limitations may be known. The programs in the online material are also provided in the same spirit. They may not necessarily give the correct result in all cases, but since the algorithm used is known to the readers, they can modify these to suit their requirements.

The programs in this Appendix are all written in Fortran 77, but may require some changes before running them on a new system. They should also work with Fortran 90 or later versions with some modifications. We leave it to the readers to figure out the changes needed to run the programs on their machines. In particular, there may be some conflict in names as the Fortran library may have another subprogram with same name with same or different arguments. In such cases an explicit EXTERNAL declaration may be needed to force the compiler to use the right routine. We assume that the readers are familiar with Fortran programming. Nevertheless, in this section we give a brief description of common pitfalls in Fortran programming. It is not expected to be an exhaustive summary of Fortran programming, but just a warning signal for those who are not very familiar with the art of programming.

One aspect of Fortran programming which causes some confusion among the beginners is that, the individual program units are usually compiled separately. Hence, there is no way the machine can check inconsistencies in passing arguments to subprograms. Although, some of the modern Fortran compilers do check for such inconsistencies, in general, it is the responsibility of the user to ensure that the dummy arguments and the actual arguments match each other in type. This is also true when the actual argument is a constant, since there will be no type conversion when the subroutine is actually called. For example, consider the following statements

```
CALL ADD(1,2,S)
END
SUBROUTINE ADD(A,B,S)
S=A+B
END
```

Here the variables A and B in the subroutine are of type REAL, while the actual arguments in the calling program are integers. The computer will not convert the arguments when the CALL statement is executed. Instead the same sequence of bits i.e., . . . 001 and . . . 0010 are interpreted as REAL numbers. With the IEEE format they will be interpreted as numbers close to the underflow limit. For example, in the 32-bit IEEE format these numbers are interpreted as 1.4×10^{-45} and 2.8×10^{-45}, respectively.

Even more interesting situation can arise if the actual argument is a constant while the corresponding argument in the subroutine is changed during the execution of subroutine. In such cases, the constant can get changed resulting in unpredictable results. This kind of problems will not be detected by even those compilers which detect the conflict in type of variables while calling a subroutine. For example, consider the following statements:

```
A=3.0
CALL CHANG(2.0,A)
B=2.0*A
PRINT *,A,B
END
SUBROUTINE CHANG(A,B)
A=A+B
END
```

On most machines, this program will print the values 3 and 15 for A and B respectively. This problem arises because the constant 2.0 will be changed to 5.0 when the subroutine CHANG is executed, because the variable A is changed from 2.0 to 5.0 in the subroutine. As a result, the next statement (i.e., `B=2.0*A`) is interpreted as `B=5.0*A`. Of course, some of the modern compilers do not pass a constant directly as an argument to a subroutine, and this problem may not occur on such systems. In general, it is always safe to use variables as actual arguments while calling any subroutine.

There is another very common error while passing the dimension as arguments. Let us assume that there is a subroutine MAT to solve a system of linear equations $A\mathbf{x} = \mathbf{b}$, where A is a $n \times n$ matrix while $\mathbf{b}$ and $\mathbf{x}$ are n-vectors. Since the subroutine is written for any value of n, the actual dimension of the matrix has to be passed as an argument. If we take a simplistic attitude and write the subroutine and call statements as

```
DIMENSION A(5,5),B(5),X(5)
...
N=3
CALL MAT(N,A,B,X)
END
SUBROUTINE MAT(N,A,B,X)
DIMENSION A(N,N),B(N),X(N)
```

Here we expect to use a maximum dimension of 5, and the dimensions of arrays are declared accordingly in the calling program. But in this particular case, the matrix happens to be 3×3 and hence N is set to 3. The computer stores all arrays in its linear memory in certain order. For example, $A(\mathrm{I},\mathrm{J})$ will be the kth element of this array, where $k = \mathrm{I} + (\mathrm{J}-1)5$. Hence, if the program is run with the following matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \tag{B.1}$$

the numbers are stored in the computer's memory in the following order

$$1\ 4\ 7\ -\ -\ 2\ 5\ 8\ -\ -\ 3\ 6\ 9\ -\ -\ -\ -\ -\ -\ -\ -\ -\ -\ -\ - \tag{B.2}$$

where the dashes denote the memory locations which are not used by the current matrix. When the subroutine is called, the information about where the array A starts in the computer memory, as well as the dummy arguments used to define the dimension of A are passed on to the subroutine. Consequently, the array is dimensioned as $A(3,3)$. Hence, the array A is interpreted as 3×3 array and only the first nine elements of A will be considered in the same order as given above. As a result, the matrix is interpreted as

$$A = \begin{pmatrix} 1 & - & 5 \\ 4 & - & 8 \\ 7 & 2 & - \end{pmatrix} \tag{B.3}$$

where three of the elements are undefined. This is completely different from the matrix which was intended to be transferred. It should be noted that, this problem does not arise for one-dimensional arrays like B and X, since the first three memory locations will be correctly interpreted as the corresponding elements. To circumvent this difficulty, most of the standard subroutines include one more argument to pass the actual value of the first dimension for a two-dimensional array. The second dimension need not be passed correctly, since that is not used in computing the order of elements in the array. For example, if the dimension in the subroutine is changed to $A(5, \mathrm{N})$, the correspondence will be correctly established. Similarly, for a three-dimensional array, the first two dimensions will have to be specified correctly, while the third one need not be specified correctly. In fact, Fortran 77 allows the last dimension of an array to be represented by an asterisk. The above example can be rewritten as

```
DIMENSION A(5,5),B(5),X(5)
...
N=3
N1=5
CALL MAT(N,N1,A,B,X)
END
SUBROUTINE MAT(N,N1,A,B,X)
DIMENSION A(N1,*),B(*),X(*)
```

Here N1 is the actual value of the first dimension of the array A. In all subroutines requiring arrays of dimension two or higher, it is preferable to have additional arguments to specify the actual values of the required dimensions. If there is no provision to pass on this value then the dimension in the calling program must exactly match what is expected in the subroutine. The exact matching of dimensions is required only if the calling program also needs to refer to elements in the concerned array. If the array is merely passed on to different subroutines, then the dimensions in calling program are immaterial as long as all subroutines are provided with the same dimension information and the size of array in calling program is larger than the required size in the subroutines.

In many routines, dealing with arrays of n-dimensions, where n may be variable, it is not convenient to define multi-dimensional array. In most of these routines (e.g., BSPINTN or POLFITN) the corresponding array is treated as a one-dimensional array with the required size and the order of element is calculated explicitly. The dimensions of the array are assumed to match the actual size. Thus for a problem in four dimension with array A(4,5,6,7) in the calling program. The subroutine will treat it as a one-dimensional array AS(840), and element $\mathrm{A}(i, j, k, l)$ corresponds to $\mathrm{AS}(i + (j-1)4 + (k-1)20 + (l-1)120)$ in the subroutine. It is generally assumed that there are no gaps in the memory allocation, that is, the dimensions of array A must exactly match the size of array in the given problem. Alternately, even the calling program may also use a one-dimensional array and calculate the required index as mentioned above.

Some of the subroutines (e.g., BSPLIN, MULER2, MSTEP, RANF, SECANI, SPLEVL) assume that the value of variables is preserved after the execution is over, as these values are used during subsequent calls to the subroutine. Most Fortran compilers use static memory allocation and the values are automatically preserved. However, some of the modern compilers allocate the memory dynamically, in which case, the values may not be preserved, unless a SAVE statement is included. A save statement without arguments implies that the values of all permissible variables are preserved. Such a statement is included in some subroutines, but it might have been omitted in some cases.

As far as possible, the subroutines use the normal Fortran naming convention to determine the type of variables. Only for logical, complex or double precision variables the convention has been broken. In most such cases, an implicit statement has been used to define a new convention for determining the type of variable. All subroutines use double precision arithmetic, as on most modern computers there is little difference in execution time between single and double precision arithmetic. If the user is interested in using the single precision arithmetic, then in most cases only the IMPLICIT REAL*8 statement should be commented out or modified to REAL*4. For most subroutines this should work. Nevertheless, single precision (REAL*4) version of all subroutines is provided in a separate directory. In most of these routines the constants are still left in double precision and hence some calculations may still be done in double precision. Similarly, quadruple precision (REAL*16) version of the subroutines is also provided in a separate directory but again some of the constants are in double precision only so the accuracy may not always be higher. In particular, all routines using rational function approximations have the same constants as in double precision version which will only give accuracy of order of 10^{-15}. It is nontrivial to change these routines to get higher order accuracy as in most cases the range of approximation will have to be split to achieve higher accuracy. Further calculation of the required coefficients of rational function approximations will require higher order accuracy. Similarly, the weights and abscissas of some of the Gaussian formulae have not been calculated to higher accuracy.

Most of the subprograms have a parameter IER which is used as an error flag. After execution of the subprograms IER should be zero if no error has occurred. Nonzero values of IER indicate some error condition. Values of IER less than 100, usually (but not necessarily) indicate an error condition, which is not very serious and the result may still be useful. However, further checks may be necessary before accepting the result. In most of the interpolation and integration routines IER is set to values less than 100 even when it fails to converge to satisfactory accuracy. This is mainly because in most cases these routines will still give a reasonable accuracy. But this may not be the case in exceptional situations, due to singularity, or in case of integration in large number of dimensions, simply because the number of points allowed to be used may be completely inadequate. Even a zero value of IER does not necessarily imply that the result is correct as all possible errors are not detected by any of these subroutines. Thus additional checks are required in all cases. But in general one can expect that the result is likely to be correct if IER is zero. A value of IER > 100, usually signifies that the execution had to be terminated because of some serious error, and the result is not likely to be correct. Of course, there will be many cases where even the results in these cases are acceptable, but it will need careful analysis. A few of the subprograms (e.g., BISECT, NEWRAP, FRED, SECANI) return a negative value of IER under some special circumstances, even though the execution is successful. It may be noted that some subprograms call other subprograms. Hence, if IER is nonzero the error may have occurred in the secondary subroutines. In such cases, only the final value of IER may be returned. It may be noted that if an error condition is detected the variable that is expected to return the required result may contain some irrelevant value. Hence it is necessary to check the value of IER after every call to these routines before accepting the result. The value of IER returned by various programs has been modified since the first edition was written, to make them more or less unique between different subroutines. In the present version, when IER > 0, in most cases the first digit is determined by the topic covered by the subroutine. Thus the values 1x or 1xx are for linear algebra routines (Chapters 3 and 11), values 2x or 2xx are for interpolation and differentiation (Chapters 4 and 5), 3x or 3xx for integration (Chapters 6), 4x or 4xx for nonlinear equations (Chapter 7), 5x or 5xx for optimisation (Chapter 8), 6x or 6xx for statistics and approximation (Chapters 9, 10), 7x or 7xx for differential and integral equations (Chapters 12, 13, 14).

Many of the subprograms have a parameter REPS and/or AEPS, which specify the required accuracy. Depending on the subroutine this parameter could control either the relative or the absolute error in the final result. In all cases, some heuristic convergence test is used to check for convergence of the results. Consequently, the actual error may not necessarily be smaller than the required accuracy. On the other hand, in most of the simple cases, where the convergence is very rapid the actual error may be an order of magnitude smaller than the required accuracy.

In the following sections we give the description of each subprogram included in the online material and its usage. A brief description of each variable in the call statement also appears in the Fortran files. Some programs giving actual examples of usage are also included in the online material. Subprograms with an underscore in the names are simple variants of other subprograms with the suffix dropped and are not described separately in this Appendix. For example, GAUELM_C is a variant of GAUELM for complex arithmetic. These subprograms are also included in the list which can be found in the online material.

B.2 Roundoff Error

1. CASSUM Function routine to sum N terms of a series using the cascade sum algorithm. The Ith term is calculated by the FUNCTION TERM(I), which must be supplied by the user. If the number of terms exceeds 2^{N2MAX} ($= 2^{30}$), then true "binary" sum may not be calculated. By a simple change as indicated in the program, it is possible to use this subroutine to sum the terms in a real array TERM. In that case TERM should be a real array containing the terms to be summed. Function CASSUM_A implements this variation.

2. ROUND Function routine to round a real number X to N digits using base B. It is assumed that $\hbar B^N < 1$, since otherwise, the accuracy of computer arithmetic is not sufficient to give N digits. The returned value of the function ROUND will be the rounded value of X. This function can be used to simulate an arithmetic with required number of digits by rounding intermediate results after each arithmetic operation to required number of digits. Since it is fairly expensive to use this function routine, it should be used for small problems only.

B.3 Linear Algebraic Equations

3. GAUELM Subroutine to solve a system of N linear equations using Gaussian elimination with partial pivoting. N is the number of equations as well as the number of unknown variables. NUM is the number of different right-hand side vectors, for which the equations are to be solved. A is a real array of length $LJ \times N$ containing the matrix of coefficients. A(i, j) is the coefficient of x_j in the ith equation. The matrix will be overwritten by the subroutine. Hence, if it is required afterwards, a separate copy should be saved. X is a real array of length $LJ \times NUM$, containing the right-hand sides of the linear systems. The system will be solved for each right-hand side vector given by the columns of X. X(I, J) should contain the Ith component for the Jth right-hand side vector. After execution, the array X will contain the required solutions. Thus the right hand side vectors too will be overwritten by the subroutine and if they are required afterwards, a separate copy should be preserved. DET is an output parameter containing the value of the determinant. INC is an integer array of length N, which will contain the pivoting information after the subroutine is executed. LJ is the actual value of the first dimension of arrays A and X, as declared in the calling program ($LJ \geq N$). IER is the error parameter. $IER = 101$ indicates that $N \leq 0$, or $N > LJ$, in which case, no calculations will be performed. $IER = 121$ denotes that at some stage of elimination process, the pivot turned out to be zero, in which case, the calculations are terminated at that stage. This failure can occur only for matrices which are singular or almost singular. This error can also arise if the first dimension of the matrix is not specified correctly. IFLG is an integer parameter which determines the kind of calculations required. If $IFLG \leq 1$, the elimination is performed and IFLG is set to 2, so that next time the elimination is not performed. For higher values of IFLG it is assumed that elimination has already been performed, and the matrix A is overwritten by the triangular factors, while the array INC contains the information about interchanges. For $IFLG \leq 0$, both the elimination as well as solution for the required right-hand sides is calculated. If $IFLG = 1$, then only the elimination is performed and the value of determinant is calculated. If $IFLG \geq 2$, then only the solution for the required right-hand sides is obtained. It may be noted that the subroutine resets the value of IFLG to 2. Hence, if it is called again for a different matrix, the value of IFLG must be set to 0 or 1, to perform elimination for the new matrix. To calculate the inverse of a matrix, the subroutine can be called with NUM = N and array X set to a unit matrix of order N. In this case, the inverse will be returned in the array X. This subroutine assumes that the matrix is equilibrated and does not attempt any scaling. The determinant is calculated in the simple form, which may give overflow or underflow for some matrices. If the system of equations is ill-conditioned, the results could be unreliable and the subroutine may not give any indication of the problem. To detect ill-conditioning the test for zero pivots may be modified as indicated in the program. However, such simple tests may give misleading results in some cases. SVD should be used to check for ill-conditioning. To solve a system of equations with complex coefficients all real variables except those beginning with R should be treated as complex. This can achieved by an IMPLICIT COMPLEX(A–H, S–Z) statement and is implemented in GAUELM_C.

4. MATINV Subroutine to calculate the inverse of a square matrix using Gaussian elimination, with partial pivoting. N is the order of matrix, IA is the first dimension of arrays A and AI as specified in the calling program. A is a real array of length IA $\times$ N containing the matrix, which must be supplied at the time of calling. AI is a real array of length IA $\times$ N which will contain the inverse of A as calculated by the subroutine. IWK is an integer array of length N used as scratch space. IER is the error parameter, whose value may be set by subroutine GAUELM, which is used to calculate the inverse. Instead of GAUELM it is possible to use CROUT for calculating the triangular decomposition, but in that case an extra real scratch array of length N, will be required.

5. CROUT Subroutine to solve a system of N linear equations using Crout's algorithm for LU decomposition, with partial pivoting. N is the number of equations as well as the number of unknown variables. NUM is the number of different right-hand side vectors, for which the equations are to be solved. A is a real array of length LJ $\times$ N, containing the matrix of coefficients. A(i, j) is the coefficient of x_j in ith equation. This matrix will be overwritten by the subroutine. Hence, if it is required afterwards, a separate copy should be saved. After execution, A will contain the triangular decomposition of the original matrix. X is a real array of length LJ $\times$ NUM, containing the right-hand sides of the linear systems. The system will be solved for each right-hand side vector given by the columns of X. X(I, J) should contain the Ith component for the Jth right-hand side vector. After execution, the array X will contain the required solutions. Thus if the right-hand side vectors are required afterwards, a separate copy must be preserved. DET and IDET are the output parameters containing the value of the determinant in a scaled form. The actual value of the determinant is DET $\times$ 2^{IDET}. INC is an integer array of length N, which will contain the pivoting information after the subroutine is executed. LJ is the actual value of the first dimension of arrays A and X, as declared in the calling program (LJ $\geq$ N). IER is the error parameter. IER = 102 indicates that N $\leq$ 0, or N > LJ, in which case, no calculations will be performed. IER = 122 denotes that at some stage of *LU* decomposition the pivot turned out to be zero, in which case, the calculations are terminated at that stage. This failure can occur only for matrices which are singular or almost singular. IFLG is an integer parameter which determines the kind of calculations required. If IFLG $\leq$ 1, the *LU* decomposition is performed and IFLG is set to 2, so that next time the triangular decomposition is not performed. For higher values of IFLG, it is assumed that *LU* decomposition has already been performed, and the matrix A is overwritten by the triangular factors, while the array INC contains the information about interchanges. For IFLG $\leq$ 0, both the *LU* decomposition as well as the solution for the required right-hand sides are obtained. If IFLG = 1, then only *LU* decomposition is performed and the value of determinant is calculated. If IFLG $\geq$ 2, then only the solution for the required right-hand sides are obtained. It may be noted that, the subroutine resets the value of IFLG to 2. Hence, if it is called again for a different matrix, the value of IFLG must be set to 0 or 1, to perform *LU* decomposition for the new matrix. WK is a real array of length N used as a scratch space by the subroutine to store intermediate results. To calculate the inverse of a matrix, the subroutine can be called with NUM = N and array X set to a unit matrix of order N. In this case, the inverse will be returned in the array X. This subroutine implements implicit scaling of rows as explained in Section 3.3. If the system of equations is ill-conditioned, the results could be unreliable and the subroutine may not give any indication of the problem. To detect ill-conditioning the test for zero pivots may be modified as indicated in the program. However, such simple tests may give misleading results in some cases. SVD should be used to check for ill-conditioning. To solve a system of equations with complex coefficients all real variables except those beginning with R should be treated as complex. This can achieved by an IMPLICIT COMPLEX(A–H, S–Z) statement and is implemented in CROUT_C.

6. CROUTH Subroutine to solve a system of N linear equations using the technique of iterative refinement and the Crout's algorithm for *LU* decomposition with partial pivoting. N is the number of equations as well as the number of unknown variables. NUM is the number of different right-hand side vectors, for which the equations are to be solved. A is a real array of length LJ $\times$ N, containing the matrix of coefficients. A(I, J) is the coefficient of x_j in Ith equation. The matrix will be preserved by the subroutine. B is a real array of the same dimensions as A. After execution, B will contain the triangular decomposition of the original matrix. X is a real array of length LJ $\times$ NUM, containing the right-hand sides of the linear systems. The system will be solved for each right-hand side vector given by the columns of X. X(I, J) should contain the Ith component for the Jth right-hand side vector. After execution, the array X will contain the required solutions. DET and IDET are the output parameters containing the value of the determinant in a scaled form. The actual value of the determinant will be DET $\times$ 2^{IDET}. INC is an integer array of length N, which will contain the pivoting information after the subroutine is executed. LJ is the actual value of the first dimension of arrays A, B and X, as declared in the calling program (LJ $\geq$ N). REPS is the required relative accuracy, to which the iterative refinement is required. The iteration is terminated when the maximum change in any component of the solution is less than REPS times the maximum component of the solution vector. REPS should be greater than $\hbar$ for the arithmetic used. It should be noted that convergence of iteration does not guarantee that the solution is accurate to the specified level. A reasonable estimate of error is provided by WK(J) for the Jth right-hand side. However, for extremely ill-conditioned matrices, it may underestimate the error. IER is the error parameter. IER = 11 implies that the iterative refinement did not converge to the specified accuracy. This failure can occur if the system of equations is extremely ill-conditioned, or if the specified value of REPS is too low, or if the residues are accumulated using the same level of precision as all other calculations. To avoid this the variable D1 should generally be of higher precision than other variables. If the compiler supports REAL*16 arithmetic, D1 should be declared so. For single precision version of CROUTH, the variable D1 should be in double precision. If the matrix is ill-conditioned, increasing NITR may cause the iteration to converge, but the result is unlikely to be more reliable. IER = 102 indicates that N $\leq$ 0 or N $>$ LJ, in which case, no calculations will be performed. IER = 122 denotes that at some stage of *LU* decomposition, the pivot turned out to be zero, in which case, the calculations are terminated at that stage. This failure can occur only for matrices which are singular or almost singular. IFLG is an integer parameter which determines the kind of calculations required. If IFLG $\leq$ 1, the *LU* decomposition is performed and IFLG is set to 2, so that next time the *LU* decomposition is not performed. For higher values of IFLG, it is assumed that *LU* decomposition has already been performed and the array B contains the triangular factors, while the array INC contains the information about interchanges. For IFLG $\leq$ 0, both the *LU* decomposition as well as solution for the required right-hand sides are obtained. If IFLG = 1, then only the *LU* decomposition is performed and the value of determinant is calculated. If IFLG $\geq$ 2, then only the solution for the required right-hand sides are obtained. It may be noted that the subroutine resets the value of IFLG to 2. Hence, if it is called again for a different matrix, the value of IFLG must be set to 0 or 1, to perform *LU* decomposition for the new matrix. WK is a real array of length 2N + NUM used as a scratch space by the subroutine to store intermediate results. After execution, WK(I) will contain the estimated error for the Ith right-hand side. This subroutine requires subroutine CROUT to perform the *LU* decomposition and to calculate solution of resulting equations for a given right-hand side vector. To calculate the inverse of a matrix, the subroutine can be called with NUM = N and array X set to a unit matrix of order N. In this case, the inverse will be returned in the array X. If the system of equations is extremely ill-conditioned, the results could be unreliable and the subroutine may not give any indication of problem.

However, in most cases, WK(I) will give a reasonable estimate of the actual error. This subroutine should give correctly rounded solution to the system as represented in the computer, provided the system is not too ill-conditioned for the precision of arithmetic used in the calculations. It may be noted that for iterative refinement to converge it may be necessary to calculate the residuals using higher precision arithmetic as compared to that used for solution. Otherwise, the solution may not converge. For this purpose the variable D1 should have higher precision as compared to other variables.

7. CHOLSK Subroutine to solve a system of N linear equations with a real symmetric positive definite matrix using Cholesky decomposition. N is the number of equations as well as the number of unknown variables. NUM is the number of different right-hand side vectors, for which the equations are to be solved. A is a real array of length $\mathrm{ND} \times \mathrm{N}$, containing the matrix of coefficients. $\mathrm{A(I, J)}$ is the coefficient of x_J in Ith equation. Only the lower triangular part of the matrix may be used by the program, though the memory is reserved for the full array. This matrix will be overwritten by the subroutine. Hence, if it is required afterwards, a separate copy should be saved. After execution, A will contain the Cholesky decomposition of the original matrix in its lower triangular part. X is a real array of length $\mathrm{ND} \times \mathrm{NUM}$, containing the right-hand sides of the linear systems. The system will be solved for each right-hand side vector given by the columns of X. $\mathrm{X(I, J)}$ should contain the Ith component for the Jth right-hand side vector. After execution, the array X will contain the required solutions. DET is the output parameter containing the value of the determinant. ND is the actual value of the first dimension of arrays A and X, as declared in the calling program ($\mathrm{ND} \geq \mathrm{N}$). IER is the error parameter. $\mathrm{IER} = 103$ indicates that $\mathrm{N} \leq 0$, or $\mathrm{N} > \mathrm{ND}$, in which case, no calculations will be performed. $\mathrm{IER} = 123$ denotes that at some stage of Cholesky decomposition the pivot turned out to be zero, in which case, the calculations are terminated at that stage. This failure can occur if the triangular decomposition does not exist without pivoting or if the matrix is not positive definite or if it is almost singular. IFLG is an integer parameter which determines the kind of calculations required. If $\mathrm{IFLG} \leq 1$, the triangular decomposition is performed and IFLG is set to 2, so that next time the triangular decomposition is not performed. For higher values of IFLG, it is assumed that the Cholesky decomposition has already been performed, and the matrix A is overwritten by the triangular factor. For $\mathrm{IFLG} \leq 0$, both the decomposition as well as the solution for the required right-hand sides are obtained. If $\mathrm{IFLG} = 1$, then only decomposition is performed and the value of determinant is calculated. If $\mathrm{IFLG} \geq 2$, then only the solution for the required right-hand sides are obtained. It may be noted that, the subroutine resets the value of IFLG to 2. Hence, if it is called again for a different matrix, the value of IFLG must be set to 0 or 1, to perform Cholesky decomposition for the new matrix. To calculate the inverse of a matrix, the subroutine can be called with $\mathrm{NUM} = \mathrm{N}$ and array X set to a unit matrix of order N. In this case, the inverse will be returned in the array X. If the system of equations is ill-conditioned, the results could be unreliable and the subroutine may not give any indication of the problem.

8. GAUBND Subroutine to solve a system of N linear equations using Gaussian elimination with partial pivoting for a band matrix. The matrix is stored in the band form as explained below. N is the number of equations as well as the number of unknown variables. KB is the bandwidth of the matrix, that is, $a_{ij} = 0$, if $|i - j| > \text{KB}$. NUM is the number of different right-hand side vectors, for which the equations are to be solved. A is a real array of length $\text{LJ} \times (3\text{KB} + 1)$ containing the matrix of coefficients which is stored in the band form. $\text{A}(\text{I}, \text{J} - \text{I} + \text{KB} + 1)$ is the coefficient of x_J in the Ith equation. It may be noted that if KB is comparable to N, then the band form will require larger storage than the simple form of storing the matrix and no purpose will be served by using this subroutine instead of GAUELM. The matrix A will be overwritten by the subroutine. Hence, if it is required afterwards, a separate copy should be saved. X is a real array of length $\text{LJ} \times \text{NUM}$, containing the right-hand sides of the linear systems. The system will be solved for each right-hand side vector given by the columns of X. $\text{X}(\text{I}, \text{J})$ should contain the Ith component for the Jth right-hand side vector. After execution, the array X will contain the required solutions. DET and IDET are the output parameters containing the value of the determinant in a scaled form. The actual value of the determinant is $\text{DET} \times 2^{\text{IDET}}$. INC is an integer array of length N, which will contain the pivoting information after the subroutine is executed. LJ is the actual value of the first dimension of arrays A and X, as declared in the calling program ($\text{LJ} \geq \text{N}$). IER is the error parameter. $\text{IER} = 104$ indicates that $\text{N} \leq 0$, or $\text{N} > \text{LJ}$ or $\text{KB} > \text{N}$, in which case, no calculations will be performed. $\text{IER} = 124$ implies that at some stage of elimination process, the pivot turned out to be zero, in which case, the calculations are terminated at that stage. This failure can occur only for matrices which are singular or almost singular. This failure can also occur if the matrix is not specified correctly as explained above. IFLG is an integer parameter which determines the kind of calculations required. If $\text{IFLG} \leq 1$, the elimination is performed and IFLG is set to 2, so that next time the elimination is not performed. For higher values of IFLG it is assumed that elimination has already been performed, and the matrix A is overwritten by the triangular factors, while the array INC contains the information about interchanges. For $\text{IFLG} \leq 0$, both the elimination as well as solution for the required right-hand sides is calculated. If $\text{IFLG} = 1$, then only the elimination is performed and the value of determinant is calculated. If $\text{IFLG} \geq 2$, then only the solution for the required right-hand sides is obtained. For $\text{IFLG} = -1$, both elimination and solution are calculated without pivoting and IFLG is set to 2. This option can be used if the matrix is known to be diagonally dominant and pivoting is not necessary. It may be noted that the subroutine resets the value of IFLG to 2. Hence, if it is called again for a different matrix, the value of IFLG must be set to 0 or 1 or -1, to perform elimination for the new matrix. WK is a real array of length $3\text{KB} + 1$ used as scratch space. To calculate the inverse of a matrix, the subroutine can be called with NUM = N and array X set to a unit matrix of order N. In this case, the inverse will be returned in the array X. The inverse of a band matrix will in general not be banded and full storage will be required to store the inverse. This subroutine assumes that the matrix is equilibrated and does not attempt any scaling. If the system of equations is ill-conditioned, the results could be unreliable and the subroutine may not give any indication of the problem. To detect ill-conditioning the test for zero pivots may be modified as indicated below. However, such simple tests may give misleading results in some cases. To solve a system of equations with complex coefficients all real variables except those beginning with R should be treated as complex. This can achieved by an IMPLICIT COMPLEX(A–H, S–Z) statement. GAUBND_C provides the implementation for complex matrices.

9. SVD Subroutine to perform the singular value decomposition of a M $\times$ N matrix, $A = U\Sigma V^T$ (M $\geq$ N). This subroutine is based on the procedure *svd* in Wilkinson and Reinsch (1971). A is a real array of length LA $\times$ N containing the matrix. After execution, the matrix U of SVD will be overwritten on A. V is a real array of length LV $\times$ N, which will contain the matrix V (not V^T) of SVD. SIGMA is a real array of length N, which will contain the singular values of A, i.e., the diagonal elements of the diagonal matrix Σ. The singular values may not be arranged in a descending order. LA is the integer variable specifying the actual value of the first dimension of array A in the calling program (LA $\geq$ M). Similarly, LV is the actual value of the first dimension of array V in the calling program (LV $\geq$ N). E is a real array of length N, which is used as a scratch space to store intermediate results. IER is the error parameter. IER = 12 implies that the QR iteration used for finding SVD of the bidiagonal form did not converge to the required accuracy. In this case, the last value is accepted and calculations are continued. This situation will generally arise only if parameter REPS is too low for the arithmetic used. If a reasonable value of REPS is specified, then it normally requires only two or three iterations for each singular value, as compared to the maximum number of iterations ITMAX = 30. IER = 105 denotes that N $\leq$ 0, or N $>$ LV, or M $\leq$ 0, or M $>$ LA, or N $>$ M, in which case, no calculations are performed.

10. SVDEVL Subroutine to evaluate the solution of a system of linear equations using the SVD ($\mathbf{x} = V\Sigma^{-1}U^T\mathbf{b}$). This subroutine assumes that SVD is already performed. N is the number of variables in the linear system, while M is the number of equations. U is a real array of size LU $\times$ N containing the left-hand transformation matrix in SVD. V is a real array of length LV $\times$ N containing the right-hand transformation matrix in SVD. SIGMA is a real array of length N, containing the singular values. The singular values need not be arranged in any definite order. LU and LV are the actual values of the first dimension of arrays U and V respectively, in the calling program. B is a real array of length M containing the right-hand side vector. After execution, B will contain the required solution. WK is a real array of length N used as a scratch space by the subroutine. REPS specifies the required accuracy for zeroing the singular values. If $\sigma_i < \text{REPS} \times \max(\sigma_j)$, then the corresponding σ_i^{-1} is set to zero while calculating the solution. For a square matrix, where none of the singular values are reduced to zero, this subroutine gives the unique solution. If some of the singular values are reduced to zero, it gives solution with minimum norm to which any arbitrary combination of columns of V corresponding to zero σ_i can be added, to get the general solution. If any of the singular values are reduced to zero the routine will give the least squares solution unless the right hand side is in the range of the matrix. The right hand side is in the range if it is orthogonal to all columns of U corresponding to singular values which have been reduced to zero. For over-determined systems (M > N), this subroutine gives the least squares solution (with minimum norm if some of the singular values are reduced to zero). Before using this subroutine, the singular value decomposition should be computed using the subroutine SVD. It should be noted that, the subroutine SVD overwrites the matrix U on the original matrix A.

B.4 Interpolation

11. DIVDIF Subroutine for interpolation using the Newton's divided difference formula. XB is the x value at which interpolated value is required. NTAB is the number of entries in the table. Arrays X and F of length NTAB contain the abscissas and the function values at the corresponding points. The array X must contain abscissas in either ascending or descending order. NUSE specifies the maximum number of points to be used for interpolation. If NUSE is larger than NTAB, or the parameter NMAX in the subroutine, then it will be reduced to the minimum of these three numbers and the error flag IER will be set to 22. If NUSE is less than 1, it will be set to MIN(6, NTAB) and IER will be set to 21. In all cases, after execution, NUSE will contain the number of points actually used. Thus for subsequent calls NUSE must be reset to the required value before call. FB is the output array of length NUSE which will contain the interpolated values. FB(I) gives the interpolated value using I points, and FB(NUSE) gives the final value. AEPS specifies the required accuracy. Interpolation is continued, until two successive values differ by less than AEPS. If AEPS ≤ 0, then this convergence criterion will never be satisfied. IER is the error parameter. IER = 21 implies that NUSE < 1, in which case it is set to MIN(6, NTAB). IER = 22 implies that NUSE > NTAB, or NUSE > NMAX, in which case it is reduced appropriately. IER = 23 implies that the interpolation has not converged to the specified accuracy. DFB and DDFB are output parameters containing the first and second derivative at the same point XB. It should be noted that the convergence criterion only checks for convergence of interpolated value and not the derivatives. Hence, the error in computed derivatives may be much larger. In particular, if the derivative is required at one of the tabular points, then AEPS should be set to zero, since otherwise the interpolated value will converge immediately and execution will be terminated with a very crude estimate for the derivatives. If the derivatives are not required, then subroutine DIVDIF0 may be used. This subroutine requires the function NEARST.

12. DIVDIF0 Subroutine for interpolation using the Newton's divided difference formula. This is a simplified version of DIVDIF which avoids derivative calculation and has a flag to decide whether the first point for interpolation should be chosen from the nearest value in table or another point specified by the user. This flag may be useful if the routine is used to calculate interpolation in higher dimensions. XB is the x value at which interpolated value is required. NTAB is the number of entries in the table. Arrays X and F of length NTAB contain the abscissas and the function values at the corresponding points. The array X must contain abscissas in either ascending or descending order. NUSE specifies the maximum number of points to be used for interpolation. If NUSE is larger than NTAB, or the parameter NMAX in the subroutine, then it will be reduced to the minimum of these three numbers and the error flag IER will be set to 22. If NUSE is less than 1, it will be set to MIN(6, NTAB) and IER will be set to 21. In all cases, after execution, NUSE will contain the number of points actually used. Thus for subsequent calls NUSE must be reset to the required value before call. FB is the output array of length NUSE which will contain the interpolated values. FB(I) gives the interpolated value using I points, and FB(NUSE) gives the final value. AEPS specifies the required accuracy. Interpolation is continued, until two successive values differ by less than AEPS. If AEPS ≤ 0, then this convergence criterion will never be satisfied and NUSE points will be used for interpolation. IER is the error parameter. IER $= 21$ implies that NUSE < 1, in which case it is set to MIN(6, NTAB). IER $= 22$ implies that NUSE $>$ NTAB, or NUSE $>$ NMAX, in which case it is reduced appropriately. IER $= 23$ implies that the interpolation has not converged to the specified accuracy. IFLG is an integer variable used as a flag to decide the first point to be used for interpolation. If IFLG $= 0$ then the nearest point in the table is used to start interpolation. For other values of IFLG a user supplied value (IF1) is used, provided that it is admissible ($1 \leq$ IF1 $\leq$ NTAB). IF1 specifies the first point to be used for interpolation when IFLG $\neq 0$. For IFLG $= 0$ the subroutine uses the nearest point in that table and the value of IF1 is set to index of this point, so that next time we can use the same point if the interpolation in required at same XB with different F values. This situation arises if this routine is used for interpolation in more than one variables. This subroutine requires the function NEARST.

13. NEARST Function routine to locate the nearest value in an ordered table using a bisection algorithm. X is a real array containing the table with NTAB entries in either ascending or descending order. After execution, X(NEARST) will be the point nearest to XB.

14. SPLINE Subroutine to calculate the coefficients of cubic spline interpolation with not-a-knot boundary conditions using N tabular points. Arrays X and F of length N should contain the input data. F(I) should contain the function value at X(I). The array X must be in ascending or descending order. After execution, array C of length $3 \times$ N, will contain the coefficients of cubic spline. If XB is between X(I) and X(I + 1), $(1 \leq \text{I} < \text{N})$ the interpolant is given by `F(I)+DX*(C(1,I)+DX*(C(2,I)+DX*C(3,I)))`, where DX = XB − X(I). IER is the error parameter. IER = 201 implies that the number of points is less than 2. If the number of points is 2, linear interpolation will be used, while for N = 3 quadratic interpolation will be used. For higher values of N, cubic spline interpolation with not-a-knot boundary conditions is calculated.

15. SPLEVL Function routine for evaluating the cubic spline interpolant at XB, using the coefficients of cubic spline which have been calculated by the subroutine SPLINE. This subroutine first locates the subinterval containing the required point XB, using the technique described in Section 4.2. N is the number of data points. X and F are arrays of length N containing the data points, F(I) is the tabulated function value at X(I). C is a real array of length $3 \times$ N containing the coefficients of cubic spline. The array X must be in ascending or descending order. DFB and DDFB are output parameters containing the first and second derivatives of the tabulated function at $x =$ XB. IER is the error parameter. IER = 24 implies that XB is outside the range of table on the higher side, in which case, the cubic for last subinterval will be used. IER = 25 implies that XB is outside the range on the lower side, in which case, the cubic for the first subinterval will be used. It may be noted that spline interpolation is extremely unreliable outside the range of table and resulting value may have large errors. For extrapolation, it may be better to use polynomial interpolation with subroutine DIVDIF, though it also has limitations. IER = 201 implies that N < 2. After execution SPLEVL will contain the interpolated value of the function at XB. Before using this function, the coefficients of cubic spline must be calculated using subroutine SPLINE or any other equivalent subroutine.

16. SMOOTH Subroutine to draw a smooth curve passing through a set of data points using cubic spline interpolation. X and F are arrays of length NTAB containing the given data points, C is a real array of dimension (3, NTAB), which will contain the coefficients of cubic spline. The array X must be in either ascending or descending order. This subroutine will calculate the interpolated value of the function at NP uniformly spaced points in the interval spanned by the tabular points. XP and FP should be arrays of length NP which will contain the X and F values for the uniformly spaced output table. If NP is sufficiently large, the output arrays XP and FP can be used to plot a smooth curve through the given points. It may be noted that this subroutine does not actually plot the function but only generates data that can be used to get a smooth plot through the data points. If additional smoothing is required and the curve is not required to pass through data points then a least squares approximation should be used to generate the points using subroutine BSPFIT instead of SPLINE. IER is the error parameter. IER = 202 implies that NP < 2. Other values of IER may be set by subroutine SPLINE which is called to calculate the spline coefficients. This subroutine requires subroutine SPLINE and function SPLEVL.

17. BSPLIN Subroutine to calculate the B-spline basis functions at a specified point. X is a real array of size NX containing the knots. The knots must be in ascending order and distinct. NX is the number of knots. K is the order of B-spline, $K = 4$ implies cubic B-splines, while $K = 2$ gives linear B-splines, etc. XB is the point at which B-splines need to be evaluated. NDERIV specifies the number of derivatives to be calculated, for $\mathrm{NDERIV} \leq 0$ only the B-splines will be calculated. For $\mathrm{NDERIV} = 1$, the first derivative of B-splines will also be calculated. For $\mathrm{NDERIV} > 1$, the second derivative will also be calculated. Higher derivatives are not calculated, but the program can be modified to include calculation of higher derivatives. B, DB and DDB are real arrays of length $\mathrm{NX} + \mathrm{K} - 2$ which will contain the values of B-splines, its first and second derivative respectively. All $\mathrm{NX} + \mathrm{K} - 2$ basis functions and if required the derivatives are calculated simultaneously using the recurrence relations. Even if derivatives are not required the arrays with required size must be supplied. This subroutine first locates the subinterval containing the required point XB, using the technique described in Section 4.2. LEFT is an output variable which will give the location of XB in the table of knots, i.e., $\mathrm{X(LEFT)} \leq \mathrm{XB} \leq \mathrm{X(LEFT + 1)}$. IER is the error parameter, $\mathrm{IER} = 26$ implies that the required point is outside the table on higher side. $\mathrm{IER} = 27$ implies that the required point, XB is outside the range of table on lower side. In both these cases the basis functions will be calculated using the same recurrence relations, but the results may not be useful. $\mathrm{IER} = 203$ implies that $\mathrm{NX} < 2$, $\mathrm{K} < 1$ or $\mathrm{K} > \mathrm{KMAX} = 20$, in which case no calculations are done. The parameter KMAX in the subroutine can be trivially increased, but such high order splines may not be meaningful. WK is a real array of length $\mathrm{N} + 2\mathrm{K} + 1$ used as scratch space.

18. BSPINT Subroutine to calculate the coefficients for B-spline interpolation to a table of values. N is the number of points in the table, X and F are real arrays of length N, specifying the abscissas and function values. The abscissas X must be unique and in ascending order. F(I) is the tabulated function value at X(I). K is the order of B-spline interpolation required. $\mathrm{K} = 4$ implies cubic B-splines, while $\mathrm{K} = 2$ gives linear B-splines, etc. A is a real array of length $\mathrm{LA} \times 3\mathrm{K}$, which will contain the triangular decomposition of the matrix of equations that is solved to calculate the coefficients of expansion. Since the matrix is in band form with bandwidth $\mathrm{K} - 1$, it is stored in band form. This matrix will be required if another interpolation with same set of knots is required with different F. This is useful in B-spline interpolation in higher dimensions. LA is the first dimension of A as specified in the calling routine ($\mathrm{LA} \geq \mathrm{N}$). C is a real array of length N, which will contain the coefficients of B-spline basis functions in expansion for interpolation. XF is a real array of size NO, which will contain the knots used for B-spline definition. Since the number of B-spline basis functions with NO knots is $\mathrm{NO} + \mathrm{K} - 2$, some tabular points may have to be dropped to ensure that $\mathrm{N} = \mathrm{NO} + \mathrm{K} - 2$. This is equivalent to using the not-a-knot boundary condition in cubic spline. The number of points to be dropped depends on K. In all cases some points near both ends are dropped. For linear B-splines, $\mathrm{K} = 2$ all points are used. For cubic B-splines, $\mathrm{K} = 4$, the 2nd and $\mathrm{N}-1$ th points are dropped. NO is the number of knots used for B-spline interpolation, this would be equal to $\mathrm{N}+2-\mathrm{K}$. IFLG is an integer variable that specifies what calculation is to be done. For IFLG ≤ 1 the matrix is calculated and its triangular decomposition is computed. If execution is successful, IFLG will be set to 2, so that next time the matrix calculations will be skipped. If IFLG ≤ 0 the coefficients of expansion are also calculated. If IFLG $= -1$, the system of equations will be solved without pivoting. This option may not be used as the coefficient matrix may not be diagonally dominant and some pivot may turn out to be zero. If IFLG $= 2$, only the coefficients of expansion will be calculated using the old triangular decomposition available in A and the (hopefully new) function values F. INC is an integer array of length N, which will contain the pivoting information for solution of system of linear equations. This array will be required if another interpolation with same knots is needed. WK is a real array of length $3\mathrm{N}+\mathrm{K}+7$ used as scratch space. IER is the error parameter. IER $= 204$ implies $\mathrm{N} < \mathrm{K}$ or $\mathrm{K} < 2$, in which case no calculations are done. Other values may be set by subroutines BSPLIN or GAUBND, which are called. This subroutine requires subroutine BSPLIN to calculate the B-spline basis functions and subroutine GAUBND to solve the system of linear equations with a band matrix. The interpolant at any required point can be calculated using FUNCTION BSPEVL, using the calculated coefficients of expansion.

19. BSPEVL Function routine to calculate the value of function using available coefficients of B-spline expansion. The calculated value is given by:

$$\mathrm{BSPEVL} = \sum_{i=1}^{\mathrm{N+K-2}} \mathrm{WT}(i)\phi_i(\mathrm{X0}), \tag{B.4}$$

where ϕ_i are the B-spline basis functions. It can be used to calculate the interpolated value of the function, if the coefficients are already calculated using BSPINT. N is the number of knots, which may not be the same as the number of points in the table for interpolation. X is a real array of length N containing the knots for B-splines. X(1) should contain the first knot and the knots must be distinct and in ascending order. K is the order of B-splines, $\mathrm{K} = 2$ gives linear B-splines, while $\mathrm{K} = 4$ yields cubic B-splines, etc. NDERIV specifies the number of derivatives to be calculated. For NDERIV ≤ 0 only the function value will be calculated. For NDERIV $= 1$, the first derivative is also calculated. For NDERIV > 1, the second derivative will also be calculated. WT is the real array of length N, containing the coefficients of expansion in terms of B-spline basis functions. These coefficients must be calculated beforehand using BSPINT or any other equivalent routine for B-spline approximations. X0 is the point where the function value needs to be calculated. DF is the first derivative of function at X0, while DDF is the second derivative. The derivatives would be calculated only if NDERIV has been set appropriately. WK is a real array of length at least $4\mathrm{N}+5\mathrm{K}+2$ used as scratch space. IER is the error parameter, which should be zero after successful execution. Nonzero values of IER may be set by subroutine BSPLIN which is called to calculate the B-spline basis functions. This routine requires subroutine BSPLIN.

20. RATNAL Subroutine for rational function interpolation. XB is the x value at which interpolated value is required. NTAB is the number of entries in the given table. Arrays X and F of length NTAB contain the abscissas and the function values at the corresponding points. The array X must contain abscissas in either ascending or descending order. NUSE specifies the maximum number of points to be used for interpolation. If NUSE is larger than NTAB or the parameter NMAX in the subroutine, then it will be reduced to the minimum of these three numbers and the error flag IER may be set to 22. If NUSE is less than 1, it will be set to MIN(6, NTAB) and IER will be set to 21. In all cases, after execution, NUSE will contain the number of points actually used. Thus NUSE must be reset to required value before subsequent calls to the routine. FB is the output parameter containing the interpolated value. AEPS is a real parameter specifying the required accuracy. Interpolation will be continued until two successive values differ by less than AEPS. If AEPS ≤ 0, then this convergence criterion will never be satisfied. IER is the error parameter. IER $= 21$ implies that NUSE < 1, in which case it is set to MIN(6, NTAB). IER $= 22$ implies that NUSE $>$ NTAB, or NUSE $>$ NMAX, in which case it is reduced appropriately. IER $= 23$ implies that, the interpolation has not converged to the specified accuracy. IER $= 205$ implies that, the execution was terminated because the denominator at some stage was zero. In this case, either the interpolant has a pole at the requested point or it has a 0/0 form, and the subroutine will return with the interpolated value obtained in the previous step. This value may or may not be acceptable. The reliability of this value may be verified by running this subroutine with successively increasing value of NUSE, until it encounters the singularity. If these values are converging, then the result may be acceptable. This subroutine requires function NEARST.

21. POLY2 Subroutine for polynomial interpolation in two dimensions. (XB1, XB2) is the point at which interpolated value is required. X1 and X2 are real arrays of length N1 and N2 respectively, containing the abscissas. F is an array of length NDIM $\times$ N2 containing the function values F(I, J) $= f(\mathrm{X1(I)}, \mathrm{X2(J)})$. NDIM should be set to the actual value of the first dimension of the array F in the calling program (NDIM $\geq$ N1). NP1 and NP2 are the number of points to be used along the two axes for interpolation. The subroutine may adjust these values if required. FB is the output parameter containing the interpolated value. IER is the error parameter. IER $= 206$ implies that N1 $>$ NDIM, in which case no calculations are done. This subroutine calls DIVDIF0 for one-dimensional interpolation along X1. To improve efficiency, the derivative calculation has been removed from DIVDIF0 and a flag has been introduced to avoid locating the nearest point every time. This subroutine will require subroutine DIVDIF0 and function NEARST.

22. LINRN Subroutine to perform linear interpolation in N variables. XB is a real array of length N containing the coordinates of the point at which interpolated value is required. X is a real array of length NXD × N containing the abscissas. F is an N-dimensional array with dimension F(NDIM(1), ..., NDIM(N)) containing the function values

$$\mathrm{F}(\mathrm{I1},\mathrm{I2},\ldots,\mathrm{IN}) = f(\mathrm{X}(\mathrm{I1},1),\mathrm{X}(\mathrm{I2},2),\ldots,\mathrm{X}(\mathrm{IN},\mathrm{N})). \tag{B.5}$$

NP is an integer array of length N, and NP(I) is the number of tabular points along Ith coordinate. FB is the output parameter which will contain the interpolated value. NDIM is an integer array of length N, specifying the dimension of F as explained above. This array must specify the dimensions of F as used in the calling program. NXD is the actual value of the first dimension of X, and it should be greater than the maximum element in NP. IN is an integer array of length 2N, which is used as a scratch space to store intermediate quantities. Similarly, HN is a real array of length N, also used as a scratch space. As an example, consider the case, when we want to perform interpolation in four variables, with 4, 5, 6 and 7 points along the four coordinates, then the relevant statements could be as follows

```
DIMENSION F(5,5,6,8),X(10,5),XB(5),NP(5),NDIM(5),IWK(10),WK(5)
DATA N,NXD,(NP(I),I=1,4),(NDIM(I),I=1,4)/4,10,4,5,6,7,5,5,6,8/
. . . . . . . . . . . . . . . . .
CALL LINRN(N,XB,X,F,NP,FB,NDIM,NXD,IWK,WK,IER)
```

IER is the error parameter. IER = 207 signifies that NP(I) is less than two or greater than NDIM(I) or NXD for some I, in which case no calculations are performed. This subroutine requires function LOCATE.

23. LOCATE Function routine to locate a given point XB between two points of an ordered table using the bisection algorithm. X is the table with NP entries in either ascending or descending order. After execution, LOCATE returns a value such that, XB should be between X(LOCATE) and X(LOCATE+1), unless it is outside the range of the table. If XB is before the first point, then LOCATE = 1, while if XB is beyond the last point, then LOCATE = NP − 1.

24. BSPINT2 Subroutine to calculate the coefficients of interpolating B-spline expansion in two dimensions. NX, NY are the number of points along x and y, respectively in the table. X, Y are real arrays of length NX and NY containing the abscissas. The abscissas must be distinct and in ascending order. F is a real array of size LA × NY containing the function values. F(I, J) should contain $f(\mathrm{X}(\mathrm{I}),\mathrm{Y}(\mathrm{J}))$. K is an integer variable specifying the required order of B-splines, K = 4 for cubic B-splines, K = 2 for linear B-splines, etc. For simplicity, it is assumed that order is the same for expansions along x and y. AX is a real array of length LA × 3K which will contain the triangular decomposition of matrix for interpolation along X. AY is a real array of length LA × 3K which will contain the triangular decomposition of matrix for interpolation along Y. LA is the first dimension of arrays AX, AY, C, F as declared in the calling program. For simplicity, the first dimension is assumed to be the same in all these arrays. LA has to be greater than the maximum of NX and NY. C is a real array of length LA $\times$ NY which will contain the calculated coefficients of expansion. This array will be required to evaluate the interpolated value at any given point. XF and YF are real arrays of length MX and MY, respectively containing the knots for B-splines along X and Y. MX, MY are the actual number of knots used along x and y. These numbers may not be the same as NX, NY since some points may have to be dropped to match the number of equations and number of B-spline basis functions. IFLG is a flag to decide the nature of computations. For IFLG ≤ 1 the triangular decomposition of matrices is calculated, for larger values of IFLG it is assumed that the triangular decomposition and other information is already available in arrays AX, AY. INTX and INTY are integer arrays of lengths NX and NY respectively, which will contain the information about pivoting used to solve the system of equations for determining the coefficients for 1-dimensional interpolation. WK is a real array of length NX$\times$LA + NX + NY used as scratch space. IER is the error parameter. IER = 0 implies successful execution of the program. Nonzero values may be set by BSPINT, BSPLIN or GAUBND which are called. This subroutine requires BSPINT, BSPLIN and GAUBND. The interpolated value can be calculated using function routine BSPEV2 using the coefficients computed by BSPINT2.

25. BSPEV2 Function routine to calculate the value of function using available coefficients of B-spline expansion in two dimensions. The calculated value is given by:

$$\mathrm{BSPEV2} = \sum_{i=1}^{\mathrm{NX+K-2}} \sum_{j=1}^{\mathrm{NY+K-2}} \mathrm{WT}(i,j)\phi_i(\mathrm{X0})\psi_j(\mathrm{Y0}), \tag{B.6}$$

where $\phi_i(x)$ are the B-spline basis functions along x and $\psi_i(y)$ are the B-spline basis functions along y. It can be used to calculate the interpolated value of the function if the coefficients are already calculated using BSPINT2. NX, NY are the number of knots along X, Y, which may not be the same as the number of points in the table for interpolation. X, Y are real arrays of length NX, NY containing the knots for B-splines. The knots must be in ascending order with X(1) and Y(1) containing the first knot along respective directions. K is the order of B-splines, K = 2 gives linear B-splines, while K = 4 yields cubic B-splines, etc. For simplicity, the order is assumed to be the same along both axes. NDERIV specifies the number of derivatives to be calculated. For NDERIV ≤ 0 only the function value will be calculated. For NDERIV = 1, the first derivatives are also calculated. For NDERIV > 1, the second derivatives will also be calculated. WT is the real array of length IW $\times$ (NY + K − 2), containing the coefficients of expansion in terms of B-spline basis functions. These coefficients must be calculated beforehand using BSPINT2 or any other equivalent routine for B-spline approximations. X0, Y0 is the point where the function value needs to be calculated. DFX, DFY are the first derivatives of function with respect to x, y respectively, at (X0, Y0). while DFXX, DFXY, DFYY are the second derivatives $\partial^2 f/\partial x^2$, $\partial^2 f/\partial x \partial y$, $\partial^2 f/\partial y^2$. The derivatives would be calculated only if NDERIV has been set appropriately. WK is a real array of length at least $7 \times \max(\mathrm{NX}, \mathrm{NY}) + 8\mathrm{K} + 2$ used as scratch space. IER is the error parameter, which should be zero after successful execution. Nonzero values of IER may be set by subroutine BSPLIN which is called to calculate the B-spline basis functions. This routine requires subroutine BSPLIN.

26. BSPINTN Subroutine to calculate the coefficients for B-spline interpolation to a table of values in N dimensions. N is the number of dimensions. NK is an integer array of length N, giving the number of tabular points along each dimension. NK(I) is the number of points along x_I. X is a real array of length NXD $\times$ N specifying the abscissas along each dimension. X(I, J) is the Ith abscissa along the Jth dimension. For each dimension, the abscissas must be unique and in ascending order. NXD is the first dimension of arrays X, XF and INTX as specified in the calling program, NXD $\geq \max(\mathrm{NK}(1), \mathrm{NK}(2), \ldots, \mathrm{NK}(\mathrm{N}))$. F is a real array of length $\mathrm{NK}(1) \times \mathrm{NK}(2) \times \cdots \times \mathrm{NK}(\mathrm{N})$ containing the table of values. Since BSPINTN treats this array as a one dimensional array, the dimensions of F in the calling program must exactly match the size of the table so that there are no gaps in memory allocation. For example, the dimension could be F(NK(1), NK(2), ..., NK(N)). Alternately, it could be treated as a one dimensional array of appropriate length. K is the order of B-spline interpolation required. K $=$ 4 implies cubic B-splines, while K $=$ 2 gives linear B-splines, etc. For simplicity, K is assumed to be the same along all dimensions. AX is a real array of length NXD $\times$ 3K $\times$ N, which will contain the triangular decomposition of the matrix of equations that is solved to calculate the coefficients of expansion for each dimension. Since the matrix is in band form with bandwidth $\mathrm{K} - 1$, it is stored in band form. C is a real array of length $\mathrm{NK}(1) \times \mathrm{NK}(2) \times \cdots \times \mathrm{NK}(\mathrm{N})$, which will contain the coefficients of B-spline basis functions in expansion for interpolation. This array is also treated as one-dimensional array just like F and hence the dimensions in calling program must exactly match the size of the table. XF is a real array of size NXD $\times$ N which will contain the knots used in each dimension. This array will contain the knots used for B-spline interpolation in each dimension. Since the number of B-spline basis functions with MK(I) knots is $\mathrm{MK(I)} + \mathrm{K} - 2$, some tabular points may have to be dropped to ensure that $\mathrm{NK(I)} = \mathrm{MK(I)} + \mathrm{K} - 2$. This is equivalent to using the not-a-knot boundary conditions in cubic spline interpolation. The number of points to be dropped depends on K. In all cases some points near both ends are dropped. For linear B-splines, K $=$ 2 all points are used. For cubic B-splines, K $=$ 4, the 2nd and $\mathrm{NK(I)} - 1$ th points are dropped. MK is an integer array of length N containing the number of knots for B-splines in each dimension. MK(I) is the number of knots used for B-spline interpolation, this would be equal to $\mathrm{NK(I)} + 2 - \mathrm{K}$. INTX is an integer array of length NXD $\times$ N, which will contain the pivoting information for solution of system of linear equations. WK is a real array of length NK(1) × NK(2) × ⋯ × NK(N) + 3K, used as scratch space. IER is the error parameter. IER = 0 implies successful execution of subroutine. Nonzero values may be set by subroutines BSPINT, BSPLIN or GAUBND, which are called. It may be noted that for simplicity, this subroutine does not use IFLG to control the calculations as for large number of dimensions main effort goes in calculating the coefficients rather than in calculating the triangular decomposition of matrices. This subroutine requires subroutine BSPINT to perform interpolation in one dimension, subroutine BSPLIN to calculate the B-spline basis functions and subroutine GAUBND to solve the system of linear equations with a band matrix. The interpolant at any required point can be calculated using FUNCTION BSPEVN, using the calculated coefficients of expansion. If the first derivative of the function is also required then use BSPEVN1, while for second derivatives use BSPEVN2.

27. BSPEVN Function routine to calculate the value of function using available coefficients of B-spline expansion in n-dimensions. The calculated value is given by:

$$\begin{aligned}\mathrm{BSPEVN} = &\sum_{i_1=1}^{\mathrm{NK(1)+K-2}} \cdots \sum_{i_n=1}^{\mathrm{NK(n)+K-2}} \mathrm{WT}(i_1,\ldots,i_n)\\ &\times \phi_{i_1}^{(1)}(\mathrm{X0(1)})\cdots\phi_{i_n}^{(n)}(\mathrm{X0}(n)),\end{aligned} \tag{B.7}$$

where $\phi_{i_j}^{(j)}$ are the B-spline basis functions along jth dimension. It can be used to calculate the interpolated value of the function if the coefficients are already calculated using BSPINTN. N is the number of dimensions. NK is an integer array of length N containing the number of knots along each dimension, which may not be the same as the number of points in the table for interpolation. NK(I) is the number of knots along the Ith dimension. X is a real array of length NXD × N containing the knots for B-splines along each dimension. The knots must be distinct and in ascending order with X(I, J) containing Ith knot along Jth dimension. NXD is the first dimension of array X in the calling program. NXD must be greater than or equal to the maximum of NK(I). K is the order of B-splines, K = 2 gives linear B-splines, while K = 4 yields cubic B-splines, etc. WT is the real array of length (NK(1)+K−2)(NK(2)+K−2) ⋯ (NK(N)+K−2), containing the coefficients of expansion in terms of B-spline basis functions. Since BSPEVN treats this array as a one-dimensional array, the dimensions of this array in the calling program must exactly match the table size, so that there are no gaps in memory allocation. For example, the dimension could be WT(NK(1)+K−2, NK(2)+K−2, …, NK(N)+K−2). These coefficients WT must be calculated beforehand using BSPINTN or any other equivalent routine for B-spline approximations in n-dimensions. X0 is a real array of length N, specifying the coordinates of the point where the function value needs to be calculated. WK is a real array of length at least (NXD + K)(3N + 1) + K + 2 used as scratch space. IWK is an integer array of length at least 2N used as scratch space. IER is the error parameter, which should be zero after successful execution. Nonzero values of IER may be set by subroutine BSPLIN which is called to calculate the B-spline basis functions. This routine does not calculate the derivatives of expansion. Since calculating derivatives could require considerable extra time separate versions of this routine BSPEVN1 and BSPEVN2 are provided to calculate first and second derivatives. These routines can be used instead of BSPEVN when derivatives are required. This routine requires subroutine BSPLIN.

28. BSPEVN1 Function routine to calculate the value of function and its first derivatives using available coefficients of B-spline expansion in n-dimensions. This routine is the version of BSPEVN to calculate the first derivatives in addition to function values. It can be used when derivatives are also required. The arguments are the same as those for BSPEVN, except for array DF of length N, which will contain the calculated derivatives with respect to each of the dimensions. DF(I) will contain the first derivative with respect to x_I at the point X0. The scratch array WK should have a length of at least $(\mathrm{NXD}+\mathrm{K})(3\mathrm{N}+1)+\mathrm{K}+\mathrm{N}+2$. If only function value is required then BSPEVN should be used, while if second derivative is also required then BSPEVN2 should be used.

29. BSPEVN2 Function routine to calculate the value of function and its first and second derivatives using available coefficients of B-spline expansion in n-dimensions. This routine is the version of BSPEVN to calculate the first and second derivatives in addition to function values. It can be used when derivatives are also required. The arguments are the same as those for BSPEVN, except for array DF of length N, which will contain the calculated derivatives with respect to each of the dimensions. DF(I) will contain the first derivative with respect to x_I at the point X0. DDF is a real array of length $\mathrm{N}\times\mathrm{N}$, which will contain the calculated second derivatives. DDF(I, J) will contain $\partial^2 f/\partial x_I \partial x_J$ at the point X0. There is no provision to pass the first dimension of DDF and hence it should match the value in this subroutine. The scratch array WK should have a length of at least $(\mathrm{NXD}+\mathrm{K})(3\mathrm{N}+1)+\mathrm{K}+\mathrm{N}+\mathrm{N}^2+2$. If second derivatives are not required then it will be better to use BSPEVN1 (for first derivatives) or BSPEVN (for only function value).

B.5 Differentiation

30. DRVT Function routine to calculate derivative of a function which can be evaluated at any required point using a user supplied function routine FUNCTION F(X). This subroutine uses $h \to 0$ extrapolation to obtain accurate value of the derivative. It can evaluate the first, second, third or fourth derivative of the given function. A is the value of x at which the derivative is to be evaluated. ID is the order of the derivative required. ID can be 1, 2, 3 or 4. If any other value is specified the program will exit with the error flag IER set to 208. HH0 is the initial spacing to be used, which depends on the function and the precision of the arithmetic used. A value between 0.1 and 1 will usually be sufficient. If HH0 is too large, then sufficient accuracy may not be achieved by the program, while if it is too small, then roundoff error will dominate. AEPS and REPS specify the required absolute and relative accuracy. This subroutine exits when the difference between successive values is less than $\max(\mathrm{AEPS}, \mathrm{REPS} \times |\mathrm{DRVT}|)$, or when roundoff error is dominating. The presence of roundoff error is detected by using a simple test as explained in Section 5.3. F is the name of the function routine which calculates the given function at any required point. This routine must be supplied by the user. IER is the error parameter, which will be set to 28 if sufficient accuracy is not achieved. IER will be set to 29 if roundoff errors start dominating before adequate convergence is achieved. IER will be set to 208 if ID is outside the specified range.

B.6 Integration

31. SIMSON Subroutine to integrate a smooth function over a finite interval using composite Simpson's rule. RI is the output parameter containing the calculated value of the integral. XL and XU are the lower and upper limits for the integral. It is not essential to have XL $<$ XU. REPS and AEPS specify the required relative and absolute accuracy. The calculations are terminated when two successive values differ by less than $\max(\mathrm{AEPS}, \mathrm{REPS} \times |\mathrm{RI}|)$. DIF is an output parameter containing an estimate of (absolute) error calculated by the subroutine. N is an output parameter which will contain the number of function evaluations used by the subroutine. IER is the error parameter. IER $=$ 30 denotes that the subroutine has failed to converge to the specified accuracy. In most cases the computed value of the integral may still be approximately correct. Only in extreme cases the computed value will be far from actual value. In this case, RI will contain the best estimate for integral and DIF should contain the estimated error. The parameter NMAX in the subroutine could be increased, if larger number of function evaluations are to be allowed. However, in most case, it will be better to use another technique for evaluating the integral. FUN is the name of the function routine to calculate the integrand. FUNCTION FUN(X) must be supplied by the user.

32. SPLINT Subroutine to compute integral of a function supplied in the form of a table of values by integrating the interpolating cubic spline. Before calling this subroutine, the coefficients of cubic spline must be calculated by a call to subroutine SPLINE. SINT and TINT are the output parameters containing the value of the integral. SINT is the estimate using cubic spline, while TINT is the trapezoidal rule estimate, using only the tabular points. The difference between TINT and SINT may give some estimate of truncation error in the calculations. If the limits are not among the tabular points, then the function value at the end points are obtained using function SPLEVL. A and B are the lower and upper limits of the integral. It is essential to ensure that A $<$ B. N is the number of points in the table of values. X and F are real arrays of length N, containing the abscissas and function values. The abscissas must be supplied in ascending order. C is a real array of length $3 \times \mathrm{N}$ containing the coefficients of cubic spline, which may be calculated using the subroutine SPLINE. IER is the error parameter. IER = 31 denotes that the lower limit A is outside the limits of table, while IER = 32 denotes that the upper limit B is outside the limits of table. In such cases, the accuracy of integration may be questionable. IER = 301 implies that $\mathrm{A} > \mathrm{B}$ or $\mathrm{X}(1) > \mathrm{X}(\mathrm{N})$. This subroutine requires function SPLEVL to evaluate the cubic spline, while subroutine SPLINE will be required to calculate the coefficients of cubic spline before calling this subroutine.

33. BSPQD Function routine to compute integral of an expansion in terms of B-spline basis functions. The expansion may be obtained by interpolating or approximating a table of values. Before calling this subroutine, the coefficients of B-spline expansion must be calculated by a call to subroutine BSPINT.

$$\mathrm{BSPQD} = \int_{\mathrm{XL}}^{\mathrm{XU}} \sum_{i=1}^{\mathrm{N+K-2}} \mathrm{WT(i)}\ \phi_i(x)\ dx\ , \qquad \text{(B.8)}$$

where $\phi_i(x)$ are the B-spline basis functions of order K. N is the number of knots, X is a real array of length N containing the knots for B-splines. The knots must be in ascending order with X(1) containing the first knot. K is the order of B-splines, K = 2 gives linear B-splines, while K = 4 yields cubic B-splines, etc. WT is a real array of length $\mathrm{N} + \mathrm{K} - 2$ containing the coefficients of expansion in terms of B-spline basis functions. These coefficients may be calculated using BSPINT for interpolation in a table of values. XL and XU are the lower and upper limits of the integral. WK is a real array of length at least $5\mathrm{N} + 6\mathrm{K} + 9$ used as scratch space. IER is the error parameter. IER = 31 denotes that the lower limit XL is outside the limits of table, while IER = 32 denotes that the upper limit XU is outside the limits of table. In such cases, the accuracy of integration may be questionable. Other values of IER may be set by subroutine BSPLIN which is called to calculate B-spline basis functions. This subroutine requires function BSPLIN to evaluate the B-spline basis functions, while subroutine BSPINT will be required to calculate the coefficients of B-spline before calling this subroutine.

34. ROMBRG Subroutine to compute integral of a function over a finite interval, using Romberg integration or the $h \to 0$ extrapolation. RI is the output parameter containing the value of the integral. A and B are the lower and upper limits of the integral. It is not essential to have $\mathrm{A} < \mathrm{B}$. GI is a real array of dimension NMAX (= 13), containing the expected values of successive exponents in the error expansion for the trapezoidal rule. If $\mathrm{GI(I)} \leq 0$, then it will be set to 2I, which is the correct value for smooth functions. AEPS and REPS specify the required absolute and relative accuracy. The calculations are terminated when the successive values differ by less than $\max(\mathrm{AEPS}, \mathrm{REPS} \times |\mathrm{RI}|)$. DIF is an output parameter giving the error as estimated by the subroutine. N specifies the number of abscissas to be used for the first attempt with trapezoidal rule. After execution, N will contain the number of abscissas actually used by the subroutine. Hence, N will have to be reset after every call to the subroutine. If $N < 2$ it will be set to a default value of 2. On the other hand, if $N > NPT$ (= 100), then it will be set to a default value of 2 and the error flag IER will be set to 33. In particular, this situation may arise if N is not reset after the previous call to the subroutine. IER is the error parameter. IER = 30 denotes that the subroutine failed to converge to the specified accuracy. In this case, RI will contain the best estimate for the integral and DIF should contain the estimated error. The parameter NMAX inside the subroutine could be increased to allow for larger number of function evaluations, but it is better to use some other technique for evaluating the integral. FUN is the name of the function routine used to calculate the value of the integrand. FUNCTION FUN(X) must be supplied by the user. This subroutine should be used for smooth functions with GI(I) = 0. It can also handle algebraic singularity at one or both end points, provided correct values of GI are supplied.

35. EPSILN Subroutine to compute integral of a function over a finite interval, using ϵ-algorithm to accelerate the convergence. RI is the output parameter containing the value of the integral. A and B are the lower and upper limits of the integral. It is not essential to have $A < B$. AEPS and REPS specify the required absolute and relative accuracy. The calculations are terminated when the successive values differ by less than $\max(\mathrm{AEPS}, \mathrm{REPS} \times |\mathrm{RI}|)$. DIF is an output parameter giving the error as estimated by the subroutine. N specifies the number of abscissas to be used for the first attempt with trapezoidal rule. After execution, N will contain the number of abscissas actually used by the subroutine. Hence, N will have to be reset after every call to the subroutine. If $N < 2$, it will be set to a default value of 2. On the other hand, if $N > NPT$ (= 100), then it will be set to a default value of 2 and the error flag IER will be set to 33. In particular, this situation may arise if N is not reset after the previous call to the subroutine. IER is the error parameter. IER = 30 denotes that the subroutine failed to converge to the specified accuracy. IER = 34 denotes that at some stage while constructing the ϵ-table, the denominator was zero. In this case, the calculations are continued further after ignoring the corresponding term. This may be justified if this problem occurs in higher columns which are not converging because of roundoff error. A better strategy would be to stop calculating the higher columns. IER = 35 denotes that the roundoff error is dominating and the calculations are terminated, even though the required accuracy is not achieved. In all these cases, RI will contain the best estimate for the integral and DIF should contain the estimated error. The parameter NMAX inside the subroutine could be increased to allow for larger number of function evaluations, but that is not recommended. FUN is the name of the function routine used to calculate the value of the integrand. FUNCTION FUN(X) must be supplied by the user. This subroutine could be used for singular integrands, if only moderate accuracy is required. It will fail to converge to high accuracy because of roundoff error.

36. GAUSS Subroutine to compute integral of a function over a finite interval, using composite Gauss-Legendre formulae. RINT is the output parameter containing the value of the integral. A and B are the lower and upper limits of the integral. It is not essential to have A $<$ B. NP specifies the formula to be used. This subroutine uses a composite rule based on NP-point Gauss-Legendre formula. NP should be 2, 4, 8, 16 or 32, since these are the only formulae for which the weights and abscissas are incorporated. If NP is not equal to one of these values, then it will be set to a default value of 8, and error flag IER will be set to 36. AEPS and REPS specify the required absolute and relative accuracy. The calculations are terminated when the successive values differ by less than $\max(\mathrm{AEPS}, \mathrm{REPS} \times |\mathrm{RINT}|)$. DIF is an output parameter giving the error as estimated by the subroutine. NPT is an output parameter containing the number of function evaluations actually used by the subroutine. IER is the error parameter. IER = 30 denotes that the subroutine failed to converge to the specified accuracy. In this case, RINT will contain the best estimate for the integral and DIF should contain the estimated error. In such cases, attempt could be made to use higher order Gaussian formula, NP = 32 is the maximum value allowed by this subroutine. The parameter NMAX inside the subroutine could be increased to allow for larger number of function evaluations, but it will be preferable to use some other technique for evaluating the integral. IER = 36 implies that NP was not 2,4,8,16 or 32, in which case it is set to 8. FUN is the name of the function routine used to calculate the value of the integrand. FUNCTION FUN(X) must be supplied by the user. This subroutine should be used for smooth functions. NP could be set to a low value if only moderate accuracy is required, while for high accuracy NP may be increased. The optimum value of NP will depend on the degree of smoothness of the integrand and the accuracy required.

37. GAUCBY Subroutine to compute integral of a function using Gauss-Chebyshev formulae with weight function $1/\sqrt{(x-\mathrm{A})(\mathrm{B}-x)}$. RINT is the output parameter containing the value of the integral. A and B are the lower and upper limits of the integral. It is essential to have A $<$ B. AEPS and REPS specify the required absolute and relative accuracy. The calculations are terminated when the successive values differ by less than $\max(\mathrm{AEPS}, \mathrm{REPS} \times |\mathrm{RINT}|)$. DIF is an output parameter giving the error as estimated by the subroutine. NPT is an output parameter containing the number of function evaluations actually used by the subroutine. IER is the error parameter. IER = 30 denotes that the subroutine failed to converge to the specified accuracy. In this case, RINT will contain the best estimate for the integral and DIF should contain the estimated error. The parameter NMAX inside the subroutine could be increased to allow for larger number of function evaluations, but it will be preferable to use some other technique for evaluating the integral. FUN is the name of the function routine used to calculate the value of the integrand multiplied by $\sqrt{(x-\mathrm{A})(\mathrm{B}-x)}$. FUNCTION FUN(X) must be supplied by the user. This subroutine should be used for functions with square root singularity at both ends. It will evaluate the integral

$$\int_{\mathrm{A}}^{\mathrm{B}} \frac{\mathrm{FUN}(x)}{\sqrt{(x-\mathrm{A})(\mathrm{B}-x)}}\,dx\;. \tag{B.9}$$

38. GAUCB1 Subroutine to compute integral of a function using Gauss-Chebyshev formulae with weight function $\sqrt{(x-\mathrm{A})/(\mathrm{B}-x)}$. RINT is the output parameter containing the value of the integral. A and B are the lower and upper limits of the integral. It is essential to have A < B. AEPS and REPS specify the required absolute and relative accuracy. The calculations are terminated when the successive values differ by less than $\max(\mathrm{AEPS}, \mathrm{REPS}\times|\mathrm{RINT}|)$. DIF is an output parameter giving the error as estimated by the subroutine. NPT is an output parameter containing the number of function evaluations actually used by the subroutine. IER is the error parameter. IER = 30 denotes that the subroutine failed to converge to the specified accuracy. In this case, RINT will contain the best estimate for the integral and DIF should contain the estimated error. The parameter NMAX inside the subroutine could be increased to allow for larger number of function evaluations, but it will be preferable to use some other technique for evaluating the integral. FUN is the name of the function routine used to calculate the value of the integrand multiplied by $\sqrt{(\mathrm{B}-x)/(x-\mathrm{A})}$. FUNCTION FUN(X) must be supplied by the user. This subroutine should be used for functions with square root singularity at both ends. It will evaluate the integral

$$\int_{\mathrm{A}}^{\mathrm{B}} \frac{\mathrm{FUN}(x)\sqrt{x-\mathrm{A}}}{\sqrt{\mathrm{B}-x}}\,dx\;. \tag{B.10}$$

39. GAUCB2 Subroutine to compute integral of a function using Gauss-Chebyshev formulae with weight function $\sqrt{(x-\mathrm{A})(\mathrm{B}-x)}$. RINT is the output parameter containing the value of the integral. A and B are the lower and upper limits of the integral. It is essential to have A < B. AEPS and REPS specify the required absolute and relative accuracy. The calculations are terminated when the successive values differ by less than $\max(\mathrm{AEPS}, \mathrm{REPS}\times|\mathrm{RINT}|)$. DIF is an output parameter giving the error as estimated by the subroutine. NPT is an output parameter containing the number of function evaluations actually used by the subroutine. IER is the error parameter. IER = 30 denotes that the subroutine failed to converge to the specified accuracy. In this case, RINT will contain the best estimate for the integral and DIF should contain the estimated error. The parameter NMAX inside the subroutine could be increased to allow for larger number of function evaluations, but it will be preferable to use some other technique for evaluating the integral. FUN is the name of the function routine used to calculate the value of the integrand divided by $\sqrt{(\mathrm{B}-x)(x-\mathrm{A})}$. FUNCTION FUN(X) must be supplied by the user. This subroutine should be used for functions with square root singularity at both ends. It will evaluate the integral

$$\int_{\mathrm{A}}^{\mathrm{B}} \mathrm{FUN}(x)\sqrt{(x-\mathrm{A})(\mathrm{B}-x)}\;dx\,. \tag{B.11}$$

40. GAUSQ2 Subroutine to compute integral of a function with square root singularity over $(0,a]$ using a combination of Gauss-Legendre and a Gaussian formula with $1/\sqrt{x}$ weight function. RINT is the output parameter containing the value of the integral. A is the upper limit of the integral. A1 is the point at which the integral is broken, with the integral over [A1, A] being evaluated using Gauss-Legendre formula, while that over (0, A1] is evaluated using a Gaussian formula with $1/\sqrt{x}$ weight function. If A < A1, then A1 is set equal to A to start with. A1 is adjusted by the subroutine to achieve the required accuracy. After execution, A1 will contain the final value used by the subroutine. A1 may need to be reset after every call to GAUSQ2. If the next integral is similar to the previous one, then the value of A1 need not be reset, since it is probably the required value. AEPS and REPS specify the required absolute and relative accuracy. The calculations are terminated when the successive values differ by less than max(AEPS, REPS × |RINT|). Convergence is checked separately for the two integrals. DIF is an output parameter giving the error as estimated by the subroutine. NP is the output parameter containing the number of function evaluations actually used by the subroutine. This subroutine calls the subroutines GAUSS and GAUSQ to perform the integration over required subintervals. IER is the error parameter. IER = 30 denotes that the subroutine GAUSS failed to converge to the specified accuracy over the interval [A1, A]. IER = 37 denotes that the subroutine GAUSQ failed to converge to the specified accuracy, even when A1 was reduced to its minimum permissible value of AMN (= 0.01). This limit is provided to prevent the subroutine from getting into an infinite loop. This can happen if the integrand is not in a form $f(x)/\sqrt{x}$ as required for Gaussian formula. If necessary, the lower limit AMN can be reduced. IER = 38 denotes that both subroutines failed. In all these cases, RINT will contain the best estimate for the integral and DIF should contain the estimated error. F is the name of the function routine used to calculate the value of the integrand. F2 is the name of the function routine used to calculate `F(X)*SQRT(X)`, as required by the Gaussian formula. Note that the subroutine will find the integral

$$\int_0^{\mathrm{A}} \mathrm{F}(x)\;dx = \int_{\mathrm{A1}}^{\mathrm{A}} \mathrm{F}(x)\;dx + \int_0^{\mathrm{A1}} \frac{\mathrm{F2}(x)}{\sqrt{x}}\;dx\,. \tag{B.12}$$

Both the FUNCTION F(X), and FUNCTION F2(X) must be supplied by the user. The requirement of two separate functions is to avoid possible problems caused by singularity in integrand. If A − A1 < AEPS, then the first integral is not evaluated. This condition may need to be changed if the integral can make significant contribution, even though this condition is satisfied, which may happen if the limits are very small. This subroutine requires subroutines

GAUSS and GAUSQ. It should be used only for those functions which have square root singularity at $x = 0$. It can be used for integrands of form $f(x)/\sqrt{x}$, where $f(x)$ is regular at $x = 0$. It can also be used for integrands of form $f(x)\sqrt{x} = (xf(x))/\sqrt{x}$.

41. GAUSQ Subroutine to compute integral

$$\int_0^{\mathrm{A}} \frac{\mathrm{FUN}(x)}{\sqrt{x}}\, dx\ , \tag{B.13}$$

using Gaussian formulae with $1/\sqrt{x}$ weight function. The subroutine computes the integral using different formulae, until convergence is achieved or until the table of weights and abscissas is exhausted. RINT is the output parameter containing the value of the integral. A is the upper limit of the integral. REPS and AEPS specify the required accuracy, while DIF is an output parameter giving the error as estimated by the subroutine. The convergence criterion used is $|\mathrm{DIF}| < \max(\mathrm{REPS}|\mathrm{RINT}|, \mathrm{AEPS})$. IER is the error parameter. IER $= 30$ denotes that the subroutine failed to converge to the specified accuracy. In this case, RINT will contain the best estimate for the integral and DIF should contain the estimated error. In such cases, we can try to use a lower value of A. NPT is an output parameter containing the number of function evaluations actually used by the subroutine. FUN is the name of the function routine used to calculate the value of the integrand (multiplied by $\sqrt{x}$). FUNCTION FUN(X) must be supplied by the user. This subroutine should be used for functions which have a square root singularity (functions of form $f(x)/\sqrt{x}$ or $f(x)\sqrt{x}$) at $x = 0$. This routine should be preferably used through GAUSQ2, in which case the upper limit A may be adjusted if the required accuracy is not achieved.

42. GAULAG Subroutine to compute integral of a function over $[a, \infty)$ using a combination of Gauss-Laguerre and Gauss-Legendre quadrature formulae. RINT is the output parameter containing the value of the integral. A is the lower limit of the integral. A1 is the point at which the integral is broken, with the integral over $[\mathrm{A}, \mathrm{A1}]$ being evaluated using Gauss-Legendre formula, while that over $[\mathrm{A1}, \infty)$ is evaluated using the Gauss-Laguerre formula. If A1 $<$ A, then A1 is set equal to A to start with. A1 is adjusted by the subroutine to achieve the required accuracy. After execution, A1 will contain the final value used by the subroutine. A1 may need to be reset after every call to GAULAG. If the next integral is similar to the previous one, then the value of A1 need not be reset, since it is probably the required value. AEPS and REPS specify the required absolute and relative accuracy. The calculations are terminated when the successive values differ by less than $\max(\mathrm{AEPS}, \mathrm{REPS} \times |\mathrm{RINT}|)$. Convergence is checked separately for the two integrals. DIF is an output parameter giving the error as estimated by the subroutine. NP is the output parameter containing the number of function evaluations actually used by the subroutine. This subroutine calls the subroutines GAUSS and LAGURE to perform the integration over required subintervals. IER is the error parameter. IER $= 30$

denotes that the subroutine GAUSS failed to converge to the specified accuracy over the interval $[\mathrm{A}, \mathrm{A1}]$. IER $= 37$ denotes that the subroutine LAGURE failed to converge to the specified accuracy, even when A1 was raised to its maximum permissible value of AMAX ($= 50$). This limit is provided to prevent the subroutine from getting into an infinite loop. This can happen if the integrand is not in a form $e^{-x}f(x)$ as required for Gauss-Laguerre formula. If necessary, the upper limit AMAX can be increased. IER $= 38$ denotes that both subroutines failed. In all these cases, RINT will contain the best estimate for the integral and DIF should contain the estimated error. F is the name of the function routine used to calculate the value of the integrand. F2 is the name of the function routine used to calculate `F(X)*EXP(X)`, as required by the Gauss-Laguerre formula. Note that the subroutine will find the integral

$$\int_{\mathrm{A}}^{\infty} \mathrm{F}(x)\;dx = \int_{\mathrm{A}}^{\mathrm{A1}} \mathrm{F}(x)\;dx + \int_{\mathrm{A1}}^{\infty} e^{-x}\mathrm{F2}(x)\;dx\;. \tag{B.14}$$

Both the FUNCTION F(X), and FUNCTION F2(X) must be supplied by the user. The requirement of two separate functions is to avoid the problem of overflow and underflow, which may occur if F2 is formed by adding the exponential factor to F inside the subroutine. If $\mathrm{A1} - \mathrm{A} < \mathrm{AEPS}$, then the first integral is not evaluated. This condition may need to be changed if the integral can make significant contribution, even though this condition is satisfied, which may happen if the limits are very small. This subroutine requires subroutines GAUSS and LAGURE. It should be used only for those functions which fall off exponentially at large X. Similar subroutine can be written to handle singularities by combining the Gauss-Legendre formula with another Gaussian formula for singular weight functions.

43. LAGURE Subroutine to compute integral $\int_{\mathrm{A}}^{\infty} e^{-x}\mathrm{F}(x)\;dx$ using Gauss-Laguerre formulae. The subroutine computes the integral using different formulae, until convergence is achieved or until the table of weights and abscissas is exhausted. RINT is the output parameter containing the value of the integral. A is the lower limit of the integral. REPS and AEPS specify the required accuracy, while DIF is an output parameter giving the error as estimated by the subroutine. The convergence criterion used is $|\mathrm{DIF}| < \max(\mathrm{REPS}|\mathrm{RINT}|, \mathrm{AEPS})$. NPT is an output parameter containing the number of function evaluations actually used by the subroutine. IER is the error parameter. IER $= 30$ denotes that the subroutine failed to converge to the specified accuracy. In this case, RINT will contain the best estimate for the integral and DIF should contain the estimated error. In such cases, we can try to use a higher value of A. It may be noted that, depending on the function being integrated the higher order formula may have problems, because of underflow and overflow on computers with short range for the exponents. F is the name of the function routine used to calculate the value of the integrand (multiplied by e^x). FUNCTION F(X) must be supplied by the user. This subroutine should be used for functions which fall off exponentially at large x. It will be better to use this routine through GAULAG, which adjusts the lower limit A to achieve the required accuracy.

44. HERMIT Subroutine to compute integral $\int_{-\infty}^{\infty} e^{-x^2}\mathrm{F}(x)\,dx$ using Gauss-Hermite formulae. The subroutine computes the integral using different formulae, until convergence is achieved or until the table of weights and abscissas is exhausted. RINT is the output parameter containing the value of the integral. REPS and AEPS specify the required accuracy, while DIF is an output parameter giving the error as estimated by the subroutine. The convergence criterion used is $|\mathrm{DIF}| < \max(\mathrm{REPS}|\mathrm{RINT}|, \mathrm{AEPS})$. NPT is an output parameter containing the number of function evaluations actually used by the subroutine. IER is the error parameter. IER $= 30$ denotes that the subroutine failed to converge to the specified accuracy. In this case, RINT will contain the best estimate for the integral and DIF should contain the estimated error. It may be noted that, depending on the function being integrated the higher order formula may have problems, because of underflow and overflow on computers with short range for the exponents. F is the name of the function routine used to calculate the value of the integrand (multiplied by e^{x^2}). FUNCTION F(X) must be supplied by the user. This subroutine should be used for functions which fall off exponentially as e^{-x^2} at large $|x|$. In this case it is not possible to divide the range and hence it is not possible to apply any composite formulae or to use a combination of formulae to improve accuracy.

45. GAULG2 Subroutine to compute integral of a function with logarithmic singularity over $(0, a]$ using a combination of Gauss-Legendre and a Gaussian formula with $\log(1/x)$ weight function. RINT is the output parameter containing the value of the integral. A is the upper limit of the integral. A1 is the point at which the integral is broken, with the integral over $[\mathrm{A1}, \mathrm{A}]$ being evaluated using Gauss-Legendre formula, while that over $(0, \mathrm{A1}]$ is evaluated using a Gaussian formula with $\log(1/x)$ weight function. If $\mathrm{A} < \mathrm{A1}$, then A1 is set equal to A to start with. A1 is adjusted by the subroutine to achieve the required accuracy. After execution, A1 will contain the final value used by the subroutine. A1 may need to be reset after every call to GAULG2. If the next integral is similar to the previous one, then the value of A1 need not be reset, since it is probably the required value. AEPS and REPS specify the required absolute and relative accuracy. The calculations are terminated when the successive values differ by less than $\max(\mathrm{AEPS}, \mathrm{REPS} \times |\mathrm{RINT}|)$. This criterion is applied separately to each part of the integral. DIF is an output parameter giving the error as estimated by the subroutine. NP is the output parameter containing the number of function evaluations actually used by the subroutine. This subroutine calls the subroutines GAUSS and GAULOG to perform the integration over required subintervals. IER is the error parameter. IER $= 31$ denotes that the subroutine GAUSS failed to converge to the specified accuracy over the interval $[\mathrm{A1}, \mathrm{A}]$. IER $= 32$ denotes that the subroutine GAULOG failed to converge to the specified accuracy, even when A1 was reduced to its minimum permissible value of AMN ($= 0.01$). This limit is provided to prevent the subroutine from getting into an infinite loop. This can happen if the integrand is not in a form $f(x)\log x$ as required for Gaussian formula. If necessary, the lower limit AMN can be reduced. IER $= 34$ denotes that subroutine

GAUSS failed while evaluating the integral over [0,A1]. In case of multiple failures, the second digit of IER will be the sum of these values. In all these cases, RINT will contain the best estimate for the integral and DIF should contain the estimated error. F is the name of the function routine used to calculate the value of the integrand. F2 is the name of the function routine used to calculate `F(X)/LOG(1/X)`, as required by the Gaussian formula. Note that the subroutine will find the integral

$$\int_0^{\mathrm{A}} \mathrm{F}(x)\;dx = \int_{\mathrm{A1}}^{\mathrm{A}} \mathrm{F}(x)\;dx - \log \mathrm{A1} \int_0^{\mathrm{A1}} \mathrm{F2}(x)\;dx + \int_0^{\mathrm{A1}} \mathrm{F2}(x) \log(\mathrm{A1}/x)\;dx\;. \tag{B.15}$$

Both the FUNCTION F(X), and FUNCTION F2(X) must be supplied by the user. The requirement of two separate functions is to avoid possible problems caused by singularity in integrand. The first two integrals should not have any singularity and are evaluated using Gauss-Legendre formula, while the last integral is evaluated using Gaussian formula with logarithmic singularity. Note that the last two integrals arise when the range in integral over (0, A1] is transformed to (0, 1]. If $\mathrm{A} - \mathrm{A1} < \mathrm{AEPS}$, then the first integral is not evaluated. This condition may need to be changed if the integral can make significant contribution, even though this condition is satisfied, which may happen if the limits are very small. This subroutine requires subroutines GAUSS and GAULOG. It should be used only for those functions which have logarithmic singularity at $x = 0$.

46. GAULOG Subroutine to compute integral

$$\int_0^{\mathrm{A}} \mathrm{F}(x) \log(\mathrm{A}/x)\;dx\;, \tag{B.16}$$

using Gaussian formulae with $\log(\mathrm{A}/x)$ weight function. The subroutine computes the integral using different formulae, until convergence is achieved or until the table of weights and abscissas is exhausted. RINT is the output parameter containing the value of the integral. A is the upper limit of the integral. REPS and AEPS specify the required accuracy, while DIF is an output parameter giving the error as estimated by the subroutine. The convergence criterion used is $|\mathrm{DIF}| < \max(\mathrm{REPS}|\mathrm{RINT}|, \mathrm{AEPS})$. IER is the error parameter. IER = 30 denotes that the subroutine failed to converge to the specified accuracy. In this case, RINT will contain the best estimate for the integral and DIF should contain the estimated error. In such cases, we can try to use a lower value of A. NPT is an output parameter containing the number of function evaluations actually used by the subroutine. F is the name of the function routine used to calculate the value of the integrand (divided by $\log \mathrm{A}/x$). FUNCTION F(X) must be supplied by the user. This subroutine should be used for functions which have a logarithmic singularity at $x = 0$. It would be better to use this routine through GAULG2 which adjusts the value of A to achieve required accuracy.

47. GAUSRC Subroutine to calculate weights and abscissas of a Gaussian quadrature formula, with arbitrary weight function. This subroutine requires the recurrence relation for the corresponding orthogonal polynomials. The recurrence relation is assumed to be in the form

$$P_j(x) = (a_j x + b_j)P_{j-1}(x) - c_j P_{j-2}(x), \tag{B.17}$$

where $P_j(x)$ is the orthogonal polynomial of degree j in x. The coefficients a_j, b_j, c_j for $j = 1, 2, \ldots, n$ must be supplied. Here N is the number of abscissas in the required Gaussian formula. Output parameters W and AB are real arrays of length N, containing the weights and corresponding abscissas. COF is a real array of length $3 \times \mathrm{N}$ containing the coefficients in the recurrence relation for the polynomials. COF(1, i), COF(2, i) and COF(3, i) are respectively, a_i, b_i, c_i as defined above. These coefficients must be supplied. RI0 is the integral $\int_a^b w(x)\,dx$ for the required weight function over the corresponding interval. If this integral is not known it must be evaluated using appropriate quadrature formula before using this subroutine to calculate the weights and abscissas. IER is the error parameter. If $\mathrm{N} \leq 0$ then IER is set to 302 and no calculations are performed. IER = 321 implies that some coefficient becomes imaginary during calculation. This could happen only if coefficients for recurrence relation are not specified correctly. Other values may be set by TQL2 which is called to solve the eigenvalue problem. WK is a real array of length $\mathrm{N} \times (\mathrm{N} + 2)$ used as scratch space. There is no error check on the output and the accuracy can be tested by integrating the functions $w(x)x^n$ for $n = 0, 1, \ldots, 2\mathrm{N} - 1$, using the calculated weights and abscissas. This routine is in general better conditioned than GAUSWT, which uses only the moments. This routine requires subroutine TQL2.

48. GAULEG Subroutine to calculate weights and abscissas of a Gauss-Legendre quadrature formula. This subroutine uses the recurrence relation for Legendre polynomials

$$nP_n(x) = (2n - 1)xP_{n-1}(x) - (n - 1)P_{n-2}(x), \tag{B.18}$$

to calculate the weights and abscissas using subroutine GAUSRC. Here N is the number of abscissas in the required Gaussian formula. Output parameters W and A are real arrays of length N, containing the weights and corresponding abscissas. IER is the error parameter. IER = 0 implies successful execution. Nonzero values may be set by subroutine GAUSRC. WK is a real array of length $\mathrm{N} \times (\mathrm{N} + 2) + 3(\mathrm{N} + 1)$ used as scratch space. There is no error check on the output and the accuracy can be tested by integrating the functions x^j for $j = 0, 1, \ldots, 2\mathrm{N} - 1$, using the calculated weights and abscissas. This routine requires subroutines GAUSRC and TQL2.

49. GAUJAC Subroutine to calculate weights and abscissas of a Gauss-Jacobi quadrature formula. This subroutine uses the recurrence relation for Jacobi polynomials

$$\begin{aligned}2n(n+\alpha+\beta)(2n-2+\alpha+\beta)P_n^{\alpha,\beta}(x) =& \\ &-2(n-1+\alpha)(n-1+\beta)(2n+\alpha+\beta)P_{n-2}^{\alpha,\beta}(x) \\ &+(2n-1+\alpha+\beta)\big(\alpha^2-\beta^2+(2n+\alpha+\beta)(2n-2+\alpha+\beta)x\big)P_{n-1}^{\alpha,\beta}(x),\end{aligned} \tag{B.19}$$

to calculate the weights and abscissas using subroutine GAUSRC. Here N is the number of abscissas in the required Gaussian formula while ALP and BETA are the indices α and β in (B.19). The corresponding weight function is $w(x) = (1-x)^\alpha(1+x)^\beta$ on the interval $(-1,1)$. Output parameters W and A are real arrays of length N, containing the weights and corresponding abscissas. IER is the error parameter. IER = 0 implies successful execution. IER = 313 implies ALP ≤ -1 or BETA ≤ -1. Other values may be set by subroutine GAUSRC. WK is a real array of length N$\times$(N+2)+3(N+1) used as scratch space. There is no error check on the output and the accuracy can be tested by integrating the functions $(1-x)^\alpha(1+x)^\beta x^j$ for $j = 0,1,\ldots,2\text{N}-1$, using the calculated weights and abscissas. Setting ALP = 0 and BETA = 0 in this routine should yield the Gauss-Legendre formula. Similarly, setting ALP = $-1/2$ and BETA = $-1/2$ should yield the Gauss-Chebyshev formula. Using $\alpha = 0$ one can calculate quadrature formulae for algebraic singularity of form t^β, by transforming the lower limit to $t = 0$.

$$\int_{-1}^{1}(1-x)^\alpha(1+x)^\beta f(x)\,dx = 2^{\alpha+\beta+1}\int_0^1(1-t)^\alpha t^\beta f(2t-1)\,dt\,. \tag{B.20}$$

Thus for integration over interval $(0,1)$ the weights W(I) should be divided by $2^{1+\alpha+\beta}$ and abscissas should be (1+A(I))/2. This routine requires subroutines GAUSRC and TQL2 and function GAMMA.

50. LAGURW Subroutine to calculate weights and abscissas of a Gauss-Laguerre quadrature formula. This subroutine uses the recurrence relation for associated Laguerre polynomials

$$nL_n^\alpha(x) = (n+\alpha)\big((2n-1+\alpha)-x\big)L_{n-1}^\alpha(x) - (n-1+\alpha)^2(n+\alpha)L_{n-2}^\alpha(x), \tag{B.21}$$

to calculate the weights and abscissas using subroutine GAUSRC. Here N is the number of abscissas in the required Gaussian formula and ALP is the index α in (B.21). The corresponding weight function is $w(x) = x^\alpha e^{-x}$ on the interval $(0,\infty)$. Output parameters W and A are real arrays of length N, containing the weights and corresponding abscissas. IER is the error parameter. IER = 0 implies successful execution. IER = 313 implies ALP ≤ -1. Other values may be set by subroutine GAUSRC. WK is a real array of length N $\times$ (N + 2) + 3(N + 1) used as scratch space. There is no error check on the output and the accuracy can be tested by integrating the functions $e^{-x}x^{j+\alpha}$ for $j =$

$0, 1, \ldots, 2\mathrm{N}-1$, using the calculated weights and abscissas. For ALP $= 0$ it will yield the standard Gauss-Laguerre quadrature formula. This routine requires subroutines GAUSRC and TQL2 and function GAMMA.

51. GAUHER Subroutine to calculate weights and abscissas of a Gauss-Hermite quadrature formula. This subroutine uses the recurrence relation for Hermite polynomials

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x), \tag{B.22}$$

to calculate the weights and abscissas using subroutine GAUSRC. Here N is the number of abscissas in the required Gaussian formula. The corresponding weight function is $w(x) = e^{-x^2}$ on the interval $(-\infty, \infty)$. Output parameters W and A are real arrays of length N, containing the weights and corresponding abscissas. IER is the error parameter. IER $= 0$ implies successful execution. Nonzero values may be set by subroutine GAUSRC. WK is a real array of length $\mathrm{N} \times (\mathrm{N}+2) + 3(\mathrm{N}+1)$ used as scratch space. There is no error check on the output and the accuracy can be tested by integrating the functions $e^{-x^2}x^j$ for $j = 0, 1, \ldots, 2\mathrm{N}-1$, using the calculated weights and abscissas. This routine requires subroutines GAUSRC and TQL2.

52. GAUSWT Subroutine to calculate weights and abscissas of a Gaussian quadrature formula, with arbitrary weight function. This subroutine requires the values of moments i.e., the integrals

$$\mathrm{FMOM}(m) = \int_a^b w(x)x^m \, dx\,, \qquad m = 0, 1, \ldots, 2\mathrm{N}-1. \tag{B.23}$$

Existence of such formulae should be ensured before trying to use this subroutine. If $w(x) > 0$ throughout the interval, then such formulae should exist. Here N is the number of abscissas in the required Gaussian formula. Output parameters W and AB are real arrays of length N, containing the weights and corresponding abscissas. N should be less than NPMAX $(= 65)$. FMOM is the name of the function routine which calculates the moments as defined above. FUNCTION FMOM(M) must be supplied by the user. QGAUS is a logical parameter which specifies the type of formula required. If QGAUS = .TRUE., then a Gaussian formula is found, while if QGAUS = .FALSE., then the subroutine calculates the weights for an interpolatory formula with given weight function. In this case, the array AB must contain the required abscissas. IER is the error parameter. If $\mathrm{N} \leq 0$ or $\mathrm{N} \geq$ NPMAX, then IER is set to 303 and no calculations are performed. IER $= 322$ implies that the subroutine GAUELM failed to find the coefficients of the required polynomial. IER $= 323$ implies that the subroutine POLYR failed to find zeros of the polynomial. Both these errors can occur only for QGAUS = .TRUE. and in both these cases, no further calculations are performed. Similarly, IER $= 324$ implies that subroutine GAUELM failed to calculate the weights. In this case, the abscissas are already calculated, but the values may not be reliable. IER $= 322$, 323, 324 will normally imply that either the Gaussian formula does not exist or the weights and abscissas cannot be calculated because of ill-conditioning. In latter case, improving the accuracy of arithmetic may help. Even if IER = 0, the results could be unreliable. In particular, the zeros of polynomial may be complex, but only the real part may be returned in AB. This failure can occur either because of ill-conditioning, or because the formula does not exist. This subroutine requires subroutines POLYR and LAGITR for solving the required polynomial, and subroutine GAUELM for solving a system of linear equations. The algorithm used here is rather ill-conditioned and should not be used for large N, unless very high precision arithmetic is being used. There is no error check on the output and the accuracy can be tested by integrating the functions $w(x)x^n$ for $n = 0, 1, \ldots, 2\mathrm{N}-1$, using the calculated weights and abscissas. It may be better to use this subroutine in double precision. If the recurrence relation for the corresponding orthogonal polynomial are known then GAUSRC should be used to calculate the weights and abscissas.

53. FILON Subroutine to evaluate integrals of the form

$$\int_a^b f(x)\sin kx\;dx \qquad \text{or} \qquad \int_a^b f(x)\cos kx\;dx\;, \tag{B.24}$$

using Filon's method. RI is an output parameter which will contain the value of the integral. XL and XU are the lower and upper limits for the integral. It is not essential to have XL < XU. RK is the coefficient k, multiplying x in the sine or cosine function. QSIN is a logical variable, QSIN = .TRUE. implies that the integrand involves $\sin kx$ while QSIN = .FALSE. implies that the oscillatory factor is $\cos kx$. AEPS and REPS specify the required absolute and relative accuracy. The calculations are terminated when the successive values differ by less than $\max(\mathrm{AEPS}, \mathrm{REPS} \times |\mathrm{RI}|)$. DIF is an output parameter which should contain an estimate of (absolute) error calculated by the subroutine. N is an output parameter containing the number of function evaluations used by the subroutine. IER is the error parameter. IER = 30 denotes that the subroutine has failed to converge to the specified accuracy. In this case, RI will contain the best estimate for the integral and DIF should contain the estimated error. The parameter NMAX in the subroutine could be increased if larger number of function evaluations are to be allowed. FUN is the name of the function routine to calculate the non-oscillatory part $f(x)$ in the integrand. FUNCTION FUN(X) must be supplied by the user. The parameter THC in the parameter statement is the critical value of θ, below which the Taylor series expansion is used to evaluate the functions $\alpha(\theta)$, $\beta(\theta)$ and $\gamma(\theta)$. THC should be of the order of $(100\hbar)^{1/6}$. If other values are used, the errors could be larger.

54. ADPINT Subroutine to compute integral of a function using adaptive integration based on Gauss-Kronrod rule. RINT is the output parameter containing the value of the integral. XL and XU are the lower and upper limits of the integral. It is not essential to have XL < XU. REPS and AEPS specify the required accuracy, while DIF is an output parameter giving the error as estimated by the subroutine. The subroutine will attempt to ensure that

$\mathrm{DIF} < \max(|\mathrm{RINT}| \times \mathrm{REPS}, \mathrm{AEPS})$. However, it is not always possible to ensure that DIF is less than the expected tolerance and in some cases even though the execution is successfully completed DIF may be somewhat larger than the required accuracy. NPT is an output parameter which will contain the number of function evaluations actually used by the subroutine. NMAX is the maximum number of function evaluations that user is prepared to allow. If $\mathrm{NMAX} \leq 0$, then NMAX will be set to a default value of MAXPT ($= 100000$). IER is the error parameter. IER $= 31$ denotes that the subroutine failed to converge to the specified accuracy on at least one subinterval. IER $= 32$ denotes that this failure occurred more than IFMAX ($= 5$) times. In this case the accuracy requirement is adjusted by effectively increasing AEPS within the routine. The value of parameter AEPS is not actually changed. This failure could occur if the function has a very strong singularity at some points, or if the accuracy requirement is too high. In the former case, increasing IPMAX in the subroutine may help, provided the singularity is at $x = 0$, since otherwise it is not possible to subdivide the interval if it spans two consecutive numbers in the machine representation. It may be better to shift the singularity to $x = 0$. IER $= 325$ denotes that the subroutine failed to achieve satisfactory accuracy in NMAX function evaluations. This situation may occur either because of singularity, or because very high accuracy is required. Increasing NMAX or REPS may help in such cases. In all cases, RI will contain the best estimate for integral and DIF should contain the estimated error. If the integrand has singularities, then this estimate may be unreliable. F is the name of the function routine used to calculate the value of the integrand. FUNCTION F(X) must be supplied by the user. This subroutine also needs the subroutine KRONRD (or GAUS16). This subroutine could be used for mildly singular functions or for functions which vary by several orders of magnitude over the interval of integration. However, if very large interval is given and the function is almost constant over most of the interval, spurious convergence may take place as the subroutine may fail to detect the variation in function value. In such cases integration should be calculated over smaller subintervals and sum can be calculated to get the total integral.

55. KRONRD Subroutine to compute integral of a function using Gauss-Kronrod formula. RI is the output parameter containing the value of the integral. A and B are the lower and upper limits of the integral. It is not essential to have A $<$ B. DIF is an output parameter giving the error, as estimated by the subroutine. The subroutine uses 7-point Gauss-Legendre formula and its 15-point Kronrod extension to calculate the integral. RI will contain the estimate using 15-point Kronrod formula, while DIF contains the magnitude of the difference between the two values. N is an output parameter which will contain the number of function evaluations actually used by the subroutine. F is the name of the function routine to calculate the integrand. FUNCTION F(X) must be supplied by the user. This subroutine is used by ADPINT.

56. GAUS16 Subroutine to compute integral of a function using 16 point Gauss-Legendre formula. This subroutine can be used instead of KRONRD with ADPINT for adaptive integration. It calculates the integral using 8 and 16 point Gaussian formula and the difference between the two estimates is considered as estimated error. This routine will be less efficient as compared to KRONRD as it requires 24 function evaluations as opposed to 15 required by KRONRD. Although, the 16 point Gauss-Legendre formula will have higher accuracy as compared to 15 point Gauss-Kronrod formula, the error estimate is essentially the error in 8-point Gauss-Legendre formula and it will not be much different from that obtained by KRONRD. Since the adaptive choice of subintervals is determined by the estimated error, it will be almost same for both these routines. Thus in general KRONRD should be preferred but this routine is provided as an alternative. RI is the output parameter containing the value of the integral obtained using 16 point formula. A and B are the lower and upper limits of the integral. It is not essential to have A < B. DIF is an output parameter giving the error, as estimated by the subroutine. The subroutine uses 8 and 16-point Gauss-Legendre formulae to calculate the integral. RI will contain the estimate using 16-point Gaussian formula, while DIF contains the magnitude of the difference between the two values. N is an output parameter which will contain the number of function evaluations actually used by the subroutine. F is the name of the function routine to calculate the integrand. FUNCTION F(X) must be supplied by the user. This subroutine is used by ADPINT.

57. CAUCHY Subroutine to compute the Cauchy principal value of an integral. It uses ADPINT to perform the integral as explained in Section 6.6.10. RI is the output parameter containing the value of the integral. A and B are the lower and upper limits of the integral. It is essential to have A < B. C is the point inside the interval [A, B] where the integrand is singular. REPS and AEPS specify the required accuracy, while DIF is an output parameter giving the error as estimated by the subroutine. The subroutine will attempt to ensure that $\mathrm{DIF} < \max(|\mathrm{RINT}| \times \mathrm{REPS}, \mathrm{AEPS})$. F is the name of the function routine used to calculate the value of the integrand. FUNP is the name of the function routine used to calculate $\mathrm{F(C+X)+F(C-X)}$. IER is the error parameter. IER = 304 denotes that A > B, A > C or C > B. In this case no calculations are done. Other values of IER may be set by subroutine ADPINT, which is called twice to perform the integration. The value of IER is set to IER1+2IER2, where IER1 and IER2 are the values of IER returned by two calls to ADPINT. In these cases DIF will contain the estimated error in the computed value of RI. NPT is an output parameter which will contain the number of function evaluations actually used by the subroutine. This subroutine needs subroutines ADPINT and KRONRD. FUNCTION F(X) and FUNCTION FUNP(X) must be supplied by the user. The value of C is available through common block CAUFN. There is no provision to pass on the name of the function F(X) to FUNP(X) and if needed it will have to be put explicitly. If $\mathrm{F(C+X)+F(C-X)}$ can be simplified to remove singularity, the roundoff errors will be reduced.

58. EULER Subroutine to compute the sum of an alternating series using Euler's transformation.

$$\mathrm{SUM} = \sum_{i=1}^{\mathrm{N}} \mathrm{A0} \times (-1)^{i-1} \mathrm{TERM}(i). \tag{B.25}$$

The series may be finite or infinite. Before applying the Euler transformation, it is advisable to sum the first few terms (and last few terms, if the series is finite) separately as it helps in improving the convergence. This subroutine sums the first M1 terms and the last M2 terms separately and applies the Euler transform to the remaining terms from $\mathrm{M1} + 1$ to $\mathrm{N} - \mathrm{M2}$. The differences are calculated until the sum of each part converges to the specified accuracy. For finite series the contribution from the upper end is calculated separately. At most NMAX differences are calculated. If the sum does not converge then another attempt may be made by increasing M1 or M2. Although, NMAX can be trivially increased, but if the convergence is slow the error estimate will be unreliable and it will be better to improve convergence, which is generally achieved by increasing M1 or M2. Increasing NMAX may not help, as in most cases the roundoff error will dominate in higher order differences. N is the number of terms to be summed, while M1, M2 are the number of terms at the two ends which need to be summed separately. For summing an infinite series use $\mathrm{N} = 0$, in which case M2 is ignored. A0 specifies the sign of the first term. The sum assuming the first term to be positive is multiplied by A0. REPS and AEPS specify the required relative and absolute accuracy. The estimated error should be less than $\max(\mathrm{AEPS}, \mathrm{REPS} \times |\mathrm{SUM}|)$. DIF is the output parameter which gives the estimated error in the computed value of the sum. In most cases, actual error will be somewhat larger than DIF and can be estimated by repeating the calculations with different values of M1 and M2. N1, N2 are the number of terms from both ends actually used by the subroutine. M2 and N2 are relevant only for finite series. SUM is the calculated value of the sum. IER is the error parameter. IER = 31 implies that $\mathrm{M1} + \mathrm{M2} + 2\mathrm{NMAX} \geq \mathrm{N}$, in which case M1 is set to N and the series is summed directly. In this case although Euler transform is not used the sum should be exact apart from roundoff errors. IER = 32 implies that the series obtained after Euler transform did not converge to required accuracy at the lower end. IER = 34 implies that the series obtained after Euler transform did not converge to required accuracy at the upper end. This is relevant only for finite series. IER = 36 implies that the series obtained after Euler transform did not converge to required accuracy at both ends. TERM is the name of the function routine to calculate the terms of the series. FUNCTION TERM(I) must be supplied by the user to calculate the Ith term of the series apart from the sign as in Eq. (B.25).

59. BSPQD2 Function routine to compute integral of an expansion in terms of B-spline basis functions in two dimensions over a rectangular region. The expansion may be obtained by interpolating or approximating a table of values. Before calling this subroutine, the coefficients of B-spline expansion must be calculated by a call to the subroutine BSPINT2.

$$\mathrm{BSPQD2} = \int_{\mathrm{XL}}^{\mathrm{XU}} dx \int_{\mathrm{YL}}^{\mathrm{YU}} dy \quad \sum_{i=1}^{\mathrm{NX+K-2}} \quad \sum_{j=1}^{\mathrm{NY+K-2}} \mathrm{WT(i,j)}\ \phi_i(x)\psi_j(y), \tag{B.26}$$

where $\phi_i(x)$ are the B-spline basis functions of order K along x, and $\psi_j(y)$ are those along y. The integral is evaluated recursively. First the integral along x is evaluated for each value of j and then the integral over y is calculated. It may be noted that the function BSPQD to evaluate the integral in one dimension is not called recursively and hence one copy will be sufficient for both directions. NX, NY are the number of knots along x and y respectively. X and Y are real arrays of length NX and NY containing the knots for B-splines. The knots must be in ascending order with X(1), Y(1) containing the first knot in respective directions. K is the order of B-splines, K $=2$ gives linear B-splines, while K $=4$ yields cubic B-splines, etc. WT is a real array of length IW $\times$ (NY $+$ K $-$ 2) containing the coefficients of expansion in terms of B-spline basis functions. These coefficients may be calculated using BSPINT2 for interpolation in a table of values. IW is the first dimension of array WT as declared in the calling program. XL and XU are the lower and upper limits of the integral along x. YL and YU are the lower and upper limits of the integral along y. WK is a real array of length at least $6\max(\mathrm{NX},\mathrm{NY}) + 6\mathrm{K} + 9$ used as scratch space. IER is the error parameter. IER $=0$ implies successful execution of the subroutine. Nonzero values may be set by BSPQD which is called to perform integration in one dimension. This subroutine requires function BSPLIN to evaluate the B-spline basis functions, and function BSPQD for integration in one dimension, while subroutine BSPINT2 will be required to calculate the coefficients of B-spline before calling this subroutine.

60. BSPQDN Function routine to compute integral of an expansion in terms of B-spline basis functions in n dimensions over a hyper-rectangular region. The expansion may be obtained by interpolating or approximating a table of values. Before calling this subroutine, the coefficients of B-spline expansion must be calculated by a call to the subroutine BSPINTN.

$$\begin{aligned}\mathrm{BSPQDN} = \int_{\mathrm{XL(1)}}^{\mathrm{XU(1)}} dx_1 \cdots \int_{\mathrm{XL(N)}}^{\mathrm{XU(N)}} dx_N \sum_{i_1=1}^{\mathrm{NK(1)+K-2}} \cdots \sum_{i_N=1}^{\mathrm{NK(N)+K-2}} \\ \mathrm{WT(i_1,\ldots,i_N)}\ \phi_{i_1}^{(1)}(x_1)\cdots\phi_{i_N}^{(N)}(x_N)\end{aligned} \tag{B.27}$$

where $\phi_i^{(j)}(x)$ are the B-spline basis functions of order K along x_j. The integral is evaluated recursively. First the integral along x_1 is evaluated for each value of other indices and then the integral over x_2 is calculated and so on. It may be noted that the function BSPQD to evaluate the integral in one dimension is not called recursively and hence one copy will be sufficient for all directions. N is the number of dimensions. NK is an integer array of length N containing the number of knots along each direction. NK(I) is the number of knots along Ith dimension. X is a real array of length NXD $\times$ N containing the knots for B-splines along each dimension. The knots must be in ascending order. X(I, J) is the Ith knot along Jth dimension. It may be noted that the subroutine BSPINTN gives the list of knots in the same format and that array can be directly used in this subroutine. NXD is the first dimension of the array X as specified in the calling program, NXD $\geq \max(\mathrm{NK}(i))$. K is the order of B-splines, K $= 2$ gives linear B-splines, while K $= 4$ yields cubic B-splines, etc. WT is a real array of length $(\mathrm{NK}(1)+\mathrm{K}-2)(\mathrm{NK}(2)+\mathrm{K}-2)\cdots(\mathrm{NK}(\mathrm{N})+\mathrm{K}-2)$ containing the coefficients of expansion in terms of B-spline basis functions. These coefficients may be calculated using BSPINTN for interpolation in a table of values. The coefficients are assumed to be stored in natural Fortran order with no gaps in data. In the calling program the array should have dimension WT(NK(1)+K−2, NK(2)+K−2, ..., NK(N)+K−2). Alternately, it can be treated as a one dimensional array of required length. XL and XU are real arrays of length N containing the lower and upper limits of the integral along each direction. XL(I) is the lower limit and XU(I) is the upper limit for integration along Ith dimension. WK is a real array of length about twice that of WT used as scratch space. The actual length of WK used in the routine is a bit complicated and is given by

$$\prod_{i=2}^{N}(\mathrm{NK(i)}+\mathrm{K}-2)+\prod_{i=3}^{N}(\mathrm{NK(i)}+\mathrm{K}-2)+15+6\mathrm{K}+5\times\max(\mathrm{NK}(i)). \quad \text{(B.28)}$$

For N > 2 this should always be less than twice the length of WT in practical problems. IER is the error parameter. IER $= 0$ implies successful execution of the subroutine. Nonzero values may be set by BSPQD which is called to perform integration in one dimension. This routine requires function BSPLIN to evaluate the B-spline basis functions, and function BSPQD for integration in one dimension, while subroutine BSPINTN will be required to calculate the coefficients of B-splines before calling this subroutine.

61. MULINT Subroutine for integration over a hyper-rectangle in N dimensions using a series of product Gauss-Legendre formulae. A and B are real arrays of length N, with A(I) and B(I) specifying the lower and upper limits for the Ith variable. It is not essential to have A(I) $<$ B(I). N is an integer variable specifying the number of dimensions. There is no limit on the value of N as far as this subroutine is concerned, but the subroutine NGAUSS which is invoked to actually evaluate the integrals will have some limit. If this limit is exceeded, IER will be set to 305. M is an integer array of length N, specifying which Gaussian formula is to be used along each axis in the first approximation. The subroutine will attempt to use M(I)-point Gauss-Legendre formula along the Ith axis. Formulae with 2, 4, 8, 16 and 32 points are incorporated in the subroutine. If any other value is specified for M(I), then M(I) will be set to the default value of 2. IND is an integer array of length N, specifying the number of subdivisions to be used for the first approximation along each axis. If IND(I) < 1, then IND(I) is set to a default value of 1. After execution, the arrays M and IND will contain the final values of the formula and number of subintervals used along each axis. M[I] and IND[I] are increased to achieve the required accuracy or until the maximum number of function evaluations is reached. F is the name of the function routine to calculate the value of integrand at any given point. FUNCTION F(N, X) must be supplied by the user. (Here the first argument specifies the number of dimensions, while the second argument is a real array of length N, specifying the coordinates of the point at which the integrand is to be evaluated.) RINT is the output parameter which will contain the value of the integral. REPS and AEPS specify the accuracy required by user while DIF is an output parameter containing the estimated (absolute) error in RINT. The subroutine will attempt to ensure that $\mathrm{DIF} < \max(|\mathrm{RINT}| \times \mathrm{REPS}, \mathrm{AEPS})$. NUM is an output parameter containing the number of function evaluations actually used by the subroutine. MAXPT is the maximum limit on the number of function evaluations which the user is prepared for. If MAXPT < 1, then it is set to a default value of MAXPTS ($= 1100000$). The value of MAXPT will depend on the computer time that is available and the accuracy required. IER is the error parameter. If the integral failed to converge to the required accuracy in MAXPT function evaluations, IER will be set to 39. In this case, RINT will contain the best approximation to the integral and DIF will contain the error estimate. The error estimate may not be very reliable as the number of points may not be sufficient to check for convergence along each dimension. If a second attempt is to be made with larger MAXPT, then the values of M(I) and IND(I) should not be reset, since that will save some function evaluations in computing initial approximations, which are known to be unsatisfactory. IER $= 305$ implies that N is beyond the permissible limits. IER $= 307$ implies that the number of points exceeded MAXPT in the first attempt itself. In such cases, no approximation for the integral will be available. This subroutine requires subroutine NGAUSS.

62. NGAUSS Subroutine for integration over a hyper-rectangle in N dimensions using a product Gauss-Legendre formula. A and B are real arrays of length N, with A(I) and B(I) specifying the lower and upper limits for the Ith variable. It is not essential to have $\mathrm{A(I)} < \mathrm{B(I)}$. N is an integer variable specifying the number of dimensions. The value of N must be between 1 and 20. If N is outside these limits, IER will be set to 305. It is trivial to increase the upper limit on N by increasing NMAX but it is unlikely that subroutine will succeed for such dimensions. In any case MAXPT will also have to be increased correspondingly. M is an integer array of length N, specifying the Gaussian formula to be used along each axis. The subroutine attempts to use M(I)-point Gauss-Legendre formula along the Ith axis. Formulae with 2, 4, 8, 16 and 32 points are incorporated in the subroutine. If any other value is specified for M(I), then IER will be set to 306 and no calculations will be attempted. IND is an integer array of length N, specifying the number of subdivisions to be used along each axis. If $\mathrm{IND(I)} < 1$, then IER will be set to 306 and no calculations will be performed. F is the name of the function routine to calculate the value of integrand at any given point. The FUNCTION F(N, X) must be supplied by the user. (Here the first argument specifies the number of dimensions, while the second argument is a real array of length N specifying the coordinates of the point at which the integrand is to be evaluated.) RI is the output parameter which will contain the value of the integral. NUM is an output parameter containing the number of function evaluations actually used by the subroutine. MAXPT is the maximum limit on the number of function evaluations which the user is prepared for. If MAXPT < 1, then it is set to a default value of MAXPTS ($= 1100000$). The value of MAXPT will depend on the computer time that is available and the accuracy required. IER is the error parameter. IER $= 305$ implies that N < 1 or N $\geq$ NMAX, IER $= 306$ implies M(J) is not admissible or IND(J) < 1 for some J, IER $= 307$ implies that the number of points exceeded MAXPT. In all these cases no calculations are performed. This routine is called by MULINT which adjusts the number of abscissas to be used along each dimension depending on the required accuracy.

63. SPHND Function routine to transform the coordinates from hyper-spherical to Cartesian in n-dimensions. It can be used for integration over hyper-spherical shell in n-dimensions using subroutine MULINT, STRINT, MCARLO or EQUIDS when the function is known in terms of Cartesian coordinates. For this purpose SPHND should be passed on as the name of the function routine to calculate the integrand. The limits A(I) and B(I) of integration along Ith dimension should be $[0, \pi]$ for I $= 2, \ldots,$ N $- 1$, $[0, 2\pi]$ for I $=$ N and $[r_l, r_u]$ for I $= 1$. Here r_l and r_u are the radial coordinates of the hyper-spherical shell over which integration is required. For integration over a hyper-sphere of radius R, $r_l = 0$ and $r_u = R$. This routine gives the transformation between hyper-spherical coordinates x_i and Cartesian coordinates y_i

$$\begin{aligned}
y_1 &= x_1 \cos(x_2),\\
y_2 &= x_1 \sin(x_2)\cos(x_3),\\
y_3 &= x_1 \sin(x_2)\sin(x_3)\cos(x_4),\\
&\ldots\\
y_{n-1} &= x_1 \sin(x_2)\sin(x_3)\cdots\sin(x_{n-1})\cos(x_n),\\
y_n &= x_1 \sin(x_2)\sin(x_3)\cdots\sin(x_{n-1})\sin(x_n).
\end{aligned} \tag{B.29}$$

The integral can be written as

$$\int_{S_n} f(x)\,dV = \int_{r_l}^{r_u} dx_1 \int_0^{\pi} dx_2 \cdots \int_0^{\pi} dx_{n-1} \int_0^{2\pi} dx_n f(x) x_1^{n-1} \prod_{i=2}^{n-1} \sin^{n-i} x_i\,. \tag{B.30}$$

The function $f(x)$ is calculated after transformation to Cartesian coordinates and the volume element as given above is multiplied. N is the number of dimensions, X and Y are real arrays of length N containing the coordinates of required point. X(I) are the input hyper-spherical coordinates, while Y(I) are the Cartesian coordinates. FUNCTION FUNSPH(N, Y) is called to calculate the required function in Cartesian coordinates. Since there is no provision to pass on the name of this function, it has to be the same as what appears in the routine. Again since there is no provision to pass on an error flag, the routine terminates the execution if $N > NMAX = 50$ or if $N \leq 0$.

64. STRINT Subroutine for integration over a hyper-rectangle in N dimensions using compound monomial rules of degree 1, 3 or 5. A and B are real arrays of length N, with A(I) and B(I) specifying the lower and upper limits for the Ith variable. It is not essential to have $A(I) < B(I)$. N is an integer variable specifying the number of dimensions. There is no limit on the value of N as far as this subroutine is concerned, but the subroutine STROUD which is invoked to actually evaluate the integrals will have some limit. If this limit is exceeded, IER will be set to 309. Although a large limit on N is allowed the result is unlikely to be reliable for very large values of N. M is an integer parameter specifying the formula to be used on each subdivision. The allowed values of M are 1, 3 and 5. $M = 1$ selects the one-point formula of degree one (essentially a generalisation of the midpoint rule). $M = 3$ selects the 2N-point formula of degree 3 due to Stroud. $M = 5$ selects the $(2N^2 + 1)$-point formula of degree 5. If M is not one of these values, it will be set to the default value of 3. IND is an integer array of length N, specifying the number of subdivisions to be used for the first approximation along each axis. If $IND(I) < 1$, then it will be set to a default value of 1. After execution, the array IND will contain the final values of the number of subintervals used along each axis. F is the name of the function routine to calculate the value of the integrand at any given point. FUNCTION F(N, X) must be supplied by the user. (Here the first argument specifies the number of dimensions, while the second argument is a real array of length N, specifying the coordinates of the point at which the integrand is to be evaluated.) RINT is the output parameter which will contain the value of the integral. REPS and AEPS specify the accuracy required by user, while DIF is an output parameter containing the estimated (absolute) error in RINT. The subroutine will attempt to ensure that $\mathrm{DIF} < \max(|\mathrm{RINT}| \times \mathrm{REPS}, \mathrm{AEPS})$. NUM is an output parameter containing the number of function evaluations actually used by the subroutine. MAXPT is the maximum limit on the number of function evaluations which the user is prepared for. If $MAXPT < 1$, then it is set to a default value of MAXPTS (= 1000000). The value of MAXPT will depend on the computer time that is available and the accuracy required. IER is the error parameter. If the integral fails to converge to the required accuracy in MAXPT function evaluations, IER will be set to 39. In this case, RINT will contain the best approximation to the integral and DIF will contain the error estimate. The error estimate may not be very reliable as the number of points may not be sufficient to check for convergence along each dimension. If a second attempt is to be made with larger MAXPT, then the values of IND(I) should not be reset, since that will save some function evaluations in computing initial approximations, which are known to be unsatisfactory. IER = 308 implies that the number of points exceeded MAXPT in the first attempt itself. In such cases, no approximation for the integral will be available. IER = 309 implies that N is outside the limits accepted by STROUD. This subroutine requires subroutine STROUD.

65. STROUD Subroutine for integration over a hyper-rectangle in N dimensions using a compound monomial rule of degree 1, 3 or 5. A and B are real arrays of length N, with A(I) and B(I) specifying the lower and upper limits for the Ith variable. It is not essential to have A(I) $<$ B(I). N is an integer variable specifying the number of dimensions. The value of N must be between 1 and 50. If N is outside these limits, IER will be set to 309. M is an integer specifying the formula to be used on each subdivision. The allowed values of M are 1, 3 and 5. M $=$ 1 selects the one-point formula of degree one (essentially a generalisation of the midpoint rule). M $=$ 3 selects the 2N-point formula of degree 3 due to Stroud. M $=$ 5 selects the $(2\mathrm{N}^2 + 1)$-point formula of degree 5. If any other value is specified for M, then IER will be set to 310 and no calculations will be attempted. IND is an integer array of length N, specifying the number of subdivisions to be used along each axis. If IND(I) $<$ 1, then IER will be set to 310 and no calculations will be performed. F is the name of the function routine to calculate the value of integrand at any given point. The FUNCTION F(N, X) must be supplied by the user. (Here the first argument specifies the number of dimensions, while second argument is a real array of length N specifying the coordinates of the point at which the integrand is to be evaluated.) RI is the output parameter which will contain the value of the integral. NUM is an output parameter containing the number of function evaluations actually used by the subroutine. MAXPT is the maximum limit on the number of function evaluations which the user is prepared for. If MAXPT $<$ 1, then it is set to a default value of MAXPTS ($=$ 1000000). The value of MAXPT will depend on the computer time that is available and the accuracy required. IER is the error parameter. IER $=$ 308 implies that the number of points exceeded MAXPT. IER $=$ 309 implies that the N $<$ 1 or N $>$ NMAX. IER $=$ 310 implies that the M is not 1, 3 or 5 or IND(I) $<$ 1 for some I. In all these cases no calculations are done. This routine is called by STRINT which adjusts the number of abscissas to be used along each dimension depending on the required accuracy.

66. MCARLO Subroutine for integration over a hyper-rectangle in N dimensions using Monte Carlo method. A and B are real arrays of length N, with A(I) and B(I) specifying the lower and upper limits for the Ith variable. It is not essential to have A(I) $<$ B(I). N is an integer variable specifying the number of dimensions. The value of N must be between 1 and NMAX ($=$ 50). If N is outside these limits, IER will be set to 311. NPT is the maximum number of function evaluations to be used for integration. F is the name of the function routine to calculate the value of the integrand at any given point. The FUNCTION F(N, X) must be supplied by the user. (Here the first argument specifies the number of dimensions, while the second argument is a real array of length N specifying the coordinates of the point at which the integrand is to be evaluated.) RI is an output parameter, which will contain the value of the integral. REPS and AEPS specify the desired accuracy. ERR is an output parameter containing the (absolute) error estimate for the integral. This error estimate is obtained by using the variance σ as defined in Section 6.11. ERR is set to $2.576\sigma/\sqrt{\mathrm{NP}}$ corresponding to a confidence level of 99%. The subroutine attempts to ensure that ERR $< \max(|\mathrm{RI}| \times \mathrm{REPS}, \mathrm{AEPS})$. NP is the number of function evaluations actually used by the subroutine. IER is the error parameter. IER = 39 implies that the subroutine failed to converge to the specified accuracy. In this case, RI will contain the best estimate and ERR the corresponding error estimate. IER = 311 implies that N < 1 or N > 50 and no calculations are done. This subroutine requires FUNCTION RANF(ISEED) to generate random numbers. If a different routine is used for this purpose, then the value of seed may have to be changed.

67. RAN1 Function routine to generate a sequence of random numbers using a simple linear congruential method. The constants AM, A and AC will give overflow in integer arithmetic using 32-bits and hence these must be treated as double precision variables. SEED could be set to any positive value less than AM, before first call to the routine. After the first call, this variable should not be changed in any other program. It may be noted that this routine requires a real number as seed as opposed to RANF, which needs an integer seed as argument. Further, in this case the seed has to be positive, while for RANF it has to be negative for initialisation. This has to be kept in mind while changing the random number generator in any program.

68. RANF Function routine to generate a sequence of random numbers using a combination of three linear congruential generators. This subroutine is based on the subroutine RAN1 in Press *et al.* (2007). The constants m, a and c for each of these generators are chosen, such that there should be no overflow on a machine with 32-bit word length. ISEED could be set to any negative value, before first call to the routine. After the first call, this variable should not be changed in any other program.

69. EQUIDS Subroutine for integration over a hyper-rectangle in N dimensions using the method based on equidistributed sequences. A and B are real arrays of length N, with A(I) and B(I) specifying the lower and upper limits for the Ith variable. It is not essential to have A(I) < B(I). N is an integer variable specifying the number of dimensions. N should be between 1 and NMAX (= 21). If N is outside these limits, then IER will be set to 312. NPT is the maximum number of function evaluations to be used in computing the integral. F is the name of the function routine to calculate the value of the integrand at any given point. The FUNCTION F(N, X) must be supplied by the user. (Here the first argument specifies the number of dimensions, while the second argument is a real array of length N specifying the coordinates of the point at which the integrand is to be evaluated.) S1 and S2 are the output parameters which will contain the value of the integral. These values provide the two approximations using equidistributed sequences. If the function is sufficiently smooth, S2 is expected to be a better approximation and could in general be used as the best approximation to the integral. REPS and AEPS specify the accuracy required by the user, while DIF is an output parameter containing the estimated (absolute) error in S2. The subroutine attempts to ensure that DIF $< \max(|\mathrm{S2}| \times \mathrm{REPS}, \mathrm{AEPS})$. NP is an output parameter containing the number of function evaluations actually used by the subroutine. If the integral fails to converge to the required accuracy in NPT function evaluations, the error parameter IER will be set to 39. In this case, S2 (or S1) will contain the best approximation to the integral and DIF will contain the error estimate. Evaluation of the sum in EQUIDS involves large roundoff errors and it is recommended to use this routine with double precision. IER = 312 implies that N < 1 or N $>$ NMAX.

B.7 Nonlinear Algebraic Equations

70. BISECT Subroutine to find a real zero of a continuous function using the method of bisection. XL and XU bracket the interval containing the zero. The function must have opposite signs at these two points. This interval will be refined by bisection, and after execution XL and XU will contain the refined estimate for the interval containing the zero. It is not necessary to have XL $<$ XU. X is an output parameter which will contain the interpolated value of the zero, using the last estimate at the end points XL and XU. NB is an input parameter specifying the number of bisections to be performed. F is the name of the function routine to calculate the function. IER is the error parameter. IER = 401 implies that NB ≤ 0, in which case, no calculations are performed. IER = 421 implies that the function has the same sign at both the end points and hence bisection cannot be performed. IER = -1 implies that the function vanishes at one of the points. Hence, the required number of bisections have not been performed, but in this case, X will contain the "zero". Before invoking this subroutine, it must be ensured that the function is continuous in the interval. In particular, the sign change should not be due to a singularity within the interval. The number of bisections to be performed should be carefully chosen, since performing unnecessary bisections will not improve the accuracy of the computed zero. The FUNCTION F(X) must be supplied by the user.

71. SECANT Subroutine to calculate a real zero of a given function using the secant iteration. XL and XU specify the limits within which the zero is expected. It is essential to have XL $<$ XU i.e., XL is the lower limit and XU is the upper limit. It is not necessary that the function value should have opposite sign at these two points. The limits are only used to terminate the iteration, if it is going astray. X0 is the initial guess for the zero and it is essential to have XL $\leq$ X0 $\leq$ XU. X is an output parameter containing the computed value of the zero. REPS and AEPS specify the relative and absolute convergence criterion. The zero should be determined with an accuracy of $\max(\mathrm{AEPS}, \mathrm{REPS}|\mathrm{X}|)$. FUN is the name of the external routine to calculate the required function. FUNCTION FUN(X) must be supplied by the user. IER is the error parameter. IER = 40 implies that the calculated values of the function at the two most recent points was equal, and it is not possible to continue the iteration further. This situation can arise if the zero is multiple, or if the convergence criterion is too stringent, or sometimes just by coincidence. In the first two cases, the computed value of the zero may be reasonable. IER = 402 implies that the starting value was outside the given interval, in which case, no calculations are performed. IER = 422 implies that the iteration has gone outside the prescribed limits, and was terminated at that stage. This situation may arise, because the zero is outside or close to the specified limits. However, this is not essential and the iteration may tend to go outside for other reasons. IER = 423 implies that the iteration failed to converge to the specified accuracy. This failure could be due to various reasons, like very stringent convergence criterion, or excessive roundoff error in calculating the function, or the iteration simply failed to detect the zero. This subroutine attempts to make a reasonable guess for the second starting value, using the limiting interval and the convergence criterion. However, if the root is too large, or too small, or if the bounding interval is very large this choice may not be good. In such cases, the iteration may not converge. This problem can be rectified by choosing the proper increment DX at the first step, or by making the bounding interval smaller. If the function is calculated in the form FUN$\times 2^{\mathrm{JF}}$ to avoid overflows and underflows, then use the subroutine SECAN_2 instead of SECANT. The call statement is identical for the two routines.

72. SECANC Subroutine to calculate a complex zero of a given function using the secant iteration. This is the complex version of SECANT. In this case the root cannot be bracketed between limits and only the magnitude of root is used to check if it is within limits. The limit is only used to terminate the iteration, if it is going astray. X0 is the initial guess for the zero and it is essential to have $|\mathrm{X0}| < \mathrm{R}$. R is the limiting magnitude for the root. X is an output parameter containing the computed value of the zero. X and X0 are complex variables, while R is real. REPS and AEPS specify the relative and absolute convergence criterion. The zero should be determined with an accuracy of $\max(\mathrm{AEPS}, \mathrm{REPS}|\mathrm{X}|)$. FUN is the name of the external routine to calculate the required function. FUNCTION FUN(X) must be supplied by the user. IER is the error parameter. IER = 40 implies that the calculated values of the function at the two most recent points was equal, and it is not possible to continue the iteration further. This situation can arise if the zero is multiple, or if the convergence criterion is too stringent, or sometimes just by coincidence. In the first two cases, the computed value of the zero may be reasonable. IER = 402 implies that the starting value was outside the specified interval, in which case, no calculations are performed. IER = 422 implies that the iteration has gone outside the prescribed limits, and was terminated at that stage. This situation may arise, because the zero is outside or close to the specified limits. However, this is not essential and the iteration may tend to go outside for other reasons. IER = 423 implies that the iteration failed to converge to the specified accuracy. This failure could be due to various reasons, like very stringent convergence criterion, or excessive roundoff error in calculating the function, or the iteration simply failed to detect the zero. This subroutine attempts to make a reasonable guess for the second starting value.

However, if the roots are close then this value may not be appropriate. In such cases, the iteration may not converge. This problem can be rectified by choosing the proper increment DX at the first step. If the function is calculated in the form FUN $\times 2^{\mathrm{JF}}$ to avoid overflows and underflows, then use the subroutine SECANC_2 instead of SECANC. The call statement is identical for the two routines.

73. SECANI Subroutine to calculate a real zero of a given function using the secant iteration. The function is calculated in the form $\mathrm{F}\times 2^{\mathrm{JF}}$ to avoid overflows and underflows. It also uses the reverse communication technique for passing function values. This subroutine will return control to the calling program when it needs a function evaluation. In that case, the error parameter IER will be set to a negative value and the function should be evaluated at X. The value of the function should be returned in the variables F and JF. During the function evaluation variables other than F and JF in the call statement should not be disturbed. Before the first call, IER should be set to zero. IER ≥ 0 implies that the execution is complete and no more function evaluations are required. In that case the value of X will give the root (if IER $= 0$) or the final value of X where the function was evaluated. XL and XU specify the limits within which the zero is expected. It is essential to have XL $<$ XU i.e., XL is the lower limit and XU is the upper limit. It is not necessary that the function value should have opposite sign at these two points. The limits are only used to terminate the iteration, if it is going astray. X0 is the initial guess for the zero and it is essential to have XL $\leq$ X0 $\leq$ XU. X is an output parameter containing the value of x at which the function value is required, when IER < 0. For other values of IER it will contain the computed root or a failed approximation to it. The calculated function value is to be passed through the variables F and JF. REPS and AEPS specify the relative and absolute convergence criterion. The zero should be determined with an accuracy of $\max(\mathrm{AEPS}, \mathrm{REPS}|\mathrm{X}|)$. IER is the error parameter. If IER < 0, then a fresh function evaluation is required. IER $= 0$ implies that function execution has been successfully completed and X should contain the calculated root. IER $= 40$ implies that the calculated values of the function at the two most recent points was equal, and it is not possible to continue the iteration further. This situation can arise if the zero is multiple, or if the convergence criterion is too stringent, or sometimes just by coincidence. In the first two cases, the computed value of the zero may be reasonable. IER $= 402$ implies that the starting value was outside the given interval, in which case, no calculations are performed. IER $= 422$ implies that the iteration has gone outside the prescribed limits, and was terminated at that stage. This situation may arise, because the zero is outside or close to the specified limits. However, this is not essential and the iteration may tend to go outside for other reasons. IER $= 423$ implies that the iteration failed to converge to the specified accuracy. This failure could be due to various reasons, like very stringent convergence criterion, or excessive roundoff error in calculating the function, or the iteration simply failed to detect the zero. This subroutine attempts to make a reasonable guess for the second starting value, using the limiting interval and the convergence criterion. However, if the root is too large, or too small, or if the bounding interval is very large this choice may not be good. In such cases, the iteration may not converge. This problem can be rectified by choosing the proper increment DX at the first step, or by making the bounding interval smaller.

74. NEWRAP Subroutine to calculate a real zero of a given function using the Newton-Raphson method. This subroutine tries to estimate the multiplicity of the zero by looking at the convergence rate and then uses this estimate to accelerate the convergence. XL and XU specify the limits within which the zero is expected. It is essential to have XL $<$ XU i.e., XL is the lower limit and XU is the upper limit. It is not necessary that the function should have opposite signs at these two points. The limits are only used to terminate the iteration, if it is going astray. X0 is the initial guess for the zero and it is essential to have XL $\leq$ X0 $\leq$ XU. X is an output parameter containing the computed value of the zero. REPS and AEPS specify the relative and absolute convergence criterion. The zero should be calculated with an accuracy of $\max(\mathrm{AEPS}, \mathrm{REPS}|\mathrm{X}|)$. FUN is the name of the external routine to calculate the required function and its derivative. FUNCTION FUN(X, DF) must be supplied by the user, where DF is the first derivative of FUN at X. IER is the error parameter. IER = 403 implies that the starting value was outside the given range, in which case, no calculations are performed. IER = 424 implies that the iteration has gone outside the prescribed limits, and was terminated at that stage. This situation may arise, because the zero is outside or close to the specified limits. However, this is not essential and iteration may tend to go outside for other reasons. IER = 425 implies that the iteration failed to converge to the specified accuracy. This failure could be due to various reasons, like very stringent convergence criterion, or excessive roundoff error in calculating the function, or the iteration simply failed to detect the zero. IER = 426 implies that the calculated value of the derivative at the last point is zero, and it is not possible to continue the iteration further. In this case the zero is detected to be simple. This can happen by coincidence. This situation can happen if the zero is multiple, in which case IER is set to $126 - 100 \times$ multiplicity. In this case, the computed value of the zero may be reasonable. On successful completion of the subroutine, IER will be zero if the root is simple and IER = $-k$, if the root is detected to be multiple with multiplicity k. Function FUN(X, DF) must be supplied by the user. NEWRAC is the complex version of this subroutine.

75. BRENT Subroutine to calculate a real zero of a given function using the Brent's method. This subroutine is based on the routine given by Brent (1973). A and B specify the limits within which the zero is located. It is essential that the function has opposite signs at these two points. The values of A and B will be updated by the subroutine to locate the zero with required accuracy. X is the output parameter containing the computed value of the zero. REPS and AEPS specify the relative and absolute convergence criterion. The zero should be calculated with an accuracy of $\max(\mathrm{AEPS}, \mathrm{REPS}|\mathrm{X}|)$. F is the name of the external routine to calculate the required function. FUNCTION F(X) must be supplied by the user. IER is the error parameter. IER = 427 implies that the function has the same sign at both the end points and hence the Brent's method cannot be applied. IER = 428 implies that the iteration failed to converge to the specified accuracy. This failure could be due to the fact that the convergence criterion is too stringent. Increasing the parameter NIT in the subroutine may allow convergence, but before doing that it must be ensured that it will be meaningful to do so. Since convergence to any arbitrary accuracy does not ensure that the zero is correct to that accuracy. If the accuracy requirement cannot be satisfied within the available floating-point arithmetic, then the iteration may never converge to the specified accuracy. Before invoking this subroutine, it must be ensured that the function is continuous in the interval. In particular, the sign change should not be due to a singularity within the interval.

76. SEARCH Subroutine to search for complex zeros by looking for sign changes in the real and imaginary parts of the function. It will output an array of values in the two-dimensional plane giving the quadrant value of the function at that point. Using this array the zero may be located as explained in Section 7.7. The zeros will be searched in a rectangular region bounded by RX1 and RX2 along the real axis and by RY1 and RY2 along the imaginary axis. It is not really essential to have RX1 < RX2 or RY1 < RY2. NX and NY are the number of points along the real and imaginary axes respectively, where the function value is calculated. NX and NY should be greater than 1, if not, a default value of 21 will be used. Further, NX $\leq$ IMAX (= 41), otherwise it will be reduced to the maximum permissible value of IMAX. CFUN is the name of the function routine to calculate the function of complex variable. FUNCTION CFUN(Z) must be supplied by the user. Here Z and CFUN are both complex variables. Output is written on the Fortran logical unit 6, which may be printed on a printer or displayed on the screen.

77. ZROOT Subroutine to calculate complex zeros of a given function using Muller's method. This subroutine uses deflation to remove the known zeros. Deflation is carried out by explicitly dividing the function by factors of the form $z - z_i$, where z_i are the known zeros. This subroutine calls MULLER (or MULER2) to find the zeros. N specifies the number of zeros to be determined. CX is a complex array of length N, containing the N starting values for the iteration. NZ is the number of known zeros of the given function. CZERO is a complex array of length NZ + N containing the zeros. At the time of calling, the first NZ elements should contain the known zeros of the function. All zeros found by the subroutine will be added to this list and the value of NZ will be increased accordingly. REPS and AEPS specify the relative and absolute convergence criterion. The zeros should be calculated with an accuracy of $\max(\mathrm{AEPS}, \mathrm{REPS}|\mathrm{X}|)$. IER is the error parameter. IER = 41 implies that the Muller's iteration did not converge to the required accuracy for at least one of the zeros, but the computed root may still be acceptable at lower accuracy.

This means that the subroutine MULLER has come out with a nonzero value of IER, which is less than 100. IER = 429 implies that the iteration failed to converge for at least one of the zeros. The number of zeros successfully found can be obtained from NZ, which will give the number of known zeros including those which were known before the subroutine was called. RMAX is a parameter to specify the approximate range of z values, where zeros are required. If the iteration goes outside the region $|Z| \leq$ RMAX, then it will be terminated. This parameter may be used to ensure that iteration does not stray into a region which will yield overflow or some other problem. CF is the name of the external routine to calculate the required function. FUNCTION CF(Z) (or FUNCTION CF(Z, IX) for MULER2) must be supplied by the user. Here both CF and Z are assumed to be of type COMPLEX. Apart from this function, the subroutine also requires subroutine MULLER or MULER2. To use MULER2 instead of MULLER, change the call statement as indicated in the comments. ZROOT2 is the version of ZROOT for use with MULER2.

78. ZROOT2 Subroutine to calculate complex zeros of a given function using Muller's method. This subroutine uses deflation to remove the known zeros. Deflation is carried out by explicitly dividing the function by factors of the form $z - z_i$, where z_i are the known zeros. This subroutine calls MULER2 (or MULLER) to find the zeros. N specifies the number of zeros to be determined. CX is a complex array of length N, containing the N starting values for the iteration. NZ is the number of known zeros of the given function. CZERO is a complex array of length NZ + N containing the zeros. At the time of calling, the first NZ elements should contain the known zeros of the function. All zeros found by the subroutine will be added to this list and the value of NZ will be increased accordingly. REPS and AEPS specify the relative and absolute convergence criterion. The zeros should be calculated with an accuracy of $\max(\text{AEPS}, \text{REPS}|X|)$. IER is the error parameter. IER = 41 implies that the Muller's iteration did not converge to the required accuracy for at least one of the zeros, but the computed root may still be acceptable at lower accuracy. This means that the subroutine MULER2 has come out with a nonzero value of IER, which is less than 100. IER = 429 implies that the iteration failed to converge for at least one of the zeros. The number of zeros successfully found can be obtained from NZ, which will give the number of known zeros including those which were known before the subroutine was called. RMAX is a parameter to specify the approximate range of z values, where zeros are required. If the iteration goes outside the region $|Z| \leq$ RMAX, then it will be terminated. This parameter may be used to ensure that iteration does not stray into a region which will yield overflow or some other problem. CF is the name of the external routine to calculate the required function. FUNCTION CF(Z, IX) (or FUNCTION CF(Z) for MULLER) must be supplied by the user. Here both CF and Z are assumed to be of type COMPLEX and IX is an integer variable such that the function value is given by $f(\text{Z}) = \text{CF} \times 2^{\text{IX}}$. This form is useful for functions which will otherwise result in overflow or underflow. Apart from this function, the subroutine also requires subroutine MULER2 or

MULLER. Subroutine MULER2 uses *reverse communication* technique to pass the function value. This subroutine returns the control to the calling program whenever a function evaluation is required. This technique is useful when a large number of parameters, including some external subprogram names, are required for the function evaluation. At the first call to MULER2 the error parameter IER should be set to zero. When the control is returned to the calling program, it should check the value of IER to decide the action. If IER < 0, then a function evaluation is required and the subroutine should be called once again after calculating the function. It is important to ensure that none of the other parameters in the call statements, e.g., CX1, CX2, CX3 and IER are changed in between. Nonnegative values of IER signify that the execution of MULER2 is complete and it should not be called again. ZROOT is the version of ZROOT2 for use with MULLER.

79. MULLER Subroutine to calculate a complex zero of a given function using Muller's method. CX1, CX2 and CX3 are complex variables specifying the three starting values required for Muller's method. After execution, CX3 will contain the computed value of the zero, while CX2 and CX1 will contain the previous iterates. REPS and AEPS specify the relative and absolute convergence criterion. The root should be calculated with an accuracy of $\max(\mathrm{AEPS}, \mathrm{REPS}|\mathrm{CX3}|)$. IER is the error parameter. IER $= 42$ implies that the iteration converged to a moderate accuracy specified by the parameter REPS0 ($= 10^{-4}$), but after some stage the difference between successive iterates started increasing and the iteration was terminated. This will usually imply that specified accuracy is too stringent, or that there is a significant roundoff error in evaluating the function. In such cases, the computed value of the zero should be within the domain of indeterminacy for the function and may be acceptable. The difference $|\mathrm{CX2} - \mathrm{CX3}|$ should give an estimate of accuracy achieved. IER $= 43$ implies that the moderate convergence criterion was satisfied, but the iteration failed to converge to the required accuracy in specified maximum number of iterations NIT ($= 50$). This failure could be due to slow convergence and increasing the value of NIT, or a second attempt with the new estimate as the starting value may yield a better result. Further, the difference between CX1, CX2 and CX3 will give an estimate of the expected error in the computed zero. IER $= 404$ implies that the three starting values are not distinct, in which case no calculations are performed. IER $= 431$ implies that iteration has gone outside the specified limits i.e., $|\mathrm{CX3}| > \mathrm{RMAX}$. IER $= 432$ implies that the iteration has failed to converge to any reasonable accuracy. IER $= 433$ implies that the iteration cannot be continued further, because the denominator in the iteration function vanishes. In this case, CX1, CX2 and CX3 will contain the last three iterates. This failure can occur at multiple zeros after the iteration has entered into the domain of indeterminacy, in which case, the root may be acceptable. CF is the name of the external routine to calculate the required function. FUNCTION CF(Z) must be supplied by the user. Here CF and Z are both complex variables. NZ is the number of known zeros of the function. CZERO is a complex array of length NZ containing the known zeros. RMAX is a parameter to specify the approximate range of z values, where zeros are required. If the iteration gives $|Z| >$ RMAX, then it will be terminated. This parameter may be used to ensure that the iteration does not stray into a region which will yield overflows or other problems.

80. MULER2 Subroutine to calculate a complex zero of a given function using Muller's method. This subroutine is essentially identical to the subroutine MULLER described earlier, except for the fact that here the given function is calculated in the form $f(\mathrm{CX}) = \mathrm{CF} \times 2^{\mathrm{IX}}$, which is suitable for those functions which will normally overflow or underflow on the computer. For example, this subroutine is very useful to calculate the zeros of determinants arising out of finite difference approximation to differential equations. Apart from this, it uses the reverse communication technique for passing function values. This subroutine will return control to the calling program when it needs a function evaluation. In that case, the error parameter IER will be set to a negative value and the function should be evaluated at $z =$ CX. The value of the function should be returned in the variables CF and IX. Here CX and CF are complex variables while IX is an integer. The function value is expressed in the form explained above. During the function evaluation variables other than CF and IX in the call statement should not be disturbed. Before the first call IER should be set to zero. IER ≥ 0 imply that the execution is complete and no more function evaluations are required. Apart from CF, CX and IX other arguments as well as error exits are identical to those for MULLER. For an example of the usage of this subroutine see subroutine ZROOT2 above.

81. DELVES Subroutine to calculate complex zeros of an analytic function inside a given circle in the complex plane. This subroutine uses the quadrature based method described in Section 7.9. This method also requires the first derivative of the function. CZ is a complex variable specifying the centre of the required circle in the complex plane. RAD is a real variable specifying the radius of this circle. The roots inside a circle with centre at CZ and radius RAD should be calculated. CF is the name of the function routine used to calculate the given function and its derivative. FUNCTION CF(Z, CDF) must be supplied by the user, where CF, CDF and Z are complex variables and CDF is the first derivative of CF at Z. NZ is an output parameter containing the number of roots located by the subroutine. CZERO is a complex array of length NMAX$(= 5)$, which will contain the zeros located by the subroutine. Since this subroutine will not attempt to determine the zeros, unless the number is less than or equal to NMAX, the size of the array CZERO need not be more than NMAX. IER is the error parameter. IER $= 405$ implies that the radius RAD is zero or negative, in which case, no calculations are performed. IER $= 434$ implies that the number of zeros inside the circle exceeds NMAX and no further calculations are performed. In this case, NZ will contain the estimated number of zeros inside the circle. To find the zeros in this case, a second attempt may be made with smaller circles covering the original area. IER $= 435$ implies that the Newton-Raphson iteration for refining the calculated zero did not converge for at least one zero. In such cases, again a second attempt with smaller circle may help. These problems may be due to the fact that the contour is very close to one of the zero, in which case changing the radius may help. IER = 436 implies that the subroutine POLYC fails to find the roots of the required polynomial. In this case, attempt will be made using Newton-Raphson method on whatever values are available. Other values of IER may be set by the subroutine CONTUR, which is called to evaluate the contour integrals. REPS and AEPS specify the relative and absolute convergence criterion. The zeros should be calculated with an accuracy of $\max(\mathrm{AEPS}, \mathrm{REPS}|\mathrm{X}|)$. This subroutine requires subroutine CONTUR to calculate the contour integrals, subroutines POLYC and LAGITC to calculate the zeros of resulting polynomial, and subroutine NEWRAC to refine the calculated zeros by iterating on the original function. In addition of course, the function routine CF will be required. Before using this subroutine, it is essential to ensure that the required function is analytic everywhere inside the given circle.

82. CONTUR Subroutine to evaluate the contour integrals over a circle, as required by the subroutine DELVES for finding complex zeros of an analytic function. CF is the name of the function routine used to calculate the given function. FUNCTION CF(Z, CDF) must be supplied by the user. Here CF, CDF and Z are complex variables and CDF is the first derivative of CF at Z. CZ is a complex variable specifying the centre of the required circle in the complex plane. RAD is a real variable specifying the radius of this circle. NZ is an output parameter, containing the number of zeros inside a circle with centre at CZ and radius RAD. CS is a complex array of length NMAX, which will contain the calculated values of the contour integrals

$$\mathrm{CS(K)} = \frac{1}{2\pi i}\oint \frac{z^{\mathrm{K}} f(z)\,dz}{f'(z)} = \sum_{j=1}^{\mathrm{NZ}} z_j^{\mathrm{K}}, \qquad (\mathrm{K} = 1, 2, \ldots, \mathrm{NZ}), \tag{B.31}$$

where z_j are the zeros inside the given circle. NMAX is the maximum number of zeros to be located. IER is the error parameter. IER = 437 implies that the calculation of the first integral giving the number of roots within the given circle has not converged satisfactorily. This failure could be due to the fact that the circle is too big, or the boundary is too close to one of the zeros. A second attempt with a smaller circle or after shifting the original circle appropriately may help in this case. IER = 434 implies that the number of zeros inside the circle exceeds NMAX and no further calculations are performed. In this case, NZ will contain the estimated number of zeros inside the circle. To find the zeros in this case, a second attempt may be made with smaller circles covering the original area. IER = 44 implies that the iteration to estimate the first integral did not converge to satisfactory accuracy, but the value is less than 0.5 in magnitude. In this case, most probably there is no root inside the given circle and NZ is set to zero. IER = 45 implies that the integral did not converge to a satisfactory accuracy, but the value of NZ may still be reasonable. In this case, the calculations are continued further. IER = 46 implies that the iteration for evaluation of the higher integrals i.e., (CS(I), I=1, NZ) did not converge to satisfactory accuracy. The calculations are not aborted at this stage, since most of the effort required in this subroutine has anyway been spent by now. With little extra effort, we can as well compute the zeros and check if they are reliable. The reliability can be verified by the fact that the iteration with original function converges during the refinement phase in subroutine DELVES.

83. NEWRAC Subroutine to calculate a complex zero of a given function using the Newton-Raphson method. This is a complex version of subroutine NEWRAP, but it does not try to estimate the multiplicity of the zero. CX is the complex variable specifying the starting value. After execution, CX will contain the computed value of the zero. REPS and AEPS specify the relative and absolute convergence criterion. The zero should be calculated with an accuracy of $\max(\mathrm{AEPS}, \mathrm{REPS}|\mathrm{CX}|)$. IER is the error parameter. IER = 425 implies that the iteration did not converge. IER = 426 implies that the derivative turns out to be zero at some stage, in which case, the iteration cannot be continued further. If the zero is multiple, then this situation can arise, when the iteration is inside the domain of indeterminacy. In that case, the computed zero may be acceptable. However, in other cases, the computed zero will have no meaning if the derivative is zero. CFUN is the name of the external routine to calculate the required function and its derivative. FUNCTION CFUN(Z, CDF) must be supplied by the user, where CF, CDF and Z are all complex variables and CDF is the first derivative of the function CFUN at Z.

84. POLYR Subroutine to calculate all zeros of a polynomial of degree N, with real coefficients, using Laguerre's method. N is the degree of polynomial, A is a real array of length N + 1 containing the coefficients of the polynomial. A(I) is the coefficient of x^{I-1} in the polynomial i.e., A(1) is the constant term while A(N+1) is the coefficient of x^N. A(N+1) should be nonzero. CX is a complex array of length N, which will contain the zeros of the polynomial after the execution is complete. The roots are sorted in the order of increasing real part. However, if the subroutine is terminated abnormally, then the roots may not be sorted. IER is the error parameter. IER = 406 implies that $\mathrm{N} \leq 0$, while IER = 408 implies that $\mathrm{A(N+1)} = 0$. In such cases, no calculations are performed. IER = 430 implies that the Laguerre's iteration failed to converge at some stage. In this case, the roots already located until then, will be available in CX. IER = $n \times 11$ implies that the iteration for refining the roots did not converge for n of the roots. QREFIN is a logical parameter, QREFIN = .TRUE. specifies that the computed zeros should be "refined" by iterating with the original polynomial. QREFIN = .FALSE. specifies that the refinement is not required. This parameter may be useful, if there is some doubt that the iteration during refinement has converged to some other root, thereby giving rise to an apparent double root. WK is a real array of length N + 1, which is used as a scratch space by the subroutine to store intermediate numbers. The parameter EPS inside the subroutine should be of the order of $\hbar$ for the arithmetic being used. This value is used only to decide whether the root is real or complex.

If $z = x + iy$ is the computed root and $|y| \leq 10\mathrm{EPS} \times |x|$, then the root is assumed to be real. Depending on the outcome of this test, the subroutine performs deflation for a real root, or a pair of complex conjugate roots. This parameter EPS should be changed to appropriate value, but the exact value may not be crucial for working of the subroutine, unless there are roots with very small, but nonzero imaginary parts. This subroutine requires subroutine LAGITR to perform the Laguerre's iteration.

85. LAGITR Subroutine to find one root of a polynomial of degree N with real coefficients, using Laguerre's method. N is the degree of polynomial and A is the real array of length N+1 containing the coefficients. A(I) is the coefficient of x^{I-1}. CXI is the starting value for the Laguerre's iteration. After execution, CXI will contain the computed root. IER is the error parameter. IER = 439 implies that the iteration has failed to converge to any reasonable accuracy. IER = 438 implies that the denominator in the iteration function vanishes and the iteration cannot be continued. In practice, it is rare for this iteration to fail, unless the polynomial is too ill-conditioned for the precision of the arithmetic used.

86. POLYC Subroutine to calculate all zeros of a polynomial of degree N, with complex coefficients, using Laguerre's method. This subroutine is the complex version of POLYR. N is the degree of polynomial. COF is a complex array of length N + 1, containing the coefficients of the polynomial. COF(I) is the coefficient of x^{I-1} in the polynomial i.e., COF(1) is the constant term while COF(N+1) is the coefficient of x^{N}. COF(N+1) should be nonzero. CX is a complex array of length N, which will contain the zeros of the polynomial after the execution is complete. IER is the error parameter. IER = 406 implies that $\mathrm{N} \leq 0$, while IER = 408 implies that COF(N + 1) = 0. In such cases, no calculations are performed. IER = 430 implies that the Laguerre's iteration failed to converge at some stage. In this case, the roots already located until then, will be available in CX. IER = $n \times 11$ implies that the iteration for refining the roots did not converge for n of the roots. QREFIN is a logical parameter, QREFIN = .TRUE. specifies that the computed zeros should be "refined" by iterating with the original polynomial. QREFIN = .FALSE. specifies that the refinement is not required. This parameter may be useful, if there is some doubt that the iteration during refinement has converged to some other root, thereby giving rise to an apparent double root. CWK is a complex array of length N + 1, which is used as a scratch space by the subroutine to store intermediate numbers. This subroutine requires subroutine LAGITC to perform the Laguerre's iteration.

87. LAGITC Subroutine to find one root of a polynomial of degree N with complex coefficients, using Laguerre's method. N is the degree of polynomial and COF is a complex array of length N+1 containing the coefficients. COF(I) is the coefficient of x^{I-1}. CXI is the starting value for the Laguerre's iteration. After execution, CXI will contain the computed root. IER is the error parameter. IER = 439 implies that the iteration has failed to converge to any reasonable accuracy. IER = 438 implies that the denominator in the iteration function vanishes and the iteration cannot be continued. In practice, it is rare for this iteration to fail, unless the polynomial is too ill-conditioned for the precision of the arithmetic used.

88. DAVIDN Subroutine to solve a system of nonlinear equations using Davidenko's method. This method can be coupled with any subroutine for solving a system of nonlinear equations to improve its chances of convergence. It can be used with subroutine NEWTON for Newton's method, when the Jacobian can be easily calculated, or with subroutine BROYDN for Broyden's method, when the Jacobian is too complicated to be calculated explicitly. FCN is the name of the subroutine to calculate the vector function, i.e., the left-hand sides of the equations (the right-hand sides are assumed to be zero). This subroutine should also calculate the Jacobian if Newton's method is to be used. This subroutine must contain the COMMON/DNFUN/$\cdots$ statement to enable it to use the arguments from the common block. The value of THETA, the initial guess X0 and corresponding function values F0, which may be required to calculate the function required for the Davidenko's method. The function can be parametrised in any convenient way, but THETA = θ_0 should have the solution X0 and THETA = 0 should correspond to the required function. The subroutine FCN should be of the form

```
SUBROUTINE FCN(NP,X,F,DF)
IMPLICIT REAL*8(A-H,O-Z)
PARAMETER(NMAX=200)
DIMENSION X(NP),F(NP),DF(NP,NP)
COMMON/DNFUN/THETA,X0(NMAX),F0(NMAX)
.......................
F(1)=..........
...............
F(NP)=.........
DO I=1,NP
  F(I)=F(I)-THETA*F0(I)
ENDDO
END
```

Here NP is the number of variables, X and F are the vectors $\mathbf{x}$ and $\mathbf{f}$, DF is the Jacobian. This form is for use with subroutine NEWTON. For use with BROYDN, the variable DF should be removed from the list of argument as well as from the dimension statement. The last few statements parametrise the function for Davidenko's method. Other parametrisation mentioned in Section 7.16 can also be used, but if X(I) also occurs with THETA, then the Jacobian will also need to be modified (if Newton's method is to be used). If the function has some natural parameter, then it can be used instead of artificial parametrisation, provided the solution is known for some value of the parameter and $\theta = 0$ corresponds to the required solution. The last requirement can be removed by a trivial change in the subroutine. The subroutine FCN with above specifications must be supplied by the user.

NP is the number of variables which must be equal to the number of equations. X is a real array of length NP containing the starting values. After execution, the solution will be returned in the same array. It is not essential to supply the starting values, since these values can be supplied at the time of execution through the read statement. Apart from the starting values, the subroutine will also ask for successive values of THETA, the parameter introduced into the equations. The value of THETA should be changed from θ_0 to zero gradually, where θ_0 is the value for which X0 is the exact solution of the system of equations. The value of θ_0 is generally assumed to be 1. Steps in which the value of THETA should change will depend on the functions involved. If at any stage the iteration fails to converge, a second attempt could be made with THETA changing in smaller steps, or with a different starting value. F is a real array of length NP, which will contain the value of the vector function $\mathbf{f}$ at the point specified by the array X. REPS and AEPS specify the relative and absolute convergence criterion. All components of the root should be calculated to an accuracy of $\max(\mathrm{AEPS}, \mathrm{REPS}|\mathrm{X(I)}|)$. The subroutine initially uses a more modest criterion, which may need to be changed if some of the components of the root are very small, since an absolute criterion with $\mathrm{AEPS0} = 10^{-4}$ may be too large in that case. For such cases, change the value of AEPS0 suitably. AEPS0 may be set equal to AEPS if needed. IER is the error parameter. IER = 407 implies that the value of NP is greater than NMAX (= 200), or NP < 1. In this case, the value of NMAX can be increased and the COMMON statement changed appropriately in the SUBROUTINE FCN. IER = 440 implies that THETA did not reduce to zero, even after 100 steps. In this case, a second attempt could be made with the last value of X as the starting value. Apart from this, other values of IER may be returned by the subroutine NEWTON or BROYDN. WK is a real array of length $\mathrm{NP}^2 + \mathrm{NP}$ for subroutine NEWTON and $\max(2\mathrm{NP}^2, \mathrm{NP}^2 + 4\mathrm{NP})$ for BROYDN. WK is used as a scratch space to store intermediate results. IWK is an integer array of length NP used as a scratch space to store intermediate numbers. This subroutine requires subroutine NEWTON or BROYDN to solve the system of nonlinear equations and subroutine GAUELM to solve intermediate systems of linear equations. To use BROYDN instead of NEWTON, the call statement should be changed as indicated in the comments. DAVIDN_B is the version for use with BROYDN.

89. NEWTON Subroutine to solve a system of nonlinear equations using Newton's method. This method requires the calculation of the full Jacobian at every step. This subroutine can be used directly, if a good approximation to the root is known. Otherwise, it may be better to use it through subroutine DAVIDN to improve the chances of convergence. NP is the number of equations in the system. X is a real array of length NP containing the initial approximation to the solution. After execution, the computed solution will be returned in the same array. F is a real array of length NP containing the value of vector function **f** at the point specified by the array X. FCN is the name of the subroutine used to calculate the left-hand sides of the system of equations (the right-hand sides are assumed to be zero). The subroutine FCN(NP, X, F, DF) must be supplied by the user. (Here NP is the number of equations in the system. X and F are real arrays of length NP containing the values of **x** and the vector function **f**, respectively. DF is a two-dimensional real array of length NP $\times$ NP containing the Jacobian with $\mathrm{DF(I, J)} = \partial\mathrm{F(I)}/\partial\mathrm{X(J)}$. It should be noted that in subroutine FCN, the first dimension of array DF must be equal to NP itself.) REPS and AEPS specify the relative and absolute convergence criterion. All components of the root should be calculated to an accuracy of $\max(\mathrm{AEPS}, \mathrm{REPS}|\mathrm{X(I)}|)$. IER is the error parameter. IER = 442 implies that the iteration did not converge to the specified accuracy. IER = 441 implies that the subroutine GAUELM used to solve the intermediate systems of linear equation has failed. This failure can occur if the value of NP is unacceptable, or if the Jacobian matrix is nearly singular. WK is a real array of length $\mathrm{NP}^2+\mathrm{NP}$, used as a scratch space to store intermediate results. IWK is an integer array of length NP, used as a scratch space to store intermediate numbers.

90. BROYDN Subroutine to solve a system of nonlinear equations using Broyden's method. This method does not require the calculation of derivatives. This subroutine can be used directly if a good approximation to the root is known. Otherwise, it may be better to use it through subroutine DAVIDN_B to improve the chances of convergence. NP is the number of equations in the system. X is a real array of length NP containing the initial approximation to the solution. After execution, the computed solution will be returned in the same array. F is a real array of length NP containing the value of vector function **f** at the point specified by the array X. FCN is the name of the subroutine used to calculate the left-hand sides of the system of equations (the right-hand sides are assumed to be zero). The subroutine FCN(NP, X, F) must be supplied by the user. (Here NP is the number of equations in the system. X and F are real arrays of length NP containing the values of **x** and the vector function **f**(**x**), respectively.) REPS and AEPS specify the relative and absolute convergence criterion. All components of the root should be calculated to an accuracy of $\max(\mathrm{AEPS}, \mathrm{REPS}|\mathrm{X(I)}|)$. IER is the error parameter. IER = 442 implies that the iteration did not converge to the specified accuracy. IER = 441 implies that the subroutine GAUELM used to solve the intermediate systems of linear equation has failed. This failure can occur if the value of NP is unacceptable, or if the matrix is nearly singular. WK is a real array of length $\max(2\mathrm{NP}^2, \mathrm{NP}^2+4\mathrm{NP})$, used as a scratch space to store intermediate results. IWK is an integer array of length NP used as a scratch space to store intermediate numbers.

B.8 Optimisation

91. BRACKM Subroutine to bracket a minimum in one dimension. At the time of calling A and B should contain the two starting values for search. After execution, the triplet (A, X, B) should bracket the minimum with $F(X) < \min(F(A), F(B))$ and X in between A and B. F is the name of the function routine used to calculate the function to be minimised. FUNCTION F(X) must be supplied by the user. IER is the error parameter. IER = 501 implies that A = B and no calculations are performed. IER = 521 implies that the subroutine failed to locate a minimum. This failure can occur, if there is no minimum, or if the initial step $|B - A|$ is too large, which causes the subroutine to jump over the minimum, or if the minimum is too far off as compared to the initial step size.

92. GOLDEN Subroutine to minimise a function in one dimension using the golden section search. Before calling the subroutine, the minimum should be bracketed by the triplet (A, X, B) with X in between A and B and $F(X) < \min(F(A), F(B))$. It is not necessary to have $B > A$, but X must be between A and B. After execution, the final bracket will be overwritten on (A, X, B) with the central value X giving the best approximation to the minimiser. FX will give the value of the function at X. The parameters REPS and AEPS specify the relative and absolute convergence criterion. The interval will be subdivided until $|B - A| < \max(\text{REPS}|X|, \text{AEPS})$. IER is the error parameter. IER = 50 implies that the process of subdivision was terminated, because the function value is equal at all the three points. This is usually due to roundoff error as explained in Section 8.1. IER = 51 implies that the required accuracy was not achieved, even after NIT (= 100) subdivisions. This problem can arise if very high accuracy is required, or if the initial bracket is too large. IER = 522 implies that the input values of A, B, X are not consistent, i.e., either they do not bracket a minimum or X is not between A and B. In this case, no calculations will be performed. F is the name of the function routine used to calculate the function to be minimised. FUNCTION F(X) must be supplied by the user.

93. BRENTM Subroutine to minimise a function in one dimension using the Brent's method. This subroutine is based on the procedure given by Brent (1973). Before calling this subroutine, the minimum should be bracketed by the triplet (A, X, B) with $F(X) < \min(F(A), F(B))$ and X between A and B. After execution, the final bracket will be overwritten on (A, X, B) with the central value X giving the best approximation to the minimiser. FX will give the value of the function at X. The parameters REPS and AEPS specify the relative and absolute convergence criterion. The minimiser X should be calculated with an accuracy of $\max(\text{REPS}|X|, \text{AEPS})$. IER is the error parameter. IER = 51 implies that the required accuracy was not achieved, even after NIT (= 75) iterations. This failure can occur if very high accuracy is required or if the initial bracket is too large. IER = 523 implies that the input values of

A, B, X are not consistent, i.e., either they do not bracket a minimum or X is not between A and B. In this case, no calculations will be performed. No test for roundoff error is performed in this subroutine and the result may not be accurate to the specified accuracy, even if the interval has been reduced to the required size. F is the name of the function routine used to calculate the function to be minimised. FUNCTION F(X) must be supplied by the user.

94. DAVIDM Subroutine to minimise a function in one dimension using the Hermite cubic interpolation. This method requires the calculation of the first derivative in addition to the function value. X1 and X2 specify the two distinct starting values. These values need not bracket the minimum. After execution, X1 and X2 will contain the last two iterates with X2 giving the best approximation to the minimiser. F2 will give the value of the function at X2, while D2F gives the estimate for second derivative, which is used to distinguish between a minimum and a maximum. If D2F > 0 then X2 should be a minimiser, otherwise X2 should be a maximiser. Since D2F is merely an estimate of second derivative, if it is close to zero the nature of extremum will be difficult to determine. The parameters REPS and AEPS specify the relative and absolute convergence criterion. The minimiser X2 should be calculated with an accuracy of $\max(\mathrm{REPS}|\mathrm{X2}|, \mathrm{AEPS})$. IER is the error parameter. IER $= 52$ implies that the iteration has converged to a maximiser, rather than a minimiser. This distinction is made on the basis of the sign of the estimated second derivative and hence may not necessarily be correct if roundoff error is significant. In particular, a point of inflection can be passed on as either a minimiser or a maximiser. IER $= 502$ implies that X1 $=$ X2, in which case, no calculations will be performed. IER $= 524$ implies that the iteration cannot be continued further, since the Hermite cubic does not have a minimum and the corresponding parabolic interpolation yields a zero denominator. IER $= 525$ implies that the required accuracy was not achieved, even after NIT ($= 75$) iterations. The convergence failure can occur if very high accuracy is required, or if the starting values are far from the minimiser. F is the name of the function routine used to calculate the function to be minimised and its derivative. FUNCTION F(X, DF) must be supplied by the user. Here DF is the first derivative of F at X.

95. BFGS Subroutine to find minimum of a function of N variables using the quasi-Newton method, with BFGS formula for updating the Hessian matrix. This method requires the calculation of gradient vector in addition to the function value. X is a real array of length N, containing the starting values for the independent variables. After execution, the minimiser will be returned in the same array X. F is a real variable giving the value of the function at X. G is a real array of length N, which will contain the value of the gradient vector at X ($\mathrm{G(I)} = \frac{\partial \mathrm{F}}{\partial \mathrm{X(I)}}$). H is a two-dimensional array of length at least N $\times$ N, which will contain an approximation to the inverse of the Hessian matrix at X. The first dimension of H must be equal to N. F, G and H are output parameters, which need not be initialised at the time of calling. NUM is an output parameter containing the number of function evaluations required by the subroutine.

The parameters REPS and AEPS specify the relative and absolute convergence criterion. Each component of the minimiser X should be calculated with an accuracy of $\max(\mathrm{REPS}|X(I)|, \mathrm{AEPS})$. IER is the error parameter. IER = 53 implies that the Hessian matrix is probably singular at the final point. In this case, the iteration could have converged to a saddle point and further investigation may be required to determine the nature of the stationary point. This is detected by considering the norm of the matrix H as explained in Section 8.5. This test is not very reliable, particularly if the required accuracy is too low or too high. IER = 503 implies that $N < 1$, in which case no calculations are performed. IER = 526 implies that the iteration failed to converge to a satisfactory accuracy. Other values of IER may be set by the subroutine LINMIN, which is called to perform the line searches. FCN is the name of the subroutine which calculates the function F and the gradient vector G. WK is a real array of length 3N, which is used as a scratch space. SUBROUTINE FCN(N, X, F, G) must be supplied by the user. Here N is the number of variables, F is the function value, while X and G are real arrays of length N, containing the coordinates of the point X, where the function F and the gradient vector G are to be evaluated. Apart from FCN this subroutine requires the subroutine LINMIN to perform the line searches and the function FLNM to calculate the function and its derivative along the given line, as required by LINMIN.

96. LINMIN Subroutine to perform line search as required by the subroutine BFGS. It implements a crude, but reasonably efficient algorithm to find an acceptable minimum of a function of N variables along a given direction. Any point which satisfies the conditions (8.37) is acceptable. This subroutine requires the first derivative of the function in addition to the function value. This subroutine should only be used for line search as required by quasi-Newton methods and not for any other purpose, since the criterion for acceptance will not suffice for other purposes. X1 is the starting point from where the line search is to be performed. X2 is the first guess for the minimum. After execution, X1, F1 and DF1 will respectively contain the accepted point, the value of the function and its first derivative along the given direction at that point. The parameters REPS and AEPS specify the relative and absolute convergence criterion for the subroutine BFGS. These parameters are used here only to terminate the line search, once the interval has been reduced sufficiently. This situation should not arise normally, but occasionally because of roundoff error, or error in coding the derivatives, the subroutine may fail to find acceptable points. IER is the error parameter. IER = 527 implies that the subroutine has failed to find any point, where the function value is less than or equal to that at the starting point. This failure could be due to roundoff error, or more likely due to some error in coding the derivatives. IER = 55 or 528 implies that the subroutine failed to find an acceptable point, even though the interval has been reduced to specified tolerance. In this case, the last point is accepted if the function value is less than or equal to that at starting point (IER = 55), otherwise IER = 528. IER = 54 implies that the subroutine failed to find an acceptable point in NIT (= 15) iterations, but the function value at the last approximation is less than or equal to the value at the starting point and this point is accepted. The last two conditions are often encountered, when the iteration is close to convergence and the accuracy requirement is too high. The final point may be acceptable in such circumstances. This can be checked by verifying the value of the gradient vector in subroutine BFGS. The parameters F, V, XI, N and NUM are not used by this subroutine, but are simply passed on to the FUNCTION FLNM, for calculating the function value and the first derivative. F is the name of the subroutine used to calculate the function of N variables, V is a real array of length 3N, first N elements of which contain the direction along which the line search is to be carried out. After execution of FLNM the next N elements will contain the coordinates of the last point at which the function is evaluated, while the last N elements of this array will contain the gradient vector at the last point. XI is a real array of length N, containing the coordinates of the starting point for the line search. NUM is an integer variable to keep count of the number of function evaluations. This subroutine requires the function FLNM and the subroutine F to calculate the function. SUBROUTINE F(N, X, FX, G) must be supplied by the user. Here N is the number of variables, FX is the function value, while X and G are real arrays of length N, containing the coordinates of the point X, where the function FX and the gradient vector G are to be evaluated.

97. FLNM Subroutine to calculate the function as required for the line search routine LINMIN. FCN is the name of the subroutine used to calculate the function of N variables. X is a parameter along the line which specifies the point at which the function is to be evaluated. DF is the first derivative along the line. V is a real array of length 3N, the first N elements of which specify the direction along which the search is being carried out. Next N elements will contain the coordinates of the point at which function is evaluated, while the last N elements contain the gradient vector at this point. X0 is a real array of length N, containing the coordinates of the starting point. NUM is an integer variable which keeps a count of function evaluations. The function will be evaluated at the point with coordinates XI(I) = X0(I) + X × V(I). These coordinates are returned in N+1 to 2N element of array V. The SUBROUTINE FCN(N, XI, F, G) must be supplied by the user. Here XI is a real array of length N, containing the coordinates of the required point. F is the function value at this point. G is a real array of length N containing the gradient vector at XI.

98. NMINF Subroutine to find minimum of a function of N variables, using a direction set method. This method does not require calculation of the derivatives. X is a real array of length N, containing the starting values for the independent variables. After execution, the minimiser will be returned in the same array X. F is a real variable giving the value of the function at X. NUM is an output parameter containing the number of function evaluations required by the subroutine. The parameters REPS and AEPS specify the relative and absolute convergence criterion. The iteration is continued until the change in function value δF during one complete cycle consisting of N iterations, satisfies $|\delta F| < \max(\text{REPS}|\text{F}|, \text{AEPS})$. It may be noted that unlike subroutine

BFGS, here the convergence test is applied on function value, rather than the minimiser. This must be kept in mind while specifying REPS and AEPS. In general, relative change by REPS in function value will translate to relative variation by $\sqrt{\text{REPS}}$ in coordinates of minimiser. IER is the error parameter. IER = 504 implies that $\text{N} \leq 1$, in which case, no calculations are performed. IER = 529 implies that the iteration failed to converge to a satisfactory accuracy. Other values of IER may be set by the subroutine LINMNF, which is called to perform the line searches. FCN is the name of the subroutine which calculates the function F. SUBROUTINE FCN(N, X, F) must be supplied by the user. Here N is the number of variables, X is an array of length N containing the parameter values at which the function value is required. F is the calculated value of the function at X. WK is a real array of length 2N(N+1), which is used as a scratch space. Apart from FCN, this subroutine requires subroutine LINMNF to perform the line searches and function FLN to calculate the function as required by LINMNF, and subroutine SVD to perform the singular value decomposition.

99. LINMNF Subroutine to perform line search as required by the subroutine NMINF. It implements a crude, but reasonably efficient algorithm to find an acceptable minimum of a function of N variables along a given direction. This subroutine does not require any derivative. Any point, where the function value is less than that at the starting point is acceptable. This subroutine should only be used for line search as required by direction set methods. Because of the crude criterion for acceptance it is not suitable for a general minimisation along a given direction. X0 is the starting point, from where the line search is to be performed. X1 is the first guess for the minimum. After execution, X0 and F0 will contain the accepted point and the corresponding function value. The parameters REPS and AEPS specify the relative and absolute convergence criterion for the subroutine NMINF. These parameters are used here only to terminate the line search, once the interval has been reduced sufficiently. This situation should not arise normally, but occasionally because of roundoff error, the subroutine may fail to find an acceptable point. IER is the error parameter. IER = 56 implies that the subroutine failed to find an acceptable point and in this case, the starting point itself is accepted. The parameters F, V, XI, N and NUM are not used by this subroutine, but are simply passed on to the FUNCTION FLN, to calculate the function value. F is the name of the subroutine used to calculate the function of N variables, V is a real array of length N, containing the direction along which the line search is to be carried out. XI is a real array of length 2N, the first N elements contain the coordinates of the starting point for the line search. After execution of FLN, the next N elements will contain the coordinates of the last point at which the function is evaluated. NUM is an integer variable to keep count of the number of function evaluations. The subroutine LINMNF requires the function FLN and the subroutine F to calculate the function. SUBROUTINE F(N, X, FX) must be supplied by the user. Here N is the number of variables, X is an array of length N containing the parameter values at which the function value is required. FX is the calculated value of the function at X.

100. FLN Subroutine to calculate the function as required for the line search routine LINMNF. FCN is the name of the subroutine used to calculate the function of N variables. X is a parameter along the line which specifies the point at which the function is to be evaluated. V is a real array of length N, containing the direction along which the search is being carried out. X0 is a real array of length 2N, the first N elements of which specify the coordinates of the starting point. NUM is an integer variable which keeps a count of function evaluations. The function will be evaluated at a point with coordinates XI(I) $=$ X0(I) $+$ X $\times$ V(I). The last N elements of X0 are used to store XI. The SUBROUTINE FCN(N, XI, F) must be supplied by the user. Here XI is a real array of length N, containing the coordinates of the required point and F is the function value at this point.

101. SIMPLX Subroutine to solve a linear programming problem using the simplex method. N is the number of variables, each of which is constrained to be nonnegative. M1 is the number of constraints of the form $\mathbf{a}^T\mathbf{x} \leq b_i \geq 0$, M2 is the number of constraints of the form $\mathbf{a}^T\mathbf{x} \geq b_i \geq 0$, M3 is the number of constraints of the form $\mathbf{a}^T\mathbf{x} = b_i \geq 0$. A is a real array of length $\mathrm{IA} \times (\mathrm{N}+\mathrm{M2}+1)$. IA specifies the first dimension of A, exactly as in the calling program ($\mathrm{IA} \geq \mathrm{M1}+\mathrm{M2}+\mathrm{M3}+2$). The array A contains the tableau as explained in Section 8.7. At input the first row of A contains the cost coefficients $\mathrm{A}(1,\mathrm{i}+1) = c_i$, where $c_1x_1 + c_2x_2 + \ldots + c_Nx_N$ is the function to be minimised. The next $\mathrm{M} = \mathrm{M1}+\mathrm{M2}+\mathrm{M3}$ rows contain the coefficients for the constraints, with the rows 2 to $\mathrm{M1}+1$ containing the constraints of the first type, rows $\mathrm{M1}+2$ to $\mathrm{M1}+\mathrm{M2}+1$ containing the constraints of the second type and the remaining rows containing the constraints of the third type. $\mathrm{A}(\mathrm{i},1) = b_i$ and $\mathrm{A}(\mathrm{i},\mathrm{j}+1) = a_j$ for the ith constraint. If $\mathrm{M2}+\mathrm{M3} > 0$, then the row $\mathrm{M}+2$ will be used by the subroutine to specify the cost coefficients for the auxiliary objective function to be minimised for finding a basic feasible vector. If only constraints of the first type are present, then this row will not be required. After the execution is complete, the optimal feasible vector will be returned in the real array X of length N. F is an output parameter containing the optimum value of the objective function. IWK is an integer array of length $\mathrm{N}+\mathrm{M}+\mathrm{M2}+1$, which is used as a scratch space by the subroutine. This array will contain the permutation information about the variables. IWK(I), $2 \leq \mathrm{I} \leq \mathrm{M}+1$ gives the serial number of variable corresponding to the Ith row of the tableau. IWK(I), $\mathrm{I} > \mathrm{M}+1$ gives the serial number of variable corresponding to $(\mathrm{I}-\mathrm{M})$th column of the tableau. The variables are numbered as follows: the first N variables are the required variables, after that $\mathrm{M1}+\mathrm{M2}$ slack variables are introduced, and finally there are $\mathrm{M2}+\mathrm{M3}$ artificial variables for the auxiliary problem. IER is the error parameter. IER $= 57$ implies that the objective function is unbounded from below and no optimal feasible vector exists. IER $= 58$ implies that, there is no basic feasible vector. Hence, the constraints are inconsistent or the problem is not properly formulated. IER = 505 implies that at least one of the variables N, M1, M2 and M3 is negative or $\mathrm{IA} < \mathrm{M} + 2$. IER = 506 implies that at least one of the right-hand side coefficients in the constraints is negative. In both these cases, no calculations are performed. IER = 531 implies that the simplex algorithm failed to find the optimal vector in a reasonable number of iterations. This failure can occur either because the number of iterations required are very large, or because the simplex iteration has gone into a nonterminating loop because of degeneracy. In principle, this failure can happen for the auxiliary problem for finding a basic feasible vector also. IER = 532 implies that a basic feasible vector could not be found to start the simplex iteration. This situation can occur if the solution to the auxiliary problem is degenerate and the artificial variable on the left-hand side cannot be exchanged. This situation should not arise, unless the constraints are not linearly independent, or roundoff error is significant. AEPS is a real parameter to control roundoff error, any quantity less than AEPS in magnitude is assumed to be zero. This subroutine requires subroutine SIMPX to actually perform the simplex minimisation.

102. SIMPX Subroutine to solve linear programming problems using the simplex method. The problem is assumed to be in the standard form and the initial tableau is supplied in the real array A of length $\mathrm{IA} \times (\mathrm{N} - \mathrm{M} + 1)$ with $\mathrm{IA} \geq \mathrm{M} + 2$. IA is the first dimension of A, exactly as specified in the calling program. N is the total number of variables in the given problem, including any slack or artificial variables that have been introduced. M is the number of constraints in the problem. NV is the number of variables excluding the artificial variables (if any). This parameter is used only for the auxiliary function, to check if any artificial variable is remaining on the left-hand side. QF is a logical variable, which should be set to .TRUE. if the main objective function specified by the first row of the tableau is to be minimised. QF should be set to .FALSE. if the auxiliary objective function specified by the last row of A is to be minimised for finding the initial basic feasible vector. In that case, attempt is made to remove the artificial variables from the set of left-hand side variables, before accepting the solution. ID and IV are integer arrays of length $\mathrm{M} + 1$ and $\mathrm{N} - \mathrm{M} + 1$ respectively, used to store permutations of the original variables. IER is the error parameter. IER = 57 implies that the objective function is unbounded from below and the optimal feasible vector does not exist. IER = 531 implies that the simplex iteration has not converged in a reasonable number of iterations. This failure may be due to degeneracy, since that is not accounted for in this subroutine. AEPS is a real parameter used to control roundoff error. Any quantity less than AEPS in magnitude is assumed to be zero.

B.9 Statistical Inferences

103. SHSORT Subroutine to sort an array of length N in ascending order, using shell sort algorithm. A is a real array of length N, which is to be sorted.

After execution the sorted elements will be returned in the same array. N is the number of elements of array A to be sorted.

104. GAMMAP Function to calculate the incomplete Gamma function

$$P(a,x) = \frac{\gamma(a,x)}{\Gamma(a)} = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1}\, dt \; . \tag{B.32}$$

It may be noted that sometimes the incomplete Gamma function is defined by $\gamma(a,x)$. This should be accounted for while using the function routine. It is difficult to approximate this function over entire range of a, x values and it is possible that for some combination the value may not be reliable. A is the argument for the complete Gamma function and X is the upper limit of integration in the above equation. The function is not defined for A ≤ 0 or X < 0 and in these cases a value of -1 will be returned without any warning. For $x < 3$ it uses the power series (DiDonato & Morris 1986) to approximate

$$P(a,x) = \frac{x^a}{\Gamma(a+1)} \left(1 + a \sum_{n=1}^{\infty} \frac{(-x)^n}{(a+n)n!} \right) . \tag{B.33}$$

For $x < 1.2a$ a continued fraction approximation for complementary function

$$Q(a,x) = 1 - P(a,x) = \cfrac{\frac{e^{-x}x^a}{\Gamma(a)}}{x + \cfrac{1-a}{1 + \cfrac{1}{x + \cfrac{2-a}{1 + \frac{2}{x+\cdots}}}}} \tag{B.34}$$

is used. For other values of parameter the power series

$$P(a,x) = \frac{e^{-x}x^a}{\Gamma(a+1)} \left(1 + \sum_{n=1}^{\infty} \frac{x^n}{(a+1)(a+2)\cdots(a+n)} \right) \tag{B.35}$$

is used. This function requires Functions GAMMA and GAMMAL to calculate the (complete) Gamma function or its logarithm.

105. BETAP Function to calculate the incomplete Beta function

$$I_x(a,b) = \frac{B_x(a,b)}{B(a,b)} = \frac{1}{B(a,b)} \int_0^x t^{a-1}(1-t)^{b-1}\, dt \; . \tag{B.36}$$

It may be noted that sometimes the incomplete Beta function is defined by $B_x(a,b)$. This should be accounted for while using the function routine. It is difficult to approximate this function over entire range of a, b, x values and it is possible that for some combination the value may not be reliable. A and B are the arguments for the complete Beta function and X is the upper limit of integration in the above equation. Here A -1 is the exponent of t and B -1 is that of $1-t$ in the integral. The function is not define for A ≤ 0 or B ≤ 0 or X < 0 or X > 1 and in these cases a value of -1 will be returned without any warning. These function requires Functions BETSER, BETCON, BETCON1 and BETAI for calculating the function using different approximations depending on the arguments. Apart from these Function GAMMAL is required to calculate logarithm of Gamma function and subroutines ADPINT, KRONRD and FBETA to calculate the integral for some argument range.

106. BETSER Function routine to calculate the incomplete Beta function (B.36) using the infinite series (DiDonato & Morris 1992)

$$I_x(a,b) = \frac{x^a}{aB(a,b)}\left(1 + a\sum_{j=1}^{\infty} \frac{(1-b)(2-b)\cdots(j-b)}{j!(a+j)} x^j\right) \tag{B.37}$$

This series is useful for small values of x. This function is called by BETAP for some range of arguments. This routine should not be used to calculate the function for arbitrary values of its arguments as the approximation may not be valid. A and B are the arguments for the complete Beta function and X is the upper limit of integration in Eq. (B.36). This function needs Function GAMMAL to calculate the logarithm of Gamma function.

107. BETCON1 Function routine to calculate the incomplete Beta function (B.36) using a continued fraction (DiDonato & Morris 1992) approximation

$$I_x(a,b) = \cfrac{\cfrac{x^a(1-x)^b}{aB(a,b)}}{1 + \cfrac{d_1}{1 + \cfrac{d_2}{1+\cdots}}} \tag{B.38}$$

with

$$d_{2n} = \frac{n(b-n)x}{(a+2n-1)(a+2n)}\,, \qquad d_{2n+1} = -\frac{(a+n)(a+b+n)x}{(a+2n)(a+2n+1)}\,. \tag{B.39}$$

This function is called by BETAP for some range of arguments. This routine should not be used to calculate the function for arbitrary values of its arguments as the approximation may not be valid. A and B are the arguments for the complete Beta function and X is the upper limit of integration in Eq. (B.36). This function needs Function GAMMAL to calculate the logarithm of Gamma function.

108. BETCON Function routine to calculate the incomplete Beta function (B.36) using a continued fraction (DiDonato & Morris 1992) approximation

$$I_x(a,b) = \cfrac{\cfrac{x^a(1-x)^b}{aB(a,b)} a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_3+\cdots}}} \tag{B.40}$$

where, coefficients a_i, b_i are defined using Eq. (B.39)

$$a_1 = 1, \quad b_1 = 1+d_1, \quad a_{n+1} = -d_{2n-1}d_{2n}, \quad b_{n+1} = 1+d_{2n}+d_{2n+1}, \quad (n > 1). \tag{B.41}$$

This function is called by BETAP for some range of arguments. This routine should not be used to calculate the function for arbitrary values of its arguments as the approximation may not be valid. A and B are the arguments for the complete Beta function and X is the upper limit of integration in Eq. (B.36). This function needs Function GAMMAL to calculate the logarithm of Gamma function.

109. BETAI Function routine to calculate the incomplete Beta function (B.36) by directly evaluating the integral. This function is called by BETAP for some range of arguments. This routine should not be used to calculate the function for arbitrary values of its arguments as it may not be very efficient for all values of the arguments. A and B are the arguments for the complete Beta function and X is the upper limit of integration in Eq. (B.36). This function needs Function GAMMAL to calculate the logarithm of Gamma function, subroutines ADPINT and KRONRD to evaluate the integral. Further, the function FBETA is needed to define the integrand for the integral.

110. FBETA Function routine to calculate the integrand for calculating the incomplete beta function. This is used by Function BETAI.

111. RANGAU Function routine to generate a sequence of random numbers with Normal distribution of probability, with zero mean and unit variance. This subroutine uses algorithm given by Knuth. SEED could be set to any positive value less than AN before first call to the routine. After the first call, this variable should not be changed in any other program, unless an independent sequence of random numbers is required.

112. IRANBIN Function routine to generate a sequence of random numbers with binomial distribution. SEED is the seed for generating random numbers. It should be negative for the first call to function and should not be changed in any other program, unless an independent sequence of random number with different n or p is required. It should be noted that although the random number is an integer in this case, the SEED is of type Real. N is the number of trials in the binomial distribution and P is the probability of the event in one trial. C is a real array of length N, which is used to store the cumulative probability table for use in calculations. This array should not be modified in any other program. This is used only if the mean (np) is less than RMAX, in which case C(I-1) will be the probability of having I or less events.

113. IRANPOI Function routine to generate a sequence of random numbers with Poisson distribution. SEED is the seed for generating random numbers. It should be negative for the first call to function and should not be changed in any other program, unless an independent sequence of random number with different mean is required. It should be noted that although the random number is an integer in this case, the SEED is of type Real. RMU is the mean of Poisson distribution. P is a real array of length NMAX (=200), which is used to store the cumulative probability table for use in calculations. This array should not be modified in any other program. This is used only if the mean (RMU) is less than some critical value, in which case P(I-1) will be the probability of having I or less counts.

114. PCOR Function routine to calculate the probability that two uncorrelated sequences of length $(n+2)$ will give a correlation coefficient exceeding $|x|$. For large even values of n the series can give large roundoff errors in which case the distribution is approximate by normal distribution. N is the number of degrees of freedom, i.e., $\mathrm{N}+2$ is the length of the sequences and XX is the value of correlation coefficient. Since the probability distribution of correlation coefficient for uncorrelated data is symmetric about $x = 0$, the probability is calculated for $|x|$. This function requires Function GAMMAL to calculate the logarithm of Gamma function and Function ERF to calculate the Error function.

B.10 Functional Approximations

115. POLFIT Subroutine to perform least squares polynomial fit using orthogonal polynomials. N is the number of data points, M is the degree of polynomial to be fitted. X, F and SIG are real arrays of length N containing the data points. F(I) is the value of function at X(I) and SIG(I) is the corresponding error estimate in F(I). If error estimates are not available then all SIG(I) can be set to one. A is a real array of length $\mathrm{M}+1$, which will contain the coefficients of the orthogonal polynomials in the calculated fit. ALP and BETA are real arrays of length $\mathrm{M}+1$, which will contain the coefficients α_i and β_i as defined in recurrence relation for orthogonal polynomials. $\mathrm{ALP(I)} = \alpha_I$ and $\mathrm{BETA(I+1)} = \beta_I$. Y is a real array of length N, Y(I) will contain the calculated value of function at X(I) using the least squares fit. H is a real array of length $\mathrm{M}+1$ containing the χ^2. H(I+1) will contain the χ^2 using polynomial of degree I,

$$\mathrm{H(M+1)} = \sum_{j=1}^{\mathrm{N}} \left(\frac{\mathrm{Y}(j) - \mathrm{F}(j)}{\mathrm{SIG}(j)} \right)^2 . \tag{B.42}$$

As explained in Section 10.2.2, this estimate of H(I) may have a significant roundoff error. GAM is a real array of length $\mathrm{M}+1$, which will contain the quantity γ_i for the orthogonal polynomials as defined in Section 10.2.2. WK is a real array of length 2N, which is used as a scratch space to store intermediate quantities. IER is the error parameter. IER = 601 implies that $\mathrm{M} \geq \mathrm{N}$ or $\mathrm{M} < 0$, in which case, no calculations are performed. IER = 621 implies that one of the $\gamma_i = 0$, which can happen if the points X(I) are not distinct. The fitted polynomial can be calculated at any point using the coefficients A, ALP and BETA by subroutine POLEVL.

116. POLEVL Subroutine to evaluate the approximating polynomial, and its derivatives using Clenshaw's recurrences. This subroutine can be used to calculate the value of approximating polynomial at any point X, after the required coefficients a_j, α_j and β_j have been calculated by subroutine POLFIT. M (> 0) is the degree of polynomial. A, ALP and BETA are real arrays of length M + 1, with A($j + 1$) $= a_j$, ALP(j) $= \alpha_j$ and BETA($j + 1$) $= \beta_j$. These coefficients must be calculated before calling the subroutine POLEVL. X is the value of the independent variable, at which the approximation has to be evaluated. F, DF and DDF are output parameters containing the calculated values of the function and its first and second derivatives, respectively.

117. POLFIT1 Subroutine to perform least squares polynomial fit using orthogonal polynomials. This is a version of POLFIT which can handle multiple fits, i.e., multiple sets of function values over the same set of abscissas and errors. This is useful for recursive use in multiple dimensions. N is the number of data points, M is the degree of polynomial to be fitted. NUM is the number of different right hand sides (function values) to be fitted. X and SIG are real arrays of length N containing the data points and errors. F is a real array of length N $\times$ NUM containing the function values for each set of points. F(I, J) is the value of function in Jth set at X(I) and SIG(I) is the corresponding error. The errors are assumed to be the same in all sets. If error estimates are not available then all SIG(I) can be set to one. The first dimension of F in the calling program must be equal to N. A is a real array of length (M + 1) $\times$ NUM, which will contain the coefficients of the orthogonal polynomials in the calculated fit for each set. The first dimension of A in the calling program must be equal to M + 1. ALP and BETA are real arrays of length M + 1, which will contain the coefficients α_i and β_i as defined in recurrence relation for orthogonal polynomials. GAM is a real array of length M + 1, which will contain the quantity γ_i for the orthogonal polynomials as defined in Section 10.2.2. WK is a real array of length 2N, which is used as a scratch space to store intermediate quantities. IER is the error parameter. IER = 601 implies that M $\geq$ N or M < 0, in which case, no calculations are performed. IER = 621 implies that one of the $\gamma_i = 0$, which can happen if the points X(I) are not distinct. The fitted polynomial can be calculated at any point using the coefficients A, ALP and BETA by subroutine POLEVL.

118. POLORT Subroutine to evaluate the orthogonal polynomials, and its derivatives. It may be noted that this subroutine calculates the value of orthogonal polynomials at the given point as opposed to POLEVL which calculates the value of fitted function using the coefficients of these polynomials. This subroutine can be used to calculate the value of orthogonal polynomial basis functions at any point X, after the required coefficients α_j and β_j have been calculated by subroutine POLFIT or POLFIT1. M (> 0) is the degree of polynomial. ALP and BETA are real arrays of length M + 1, with ALP(j) $= \alpha_j$ and BETA($j + 1$) $= \beta_j$. These coefficients must be calculated before calling the subroutine POLORT. X is the value of the independent variable, at which the polynomials have to be evaluated. F, DF and DDF are real arrays of length $\mathrm{M}+1$, which will contain the calculated values of the $\mathrm{M}+1$ orthogonal polynomials and its first and second derivatives, respectively at X.

119. POLFIT2 Subroutine to perform least squares polynomial fit using orthogonal polynomials in two dimensions. The data values must be available at a rectangular grid of points and weights are assumed to be equal for all points. NX is the number of data points along X axis, NY is the number of data points along Y axis. X is a real array of length NX containing the points along X axis, while Y is a real array of length NY containing the points along the Y axis. F is a real array of length $\mathrm{LA} \times \mathrm{NY}$ containing the function values. F(I, J) is the value at X(I), Y(J). AX is a real array of length $\mathrm{IC} \times 3$ containing information about the fit along X direction. AX(I, 1), AX(I+1, 2), AX(I+1, 3) will respectively contain the coefficients $\alpha_i, \beta_i, \gamma_i$ for the orthogonal polynomials. AY is a real array of length $\mathrm{IC} \times 3$ containing information about the fit along Y direction. AY(I, 1), AY(I+1, 2), AY(I+1, 3) will respectively contain the coefficients $\alpha_i, \beta_i, \gamma_i$ for the orthogonal polynomials. The arrays AX and AY will be calculated by the subroutine. LA is the first dimension of arrays F and FY as declared in the calling program, $\mathrm{LA} \geq \max(\mathrm{NX}, \mathrm{NY})$. C is a real array of length $\mathrm{IC} \times (\mathrm{MY}+1)$ containing the fitted coefficients for the polynomial fit in two dimensions. The fitted polynomial would be

$$\sum_{i=1}^{\mathrm{MX}+1} \sum_{j=1}^{\mathrm{MY}+1} \mathrm{C}(i,j)\phi_i(x)\psi_j(y), \tag{B.43}$$

where $\phi_i(x)$ and $\psi_j(y)$ are the orthogonal polynomials in x and y respectively. IC is the first dimension of arrays AX, AY and C as declared in the calling program, $\mathrm{IC} > \max(\mathrm{MX}, \mathrm{MY})$. MX is the required degree of polynomial in X. MY is the required degree of polynomial in Y. FY is a real array of length $\mathrm{LA} \times \mathrm{NY}$ containing the fitted values of the function at each of the tabular points. WK is a real array of length $\max(\mathrm{NX} \times (\mathrm{NY}+\mathrm{MY}+1), 6(\max(\mathrm{NX}, \mathrm{NY})+2))$, which is used as a scratch space to store intermediate quantities. AW is a real array of length $\mathrm{LA} \times 3$, which is used as a scratch space to store intermediate quantities. CHISQ is calculated value of χ^2 for the fit. IER is the error parameter. $\mathrm{IER} = 602$ implies that $\mathrm{IC} < \mathrm{MX}+1$ or $\mathrm{LA} < \max(\mathrm{NX}, \mathrm{NY})$. $\mathrm{IER} = 603$ implies that $\mathrm{NX} \leq \mathrm{MX}$, $\mathrm{NY} \leq \mathrm{MY}$, $\mathrm{MX} < 0$ or $\mathrm{MY} < 0$. In all these cases, no calculations are performed. The fitted polynomial can be calculated at any point using the coefficients AX, AY, C by subroutine POLEV2. This subroutine requires subroutines POLFIT1, POLEV2 and POLORT.

120. POLEV2 Subroutine to evaluate the approximating polynomial, and its derivatives using expansion in orthogonal polynomials in 2 dimensions. This subroutine can be used to calculate the value of approximating polynomial at any point, after the required coefficients c_{ij} as well as other auxiliary coefficients needed to define the orthogonal polynomials have been calculated by subroutine POLFIT2. NX is the degree of polynomial in X, NY is the degree of polynomial in Y. AX is a real array of length LA $\times$ 2 containing the coefficients α_i and β_i for orthogonal polynomials in X. AX(I, 1) contains α_i and AX(I+1, 2) contains β_i. AY is a real array of length LA $\times$ 2 containing the coefficients α_i and β_i for orthogonal polynomials in Y. AY(I, 1) contains α_i and AY(I+1, 2) contains β_i. These coefficients must be calculated before calling the subroutine POLEV2 using POLFIT2. LA is the first dimension of arrays AX, AY and WT in the calling program, LA $>$ max(NX, NY). WT is a real array of length LA $\times$ (NY + 1), containing the coefficients of the fit. X0, Y0 are the coordinates of the point at which function value needs to be calculated. F is the output parameter containing the calculated values of the function at (X0, Y0). DFX and DFY are respectively, $\partial F/\partial x$ and $\partial F/\partial y$, while DFXX, DFXY, DFYY are the second derivative $\partial^2 F/\partial x^2$, $\partial^2 F/\partial x\partial y$, $\partial^2 F/\partial y^2$. WK is a real array of length 6(max(NX, NY)+2) used as scratch space to store intermediate quantities. IER is the error parameter. IER = 604 implies that max(NX, NY) $\geq$ LA, in which case no calculations are done. The subroutine requires POLORT to calculate the orthogonal basis functions in one dimensions.

121. POLFITN Subroutine to perform least squares polynomial fit using orthogonal polynomials in N dimensions. The data values must be available at a hyper-rectangular grid of points and weights are assumed to be equal for all points. N is the number of dimensions. NK is an integer array of length N containing the number of data points along each axis. NK(I) is the number of points along Ith axis. X is a real array of length LA $\times$ N containing the points along each axis, X(I, J) is the Ith point along the Jth dimension. F is a real array of length NK(1) $\times$ NK(2) $\times \cdots \times$ NK(N) containing the function values. F is treated as a one dimensional array in this routine and hence the dimensions of array in the calling program must exactly match the number of points in each dimension, e.g., F(NK(1), NK(2), ..., NK(N)). Alternately, F can be treated as a one dimensional array of required length in the calling program also. AX is a real array of length LA $\times$ (3N + 3) containing information about the fit along each direction. AX(I, 3J−2), AX(I+1, 3J−1), AX(I, 3J) will respectively contain the coefficients $\alpha_i, \beta_i, \gamma_i$ for the orthogonal polynomials along Jth dimension. The rest of the array is used as scratch space while calculating fits in one dimension. LA is the first dimension of arrays X and AX as declared in the calling program, LA $\geq$ max(NK(I)). C is a real array of length (MK(1) + 1)(MK(2) + 1) $\cdots$ (MK(N) + 1) containing the fitted coefficients for the polynomial fit in N dimensions. The fitted polynomial would be

$$\sum_{i_1=1}^{\mathrm{MK(1)}} \sum_{i_2=1}^{\mathrm{MK(2)}} \cdots \sum_{i_N=1}^{\mathrm{MK(N)}} \mathrm{C}(i_1, i_2, \ldots, i_N)\phi_{i_1}(x_1)\phi_{i_2}(x_2)\cdots\phi_{i_N}(x_N), \tag{B.44}$$

where $\phi_{i_j}(x_j)$ are the orthogonal polynomials along jth dimension. C is treated as a one dimensional array in this routine and hence the dimensions of array in the calling program must exactly match the number of polynomials in each dimension, e.g. C(MK(1)+1, MK(2)+1, ..., MK(N)+1). MK is an integer array of length N containing the required degree of polynomial in each direction. FY is a real array of the same length and shape as F which will contain the fitted value of the function at each of the tabular points. WK is a real array of length 2NK(1) $\times$ NK(2) $\times \cdots \times$ NK(N), which is used as a scratch space to store intermediate quantities. IWK is an integer array of length N used as scratch space. CHISQ is calculated value of χ^2 for the fit. IER is the error parameter. IER = 605 implies that LA < max(NK(I)). In this case, no calculations are performed. The fitted polynomial can be calculated at any point using the coefficients C and AX by subroutine POLEVN, POLEVN1 or POLEVN2. This subroutine requires subroutines POLFIT1, POLEVN, POLORT.

122. POLEVN Subroutine to evaluate the approximating polynomial using expansion in orthogonal polynomials in N dimensions. This subroutine can be used to calculate the value of approximating polynomial at any point, after the required coefficients as well as other auxiliary coefficients needed to define the orthogonal polynomials have been calculated by subroutine POLFITN. N is the number of dimensions. NK is an integer array of length N containing the degree of polynomial in each dimension. AX is a real array of length LA $\times$ (3N + 3) containing the coefficients α_i and β_i for orthogonal polynomials in X. AX(I, 3J$-$2) contains α_i and AX(I+1, 3J$-$1) contains β_i for orthogonal polynomials along Jth dimension. These coefficients must be calculated before calling the subroutine POLEVN using POLFITN. LA is the first dimension of array AX in the calling program, This must be the same as what was used in call to POLFITN while calculating the coefficients. WT is a real array of length (MK(1) + 1)(MK(2) + 1) $\cdots$ (MK(N) + 1), containing the coefficients of the fit. This array is treated as one-dimensional array in the subroutine and hence its dimensions in the calling program must exactly match the number of orthogonal polynomials in each dimension. X0 is a real array of length N containing the coordinates of the point at which function value needs to be calculated. F is the output parameter containing the calculated values of the function at X0. WK is a real array of length 3N $\times$ LA used as scratch space. IWK is an integer array of length N used as scratch space. This subroutine requires POLORT to calculate the orthogonal basis functions in one dimensions. This subroutine does not calculate the derivatives of F. If first derivatives are required then one can use POLEVN1, while for second derivatives use POLEVN2.

123. POLEVN1 Subroutine to evaluate the approximating polynomial and its first derivative using expansion in orthogonal polynomials in N dimensions. This subroutine can be used to calculate the value of approximating polynomial at any point, after the required coefficients as well as other auxiliary coefficients needed to define the orthogonal polynomials have been calculated by subroutine POLFITN. This is the version of POLEVN which also calculates the first derivatives. The arguments are same as those for POLEVN, except for array DF of length N, which will contain the calculated derivatives with respect to each of the dimensions. DF(i) will contain $\partial F/\partial x_i$. The scratch array WK should have a length of at least 3N $\times$ LA + N. This subroutine does not calculate the second derivatives of F. If second derivatives are required then one can use POLEVN2, while if no derivatives are required then use POLEVN.

124. POLEVN2 Subroutine to evaluate the approximating polynomial and its first and second derivatives using expansion in orthogonal polynomials in N dimensions. This subroutine can be used to calculate the value of approximating polynomial at any point, after the required coefficients as well as other auxiliary coefficients needed to define the orthogonal polynomials have been calculated by subroutine POLFITN. This is the version of POLEVN which also calculates the first and second derivatives. The arguments are same as those for POLEVN, except for arrays DF and DDF. DF is a real array of length N which will contain the first derivatives of F at X0. DF(i) will contain $\partial F/\partial x_i$. DDF is a real array of length N^2 which will contain the second derivatives of F at X0. DDF(i, j) will contain $\partial^2 F/\partial x_i \partial x_j$. The first dimension of DDF in the calling program must be equal to N. Further, in this case WK is a real array of length $3\mathrm{N} \times \mathrm{LA} + \mathrm{N} + \mathrm{N}^2$ used as scratch space. This subroutine calculates the first and second derivatives of F. If second derivatives are not required then one can use POLEVN1, while if no derivatives are required then use POLEVN.

125. LLSQ Subroutine to calculate a general linear least squares fit in K dimensions. It uses singular value decomposition (SVD) to solve the system of equations. The tabular points could have arbitrary distribution in K-space and the basis functions also are arbitrary functions to be defined by the user. N is the number of tabular points, M is the number of basis functions, K is the number of dimensions. X is a real array of length IX $\times$ N containing the coordinates of tabular points. X(I, J) is the Ith coordinate of Jth tabular point. IX is the first dimension of X in the calling program (IX $\geq$ K). For fitting in one dimension, IX can be set to 1 and array X can be passed on as a one dimensional array of length N. F is a real array of length N, containing the function values. F(I) is the function value at (X(1, I), X(2, I), ..., X(K, I)). This is treated as one dimensional array in the subroutine and hence should not have any gaps in storage. This allows the routine to be used for a general distribution of points not necessarily along a hyper-rectangular mesh. EF is a real array of length N, containing the estimated error in F. This is only used for determining the weights associated with each points. Thus, it solves the following system of equations

$$\sum_{i=1}^{\mathrm{M}} \frac{\mathrm{A}(i)}{\mathrm{EF}(j)} \phi_i(\mathrm{X}(1,j),\ldots,\mathrm{X}(\mathrm{K},j)) = \frac{\mathrm{F}(j)}{\mathrm{EF}(j)}, \qquad j = 1,2,\ldots,\mathrm{N}; \tag{B.45}$$

using SVD. A is a real array of length N, which will contain the fitted coefficients. Although, there are only M coefficients the array must have a length of at least N, since the remaining elements are used as scratch space. U is a real array of length IU $\times$ M which will contain the matrix U from SVD of the design matrix. V is a real array of length IV $\times$ M which will contain the matrix V from SVD of the design matrix, ($G = U\Sigma V^T$). IU is the first dimension of U in the calling program (IU $\geq$ N). IV is the first dimension of V and COV in the calling program (IV $\geq$ M). SIGMA is a real array of length M, which will contain the singular values of the design matrix. Y is a real array of length N which will contain the fitted values of the function at the tabular points. WK is a real array of length M used as scratch space. PHI is the name of the subroutine to calculate the basis functions at a given point. REPS is the required accuracy for the solution of equations. All singular values less than REPS times the largest singular value will be set to zero during solution. This parameter can be used to eliminate the linear combinations of basis functions that contribute little to the fit. CHISQ is the minimum value of χ^2 defined by

$$\chi^2 = \sum_{i=1}^{\mathrm{N}} \left(\frac{\mathrm{F}(i)}{\mathrm{EF}(i)} - \sum_{j=1}^{\mathrm{M}} \frac{\mathrm{A}(j)}{\mathrm{EF}(i)} \phi_j(\mathrm{X}(1,i), \ldots, \mathrm{X}(\mathrm{K},i)) \right)^2 , \qquad (\mathrm{B}.46)$$

COV is an array of length IV $\times$ M which will contain the covariance matrix between fitted parameters. COV(I, J) is the covariance between A(I) and A(J). The diagonal elements are the variance in fitted parameters. IER is the error parameter, IER $=$ 606 implies that M $>$ N, M ≤ 0, N ≤ 0 or K $>$ IX. IER $=$ 607 implies that EF(I) are not all positive. In both these cases no calculations are done. The SUBROUTINE PHI(M, X, FX) must be supplied by the user to calculate the required basis functions at a given point. Here M is the number of basis functions, X is a real array of length K containing the coordinates of the point at which the basis functions need to be calculated. FX is a real array of length M containing the calculated basis functions at X. FX(I) should give $\phi_I(\mathrm{X})$. For polynomial fits in one dimension $\phi_i(x) = x^{i-1}$, but in that case it may be better to use POLFIT to calculate the fit. Apart from PHI it also needs subroutines SVD and SVDEVL.

126. BSPFIT Subroutine to calculate linear least squares fit to B-spline basis functions in one dimension. It uses singular value decomposition (SVD) to solve the system of equations and also allows regularisation to be incorporated. N is the number of tabular points, X is a real array of length N containing the coordinates of tabular points. F is a real array of length N, containing the function values. F(I) is the function value at X(I). EF is a real array of length N, containing the estimated error in F. This is only used for determining the weights associated with each points. Thus, it solves the following system of equations

$$\sum_{i=1}^{\mathrm{NO+K-2}} \frac{\mathrm{C}(i)}{\mathrm{EF}(j)} \phi_i(\mathrm{X}(j)) = \frac{\mathrm{F}(j)}{\mathrm{EF}(j)} , \qquad j = 1, 2, \ldots, \mathrm{N}; \qquad (\mathrm{B}.47)$$

using SVD. Here $\phi_i(x)$ are the B-spline basis functions. K is the order of B-splines required, K $= 4$ for cubic B-splines and K $= 2$ for linear B-splines, etc. A is a real array of length LA $\times$ (NO $+$ K $-$ 2) which will contain the matrix U from SVD of the design matrix. V is a real array of length IV $\times$ (NO $+$ K $-$ 2) which will contain the matrix V from SVD of the design matrix, ($G = U\Sigma V^T$). LA is the first dimension of A in the calling program, LA $\geq$ N, when RLM ≤ 0 and LA $\geq$ 2N, when RLM > 0. IV is the first dimension of V and COV in the calling program (IV $\geq$ NO + K − 2). SIGMA is a real array of length NO + K − 2, which will contain the singular values of the design matrix. C is a real array of length 2N, which will contain the fitted coefficients. Although, there are only NO + K − 2 coefficients the array must have a length of at least 2N, since the remaining elements are used as scratch space. XF is a real array of length NO containing the knots required to define the B-spline basis functions. NO is the number of knots for defining B-splines. The knots must be distinct and in ascending order with XF(1) containing the first knot. This will yield (NO + K − 2) B-spline basis functions for fitting. Y is a real array of length N which will contain the fitted values of the function at the tabular points. IFLG is an integer variable that specifies what calculation is to be done. For IFLG ≤ 1 the matrix is calculated and its SVD is computed. If execution is successful, IFLG will be set to 2, so that next time the matrix calculations will be skipped. If IFLG ≤ 0 the coefficients of expansion are also calculated and the fitted values Y as well as the CHISQ and COV are computed. If IFLG = 2, only the coefficients of expansion will be calculated using the old SVD available in arrays A, V and SIGMA and the (hopefully new) function values F. For IFLG = 2 the fitted values Y, CHISQ and COV are not calculated. If IFLG = 3, the coefficients of expansion as well as the fitted values Y and CHISQ are calculated using the old SVD. WK is a real array of length 4NO + 5K + 2 used as scratch space. REPS is the required accuracy for the solution of equations. All singular values less than REPS times the largest singular value will be set to zero during solution. This parameter can be used to eliminate the linear combinations of basis functions that contribute little to the fit. RLM is the regularisation parameter λ for smoothing. If $\lambda \leq 0$ no regularisation is applied, while for $\lambda > 0$ regularisation is applied using either first or second derivative. IDE is the integer parameter which specifies the order of derivative to be used for regularisation. This is used only if $\lambda > 0$, in which case it must be either 1 or 2. For IDE = 1 first derivative smoothing is used, while for IDE = 2 second derivative smoothing is applied. The regularisation is applied at all tabular points, making the number of equations 2N. CHISQ is the minimum value of χ^2 defined by

$$\chi^2 = \sum_{i=1}^{\mathrm{N}} \left(\frac{\mathrm{F}(i)}{\mathrm{EF}(i)} - \sum_{j=1}^{\mathrm{NO+K-2}} \frac{\mathrm{A}(j)}{\mathrm{EF}(i)} \phi_j(\mathrm{X}(i)) \right)^2 , \qquad \text{(B.48)}$$

COV is an array of length IV × M which will contain the covariance matrix between fitted parameters. COV(I, J) is the covariance between C(I) and C(J). The diagonal elements are the variance in fitted parameters. IER is the error parameter, IER = 608 implies that NO + K − 2 > N, or K < 2. IER = 609 implies that RLM > 0 but IDE is not 1 or 2. IER = 610 implies that EF(I) are not all positive. In all these cases no calculations are done. Other values of IER may be set by SVD or BSPLIN. This subroutine requires subroutines BSPLIN, BSPEVL, SVD, and SVDEVL.

127. BSPFIT2 Subroutine to calculate linear least squares fit to B-spline basis functions in two dimensions. It uses singular value decomposition (SVD) to solve the system of equations and also allows regularisation to be incorporated, but it is restricted to equal weights for all points. The subroutine solves the system of equations in one dimension at a time, that is why the weights have to be equal. For general least squares solution with varying weights users can try BSPFITW2, which solves the system of equations for 2 dimensions directly and hence will require much more time. NX is the number of tabular points along x-axis. NY is the number of tabular points along y-axis. X and Y are real arrays of length NX, NY containing the coordinates of tabular points. F is a real array of length LA $\times$ NY, containing the function values. F(I, J) is the function value at (X(I), Y(J)). K is the order of B-splines required, K $= 4$ for cubic B-splines and K $= 2$ for linear B-splines, etc. For simplicity, the order is assumed to be the same in both directions. AX is a real array of length LA $\times$ (MX $+$ K $- 2$) which will contain the matrix U from SVD of the design matrix for fit along x-axis. AY is a real array of length LA $\times$ (MY $+$ K $- 2$) which will contain the matrix U from SVD of the design matrix for fit along y-axis. LA is the first dimension of arrays F, AX, AY, C and FY as declared in the calling program. All these arrays should have the same first dimension. LA must be at least $2 \times \max(\mathrm{NX}, \mathrm{NY})$ when RLM > 0, otherwise LA must be at least $\max(\mathrm{NX}, \mathrm{NY})$. VX is a real array of length IV $\times$ (MX $+$ K $- 2$) which will contain the matrix V from SVD of the design matrix for fit along the x-axis. VY is a real array of length IV $\times$ (MY $+$ K $- 2$) which will contain the matrix V from SVD of the design matrix for fit along the y-axis. IV is the first dimension of VX and VY in the calling program ($\mathrm{IV} \geq \max(\mathrm{MX}, \mathrm{MY}) + \mathrm{K} - 2$). SIGMAX is a real array of length MX $+$ K $- 2$, which will contain the singular values of the design matrix for fit along the x-axis. SIGMAY is a real array of length MY $+$ K $- 2$, which will contain the singular values of the design matrix for fit along the y-axis. C is a real array of length LA $\times$ NY, which will contain the fitted coefficients. Although, there are only $(\mathrm{MX} + \mathrm{K} - 2)(\mathrm{MY} + \mathrm{K} - 2)$ coefficients the array must have a larger length, since the remaining elements are used as scratch space. XF is a real array of length MX containing the knots required to define the B-spline basis functions along x. YF is a real array of length MY containing the knots required to define the B-spline basis functions along y. MX is the number of knots for defining B-splines along x-axis. MY is the number of knots for defining B-splines along y-axis. The knots must be distinct and in ascending order with XF(1), YF(1) containing the first knot. FY is a real array of length LA $\times$ NY which will contain the fitted values of the function at the tabular points. WK is a real array of length LA $\times$ NX $+$ NX $+$ NY used as scratch space. REPS is the required accuracy for the solution of equations. All singular values less than REPS times the largest singular value will be set to zero during solution. This parameter can be used to eliminate the linear combinations of basis functions that contribute little to the fit. RLM is the regularisation parameter λ for smoothing. If $\lambda \leq 0$ no regularisation is applied, while for $\lambda > 0$ regularisation is applied using either first or second derivative.

IDE is the integer parameter which specifies the order of derivative to be used for regularisation. This is used only if $\lambda > 0$, in which case it must be either 1 or 2. For IDE = 1 first derivative smoothing is used, while for IDE = 2 second derivative smoothing is applied. The regularisation is applied at all tabular points, making the number of equations 2 times larger. CHISQ is the minimum value of χ^2 defined by

$$\chi^2 = \sum_{i=1}^{\mathrm{NX}} \sum_{j=1}^{\mathrm{NY}} \left(\mathrm{F}(i,j) - \sum_{j1=1}^{\mathrm{MX+K-2}} \sum_{j2=1}^{\mathrm{MY+K-2}} \mathrm{A}(j1,j2)\phi_{j1}(\mathrm{X}(i))\psi_{j2}(\mathrm{Y}(j)) \right)^2 . \tag{B.49}$$

Here $\phi_j(x)$ are the basis functions in x and $\psi_j(y)$ are those in y. IER is the error parameter, IER = 608 implies that $\mathrm{MX}+\mathrm{K}-2 > \mathrm{NX}$, $\mathrm{MY}+\mathrm{K}-2 > \mathrm{NY}$, or $\mathrm{K} < 2$. IER = 609 implies that RLM > 0 but IDE is not 1 or 2. In all these cases no calculations are done. Other values of IER may be set by BSPFIT, SVD or BSPLIN. This subroutine requires subroutines BSPFIT, BSPLIN, BSPEVL, BSPEV2, SVD and SVDEVL.

128. BSPFITW2 Subroutine to calculate linear least squares fit to B-spline basis functions in two dimensions. It uses singular value decomposition (SVD) to solve the system of equations and also allows regularisation as well as nonuniform weights to be incorporated. This subroutine solves the system of equations directly in two dimensions and hence will require much more time as compared to BSPFIT2. NX is the number of tabular points along x-axis. NY is the number of tabular points along y-axis. X and Y are real arrays of length NX, NY containing the coordinates of tabular points. F is a real array of length IC×NY, containing the function values. F(I, J) is the function value at (X(I), Y(J)). EF is a real array of length IC × NY, containing the estimated errors in F. These values are used to choose the weight for each equation for least squares solution. K is the order of B-splines required, K = 4 for cubic B-splines and K = 2 for linear B-splines, etc. For simplicity, order of B-splines is assumed to be the same along each axes. A is a real array of length $\mathrm{LA} \times (\mathrm{MX}+\mathrm{K}-2)(\mathrm{MY}+\mathrm{K}-2)$ which will contain the matrix U from SVD of the design matrix. LA is the first dimension of array A as declared in the calling program. LA must be at least $3\mathrm{NX} \times \mathrm{NY}$ when RLM > 0, otherwise LA must be at least $\mathrm{NX} \times \mathrm{NY}$. V is a real array of length $\mathrm{IV} \times (\mathrm{MX}+\mathrm{K}-2)(\mathrm{MY}+\mathrm{K}-2)$ which will contain the matrix V from SVD of the design matrix. IV is the first dimension of V in the calling program, $\mathrm{IV} \geq (\mathrm{MX}+\mathrm{K}-2)(\mathrm{MY}+\mathrm{K}-2)$. SIGMA is a real array of length $(\mathrm{MX}+\mathrm{K}-2)(\mathrm{MY}+\mathrm{K}-2)$, which will contain the singular values of the design matrix. C is a real array of length $\mathrm{IC} \times (\mathrm{MY}+\mathrm{K}-2)$, which will contain the fitted coefficients. IC is the first dimension of arrays C, F, EF, FY as declared in the calling program, IC must be at least NX. XF is a real array of length MX containing the knots required to define the B-spline basis functions along x. YF is a real array of length MY containing the knots required to define the B-spline basis functions along y. MX is the number of knots for defining B-splines along x-axis. MY is the number of knots for defining B-splines along y-axis. The knots must be distinct and in ascending order with XF(1), YF(1) containing the first knot. FY is a real array of length IC$\times$NY which will contain the fitted values of the function at the tabular points. WK is a real array of length $3\mathrm{NX}\times\mathrm{NY}+(\mathrm{MX}+\mathrm{K})(\mathrm{MY}+\mathrm{K})$ used as scratch space, when $\mathrm{RLM}>0$. If $\mathrm{RLM}<0$ the required length of WK is $\mathrm{NX}\times\mathrm{NY}+(\mathrm{MX}+\mathrm{K})(\mathrm{MY}+\mathrm{K})$. REPS is the required accuracy for the solution of equations. All singular values less than REPS times the largest singular value will be set to zero during solution. This parameter can be used to eliminate the linear combinations of basis functions that contribute little to the fit. RLM is the regularisation parameter λ for smoothing. If $\lambda\le 0$ no regularisation is applied, while for $\lambda>0$ regularisation is applied using either first or second derivative. IDE is the integer parameter which specifies the order of derivative to be used for regularisation. This is used only if $\lambda>0$, in which case it must be either 1 or 2. For $\mathrm{IDE}=1$ first derivative smoothing is used, while for $\mathrm{IDE}=2$ second derivative smoothing is applied. The regularisation is applied at all tabular points, making the number of equations 3 times larger. CHISQ is the minimum value of χ^2 defined by

$$\sum_{i=1}^{\mathrm{NX}}\sum_{j=1}^{\mathrm{NY}}\left(\frac{\mathrm{F}(i,j)-\sum_{j1=1}^{\mathrm{MX+K-2}}\ \sum_{j2=1}^{\mathrm{MY+K-2}}\mathrm{A}(j1,j2)\phi_{j1}(\mathrm{X}(i))\psi_{j2}(\mathrm{Y}(j))}{\mathrm{EF}(i,j)}\right)^2 . \tag{B.50}$$

Here $\phi_j(x)$ are the basis functions in x and $\psi_j(y)$ are those in y. This routine does not calculate the covariance matrix, but this can be easily added following BSPFIT. IER is the error parameter, IER = 608 implies that $\mathrm{MX}+\mathrm{K}-2>\mathrm{NX}$, $\mathrm{MY}+\mathrm{K}-2>\mathrm{NY}$, or $\mathrm{K}<2$. IER = 609 implies that $\mathrm{RLM}>0$ but IDE is not 1 or 2. IER = 610 implies that EF(I, J) are not all positive. In all these cases no calculations are done. Other values of IER may be set by SVD or BSPLIN. This subroutine requires subroutines BSPLIN, BSPEV2, SVD and SVDEVL.

129. BSPFITN Subroutine to calculate linear least squares fit to B-spline basis functions in N dimensions. It uses singular value decomposition (SVD) to solve the system of equations and also allows regularisation to be incorporated, but it is restricted to equal weights for all points. The subroutine solves the system of equations in one dimension at a time, that is why the weights have to be equal. For general least squares solution with varying weights users can try BSPFITWN, which solves the system of equations for N dimensions directly and hence will require much more time. N is the number of dimensions. NK is an integer array of length N containing the number of tabular points along each axis. X is a real array of length LA $\times$ N containing the coordinates of tabular points. X(I, J) is the Ith point along Jth axis. F is a real array of length $\mathrm{NK}(1)\times\mathrm{NK}(2)\times\cdots\times\mathrm{NK}(\mathrm{N})$, containing the function values. $\mathrm{F}(i_1,i_2,\ldots,i_N)$ is the function value at (X(i_1,1), X(i_2, 2), …, X(i_N, N)). This is treated as a one-dimensional array and hence its dimensions in the calling program must exactly match the size in each dimension, e.g., the dimensions could be F(NK(1), NK(2), ..., NK(N)). Alternately, it could be treated as a one dimensional array in the calling program also. K is the order of B-splines required, K $= 4$ for cubic B-splines and K $= 2$ for linear B-splines, etc. For simplicity, the order is assumed to be the same along each dimension. A is a real array of length LA $\times$ IV $\times$ N which will contain the matrix U from SVD of the design matrix for fit along each axis. LA is the first dimension of array X, as declared in the calling program. The first dimension of A should be LA$\times$IV. LA must be at least $2\times\max(\mathrm{NK(I)})$ when RLM > 0, otherwise LA must be at least max(NK(I)). V is a real array of length $\mathrm{IV}^2 \times \mathrm{N}$ which will contain the matrix V from SVD of the design matrix for fit along each axis. IV is the first dimension of XF and SIGMA in the calling program ($\mathrm{IV} \geq \mathrm{K} - 2 + \max \mathrm{MK(I)}$). The first dimension of V in calling program is IV^2. SIGMA is a real array of length IV $\times$ N, which will contain the singular values of the design matrix for fit along each axis. C is a real array of length $\mathrm{NK(1)} \times \mathrm{NK(2)} \times \cdots \times \mathrm{NK(N)}$, which will contain the fitted coefficients. If RLM > 0 this length must be 2 times larger. Note that although there are only $(\mathrm{MK(1)} + \mathrm{K} - 2)(\mathrm{MK(2)} + \mathrm{K} - 2) \cdots (\mathrm{MK(N)} + \mathrm{K} - 2)$ coefficients the array must have a larger length, since the remaining elements are used as scratch space. This is treated as a one-dimensional array and hence its dimensions in the calling program must exactly match the size in each dimension, e.g., the dimensions could be $\mathrm{C}(\mathrm{MK(1)} + \mathrm{K} - 2, \mathrm{MK(2)} + \mathrm{K} - 2, \ldots, \mathrm{MK(N-1)} + \mathrm{K} - 2, \mathrm{NX})$. Here the last dimension has to be increased suitably to accommodate the scratch space required. The last dimension NX should be chosen such that the total size is greater than the required value. If this array is only passed on to BSPEVN or equivalent routines to calculate the function value at any required point, then the exact dimensions in calling program are immaterial as long as the total length is larger than the required value. XF is a real array of length IV $\times$ N containing the knots required to define the B-spline basis functions in each dimension. The knots must be distinct and in ascending order. XF(I, J) is the Ith knot along Jth dimension. MK is an integer array of length N containing the number of knots for defining B-splines along each axis. FY is a real array of same size and shape as F which will contain the fitted values of the function at the tabular points. WK is a real array of length $\mathrm{LA} + \mathrm{NK(1)} \times \mathrm{NK(2)} \times \cdots \times \mathrm{NK(N)}$ used as scratch space. If RLM > 0 then this size must be twice as large. IWK is an integer array of length 3N used as scratch space. REPS is the required accuracy for the solution of equations. All singular values less than REPS times the largest singular value will be set to zero during solution. This parameter can be used to eliminate the linear combinations of basis functions that contribute little to the fit. RLM is the regularisation parameter λ for smoothing. If $\lambda \leq 0$ no regularisation is applied, while for $\lambda > 0$ regularisation is applied using either first or second derivative. IDE is the integer parameter which specifies the order of derivative to be used for regularisation. This is used only if $\lambda > 0$, in which case it must be either 1 or 2. For IDE $= 1$ first derivative smoothing is used, while for IDE $= 2$ second derivative smoothing is applied. The regularisation is applied at all tabular points, making the number of equations 2 times larger. CHISQ is the minimum value of χ^2 obtained using the fitted coefficients. IER is the error parameter. IER = 609 implies that RLM > 0 but IDE is not 1 or 2. In this case no calculations are done. Other values of IER may be set by BSPFIT, SVD or BSPLIN. This subroutine requires subroutines BSPFIT, BSPLIN, BSPEVL, BSPEVN, SVD and SVDEVL.

130. BSPFITWN Subroutine to calculate linear least squares fit to B-spline basis functions in N dimensions. It uses singular value decomposition (SVD) of the full set of equations to obtain the fits and also allows regularisation to be incorporated. It incorporates unequal weights, but can require several orders of magnitude larger time as compared to BSPFITN for the same size of table. The memory required is also much larger. Hence, it should be used only if BSPFITN is not suitable because of highly varying weights. N is the number of dimensions. NK is an integer array of length N containing the number of tabular points along each axis. X is a real array of length LA × N containing the coordinates of tabular points. X(I, J) is the Ith point along Jth axis. F is a real array of length NK(1) × NK(2) × ⋯ × NK(N), containing the function values. F$(i_1, i_2, \ldots, i_N)$ is the function value at (X(i_1, 1), X(i_2, 2), ..., X(i_N, N)). This is treated as an one-dimensional array and hence its dimensions in the calling program must exactly match the size in each dimension, e.g., the dimensions could be F(NK(1), NK(2), ..., NK(N)). EF is a real array of same size and shape as F, containing the estimated errors in F. K is the order of B-splines required, K = 4 for cubic B-splines and K = 2 for linear B-splines, etc. For simplicity, the order is assumed to be the same along each dimension. A is a real array of length

$$\begin{aligned}&\mathrm{NK(1)NK(2)\cdots NK(N)}\\ &\qquad\times(\mathrm{MK(1)+K-2})(\mathrm{MK(2)+K-2})\cdots(\mathrm{MK(N)+K-2}),\end{aligned} \tag{B.51}$$

which will contain the matrix U from SVD of the design matrix. If RLM > 0 the required size will be N + 1 times larger. LA is the first dimension of array X, as declared in the calling program. LA must be at least max(NK(I)). V is a real array of length

$$[(\mathrm{MK(1)+K-2})(\mathrm{MK(2)+K-2})\cdots(\mathrm{MK(N)+K-2})]^2, \tag{B.52}$$

which will contain the matrix V from SVD of the design matrix. IV is the first dimension of XF in the calling program (IV $\geq$ max(MK(I))). SIGMA is a real array of length $(\mathrm{MK(1)+K-2})(\mathrm{MK(2)+K-2})\cdots(\mathrm{MK(N)+K-2})$, which will contain the singular values of the design matrix. C is a real array of length NK(1) × NK(2) × ⋯ × NK(N), which will contain the fitted coefficients. If RLM > 0 this length must be N + 1 times larger. Note that although there are only $(\mathrm{MK(1)+K-2})(\mathrm{MK(2)+K-2})\cdots(\mathrm{MK(N)+K-2})$ coefficients the array must have a larger length, since the remaining elements are used as scratch space. This is treated as a one-dimensional array and hence its dimensions in the calling program must exactly match the size in each dimension, e.g., the dimensions could be

$C(MK(1) + K - 2, MK(2) + K - 2, \ldots, MK(N - 1) + K - 2, NX)$.
Here the last dimension has to be increased suitably to accommodate the scratch space required. The last dimension NX should be chosen such that the total size is greater than the required value. If this array is only passed on to BSPEVN or equivalent routines to calculate the function value at any required point, then the exact dimensions in calling program are immaterial as long as the total length is larger than the required value. XF is a real array of length IV $\times$ N containing the knots required to define the B-spline basis functions in each dimension. The knots must be distinct and in ascending order. XF(I, J) is the Ith knot along Jth dimension. MK is an integer array of length N containing the number of knots for defining B-splines along each axis. FY is a real array of same size and shape as F which will contain the fitted values of the function at the tabular points. WK is a real array of length $N \times (LA + 6) + LA + 2K + 2$ used as scratch space. IWK is an integer array of length 3N used as scratch space. REPS is the required accuracy for the solution of equations. All singular values less than REPS times the largest singular value will be set to zero during solution. This parameter can be used to eliminate the linear combinations of basis functions that contribute little to the fit. RLM is the regularisation parameter λ for smoothing. If $\lambda \leq 0$ no regularisation is applied, while for $\lambda > 0$ regularisation is applied using either first or second derivative. IDE is the integer parameter which specifies the order of derivative to be used for regularisation. This is used only if $\lambda > 0$, in which case it must be either 1 or 2. For IDE $= 1$ first derivative smoothing is used, while for IDE $= 2$ second derivative smoothing is applied. The regularisation is applied at all tabular points, making the number of equations $N + 1$ times larger. CHISQ is the minimum value of χ^2 obtained using the fitted coefficients. IER is the error parameter. IER $= 608$ implies that LA or IV are not large enough to store the required quantities. IER $= 609$ implies that RLM > 0 but IDE is not 1 or 2. In these cases no calculations are done. Other values of IER may be set by SVD or BSPLIN. This subroutine requires subroutines BSPLIN, BSPEVN, SVD and SVDEVL.

131. LINFITXY Subroutine to calculate the least squares straight line fit when there is error in both x and y values, using the technique described in Section 10.2.4. For simplicity it is assumed that all data points have the same errors and correlation. N is the number of data points. X and Y are real arrays of length N, specifying the data values. Fit of the form $Y(I) = a + b \times X(I)$ is to be calculated. SIGX and SIGY are the estimated errors in X and Y values, while RHO is the correlation between the errors in X and Y. XI and YI are the arrays which will contain the fitted values of X and Y. Since both X and Y have errors fitted values of both need to be calculated. A and B are the fitted values of the intercept and slope, respectively. Thus $YI(I) = A + B \times XI(I)$. CHI is the value of χ^2 at the fit as defined by Eq. (10.49). IER is the error parameter. IER $= 617$ implies that the discriminant of the quadratic (10.59) is negative and the calculations are aborted.

132. NLLSQ Subroutine to calculate the χ^2 function for a nonlinear least squares fit for use with subroutine BFGS. By suppressing the gradient calculations it can also be used with subroutine NMINF. This is the function to be minimised by BFGS. N is the number of parameters to be fitted, A is a real array of length N containing the values of the parameters at which the function is required to be calculated. F is the calculated value of the function. G is a real array of length N, containing the calculated derivatives. G(i) will contain $\frac{\partial F}{\partial a_i}$. The data points are passed through common block ZZFUN, which must be initialised in the calling program before calling BFGS. The parameter NP must be set to a value matching the array sizes in the calling program. The common block contains the following variables: X, FX, EF are real arrays of length NP containing respectively, the values of abscissas, function values and estimated errors in function values at each point. NN is the number of data points in the table, NN $\leq$ NP. FX1 is a real array of length NP which will contain the fitted value of the function at all points. Although EF(I) should contain the estimated error in FX(I), in many cases it is found that multiplying all elements of EF by suitable constant improves the convergence of BFGS significantly without changing the minimum. Only the value of χ^2 will need to be scaled suitably. This subroutine needs FCN to calculate the function value at any X for specified values of the parameters. It may be noted that there is no provision to pass on the name of a subroutine to this subprogram and hence the name has to be explicitly changed in the subroutine. SUBROUTINE FCN(N, A, X, F, DF) must be supplied by the user. Here N is the number of parameters, A is a real array of length N containing the values of parameters. X is a real variable specifying the point where the function value needs to be calculated. F is a real variable which should contain the calculated function value F(X, A), while DF is a real array of length N containing the derivatives of F. DF(I) should give $\frac{\partial F}{\partial A_i}$. By suppressing the derivative calculations NLLSQ can also be used with subroutine NMINF. The corresponding version may be found in NLLSQ_F.

133. DFT Subroutine to calculate the discrete Fourier transform (DFT) using normal sum. This program is applicable to arbitrary number of points, but it requires $O(N^2)$ arithmetic operations, and hence should be used only when N is relatively small. N is the number of data points. CG is a complex array of length N, which should contain the data points. CF is a complex array of length N which will contain the Fourier transform of CG. IFLG is a flag, if IFLG ≥ 0 the DFT is calculated, while if IFLG < 0 the inverse DFT will be calculated. IER is the error parameter. IER $= 611$ implies that N < 2, in which case, no calculations are performed. It may be noted that when inverse transform is calculated the result will need to be divided by N to match the original data. The subroutine does not perform the division. It may be noted that even while calculating the inverse transform the input should be provided in array CG and calculated transform will be available in array CF.

134. FFT Subroutine to calculate the discrete Fourier transform (DFT) using a FFT algorithm. N is the number of data points which must be equal to a power of 2. CG is a complex array of length N, which should contain the data points when calling the subroutine. After execution, the DFT will be overwritten on the same array CG. Hence, if necessary a copy of the original data should be preserved for later use before calling the subroutine. IFLG is a flag, if IFLG ≥ 0 the DFT is calculated, while if IFLG < 0 the inverse DFT will be calculated. IER is the error parameter. IER = 611 implies that N < 2, in which case, no calculations are performed. IER = 631 implies that N is not a power of 2. This test is performed towards the end of the calculation and hence in this case, the contents of array CG will be destroyed. It may be noted that when inverse transform is calculated the result will need to be divided by N to match the original data. The subroutine does not perform the division.

135. FFTR Subroutine to calculate the discrete Fourier transform (DFT) of real data using a FFT algorithm. N is the number of data points which must be equal to a power of 2. CG is a complex array of length N/2, which should contain the data points when calling the subroutine. The data should be stored in the natural order with REAL(CG($j+1$)) $= g_{2j}$ and AIMAG(CG($j+1$)) $= g_{2j+1}$ for $j = 0, 1, \ldots, \mathrm{N}/2 - 1$. In fact, in the calling program, the array CG can be treated as a real array of length N. After execution, the DFT will be overwritten on the same array CG as explained in Section 10.6. If necessary, another copy of the original data should be preserved for later use before calling the subroutine. IFLG is a flag. If IFLG ≥ 0 the DFT is calculated, while if IFLG < 0 the inverse DFT will be calculated. IER is the error parameter. IER = 611 implies that N < 4 and no calculations are done. IER = 631 implies that N is not a power of 2. Since this test is performed towards the end of calculations, the contents of array CG will be destroyed. It may be noted that when inverse transform is calculated the result will need to be divided by N/2 to match the original data. The subroutine does not perform the division. This subroutine requires subroutine FFT for calculating the DFT of complex data.

136. FFTN Subroutine to calculate the discrete Fourier transform (DFT) in n dimensions using a FFT algorithm. ND is the number of dimensions. NN is an integer array of length ND. NN(I) is the number of data points along the Ith coordinate, which must be equal to a power of 2. CG is a complex array of length NN(1) $\times$ NN(2) $\times \ldots \times$ NN(ND), which should contain the data points when calling the subroutine. The data should be stored in the normal Fortran order with

$$\begin{aligned}\mathrm{CG}\,(1 + j_1 + j_2\mathrm{NN}(1) + j_3\mathrm{NN}(1)\mathrm{NN}(2) + \cdots + j_{ND}\mathrm{NN}(1)\ldots\mathrm{NN}(\mathrm{ND}-1))\\ = g_{j_1,j_2,\ldots,j_{ND}}\,,\end{aligned} \tag{B.53}$$

for $0 \leq j_r < \mathrm{NN}(r) - 1$. In fact, in the calling program CG can be treated as a ND-dimensional complex array with dimension CG(NN(1), NN(2), ..., NN(ND)) and CG($j_1+1, j_2+1, \ldots, j_n+1$) $= g_{j_1,j_2,\ldots,j_n}$. It may be noted that the dimensions of this ND-dimensional array must be exactly equal to the number of data points in the corresponding variables. After execution, the DFT will be overwritten on the same array CG. Hence, if necessary a copy of original data should be preserved for later use, before calling the subroutine. IFLG is a flag. If IFLG ≥ 0 the DFT is calculated, while if IFLG < 0 the inverse DFT will be calculated. IER is the error parameter. IER $= 631$ implies that at least one of the NN(I), $(\mathrm{I} = 1, 2, \ldots, \mathrm{ND})$ is not a power of 2. Since this test is performed towards the end of calculations, the contents of array CG will be destroyed. It may be noted that when inverse transform is calculated the result will need to be divided by $\mathrm{NN}(1) \times \mathrm{NN}(2) \times \cdots \times \mathrm{NN}(\mathrm{ND})$ to match the original data. The subroutine does not perform the division.

137. LAPINV Subroutine to calculate the inverse Laplace transform of a given function $F(s)$. N is the number of points at which the value of the inverse function is required. T is a real array of length N containing the points t_i, at which the inverse transform is required. The elements t_i need not be in any order, but the last element $\mathrm{t_N}$ should be the largest or close to the largest, since this element is used to control the value of T_0 as explained in Section 10.8. F is a real array of length N, which will contain the value of the required function at t_i after execution of the subroutine. CFS is the name of the function routine used to calculate the function $F(s)$. ALPHA is an estimate for the exponential order of the function $f(t)$ as explained in Section 10.8. REPS is the convergence parameter. The results are normally expected to have a relative accuracy of REPS. However, as explained in the text, this is not guaranteed when the convergence is slow. The results can be improved by either increasing the value of the parameter NMAX in the subroutine, or by removing the discontinuity which is causing the slow convergence. The second alternative will be more effective. IER is the error parameter. IER $= 61$ implies that the ϵ-algorithm failed to converge for at least one of the points. IER $= 62$ implies that the ϵ-algorithm encountered a zero denominator at some stage. Since only the last value of IER will be retained, the error flag may be misleading in those cases, where failure has occurred at more than one points. FUNCTION CFS(CS) must be supplied by the user. Here both CFS and CS are complex variables.

138. POLD Function routine to evaluate a polynomial and its derivatives at any point. N is the degree of polynomial. A is a real array of length $\mathrm{N} + 1$ containing the coefficients of the polynomial. A(1) should contain the constant term and A(N+1) should be the coefficient of $\mathrm{X^N}$. X is the point at which polynomial is to be evaluated. ND is the number of derivatives to be evaluated. It would evaluate the first ND derivatives. The first derivative is always evaluated, irrespective of the value of ND. PD is a real array of length ND, which will contain the computed values of the derivatives. PD(I) will contain the Ith derivative of polynomial. The value of polynomial is returned as POLD.

139. RMK Function routine to evaluate a rational function at any point. M is the degree of numerator, while K is the degree of denominator. A is a real array of length $\mathrm{M} + 1$ containing the coefficients of the polynomial in the numerator. A(1) should contain the constant term and A(M+1) should be the coefficient of $\mathrm{X^M}$. B is a real array of length $\mathrm{K} + 1$ containing the coefficients of the polynomial in the denominator. B(1) should contain the constant term and

B(K+1) should be the coefficient of X^K. X is the point at which the rational function is to be evaluated. The value of rational function is returned as RMK.

140. RMK1 Function routine to evaluate a rational function at any point. This is the same as RMK, except that the constant term for polynomial in the denominator is assumed to be one and hence is not supplied. M is the degree of numerator, while K is the degree of denominator. A is a real array of length M + 1 containing the coefficients of the polynomial in the numerator. A(1) should contain the constant term and A(M+1) should be the coefficient of X^M. B is a real array of length K containing the coefficients of the polynomial in the denominator. B(I) should contain the coefficient of X^I. X is the point at which the rational function is to be evaluated. The value of rational function is returned as RMK1.

141. RMKD Function routine to evaluate a rational function and its first derivative at any point. M is the degree of numerator, while K is the degree of denominator. A is a real array of length M + 1 containing the coefficients of the polynomial in the numerator. A(1) should contain the constant term and A(M+1) should be the coefficient of X^M. B is a real array of length K + 1 containing the coefficients of the polynomial in the denominator. B(1) should contain the constant term and B(K+1) should be the coefficient of X^K. X is the point at which the rational function is to be evaluated. DF will contain the computed value of the derivative. The value of rational function is returned as RMKD.

142. RMKD1 Function routine to evaluate a rational function and its first derivative at any point. This is the same as RMKD, except that the constant term for polynomial in the denominator is assumed to be one and hence is not supplied. M is the degree of numerator, while K is the degree of denominator. A is a real array of length M + 1 containing the coefficients of the polynomial in the numerator. A(1) should contain the constant term and A(M+1) should be the coefficient of X^M. B is a real array of length K containing the coefficients of the polynomial in the denominator. B(I) should contain the coefficient of X^I. X is the point at which the rational function is to be evaluated. DF will contain the computed value of the derivative. The value of rational function is returned as RMKD1.

143. PADE Subroutine to calculate the coefficients of Padé approximation $R_{mk}(x)$ from the known coefficients of Maclaurin series. M and K are the degrees of polynomials in the numerator and the denominator, respectively. A is a real array of length M + K + 1, which will contain the coefficients of the Padé approximation. A(i), ($i = 1, \ldots, \mathrm{K}$) is the coefficient of x^i in the denominator, the constant term being assumed to be unity, A(K + i + 1), ($i = 0, 1, \ldots, \mathrm{M}$) is the coefficient of x^i in the numerator, which gives the Padé approximation

$$R_{\mathrm{MK}}(x) = \frac{\mathrm{A}(\mathrm{K}+1) + \mathrm{A}(\mathrm{K}+2)x + \mathrm{A}(\mathrm{K}+3)x^2 + \cdots + \mathrm{A}(\mathrm{K}+\mathrm{M}+1)x^{\mathrm{M}}}{1 + \mathrm{A}(1)x + \mathrm{A}(2)x^2 + \cdots + \mathrm{A}(\mathrm{K})x^{\mathrm{K}}} \,, \tag{B.54}$$

C is a real array of length M+K+1 containing the coefficients of the Maclaurin series for the required function. C$(i+1)$ should contain the coefficient of x^i in the Maclaurin series. These coefficients must be supplied by the user. IER is the error parameter. IER = 612 implies that either M < 0 or K < 0, in which case no calculations are performed. Other values of IER may be set by the subroutine GAUELM, which is called to solve the system of linear equations. WK is a real array of length K^2 used as a scratch space to store intermediate quantities. IWK is an integer array of length K, used as a scratch space to store intermediate quantities. This subroutine requires subroutine GAUELM to solve the system of linear equations.

144. CHEBCF Subroutine to convert a power series into a series of Chebyshev polynomials and vice versa. N is the degree of polynomial. C and P are real arrays of length N + 2 containing respectively, the coefficients of the Chebyshev and the power series expansions. C$(i+1)$ is the coefficient of $T_i(x)$ in Chebyshev expansion, while P$(i+1)$ is the coefficient of x^i in the power series. It should be noted that C(1) is the coefficient of $T_0(x)$ and is not doubled as in the normal Chebyshev expansions. IFLG is the flag which decides the type of conversion required. If IFLG = 0, then coefficients of Chebyshev expansion will be calculated. In that case, the power series coefficients must be supplied. After execution, the array C will contain the coefficients of Chebyshev expansion, while the array P is unaffected. If IFLG $\neq 0$, then the coefficients of power series will be calculated. In that case, the coefficients of Chebyshev expansion must be supplied and after execution, the array P will contain the coefficients of power series, while the contents of C will be destroyed, since this array is used as a scratch space by the subroutine.

145. CHEBEX Subroutine to calculate the coefficients of Chebyshev expansion of a function that can be evaluated at any required point. It uses orthogonality of Chebyshev polynomials over a set of discrete points to find the coefficients and the value will only be approximately correct.

$$\mathrm{FUN}(x) = \frac{1}{2}c_0 + \sum_{i=1}^{\mathrm{N}-1} c_i T_i(x). \tag{B.55}$$

N is the number of coefficients required. This number should be much larger than the actual number of coefficients needed. The accuracy of computed coefficients increases with N. There is no check to test the accuracy and it has to be ascertained by recomputing the coefficients with larger N (say 2N) and comparing the two values. C is a real array of length N, which will contain the computed coefficients. FUN is the name of the function routine supplied to calculate the required function. IER is the error parameter. IER = 613 implies that N < 10, in which case no calculations are done. Function FUN(X) must be supplied by the user.

146. CHEBAP Subroutine to calculate the coefficients of Rational function Chebyshev approximations $T_{mk}(x)$ from the known coefficients of expansion in Chebyshev polynomials. It should be noted that, this subroutine does not generate minimax approximations, but if the coefficients of Chebyshev expansion fall off rapidly, then the approximation will be close to minimax. M and K are the degrees of polynomials in the numerator and the denominator, respectively. A is a real array of length $\mathrm{M}+\mathrm{K}+1$, which will contain the coefficients of the rational function approximation. $\mathrm{A}(i)$, $(i=1,\ldots,\mathrm{K})$ is the coefficient of $T_i(x)$ in the denominator, the constant term being assumed to be unity, $\mathrm{A}(\mathrm{K}+i+1)$, $(i=0,1,\ldots,\mathrm{M})$ is the coefficient of $T_i(x)$ in the numerator, which gives the approximation $R_{\mathrm{MK}}(x)$ as

$$\frac{\mathrm{A}(\mathrm{K}+1)+\mathrm{A}(\mathrm{K}+2)T_1(x)+\mathrm{A}(\mathrm{K}+3)T_2(x)+\cdots+\mathrm{A}(\mathrm{K}+\mathrm{M}+1)T_{\mathrm{M}}(x)}{1+\mathrm{A}(1)T_1(x)+\mathrm{A}(2)T_2(x)+\cdots+\mathrm{A}(\mathrm{K})T_{\mathrm{K}}(x)}\,. \tag{B.56}$$

C is a real array of length $\mathrm{M}+2\mathrm{K}+1$, containing coefficients of the Chebyshev series for the required function. $\mathrm{C}(i+1)$ should contain the coefficient of $T_i(x)$ in the Chebyshev series. These coefficients must be supplied by the user. Following the usual convention of Chebyshev expansion, the coefficient of $T_0(x)$ in the expansion is C(1)/2. IER is the error parameter. IER = 612 implies that either $\mathrm{M}<0$ or $\mathrm{K}<0$, in which case, no calculations are performed. Other values of IER may be set by the subroutine GAUELM, which is called to solve the system of linear equations. WK is a real array of length K^2, used as a scratch space to store intermediate quantities. IWK is an integer array of length K, used as a scratch space to store intermediate quantities. This subroutine requires subroutine GAUELM to solve the system of linear equations.

147. REMES Subroutine to calculate the minimax rational function approximation for a given function over a finite interval, using the second algorithm of Remes. M and K are the degrees of polynomials in the numerator and the denominator, respectively. N is the number of points which will be used for initial scan of extrema in the error curve. This number should be at least $3(\mathrm{M}+\mathrm{K}+1)$, in order to be able to isolate different extrema. XL and XU are respectively, the lower and upper limits of the interval over which the approximation is required. A is a real array of length $\mathrm{M}+\mathrm{K}+2$, which will contain the coefficients in the same form as that given by the subroutine PADE. $\mathrm{A}(i)$, $(i=1,2,\ldots,\mathrm{K})$ is the coefficient of x^i in the denominator, the constant term being unity, while $\mathrm{A}(\mathrm{K}+i+1)$, $(i=0,1,\ldots,\mathrm{M})$ is the coefficient of x^i in the numerator. If IFLG = 0, then the initial guess for these coefficients must be supplied. X and F are arrays of length N with F(I) containing the value of the function at X(I). These values need not be stored before calling the subroutine, since the subroutine itself selects a uniform mesh and calculates the value of the function at the required points. EX is a real array of length $\mathrm{M}+\mathrm{K}+5$ containing the extrema of the error curve. If IFLG = 2, then the initial guess for the extrema must be supplied, otherwise these values are not required. In all cases, after execution, EX will contain the extrema of the error curve for the final approximation. IE is the number of extrema, which should be $\mathrm{M}+\mathrm{K}+2$, if the error curve is of the standard form. Some types of nonstandard error curves can be handled by this subroutine (Example 10.11). EMAX is the maximum error in the final approximation calculated by the subroutine. EPS is the required tolerance. The iteration for calculating the coefficients of rational function is continued until the maximum error differs by less than EPS. The Remes iteration is continued until the difference between different extrema of error curve is less than 1% of the maximum error. EPSM specifies the tolerance for finding the extrema of the error curve. In general, it is found that a moderate value for EPS and EPSM is enough to find approximations even to very high accuracy. IFLG is an integer parameter, which specifies the nature of initial approximation for the Remes algorithm. If IFLG $= 0$, then the iteration is started from a known initial approximation. In this case, the coefficients of initial approximation must be supplied in the array A. If IFLG $= 1$, then no initial approximation is required and the first iteration is performed by assuming that the extrema of error curves are given by those of $T_{\mathrm{M+K+1}}(x)$. This is the most useful case, if no approximation of the right form and with the correct number of extrema is known. If IFLG $= 2$, then iteration is started with an initial approximation for the extrema of the error curve. In this case, the approximate location of the extrema must be supplied in the array EX and IE ($\geq \mathrm{M}+\mathrm{K}+2$) should be set equal to the number of extrema. If the error curve is expected to be nonstandard, then it will be preferable to supply only $\mathrm{M}+\mathrm{K}+2$ extrema, where the error is expected to be the largest and is alternating in sign. IER is the error parameter. IER $= 614$ implies that either $\mathrm{M} < 0$, or $\mathrm{K} < 0$, or $\mathrm{XU} \leq \mathrm{XL}$, or $\mathrm{M}+\mathrm{K}+2 > \mathrm{NMAX}$, in which case, no calculations are performed. The last requirement arises because of the dimensions of array AA in the common block, which is used to transfer the coefficients to an auxiliary function routine for finding extrema of the function. IER $= 632$ implies that the Remes iteration failed to converge to the specified accuracy in NIT ($= 30$) iterations. This failure could be due to roundoff error, or because the starting values are not sufficiently close. IER $= 633$ implies that at some stage the error curve does not have the required number of extrema and hence the iteration cannot proceed further. This condition does not necessarily imply that the error curve is nonstandard. Apart from these, other values of IER may be set by the subroutine BRENTM which is called to find the extrema, or the subroutine GAUELM which is called to solve the system of linear equations. WK is a real array of length $(\mathrm{K}+\mathrm{M}+2)^2$ used as a scratch space. IWK is an integer array of length $\mathrm{M}+\mathrm{K}+2$, used as a scratch space to store intermediate numbers. This subroutine requires the subroutines BRENTM and GAUELM and function routines FM, FUN and FUND. FUNCTION FUN(X) calculates the required function, while FUNCTION FUND(X) calculates the weight function. Here X, FUN and FUND are real variables. The subroutine generates approximation of the form $\mathrm{FUN}(X) \approx \mathrm{FUND}(X) R_{\mathrm{MK}}(X)$. If $\mathrm{FUND}(x) = 1$ and $\mathrm{FUN}(x) = f(x)$, then the subroutine will calculate minimax approximation to $f(x)$ with respect to the absolute error, while if $\mathrm{FUND}(x) = 1/f(x)$

and FUN$(x) = 1$, then the approximation will be obtained with respect to the relative error, provided $f(x) \neq 0$ throughout the required interval. It should be noted that the names of the function routines FUN and FUND are fixed and cannot be changed. This is forced by the fact that Fortran does not allow external function names to be passed via the common block and the subroutine BRENTM does not pass any other argument to the function routine, which it uses for minimisation. This defect can be corrected by modifying BRENTM to pass on extra arguments to the function routine which is called for minimisation. (This approach has been followed in subroutine BFGS, which needs to call line search routine LINMIN.) The functions FUN and FUND must be supplied by the user. The common block ZZFN is used to pass on parameters to FUNCTION FM(X), which calculates the function to be minimised. In the common block AA is a real array of length NMAX which contains the coefficients of rational function approximation. SI is a real variable which is set to positive value if minimum is to be found and to negative value when maximum is to be found. While SI > 10 implies that function value, FUN(X), is known and need not be calculated. MM and KK are integer variables which are equal to M and K respectively. The rational function approximation can be computed at any required value of X, using function RMK1(M, K, A(K+1), A, X).

148. FM Function subroutine to be used with subroutine REMES. This routine is called by BRENTM to locate the extrema of error curve. It calculates the difference between the actual function and the rational function approximation weighted by the required weight function. This difference is multiplied by SI which is set to a negative value if we want to find a maxima. For initial scan SI ≥ 10, in which case, function value is not calculated, since it is already known.

149. GAMMA Function routine to calculate the Gamma function for a real argument

$$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}\,dt\,, \qquad (x > 0). \tag{B.57}$$

The computed value should generally have relative accuracy of 10^{-15}. The routine first calculates the value of Gamma function at $|x|$. The values for negative argument are then calculated using

$$-x\Gamma(-x)\Gamma(x) = \frac{\pi}{\sin(\pi x)}\,. \tag{B.58}$$

For $x > 1000$ the Stirling's formula is used to calculate the function value. For $8 < x \leq 1000$ a rational function approximation of form

$$\log\Gamma(x) = (x - \frac{1}{2})\log x - x + \log(\sqrt{2\pi}) + \frac{1}{x}R_{mk}(1/x^2), \tag{B.59}$$

is used. For $0 < x \leq 8$ the range is first translated to $[2, 3]$ using $\Gamma(x+1) = x\Gamma(x)$ and then a rational function approximation over $[2, 3]$ is used to approximate the function value. This routine does not check for overflow or invalid arithmetic operations. Since $\Gamma(x)$ increases very rapidly with x, the function evaluation will lead to overflow for x larger than approximately 170 on a 53 bit (REAL*8) arithmetic. Similarly, $\Gamma(x)$ diverges when x is a negative integer or zero. If overflow is expected, then it may be better to use GAMMAL instead which gives the logarithm of $\Gamma(x)$.

150. GAMMAL Function routine to calculate the natural logarithm of Gamma function for a real argument. It uses the same approximations as those used by Function GAMMA, except that for large arguments ($|x| > 8$), it directly calculates the logarithm and hence should not give overflow. It gives $\log(|\Gamma(X)|)$ and hence sign should be accounted separately.

151. ERF Function routine to calculate the Error function

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\,dt\,, \tag{B.60}$$

for a real argument. This routine uses rational function approximations of the form

$$\begin{aligned} &\mathrm{erf}(x) \approx xR_{mk}(x^2), \qquad (0 \le x < 2);\\ &\mathrm{erf}(x) \approx 1 - \tfrac{e^{-x^2}}{x} R'_{m'k'}(\tfrac{1}{x^2}), \qquad (2 \le x < \infty). \end{aligned} \tag{B.61}$$

While for $x < 0$ it uses $\mathrm{erf}(-x) = -\mathrm{erf}(x)$ to evaluate the function value. Each of the rational function approximation has relative accuracy of better than 10^{-15}. For large x the value may be indistinguishable from 1.

152. ERFC Function routine to calculate the complementary Error function

$$\mathrm{erfc}(x) = 1 - \mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2}\,dt\,, \tag{B.62}$$

for a real argument. This routine uses the same approximations as those used by ERF(X). The value should generally have relative accuracy of 10^{-15}. For small arguments the value of function may not be distinguishable from unity, while for large negative arguments it may be indistinguishable from 2.

153. BJ0 Function routine to calculate the Bessel function of the first kind of order zero, $J_0(x)$ for a real argument. For $|x| < 8$ it uses a rational function approximation to $J_0(x) \approx R_{mk}(x^2)$, while for larger arguments it uses the asymptotic form

$$J_0(x) = \sqrt{\frac{2}{\pi x}} \left(P_0(x)\cos(x - \frac{\pi}{4}) - Q_0(x)\sin(x - \frac{\pi}{4}) \right). \tag{B.63}$$

The functions $P_0(x)$ and $Q_0(x)$ are approximated by rational function approximations of form

$$P_0(x) \approx R_{mk}(1/x^2); \qquad Q_0(x) \approx \frac{1}{x} R_{m'k'}(1/x^2). \tag{B.64}$$

All the rational function approximations have absolute accuracy of better than 10^{-15}, but the resulting function values may not have the same relative accuracy at all points. Near the zeros of $J_0(x)$ the relative accuracy would be lower. For large arguments, $|x| > 8$, where the asymptotic formula is used there could be some cancellation between the two terms and consequent loss of significant figures close to zeros of $J_0(x)$.

154. BJ1 Function routine to calculate the Bessel function of the first kind of order one, $J_1(x)$ for a real argument. For $|x| < 8$ it uses a rational function approximation to $J_1(x) \approx xR_{mk}(x^2)$, while for larger arguments it uses the asymptotic form

$$J_1(x) = \sqrt{\frac{2}{\pi x}} \left(P_1(x) \cos(x - \frac{3\pi}{4}) - Q_1(x) \sin(x - \frac{3\pi}{4}) \right). \tag{B.65}$$

The functions $P_1(x)$ and $Q_1(x)$ are approximated by rational function approximations of form

$$P_1(x) \approx R_{mk}(1/x^2); \qquad Q_1(x) \approx \frac{1}{x} R_{m'k'}(1/x^2). \tag{B.66}$$

All the rational function approximations have absolute accuracy of better than 10^{-15}, but the resulting function values may not have the same relative accuracy at all points. Near the zeros of $J_1(x)$ the relative accuracy would be lower. For large arguments, $|x| > 8$, where the asymptotic formula is used there could be some cancellation between the two terms and consequent loss of significant figures close to zeros of $J_1(x)$.

155. BJN Subroutine to calculate the Bessel function of the first kind of integral order, $J_n(x)$ for a real argument. This routine calculates $J_k(x)$ for k between 0 and n. Since a recurrence relation is used to calculate these values, all the values are returned. Here N is the maximum order of Bessel functions required. For positive N, $J_k(x)$ with positive $k = 0, 1, \ldots, \mathrm{N}$ will be calculated. For negative N, $J_k(x)$ with negative $k = 0, -1, \ldots, -\mathrm{N}$ are calculated. XB is the argument at which the function values are required. BJ is a real array which will contain the computed values of the Bessel functions. $\mathrm{BJ}(|i|+1) = J_i(\mathrm{XB})$. Since this array is also used as scratch space the length of the array must be at least, $|\mathrm{N}|+16+\max(25, 5\sqrt{|N|})$. For $|\mathrm{N}| < |\mathrm{XB}|$ the recurrence relation is stable in forward direction and hence it is used to calculate $J_k(x)$ in a straightforward manner using the values of $J_0(x)$ and $J_1(x)$. For smaller XB, the recurrence relation is unstable in forward direction and hence is used in backward direction as explained in {2.38} In this case for some suitable n_1, we start using $J_{n_1} = 0$ and $J_{n_1-1} = 1$ and calculate lower order functions using the recurrence relation

$$J_{k-1}(x) = \frac{2k}{x} J_k(x) - J_{k+1}(x), \qquad k = n_1 - 1, \ldots, 2, 1. \tag{B.67}$$

The normalisation is then found using the relation

$$J_0(x) + 2 \sum_{k=1}^{\infty} J_{2k}(x) = 1. \tag{B.68}$$

Thus we can divide all values by the calculated sum to obtain the correct values for $J_k(x)$. To use this technique we need to select a suitably large value of n_1 to start the recurrence relation. Accuracy of resulting function values can be checked by comparing the value of $J_{n+1}(x)$ without normalisation. The reciprocal of this value will give an estimate of accuracy achieved by this process. Thus if the computed value of $J_{n+1}(x)$ is not large enough an error message is printed out by the routine. This should not normally happen, but in case for some combination of N and XB, the chosen value of n_1 is not large enough, the error message will be printed out. In such cases the choice of N1 in the routine should be increased suitably. In principle, this technique can be used for all values of x, but n_1 has to be larger than x and hence for large x, we will need lot of computation using this technique. Further, the use of the recurrence relation in this form results in overflow even when the required values are within the range of computer arithmetic. This problem is more severe at low x, where the values can be easily computed using the series expansion. Thus we use the series expansion for $x \leq 4$ for computing $J_n(x)$ and $J_{n-1}(x)$. After that the recurrence relation is used in backward direction to get other values. The overflow problem becomes very acute when single precision arithmetic is used and it is not recommended to use this routine in single precision. The accuracy attained by this routine will depend on the values of N and XB, but in general we expect accuracy of order of 10^{-15} which is the accuracy with which $J_0(x)$ and $J_1(x)$ are calculated. This routine needs BJ0 and BJ1 to calculate $J_0(x)$ and $J_1(x)$.

156. BY0 Function routine to calculate the Bessel function of the second kind of order zero, $Y_0(x)$ for a real argument. This function is not defined for $x \leq 0$, and the routine simply returns a value of zero without any warning or error flag. User must ensure that x is positive. For $x < 8$ it uses a rational function approximation of the form

$$Y_0(x) = \frac{2}{\pi}\left\{J_0(x)(\log(x/2)+\gamma) + x^2 R_{mk}(x^2)\right\}, \tag{B.69}$$

where γ is the Euler's constant. $J_0(x)$ is computed using a rational function approximation. For larger arguments the asymptotic form is used:

$$Y_0(x) = \sqrt{\frac{2}{\pi x}}\left(P_0(x)\sin(x-\frac{\pi}{4}) + Q_0(x)\cos(x-\frac{\pi}{4})\right). \tag{B.70}$$

The functions $P_0(x)$ and $Q_0(x)$ are approximated by rational function approximations of form

$$P_0(x) \approx R_{mk}(1/x^2); \qquad Q_0(x) \approx \frac{1}{x}R_{m'k'}(1/x^2). \tag{B.71}$$

All the rational function approximations have absolute accuracy of better than 10^{-15}, but the resulting function values may not have the same relative accuracy at all points. Near the zeros of $Y_0(x)$ the relative accuracy would be lower. For large arguments, $x > 8$, where the asymptotic formula is used there could be some cancellation between the two terms and consequent loss of significant figures close to zeros of $Y_0(x)$.

157. BJY0 Subroutine to calculate the Bessel function of the first and second kind of order zero, $J_0(x), Y_0(x)$ for a real argument. Since computation of $Y_0(x)$ involves $J_0(x)$ also, this subroutine is provided to give both values together and can be used if both functions are required. The function of the second kind is not defined for $x \leq 0$, and the routine simply returns a value of zero without any warning or error flag. The value of $J_0(x)$ should be calculated even in this case. User must ensure that x is positive if $Y_0(x)$ is also required. For calculating $J_0(x)$ alone it will be more efficient to use BJ0. This routine is essentially same as BY0, except that in this case the function values have to appear as arguments while calling the subroutine. The FUNCTION BY0 can also be modified to include BJ0 in its argument list, in which case it can return that value also. In that case the line computing BJ0 should be uncommented. Here XB is the argument at which the function values are required. BJ0 and BY0 are the computed values of $J_0(\mathrm{XB})$ and $Y_0(\mathrm{XB})$.

158. BY1 Function routine to calculate the Bessel function of the second kind of order one, $Y_1(x)$ for a real argument. This function is not defined for $x \leq 0$, and the routine simply returns a value of zero without any warning or error flag. User must ensure that x is positive. For $x < 8$ it uses a rational function approximation of the form

$$Y_1(x) = \frac{2}{\pi}\left\{J_1(x)(\log(x/2)+\gamma) - \frac{1}{x} - xR_{mk}(x^2)\right\}, \tag{B.72}$$

where γ is the Euler's constant. $J_1(x)$ is computed using a rational function approximation. For larger arguments the asymptotic form is used:

$$Y_1(x) = \sqrt{\frac{2}{\pi x}}\left(P_1(x)\sin(x-\frac{3\pi}{4}) + Q_1(x)\cos(x-\frac{3\pi}{4})\right). \tag{B.73}$$

The functions $P_1(x)$ and $Q_1(x)$ are approximated by rational function approximations of form

$$P_1(x) \approx R_{mk}(1/x^2); \qquad Q_1(x) \approx \frac{1}{x}R_{m'k'}(1/x^2). \tag{B.74}$$

All the rational function approximations have absolute accuracy of better than 10^{-15}, but the resulting function values may not have the same relative accuracy at all points. Near the zeros of $Y_1(x)$ the relative accuracy would be lower. For large arguments, $x > 8$, where the asymptotic formula is used there could be some cancellation between the two terms and consequent loss of significant figures close to zeros of $Y_1(x)$.

159. BJY1 Subroutine to calculate the Bessel function of the first and second kind of order one, $J_1(x), Y_1(x)$ for a real argument. Since computation of $Y_1(x)$ involves $J_1(x)$ also, this subroutine is provided to give both values together and can be used if both functions are required. The function of the second kind is not defined for $x \leq 0$, and the routine simply returns a value of zero without any warning or error flag. The value of $J_1(x)$ should be calculated even in this case. User must ensure that x is positive if $Y_1(x)$ is also required. For calculating $J_1(x)$ alone it will be more efficient to use BJ1. This routine is essentially same as BY1, except that in this case the function values have to appear as arguments while calling the subroutine. The FUNCTION BY1 can also be modified to include BJ1 in its argument list, in which case it can return that value also. In that case the line computing BJ1 should be uncommented. Here XB is the argument at which the function values are required. BJ1 and BY1 are the computed values of $J_1(\mathrm{XB})$ and $Y_1(\mathrm{XB})$.

160. BYN Subroutine to calculate the Bessel function of the second kind of integral order, $Y_n(x)$ for a real argument. This routine calculates $Y_k(x)$ for k between 0 and n. This function is not defined for $x \leq 0$, and the routine does not calculate the value and does not give any warning or error flag. In this case values in array BY will be preserved and may be misinterpreted as calculated value. User must ensure that x is positive. Since a recurrence relation is used to calculate these values, all the values are returned. Here N is the maximum order of Bessel functions required. For positive N, $Y_k(x)$ with positive $k = 0, 1, \ldots, \mathrm{N}$ will be calculated. For negative N, $Y_k(x)$ with negative $k = 0, -1, \ldots, -\mathrm{N}$ are calculated. XB is the argument at which the function values are required. BY is a real array of length $|\mathrm{N}| + 1$ which will contain the computed values of the Bessel functions. $\mathrm{BY}(|i| + 1) = Y_i(\mathrm{XB})$. Since the recurrence relation for $Y_n(x)$

$$Y_{n+1}(x) = \frac{2n}{x} Y_n(x) - Y_{n-1}(x), \tag{B.75}$$

is stable in the forward direction for all x, these values are computed in a straightforward manner using the values of $Y_0(x)$ and $Y_1(x)$. This routine needs BY0 and BY1 to calculate $Y_0(x)$ and $Y_1(x)$.

161. SPHBJN Subroutine to calculate the spherical Bessel function of integral order,

$$j_n(x) = \sqrt{\frac{\pi}{2x}} J_{n+1/2}(x), \tag{B.76}$$

for a real argument. This routine calculates $j_k(x)$ for k between 0 and n. Since a recurrence relation is used to calculate these values, all the values are returned. Here N is the maximum order of Bessel functions required. For positive N, $j_k(x)$ with positive $k = 0, 1, \ldots, \mathrm{N}$ will be calculated. For negative N, $j_k(x)$ with negative $k = 0, -1, \ldots, -\mathrm{N}$ are calculated. XB is the argument at which the function values are required. BJ is a real array which will contain the computed values of the Bessel functions. $\mathrm{BJ}(|i| + 1) = j_i(\mathrm{XB})$. Since this array is also used as scratch space the length of the array must be at least, $|\mathrm{N}| + 16 +$

$\max(25, 5\sqrt{|N|})$. For |N| < |XB| or for N < 0 the recurrence relation is stable in forward direction and hence it is used to calculate $j_k(x)$ in a straightforward manner using the values of

$$j_0(x) = \frac{\sin x}{x}; \qquad j_1(x) = \frac{\sin x}{x^2} - \frac{\cos x}{x} \qquad \text{or} \qquad j_{-1}(x) = \frac{\cos x}{x} \,. \tag{B.77}$$

For smaller XB and N > 0, the recurrence is unstable in forward direction and hence is used in backward direction as explained in {2.38} In this case for some suitable n_1, we start using $j_{n_1} = 0$ and $j_{n_1-1} = 1$ and calculate lower order functions using the recurrence relation

$$j_{k-1}(x) = \frac{2k+1}{x} j_k(x) - j_{k+1}(x), \qquad k = n_1 - 1, \ldots, 2, 1. \tag{B.78}$$

The normalisation is then found using the value of $j_0(x)$. Thus we can divide all values by the calculated ratio S of $j_0(x)$ as computed from recurrence relation to its actual value, to obtain the correct values for $j_k(x)$. To use this technique we need to select a suitably large value of n_1 to start the recurrence relation. Accuracy of resulting function values can be checked by comparing the value of $j_{n+1}(x)$ before normalisation. Its reciprocal will give an estimate of accuracy achieved by this process. Thus if the computed value of $j_{n+1}(x)$ is not large enough an error message is printed out by the routine. This should not normally happen, but in case for some combination of N and XB, the chosen value of n_1 is not large enough, the error message will be printed out. In such cases the choice of N1 in the routine should be increased suitably. In principle, this technique can be used for all values of x, but n_1 has to be larger than x and hence for large x, we will need lot of computation using this technique. Further, the use of the recurrence relation in this form results in overflow even when the required values are within the range of computer arithmetic. This problem is more severe at low x, where the values can be easily computed using the series expansion. Thus we use the series expansion for $x \leq 4$ for computing $j_n(x)$ and $j_{n-1}(x)$. After that the recurrence relation is used in backward direction to get other values. The overflow problem becomes very acute when single precision arithmetic is used and it is not recommended to use this routine in single precision. The accuracy attained by this routine will depend on the arithmetic used. In general, we expect to achieve accuracy of order of $\hbar$ in these computations, since $j_0(x)$ and $j_1(x)$ can be expressed in terms of trigonometric functions.

162. BI0 Function routine to calculate the modified Bessel function of the first kind of order zero, $I_0(x)$ for a real argument. For $|x| < 8$ it uses a rational function approximation to $I_0(x) \approx R_{mk}(x^2)$, while for larger arguments it uses the asymptotic form

$$I_0(x) = \frac{e^{|x|}}{\sqrt{|x|}} R_{mk}(1/|x|). \tag{B.79}$$

All the rational function approximations have absolute accuracy of better than 10^{-15}, but the resulting function values may not have the same relative accuracy at all points.

163. BI1 Function routine to calculate the modified Bessel function of the first kind of order one, $I_1(x)$ for a real argument. For $|x| < 8$ it uses a rational function approximation to $I_1(x) \approx xR_{mk}(x^2)$, while for larger arguments it uses the asymptotic form

$$I_1(x) = \frac{e^{|x|}}{\sqrt{|x|}} R_{mk}(1/|x|). \tag{B.80}$$

All the rational function approximations have absolute accuracy of better than 10^{-15}, but the resulting function values may not have the same relative accuracy at all points.

164. BIN Subroutine to calculate the modified Bessel function of the first kind of positive integral order, $I_n(x)$ for a real argument. This routine calculates $I_k(x)$ for k between 0 and n. Since a recurrence relation is used to calculate these values, all the values are returned. Here N is the maximum order of Bessel functions required, $I_k(x)$ with $k = 0, 1, \ldots, \mathrm{N}$ will be calculated. N must be positive, otherwise |N| will be used. XB is the argument at which the function values are required. BI is a real array which will contain the computed values of the Bessel functions. $\mathrm{BI}(i+1) = I_i(\mathrm{XB})$. Since this array is also used as scratch space the length of the array must be at least, $\mathrm{N} + 16 + \max(25, 5\sqrt{N})$. For $\mathrm{N} < \mathrm{XB} - 10$ the recurrence relation is stable in forward direction and hence it is used to calculate $I_k(x)$ in a straightforward manner using the values of $I_0(x)$ and $I_1(x)$. For smaller XB, the recurrence is unstable in forward direction and hence is used in backward direction as explained in {2.38} In this case for some suitable n_1, we start using $I_{n_1} = 0$ and $I_{n_1-1} = 1$ and calculate lower order functions using the recurrence relation

$$I_{k-1}(x) = \frac{2k}{x} I_k(x) + I_{k+1}(x), \qquad k = n_1 - 1, \ldots, 2, 1. \tag{B.81}$$

The normalisation is then found using the directly computed value of $I_0(x)$. Thus we can divide all values by the calculated ratio for $I_0(x)$ to obtain the correct values for $I_k(x)$. To use this technique we need to select a suitably large value of n_1 to start the recurrence relation. Accuracy of resulting function values can be checked by comparing the value of $I_{n+1}(x)$ before normalisation. Its reciprocal will give an estimate of accuracy achieved by this process. Thus if the computed value of $I_{n+1}(x)$ is not large enough an error message is printed out by the routine. This should not normally happen, but in case for some combination of N and XB, the chosen value of n_1 is not large enough, the error message will be printed out. In such cases the choice of N1 in the routine should be increased suitably. In principle, this technique can be used for all values of x, but n_1 has to be larger than x and hence for large x, we will need lot of computation using this technique. Further, the use of the recurrence relation in this form results in overflow even when the required values are within the range of computer arithmetic. This problem is more severe at low x, where the values can be easily computed using the series expansion. Thus we use the series expansion for $x \leq 4$ for computing $I_n(x)$ and $I_{n-1}(x)$. After that the recurrence relation is used in backward direction to get other values. The overflow problem becomes very acute when single precision arithmetic is used and it is not recommended to use this routine in single precision. The accuracy attained by this routine will depend on the values of N and XB, but in general, we expect accuracy of order of 10^{-15} which is the accuracy with which $I_0(x)$ and $I_1(x)$ are calculated. This routine needs BI0 and BI1 to calculate $I_0(x)$ and $I_1(x)$.

165. BK0 Function routine to calculate the modified Bessel function of the second kind of order zero, $K_0(x)$ for a real argument. This function is not defined for $x \leq 0$, and the routine simply returns a value of zero without any warning or error flag. User must ensure that x is positive. For $x < 8$ it uses a rational function approximation of the form

$$K_0(x) = -I_0(x)(\gamma + \log(x/2)) + x^2 R_{mk}(x^2), \tag{B.82}$$

where γ is the Euler's constant and $I_0(x)$ is computed using appropriate rational function approximation. For larger arguments the asymptotic form is used:

$$K_0(x) = \frac{e^{-x}}{\sqrt{x}} R_{mk}(1/x). \tag{B.83}$$

All the rational function approximations have absolute accuracy of better than 10^{-15}, but the resulting function values may not have the same relative accuracy at all points when different terms are combined.

166. BK1 Function routine to calculate the modified Bessel function of the second kind of order one, $K_1(x)$ for a real argument. This function is not defined for $x \leq 0$, and the routine simply returns a value of zero without any warning or error flag. User must ensure that x is positive. For $x < 8$ it uses a rational function approximation of the form

$$K_1(x) = I_1(x)(\gamma + \log(x/2)) + \frac{1}{x} - x R_{mk}(x^2), \tag{B.84}$$

where γ is the Euler's constant and $I_1(x)$ is computed using appropriate rational function approximation. For larger arguments the asymptotic form is used:

$$K_1(x) = \frac{e^{-x}}{\sqrt{x}} R_{mk}(1/x). \tag{B.85}$$

All the rational function approximations have absolute accuracy of better than 10^{-15}, but the resulting function values may not have the same relative accuracy at all points when different terms are combined.

167. BKN Subroutine to calculate the modified Bessel function of the second kind of positive integral order, $K_n(x)$ for a real argument. This routine calculates $K_j(x)$ for j between 0 and n. This function is not defined for $x \leq 0$, and the routine simply returns without any warning or error flag. In this case values in array BK will be preserved and may be misinterpreted as calculated value. User must ensure that x is positive. Since a recurrence relation is used to calculate these values, all the values are returned. Here N is the maximum order of Bessel functions required, $K_j(x)$ with $j = 0, 1, \ldots, \mathrm{N}$ will be calculated. N must be positive, otherwise $|\mathrm{N}|$ will be used. X is the argument at which the function values are required. BK is a real array of length $\mathrm{N}+1$ which will contain the computed values of the Bessel functions. $\mathrm{BK}(i+1) = K_i(\mathrm{XB})$. Since the recurrence relation is stable in forward direction for all x, it is used in a straightforward manner using the values of $K_0(x)$ and $K_1(x)$. This routine needs BK0 and BK1 to calculate $K_0(x)$ and $K_1(x)$.

168. DAWSON Function routine to calculate the Dawson's integral

$$D(x) = e^{-x^2} \int_0^x e^{t^2}\, dt\,, \tag{B.86}$$

using rational function approximation. The range is split into 4 parts. In $[0, 2.5), [2.5, 4.0), [4.0, 5.5)$ approximation of the form $D(x) = xR_{mk}(x^2)$ is used while for higher values an approximation of the form $D(x) = (1/x)R_{mk}(1/x^2)$ is used. All approximations have a relative accuracy of 10^{-15}.

169. FERMM05 Function routine to calculate the Fermi integrals

$$F_k(x) = \int_0^\infty \frac{t^k\, dt}{e^{t-x}+1}\,, \tag{B.87}$$

for $k = -1/2$ using rational function approximations. The range is split into 3 parts. For $x < 2$ (including all negative values) an approximation of the form $F_k(x) = e^x R_{mk}(e^x)$ is used. For $2 \leq x < 10$ an approximation of form $F_k(x) = R_{mk}(x)$ is used, while for higher values of x an approximation of the form $F_k(x) = x^{k+1} R_{mk}(1/x^2)$ is used. All the approximations have a relative accuracy of approximately 10^{-15}.

170. FERM05 Function routine to calculate the Fermi integrals for $k = 1/2$ using approximations similar to those for $k = -1/2$.

171. FERM15 Function routine to calculate the Fermi integrals for $k = 3/2$ using approximations similar to those for $k = -1/2$.

172. FERM25 Function routine to calculate the Fermi integrals for $k = 5/2$ using approximations similar to those for $k = -1/2$.

173. PLEG Subroutine to calculate Legendre polynomial for a specified value of x. L is the order of polynomial and X is the argument at which the polynomial value is required. P is a real array of length $\mathrm{L}+1$, which will contain the computed values of the Legendre polynomials for all orders up to L, $\mathrm{P}(j+1) = P_j(X)$. The recurrence relation is used to compute the values.

174. PLM Subroutine to calculate associated Legendre functions, $P_l^m(x)$ for a specified value of x. L, M define the order of Legendre functions, $\mathrm{L} > 0$ and $|\mathrm{M}| \leq \mathrm{L}$. No error message is issued if L, M do not satisfy these constraints. X is the argument at which the function value is required. P is a real array of length L+1, which will contain the computed values of the associated Legendre functions. $\mathrm{P}(j+1) = P_j^{\mathrm{M}}(X)$ for $j \geq \mathrm{M}$. The routine first computes the value of $P_{\mathrm{M}}^{\mathrm{M}}$. For positive m,

$$P_m^m = \prod_{i=1}^{m} (2i-1)(1-x^2)^{m/2}. \tag{B.88}$$

For negative m there is extra normalisation factor. Using this value the recurrence relation

$$(n-m)P_n^m(x) = (2n-1)xP_{n-1}^m(x) - (n-1+m)P_{n-2}^m(x), \tag{B.89}$$

is used to compute the value of $P_{\mathrm{L}}^{\mathrm{M}}$.

175. YLM Function routine to calculate the Spherical harmonic,

$$Y_l^m(\theta, \phi) = (-1)^m c_{lm} P_l^m(\cos\theta) e^{im\phi}, \tag{B.90}$$

where

$$c_{lm}^2 = \frac{(2l+1)(l-m)!}{4\pi(l+m)!}\ , \tag{B.91}$$

is chosen to ensure that integral of $|Y_l^m|^2$ over the unit sphere is 1. L is the degree, M is the azimuthal order. $\mathrm{L} > 0$, $|\mathrm{M}| \leq \mathrm{L}$ and $\mathrm{L} < \mathrm{LMAX} = 5001$. The last requirement arises from the dimensions of the array P in the function routine. The value of LMAX can be increased if needed. THETA and PHI are real variables specifying the angular coordinates θ and ϕ respectively. YLM is the complex value of spherical harmonic, and must be declared to be complex in the calling program. It is possible to use the argument $x = \cos\theta$ instead of θ by commenting out the line computing X. This routine requires PLM to compute the associated Legendre functions. YLM_X is the version of YLM with $x = \cos\theta$ as the argument.

176. MINMAX Subroutine to calculate minimax rational function approximation of discrete data using a differential correction algorithm. M and K are the degrees of polynomials in the numerator and the denominator, respectively. N is the number of data points which should be at least M + K + 2. A is a real array of length M + K + 2, which will contain the coefficients in the same form as that given by the subroutine PADE. A(i), ($i = 1, 2, \ldots, \mathrm{K}$) is the coefficient of x^i in the denominator, the constant term being unity, while $\mathrm{A}(\mathrm{K}+i+1)$, ($i = 0, 1, \ldots, \mathrm{M}$) is the coefficient of x^i in the numerator. At the time of calling the subroutine, this array should contain the initial approximation for the coefficients. If K = 0, then the algorithm will converge from arbitrary initial approximation and hence all elements of A may be set to zero or any other suitable value. For $K \neq 0$ the iteration is unlikely to converge, unless the starting values are close to the actual values of the coefficients. X and F are real arrays of length N, containing the input data, with F(I) giving the value of the function at X(I). These values must be supplied. EMAX is the maximum error in the final approximation calculated by the subroutine. EPS is the required tolerance, the iteration is continued until the change in EMAX is less than EPS. IER is the error parameter. IER = 615 implies that either $M < 0$, or $K < 0$, or $M + K + 2 > N$, in which case, no calculations are performed. IER = 634 implies that the iteration failed to converge to the specified accuracy. Other values of IER may be set by the subroutine SIMPX. WK is a real array of length $(3N + 1)(M + K + 5)$, which is used as a scratch space to store intermediate results. IWK is an integer array of length $3N + M + K + 3$, used as a scratch space. This subroutine requires the subroutine SIMPX to solve the LP problem.

177. POLYL1 Subroutine to calculate polynomial L_1-approximation of discrete data. M is the degree of polynomial. N is the number of data points, which should be at least $M + 2$. A is a real array of length $M + 2$, which will contain the coefficients. $A(i + 1)$, $(i = 0, 1, 2, \ldots, M)$ is the coefficient of x^i. X and F are real arrays of length N, containing the input data, with F(I) containing the value of the function at X(I). These values must be supplied by the user. ESUM is the sum of the magnitude of the error at all points (i.e., the L_1 norm of the residual) in the final approximation calculated by the subroutine. EPS is the expected level of roundoff error, this parameter is passed on to the subroutine SIMPL1 for simplex iteration. It is used to decide the sign of cost coefficients and pivot elements. IER is the error parameter. IER = 616 implies that either $M < 0$, or $M+2 > N$, in which case no calculations are performed. Other values of IER may be set by the subroutine SIMPL1. WK is a real array of length $(N + 2)(M + 3)$, which is used as a scratch space to store intermediate results. IWK is an integer array of length $N + M + 3$ used as a scratch space. This subroutine requires the subroutine SIMPL1 to solve the LP problem using a slightly modified simplex algorithm.

178. LINL1 Subroutine to calculate a general linear L_1-approximation in terms of arbitrary basis functions for discrete data. Approximation of the form

$$f(x) \approx \sum_{i=1}^{M} a_i \phi_i(x), \tag{B.92}$$

are sought, where $\phi_i(x)$ are the basis functions. There is no restriction on the basis functions, apart from the fact that they should be independent on the set of points in the table. This program can be used for approximation in multiple dimensions as points need not be restricted to one dimension. M is the number of basis functions. N is the number of data points, which should be at least $M + 1$. A is a real array of length $M + 1$, which will contain the coefficients of approximation. $A(i)$, $(i = 1, 2, \ldots, M)$ is the coefficient of ϕ_i. F is a real array of length N, containing the input data, F(I) should contain the value of the function at Ith point. These values must be supplied by the user. G is a real array of length IG $\times$ N containing the values of basis functions at each point in the table, G$(i,j) = \phi_i(x_j)$. These values must be supplied by the user. It may be noted that in this case the coordinates x_j are not required by the routine and it could even be a vector for approximation in multiple dimension. It is users responsibility to calculate the values $\phi_i(x_j)$ accordingly. IG is the first dimension of G as specified in the calling program (IG $\geq$ M). ESUM is the sum of the magnitude of the error at all points (i.e., the L_1 norm of the residual) in the final approximation calculated by the subroutine. EPS is the expected level of roundoff error, this parameter is passed on to the subroutine SIMPL1 for simplex iteration. It is used to decide the sign of cost coefficients and pivot elements. IER is the error parameter. IER = 616 implies that either M ≤ 0, or M$+1 >$ N, in which case no calculations are performed. Other values of IER may be set by the subroutine SIMPL1. WK is a real array of length (N + 2)(M + 3), which is used as a scratch space to store intermediate results. IWK is an integer array of length N + M + 3 used as a scratch space. This subroutine requires the subroutine SIMPL1 to solve the LP problem using a slightly modified simplex algorithm.

179. SIMPL1 Subroutine to solve LP problems using a modified version of the simplex algorithm, specially suitable for LP problems arising in L_1-approximations. This subroutine is called by subroutine POLYL1 or LINL1 and is similar to subroutine SIMPX. The LP problem is assumed to be in the standard form and the initial tableau is supplied in the real array A of length IA $\times$ (N $-$ M + 1) with IA $\geq$ M + 2. IA is the first dimension of A, exactly as specified in the calling program. N is the total number of variables in the given problem. M is the number of constraints in the problem. ID and IV are integer arrays of length M + 1 and N $-$ M + 1 respectively, used to store permutations of the original variables. IER is the error parameter. IER = 63 implies that the objective function is unbounded from below and the optimal feasible vector does not exist. IER = 635 implies that the simplex iteration has not converged in a reasonable number of iterations. This failure may be due to degeneracy, since that is not accounted for in this subroutine. AEPS is a real parameter used to control roundoff error. Any quantity less than AEPS in magnitude is assumed to be zero. It should be noted that, this subroutine does not explicitly take care of degeneracy. Since the problems arising out of L_1-approximation have very high degree of degeneracy, this subroutine may fail to converge in some cases, because of cycling as explained in Section 8.7.

B.11 Algebraic Eigenvalue Problem

180. INVIT Subroutine to find a real eigenvalue and the corresponding eigenvector of a general real matrix using inverse iteration. M is the order of the matrix. A is a real array of length IA $\times$ M containing the input matrix. IA is the first dimension of A as specified in the calling program. P is a real variable specifying the initial shift to be used. This shift should be close to the required eigenvalue. U is a real array of length M, which should contain an initial approximation to the eigenvector. This approximation need not be close to the required eigenvector, but must be nonzero. The initial vector U need not be normalised. In most cases setting all components of U to 1 will be enough, but if the initial vector so chosen is orthogonal to the required eigenvector there may be some problem in convergence. In that case an arbitrary choice which is not orthogonal to eigenvector will be required. For multiple eigenvalues if more than one eigenvectors are required, then initial vector should be chosen to be orthogonal to all known eigenvectors with same eigenvalue to ensure that an independent eigenvector is found. After execution, the array U will contain the required eigenvector, which is normalised such that the maximum component is unity. IFLG is a flag to decide the kind of iteration required. If IFLG $= 0$, then the shift P is kept fixed. If IFLG $= 1$, then the shift is varied after each iteration, using the computed Rayleigh quotient. If IFLG $= 2$, then the shift is varied using $\max(\mathbf{v_{s+1}})$ at each iteration. IFLG $= 0$ should be used when the eigenvalue is already known accurately, or when iteration is strictly required to converge to the eigenvalue nearest to P. In other cases, we may use IFLG $= 1$, if the matrix is Hermitian and IFLG $= 2$ otherwise. In practice, it is found that, even for some non-Hermitian matrices, IFLG $= 1$ often gives faster convergence, but that cannot be assumed. After execution, EI and ERC give the estimated eigenvalue (corrected for the shift P). ERC is the estimate using Rayleigh quotient, while EI is the estimate given by simple inverse iteration. ERC is relevant only for Hermitian matrices, while EI is applicable to all matrices. REPS is the required (absolute) tolerance. The iteration is continued until the maximum change in the eigenvalue and the M components of the eigenvector is less than REPS. WK is a real array of length $\mathrm{M}^2 + \mathrm{M}$ used as a scratch space. IWK is an integer array of length M used as a scratch space. NIT is an integer variable specifying the maximum number of iterations required to be performed. If NIT ≤ 0, then a default value of NIT0 ($= 100$) will be used. IER is the error parameter. IER $= 106$ implies that $\mathrm{M} \leq 1$ or $\mathrm{M} > \mathrm{IA}$, in which case, no calculations are performed. IER $= 141$ implies that the vector is zero at some stage, which is usually due to either the matrix A or the initial vector U being zero. If the matrix is nonzero, then using a different initial vector or changing the shift P may overcome this problem. IER $= 142$ implies that the inverse iteration has failed to converge, which could be either because the starting shift P is not sufficiently close to an eigenvalue, or because REPS is too small, or because the corresponding eigenvalue has a nonlinear divisor. Apart from these, other values of IER may be set by subroutine GAUELM, which is called to solve the system of linear equations. In particular, IER $= 121$ implies that one of pivots during the Gaussian elimination is zero. This problem can usually be overcome by perturbing the shift slightly. If this problem occurs persistently, it may be better to modify the Gaussian elimination routine, such that zero pivots are replaced by a suitably chosen small number. To find complex eigenvalues of a real matrix, all normal Fortran real variables except those starting with A and R should be treated as complex, which can be achieved by using an IMPLICIT COMPLEX(B–H, P, S–Z) statement. Apart from this, the definition of Rayleigh quotient will need to be modified as indicated in the program. This modification is required only if IFLG = 1, since in other cases, the value of Rayleigh quotient is not used. This is implemented in INVIT_C. To find eigenvalues of a complex matrix, the array A should also be declared as complex. This case is implemented in INVIT_CC. In both these cases, a complex version of GAUELM (GAUELM_C) will be required. To find an eigenvector of a generalised eigenvalue problem $A(\lambda)\mathbf{x} = \mathbf{0}$, use P = 0, IFLG = 0, and the matrix A should be set to $A(\lambda)$ with λ equal to the known eigenvalue. In this case the eigenvalue has to be determined before finding the eigenvector. To find the left eigenvector of the matrix take transpose of the matrix while applying the shift. This is implemented in INVIT_L, for real eigenvalues and INVIT_CL for complex eigenvalues.

181. TRED2 Subroutine to reduce a real symmetric matrix to tridiagonal form using Householder's method. This subroutine is based on the procedure *tred2* in the *Handbook*. A is a real array of length IA × N containing the matrix. After execution, the array A will be overwritten by the transformation matrix Q, such that $Q^T\mathrm{A}Q$ is tridiagonal. This matrix may be required for back-transforming the eigenvectors of tridiagonal matrix to that of the original input matrix A. N is the order of the matrix. IA is the first dimension of the array A, as specified in the calling program, IA $\geq$ N. D and E are real arrays of length N, giving the diagonal and off-diagonal elements of the reduced tridiagonal matrix with D$(i) = a_{ii}$ and E$(i+1) = a_{i,i+1} = a_{i+1,i}$. REPS is the tolerance which should be equal to $\eta/\hbar$, where η is the smallest positive number that is representable in the computer and $\hbar$ is the machine accuracy. For single precision (REAL*4) arithmetic use REPS = 10^{-30}. For double precision (REAL*8) arithmetic use REPS = 10^{-300}. IER is the error parameter. IER = 107 implies that N $\leq$ 1 or N $>$ IA, in which case, no calculations are performed.

182. TRBAK Subroutine to perform back-transformation on eigenvectors of reduced tridiagonal matrix to obtain the eigenvectors of the original real symmetric matrix reduced by TRED2. This back-transformation is not required if the eigenvectors are calculated using the subroutine TQL2, but will be required if the eigenvectors are calculated by the subroutine TRIDIA. A is a real array of length IA × N containing the transformation matrix Q generated by the subroutine TRED2. The last column of A is used as a scratch space by the subroutine and hence its contents will be destroyed during execution. IA is the first dimension of A, as specified in the calling routine. N is the order of the matrix. Z is a real array of length IZ × NZ containing the eigenvectors of the reduced tridiagonal matrix. After execution, this array will be overwritten by the eigenvectors of the original matrix A. IZ is the first dimension of array Z, as specified in the calling program. NZ is the number of eigenvectors. Thus, Z(i, j) should contain the ith component of the jth eigenvector. This subroutine simply performs the matrix multiplication AZ to obtain the required eigenvectors.

183. TQL2 Subroutine to find eigenvalues and eigenvectors of ZTZ^T using the QL algorithm, where T is a symmetric tridiagonal matrix and Z is an orthogonal matrix. This subroutine is based on the procedure *tql2* in the *Handbook.* If the matrix Z is the transformation matrix, which reduces a real symmetric matrix to tridiagonal form, then this routine gives the eigenvectors of the original matrix. This subroutine can be used to find eigenvalues and eigenvectors of a real symmetric matrix, after it is reduced to a tridiagonal form using TRED2. In that case, Z will be the transformation matrix generated by TRED2. To find eigenvalues and eigenvectors of a symmetric tridiagonal matrix, set the matrix Z to an identity matrix. Z is a real array of length IZ $\times$ N, containing the transformation matrix. After execution, this array will be overwritten by the eigenvectors of the original matrix ZTZ^T, with ith column containing the ith eigenvector. N is the order of the matrix. IZ is the first dimension of Z, as specified in the calling program. D is a real array of length N, containing the diagonal elements of the tridiagonal matrix with D$(i) = t_{ii}$. After execution, the array D will be overwritten by the eigenvalues of the matrix. The eigenvalues are sorted in the ascending order. E is a real array of length N containing the off-diagonal elements of the tridiagonal matrix, with E$(i+1) = t_{i,i+1} = t_{i+1,i}$. E is used as a scratch space by the subroutine and hence its contents will be destroyed during the execution. REPS is the tolerance which should be equal to $\hbar$, the machine accuracy. IER is the error parameter. IER $= 108$ implies that N ≤ 1 or N $>$ IZ, in which case, no calculations are performed. IER $= 143$ implies that the QL algorithm failed to converge for some eigenvalue, in which case, the calculations are abandoned.

184. TRIDIA Subroutine to find some eigenvalues and eigenvectors of a symmetric tridiagonal matrix using the Sturm sequence property, coupled with the inverse iteration method. N is the order of the matrix. E and D are real arrays of length N containing respectively, the off-diagonal and diagonal elements of the symmetric tridiagonal matrix, with E$(i+1) = a_{i,i+1} = a_{i+1,i}$ and D$(i) = a_{ii}$. It is assumed that all the off-diagonal elements are nonzero. Otherwise, the matrix should be split into two or more parts and each part should be considered independently. M1 and M2 are integers specifying which eigenvalues are to be calculated. If eigenvalues λ_i are sorted in increasing order, then all eigenvalues from λ_{M1} to λ_{M2} will be determined. If M1 $>$ M2, then no calculations are performed. EI is a real array of length M2 $-$ M1 $+ 1$, which will contain the eigenvalues after execution. EPS1 is a real parameter, which specifies the accuracy to which the eigenvalues should be located by bisection. This parameter is required in some cases, since the inverse iteration with variable shift, which is used to determine eigenvalues accurately may not always converge to the nearest eigenvalue. If that happens, then the parameter EPS1 can be decreased. A moderate value of the order of 0.1 or 0.01 times the typical eigenvalues should be normally sufficient. Increasing EPS1 will improve the efficiency, provided the inverse iteration converges. REPS is the required tolerance in the eigenvalues and eigenvectors. EV is a real array of length IV $\times$ (M2 $-$ M1 $+ 1$) which will contain the calculated eigenvectors. EV(i, j) will contain the ith component of the jth eigenvector. IV is the first dimension of the array EV as specified in the calling program, IV $\geq$ N. WK is a real array of length 7N used as a scratch space. IER is the error parameter. IER = 109 implies that N ≤ 1 or N > IV or M1 < 1 or M2 > N, in which case, no calculations are performed. Other values of IER may be set by subroutine TINVIT, which is called to calculate the eigenvalues and eigenvectors. Only the last nonzero value of IER will be retained. This subroutine requires the subroutine STURM to locate the eigenvalues and subroutine TINVIT to find the eigenvalues and eigenvectors of a symmetric tridiagonal matrix and function RAN1 to calculate random numbers.

185. STURM Subroutine to locate required eigenvalues of a real symmetric tridiagonal matrix using the method of bisection on the Sturm sequence. It is assumed that all the off-diagonal elements are nonzero. Otherwise, the matrix should be split into two or more parts and each part should be considered independently. N is the order of the matrix. E and D are real arrays of length N containing respectively, the off-diagonal and diagonal elements of the symmetric tridiagonal matrix, with $\mathrm{E}(i+1) = a_{i,i+1} = a_{i+1,i}$ and $\mathrm{D}(i) = a_{ii}$. M1 and M2 are integers specifying which eigenvalues are to be located. If eigenvalues λ_i are sorted in increasing order, then all eigenvalues from λ_{M1} to λ_{M2} will be located. If M1 > M2, then no calculations are performed. EL and EU are real arrays of length M2. After execution, the ith eigenvalue should be located in the interval $(\mathrm{EL}(i), \mathrm{EU}(i))$. NUM is an integer variable which will contain the number of times the Sturm sequence was evaluated for locating the eigenvalues to the specified accuracy. REPS is the specified accuracy to which the eigenvalues are to be located. Bisection is continued until $|\mathrm{EU}(i) - \mathrm{EL}(i)| <$ REPS and the eigenvalue has been isolated. At least three bisections are performed after the eigenvalues are isolated, that is $\mathrm{EU}(i-1) < \mathrm{EL}(i)$. If at any stage the interval is too small to be bisected further, then the iteration is naturally terminated. This situation can arise when the matrix has very close eigenvalues. WK is a real array of length N used as a scratch space. IER is the error parameter. IER = 110 implies that M1 < 1 or M2 > N, in which case no calculations are done.

186. TINVIT Subroutine to find a specified eigenvalue and eigenvector of a symmetric tridiagonal matrix using inverse iteration. N is the order of the matrix. E and D are real arrays of length N containing respectively, the off-diagonal and diagonal elements of the symmetric tridiagonal matrix, with $\mathrm{E}(i+1) = a_{i,i+1} = a_{i+1,i}$ and $\mathrm{D}(i) = a_{ii}$. EL and EU are real variables giving the lower and upper limit on the eigenvalue. The iteration will not go beyond these limits. EI is a real variable, which will return the calculated eigenvalue. EV is a real array of length N, which will contain the calculated eigenvector. For IFLG $\neq 0$ it should contain the previous eigenvector determined by the program as input, so that the routine can choose initial vector orthogonal to it. REPS is the required accuracy. The iteration is continued until either of the following three conditions are satisfied, (1) the relative change in the eigenvalue as estimated using the Rayleigh quotient is less than REPS, (2) $\max(\mathbf{v_{j+1}})\lambda >$

1/REPS (where λ is an estimate for the eigenvalue), (3) $\|\mathbf{u_{j+1}} - \mathbf{u_j}\|_1 <$ REPS. IFLG is an integer variable used as a flag. If IFLG = 0, then the initial vector is chosen randomly. Otherwise, the initial vector is chosen to be orthogonal to the vector EV. This device can be used to ensure that an independent vector is found when two or more eigenvalues are very close or equal. Since only one previous eigenvector is used, this device may not be very effective for eigenvalues of multiplicity greater than two. IER is the error parameter. IER = 144 implies that the inverse iteration failed to converge to the specified accuracy. U and B are real arrays of length N and 4N respectively, which are used as a scratch space to store intermediate quantities. NUM is an integer variable, which will contain the number of iterations required by the subroutine. This subroutine requires FUNCTION RAN1(SEED) to generate random numbers. If a different routine is used for this purpose, the seed should be changed appropriately.

187. HEREVP Subroutine to find all eigenvalues and eigenvectors of a complex Hermitian matrix. This subroutine converts the eigenvalue problem for a Hermitian matrix into that for a real symmetric matrix of order 2N and solves this expanded problem. The expanded eigenvalue problem is solved by reducing the matrix to tridiagonal form using TRED2 and then solving the eigenvalue problem for tridiagonal matrix using TQL2. ZA is a complex array of length IA $\times$ N, containing the matrix. The subroutine preserves the contents of this array. N is the order of the matrix. IA is the first dimension of ZA, as specified in the calling program. EI is a real array of length N, which will contain the eigenvalues of the matrix ZA. The eigenvalues are sorted in the ascending order. ZV is a complex array of length IZ $\times$ N, which will contain the eigenvectors of the complex matrix. ZV(I, J) will contain the Ith component of Jth eigenvector. IZ is the first dimension of array ZV as declared in the calling program. WK is a real array of length 2N $\times$ (2N + 2) used as scratch space. REPS is the tolerance which should be of the order of $\hbar$, the machine accuracy. IER is the error parameter. IER = 111 implies that N $\leq$ 1 or N > IA or N > IZ, in which case, no calculations are performed. Other values may be set by TRED2 or TQL2. The subroutine requires TRED2 and TQL2 to solve the eigenvalue problem for equivalent real symmetric matrix. Since every eigenvalue of the equivalent real matrix is repeated, there may be problem in isolating the eigenvectors when the eigenvalue of complex matrix is multiple. If there is some problem all eigenvectors of the 2N $\times$ 2N real matrix should be preserved so that the right combinations can be identified. This subroutine picks alternate eigenvectors.

188. BALANC Subroutine for reducing the norm of a matrix by exact diagonal similarity transformations. This subroutine is based on the procedure *balance* in the *Handbook*. A is a real array of length IA $\times$ N containing the matrix. After execution, the balanced matrix will be overwritten on the same array. N is the order of the matrix and IA is the first dimension of array A, as declared in the calling program. B is the base of floating-point representation in the machine. For most machines B = 2. LOW and IGH are integer variables, such that in the balanced matrix $a_{ij} = 0$ if $i > j$ and ($j <$ LOW or

$i >$ IGH). This essentially means that after balancing we have to only consider the sub-matrix in rows and columns from LOW to IGH, since other eigenvalues are already isolated by subroutine BALANC. D is a real array of length N containing the information about transformation. The elements D(LOW) to D(IGH) will contain the elements of diagonal matrix used for balancing, while other elements will contain the permutations used to isolate eigenvalues. IER is the error parameter. IER $= 112$ implies that N ≤ 1 or N $>$ IA, in which case, no calculations are performed.

189. BALBAK Subroutine to perform back-transformation of a set of right eigenvectors from those of the balanced matrix to that for the original matrix. This subroutine is based on the procedure *balbak* in the *Handbook*. N is the order of the matrix. LOW and IGH are integer variables obtained by the subroutine BALANC. CZ is a complex array of length IZ $\times$ M containing the eigenvectors of the balanced matrix. After execution, the eigenvectors of the original matrix will be overwritten on the same array. M is the number of eigenvectors, and IZ is the first dimension of array CZ as declared in the calling program. D is a real array of length N, containing the information about transformation as generated by the subroutine BALANC. This subroutine can be modified as explained in the Fortran file, to back-transform left eigenvectors and the corresponding version is implemented in BALBAK_L.

190. ELMHES Subroutine for reducing a general real matrix to an upper Hessenberg form using real stabilised elementary similarity transformation. This subroutine is based on the procedure *elmhes* in the *Handbook*. A is a real array of length IA $\times$ N containing the matrix. After execution, the reduced matrix will be overwritten on the same array. Information about the transformation (i.e., the elements $m_{i,r+1}$) will also be written on the (i,r) element of the same array. N is the order of the matrix and IA is the first dimension of array A as declared in the calling program. LOW and IGH are integer variables as given by the subroutine BALANC while balancing the matrix. This subroutine only processes the sub-matrix in rows LOW to IGH. If the matrix is not balanced, then set LOW $= 1$ and IGH $=$ N before calling this subroutine. INC is an integer array of length N, containing the information about the row and column interchanges used during the reduction. IER is the error parameter. IER $= 113$ implies that N ≤ 1 or N $>$ IA, in which case, no calculations are performed.

191. HQR Subroutine to find eigenvalues of an upper Hessenberg matrix using QR algorithm. This subroutine is based on the procedure *hqr* in the *Handbook*. It does not keep track of the transformations and hence cannot be used to find eigenvectors. H is a real array of length IH $\times$ NN containing the matrix. During execution, the contents of H will be destroyed. NN is the order of the matrix and IH is the first dimension of H as specified in the calling program. ER and EI are real arrays of length NN, which will contain the real and imaginary parts of the eigenvalues. REPS is the tolerance which should be of the order of $\hbar$, the machine accuracy. IER is the error parameter. IER $= 114$ implies that N ≤ 1 or N $>$ IH, in which case, no calculations are performed. IER $= 145$ implies that the QR iteration failed to converge at some stage and the calculations are abandoned. In this case, the eigenvalues which are already isolated should be available in arrays ER and EI.

B.12 Ordinary Differential Equations

192. RKM Subroutine to solve initial value problems in ordinary differential equations, using a second or fourth-order Runge-Kutta method with adaptive step size control. This subroutine accepts a system of first-order differential equations. N is the number of first-order equations. Y is a real array of length N, which should contain the initial values of the variables at input. After successful execution, it will contain the solution at the required point. If the execution is aborted in between, then it will contain the solution at some intermediate point T0, up to where the integration was successful. DY is a real array of length N, containing the first derivative of the solution. The derivative need not be supplied at the time of calling. DIF is the name of the external subroutine, which is invoked for calculating the right-hand side of the differential equation $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$. H is the initial step size to be tried. The subroutine will adjust the step size if necessary, using the technique described in Section 12.4. After execution, H will contain the current value of the step size. T0 is the initial value of the independent variable t, at which the initial conditions are specified. After execution, the value of T0 will be updated, to the point up to which the integration is successful. TN is the value of t at which the solution is required. TN need not be greater than T0, i.e., the solution can be calculated in forward or backward direction. If the integration is successful, then T0 will be set equal to TN. Thus, if the solution is required at several points, then the subroutine can be called repeatedly after changing TN between every call. Other variables need not be changed. REPS is the required relative accuracy in each component of the solution. This subroutine only tries to estimate the local truncation error and the computed solution may not be correct to the specified accuracy. NSTEP is the number of steps used by the Runge-Kutta method, each step may require 10 or 11 function evaluations with fourth-order Runge-Kutta method. Each step may not be successful. NMAX is the maximum number of steps that the subroutine is allowed to use. If NMAX ≤ 0, then a default value of NMX ($= 10000$) will be used. IER is the error parameter. IER $= 701$ implies that N ≤ 0, in which case, no calculations are performed. IER $= 721$ implies that step size has become smaller than REPS$|$TN $-$ T0$|$. If in a small region the step size is actually expected to be smaller than this limit, then the corresponding statement in the subroutine may be modified. IER $= 722$ implies that the step size is too small for the arithmetic used (i.e., H $< \hbar|$T0$|$) and the calculations are aborted. IER $= 723$ implies that the integration could not be completed in the specified number (NMAX) of steps. The failure of this routine can arise either because of singularity, or stiffness, or because the specified accuracy is too high. WK is a real array of length 5N, which is used as a scratch space by the subroutine. This subroutine requires subroutine RK4 or RK2 to perform one step of Runge-Kutta integration. Subroutine RK4 uses a fourth-order Runge-Kutta method, while RK2 uses a second-order method. To use RK2 the call statements as well as the parameter statement in the beginning of the subroutine should be changed as indicated. RKM_2 is the version of RKM for second-order method. Further, the subroutine DIF(T, N, Y, DY) must be supplied by the user to calculate the right-hand sides of the equations, i.e., the derivatives Y'. Here T is the value of independent variable and N is the number of equations. Y is the real array containing the dependent variables at time $t = T$, where the derivatives need to be evaluated. DY is the real array which will contain the calculated value of the derivatives. DY(i) should contain $Y'(i) = f_i(T, Y)$.

193. RK4 Subroutine to perform one step of integration using a fourth-order Runge-Kutta method. N is the number of first-order differential equations in the system. T is the value of independent variable at the initial point. This value is not updated by the subroutine. H is the step size to be used. Y0, DY0 and Y1 are real arrays of length N. Y0 contains the initial value, while after execution, Y1 will contain the final value of the solution vector at $t = T + H$. DY0 contains the first derivative at Y0. Y0 and DY0 must be supplied at the time of calling the subroutine. DIF is the name of the external subroutine used to calculate the right-hand sides of the differential equations. WK is a real array of length 2N used as a scratch space. The subroutine DIF as described in the write up for subroutine RKM must be supplied by the user. This subroutine is called by subroutine RKM, but it can be used independently to integrate an equation with constant step size. This can be achieved by repeatedly calling the subroutine after updating the value of T $(= T + H)$, Y0 (= Y1) and DY0 (using DIF).

194. RK2 Subroutine to perform one step of integration using a second-order Runge-Kutta method. All arguments have the same meaning as that for the subroutine RK4.

195. MSTEP Subroutine to solve initial value problems in ordinary differential equations, using a fourth-order multistep method with adaptive step size control. This subroutine accepts a system of first-order differential equations. It can be used with either subroutine ADAMS for Adams-Bashforth-Moulton predictor-corrector method, or subroutine GEAR for stiffly stable fourth-order method. The change can be affected by choice of a flag. By setting the flag appropriately, this subroutine can also be used to integrate the equation using a fixed step size. This subroutine is a crude implementation of multistep methods with simple techniques for adjusting the step-size which should work on simple problems. A more sophisticated implementation will be required to handle a larger fraction of equations efficiently. N is the number of first-order equations. Y is a real array of length 7N, the first N elements of which should contain the initial values of the solution at input. After execution, it will contain the solution at some intermediate points. The contents of this array must be preserved, if a second call to the subroutine is required for continuing the integration further. DY is a real array of length 7N, containing the first derivatives of the solution. The first derivative need not be supplied at the time of calling. The arrays Y and DY are used to store the solution at seven most recent points. The second index of the array is increased in a circular manner, to avoid frequent copying of the array elements. DIF is the name of the external subroutine, which is invoked for calculating the right-hand sides of the differential equations $\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$. H is the initial step size to be tried. This subroutine generates the starting values using the fourth-order Runge-Kutta method, which will adjust the step size if necessary. After the starting values are generated, the step size can be doubled or halved as explained in Section 12.3. After execution, H will contain the current value of the step size. T0 is the initial value of the independent variable t, at which the initial conditions are specified. After execution, the value of T0 will be updated to the point up to which the integration is successful. If the execution is successful the final value of T0 may be beyond the requested value of TN. The solution at TN is computed using interpolation between values of solution at different values of t. TN is the value of t at which the solution is required. If the solution is required at several points, then the subroutine can be called repeatedly after changing TN between every call. Other variables (including the scratch arrays) should not be changed, because the subroutine uses the previous values of the solution as the starting values for the new problem. YF is a real array of length N, which will contain the final value of the solution at the required point TN, provided the integration is successful. If the integration is aborted at some intermediate point after the starting values were calculated, then this array will contain the solution at the last successful point. REPS is the required relative accuracy in each component of the solution. This subroutine only tries to estimate the local truncation error. Hence, the computed solution may not be correct to the specified accuracy. NSTP is the number of evaluations of the right-hand side required by the subroutine. NSTP is initialised only during the first call (IFLG $= 0$ or 1) and hence it will accumulate the number of calls to DIF from the first call, until IFLG is reset to 0 or 1. NMAX is the maximum number of function evaluations that the subroutine is allowed to use. If NMAX ≤ 0, then a default value of NMX ($= 100000$) will be used. IER is the error parameter. IER $= 702$ implies that N ≤ 0, in which case, no calculations are performed. IER $= 724$ or 725 implies that subroutine STRT4 failed to generate the required starting values. IER $= 726$ implies that step size has become smaller than REPS$|$TN $-$ T0$|$. If in a small region the step size is actually expected to be smaller than this limit, then the corresponding statement in the subroutine may be modified. IER $= 727$ implies that the step size is too small for the arithmetic used (i.e., H $< \hbar|$T0$|$) and the calculations are aborted. IER $= 728$ implies that the integration could not be completed in the specified number (NMAX) of function evaluations. The failure of this routine can arise either because of singularity, or stiffness (with ADAMS), or because the specified accuracy is too high. For integration with fixed step size, IER may be set to 729 if the corrector fails to converge. IFLG is an integer variable used as a flag by the subroutine. If IFLG = 0 or 1, then the integration is started by generating fresh starting values using subroutine STRT4. For higher values of IFLG it is assumed that the starting values are already available in the arrays Y and DY. The subroutine itself resets the value of IFLG after generating the starting values. Hence, this parameter need not be reset between two calls, unless it is required to generate starting values again. If IFLG = 0 or 2, then the step size is adjusted according to the required accuracy. For IFLG = 1 and 3, the step size is kept fixed and no attempt is made to check the truncation error. However, the parameter REPS is used to check for convergence of the corrector iteration. In this case, if the iteration on corrector fails to converge, then IER will be set to 729. IST is an integer parameter to decide which multistep method is to be used. If IST = 0 the fourth-order Adams-Bashforth-Moulton predictor-corrector method is used, while for other values of IST, the fourth-order stiffly stable method due to Gear is used. WK is a real array, which is used as a scratch space by the subroutine. The length of this array should be 2N for Adams method and $3\mathrm{N} + \mathrm{N} \times \max(\mathrm{N}+4, 2\mathrm{N})$ for stiffly stable method. IWK is an integer array which is used as a scratch space. The length of this array should be N, if subroutine GEAR is used. Subroutine ADAMS does not need this array and a dummy array of length one will be sufficient in that case. This subroutine requires subroutine ADAMS, GEAR and GAUELM to perform the integration and subroutine STRT4 and RK4 to generate the starting values. In addition the subroutine DIF(T, N, Y, DY) must be supplied by the user to calculate the right-hand sides of the equations, i.e., the derivatives Y'. Here T is the value of independent variable and N is the number of equations. Y is the real array containing the dependent variables at time $t = \mathrm{T}$, where the derivatives need to be evaluated. DY is the real array which will contain the calculated value of the derivatives. DY(i) should contain $\mathrm{Y}'(i) = f_i(\mathrm{T}, \mathrm{Y})$.

196. ADAMS Subroutine to perform one step of solution of an initial value problem, using a fourth-order Adams-Bashforth-Moulton predictor-corrector method. This subroutine can be easily modified to use any other predictor-corrector formula. This subroutine is called by subroutine MSTEP to perform one step of integration. N is the number of first-order differential equations, Y and DY are real arrays of length 7N, as used by subroutine MSTEP. DIF is the name of the subroutine used to calculate the right-hand sides of the equations. H is the step size and T is the value of independent variable at which the solution is required. REPS is the specified accuracy. This subroutine does not check for the truncation error, but the parameter REPS is used to check for the convergence of corrector iteration. NSTP is the number of function evaluations used so far. This number is updated by the subroutine. IJ, IJM1, IJM2, IJM3 and IJM4 are respectively, the indices $j+1$, j, $j-1$, $j-2$ and $j-3$ in the predictor-corrector formula. IER is the error parameter. IER = 729 implies that the iteration on corrector has failed to converge. WK is a real array of length 2N, which is used to transmit the predicted value to subroutine MSTEP.

Subroutine DIF as described in the writeup for subroutine MSTEP must be supplied by the user.

197. STRT4 Subroutine to generate the starting values $\mathbf{y_1}$, $\mathbf{y_2}$ and $\mathbf{y_3}$ using a fourth-order Runge-Kutta method. The truncation error is estimated by performing one integration with double step size. This subroutine is called by the subroutine MSTEP. N is the number of first-order differential equations to be solved. Y and DY are real arrays of length 4N, containing the values of $\mathbf{y}$ and $\mathbf{y}'$ at the four starting points. While calling the subroutine, the initial values $\mathbf{y_0}$ must be stored in the first N elements of Y. Other elements of these arrays will be calculated by the subroutine. DIF is the name of the subroutine used to calculate the right-hand sides of the differential equations. H is the initial guess for the step size. If necessary the step size will be adjusted by the subroutine. T is the value of the independent variable at the initial point. If the execution is successful, then T is updated to T + 3H. REPS is the specified relative accuracy. IFLG is an integer parameter used as a flag. If IFLG = 0, then the step size may be adjusted while for other values of IFLG the step size will not be adjusted. For IFLG $\neq$ 0, the value of REPS is redundant, since no attempt is made to check the truncation error. TSTEP is the size of interval over which the integration is requested. It is used only for convergence check. NSTP is an integer variable which contains the number of function evaluations used so far. This value is updated by the subroutine. IER is an error parameter. IER = 724 implies that the routine failed to find the starting values, because the step size became too small. IER = 725 implies that the routine failed to find the starting values in the specified number (NIT = 10) of attempts. WK is a real array of length 2N used as a scratch space by the subroutine. This subroutine requires subroutines RK4 and DIF. Subroutine DIF as described in the writeup for subroutine MSTEP must be supplied by the user.

198. GEAR Subroutine to perform one step of solution of initial value problem using a fourth-order stiffly stable method based on the backward differentiation formula. This subroutine uses the Broyden's method to solve the implicit corrector formula. This subroutine is called by the subroutine MSTEP to perform one step of integration. N is the number of first-order differential equations, Y and DY are real arrays of length 7N as used by subroutine MSTEP. DIF is the name of the subroutine used to calculate the right-hand sides of the equations. H is the step size and T is the value of the independent variable at which the solution is required. REPS is the specified accuracy. This subroutine does not check for the truncation error, but the parameter REPS is used to check for the convergence of corrector iteration. NSTP is the number of function evaluations used so far. This number is updated by the subroutine. IJ, IJM1, IJM2, IJM3 and IJM4 are respectively, the indices $j+1$, j, $j-1$, $j-2$ and $j-3$ in the corrector formula. IFLAG is an integer variable used as a flag. If IFLAG = 0, initial approximation to the Jacobian is generated, otherwise the old approximation is used. In any case, Broyden's method will be used to update the initial approximation to the inverse of the Jacobian. If IFLAG = 0, it is set to 1 after calculation of Jacobian so that the Jacobian is not calculated again next time. IER is the error parameter. IER = 729 implies that the corrector iteration fails to converge. WK1 and WK are real arrays of length 3N and $\mathrm{N}\times\max(2\mathrm{N},\mathrm{N}+4)$ respectively, used as a scratch space. IWK is an integer array of length N used as a scratch space. Subroutine DIF as described in write-up for MSTEP must be supplied by the user. This subroutine also requires subroutine GAUELM to solve systems of linear equations. The parameter CFAC in the subroutine may need to be reduced if the subroutine MSTEP needs to adjust the step size too often.

199. EXTP Subroutine to solve initial value problems in ordinary differential equations using extrapolation method. This subroutine accepts a system of first-order differential equations. N is the number of first-order equations. Y is a real array of length N, which should contain the initial values of solution at input. After successful execution, it will contain the solution at the required point. If the execution is aborted in between, then it will contain the solution at some intermediate point T0, up to where the integration was successful. DY is a real array of length N containing the first derivative of the solution. The first derivative need not be supplied at the time of calling. DIF is the name of the external subroutine, which is invoked for calculating the right-hand sides of the differential equations $\mathbf{y}' = \mathbf{f}(t,\mathbf{y})$. H is the initial step size to be tried. The subroutine will adjust the step size if necessary, using the crude technique described in Section 12.5. After execution, H will contain the current value of the step size. T0 is the initial value of the independent variable t, at which the initial conditions are specified. After execution, the value of T0 will be updated to the point, up to which the integration is successful. TN is the value of t at which the solution is required. If the integration is successful, then T0 will be set equal to TN. Thus, if the solution is required at several points, then the subroutine can be called repeatedly after changing TN between every call. Other variables need not be changed. REPS is the required relative accuracy in each component of the solution. This subroutine only tries to estimate the local truncation error. Hence, the computed solution may not be correct to the specified accuracy. NSTEP is the number of function evaluations used by the subroutine. This number is initialised to zero at every call to EXTP. Thus if EXTP is called repeatedly, then the total number of function evaluations used will need to be accumulated separately. NMAX is the maximum number of function evaluations that the subroutine is allowed to use. If $\mathrm{NMAX} \leq 0$, then a default value of NMX (= 100000) will be used. IER is the error parameter. IER = 703 implies that $\mathrm{N} \leq 0$, in which case, no calculations are performed. IER = 730 implies that step size has become smaller than $\mathrm{REPS}|\mathrm{TN} - \mathrm{T0}|$. If in a small region the step size is actually expected to be smaller than this limit, then the corresponding statement in the subroutine may be modified. IER = 731 implies that the step size is too small for the arithmetic used (i.e., $\mathrm{H} < \hbar|\mathrm{T0}|$) and the calculations are aborted. IER = 732 implies that the integration could not be completed in the specified number (NMAX) of steps. IER = 733 implies that the denominator in the rational function extrapolation is zero. The failure of this routine can arise either because of singularity, or stiffness, or because the specified accuracy is too high. For rational function extrapolation, the failure can also be due to the fact that denominator has become zero. In this case, polynomial extrapolation can be tried. WK is a real array of length 39N, which is used as a scratch space by the subroutine. IFLG is an integer variable used to decide which type of extrapolation is to be used. If IFLG = 0, then polynomial extrapolation is used, otherwise rational function extrapolation is used. The subroutine DIF(T, N, Y, DY) must be supplied by the user to calculate the right-hand sides of the equations, i.e., the derivatives Y′. Here T is the value of independent variable and N is the number of equations. Y is the real array containing the dependent variables at time $t = \mathrm{T}$, where the derivatives need to be evaluated. DY is the real array which will contain the calculated value of the derivatives. DY(i) should contain $\mathrm{Y}'(i) = f_i(\mathrm{T}, \mathrm{Y})$.

200. FDM Subroutine to solve a two-point boundary value problem with separable boundary conditions, using finite difference method as explained in Section 12.8. If a uniform mesh is used, then it is also possible to estimate the first-order correction, using the method of deferred correction. N is the number of mesh points to be used. M is the number of first-order differential equations in the system. ML is the number of boundary conditions at the first boundary (i.e., T(1)). PAR is a real array which is passed on to the subroutine EQN and BCS for calculating the required information about the equation and the boundary conditions. This array can be used to pass any parameters that may be required by these subroutines. This array is not used by subroutine FDM, or any other routine called by it except EQN and BCS, and its size can be arbitrary depending on the requirements. X is a real array of length $\mathrm{M} \times \mathrm{N}$, which should contain the initial guess for the solution. After execution, it will contain the computed solution. If the problem is linear, then there is no need for initial approximation and this array could be initialised to zero or any other convenient value. The subroutine FDM treats it as a two-dimensional array of dimension (M, N) with X(I, J) containing the Ith component at Jth mesh point. Unfortunately, there is no provision to pass the first dimension of the array X, as specified in the calling program. Hence, to obtain meaningful results, the first dimension of the array X in the calling program must also be exactly equal to M. This problem arises because, while solving the finite difference equations, it is convenient to assume that the array elements are stored in consecutive positions without any gaps. XC is also a real array of length $\mathrm{M} \times \mathrm{N}$, which will contain the solution after applying the deferred correction. This array is also stored in the same format as the array X and the same remarks apply here too. Even if deferred correction is not requested, this array should be provided, since it is also used as a scratch space during the computations. The difference $\mathrm{XC(I,J)} - \mathrm{X(I,J)}$ gives an estimate of the truncation error in X(I, J) provided the deferred correction is calculated. T is a real array of length N, containing the mesh points t_j, $(j = 1, \ldots, \mathrm{N})$, to be used in the finite difference approximation. This array must be supplied by the user. If deferred correction is required, then the mesh must be uniformly spaced, otherwise spacing can be arbitrary. In any case, the mesh points t_j must be in either ascending or descending order. EQN is the name of the subroutine, which defines the differential equations. The differential equation is assumed to be in the form $B\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$, where B is a $\mathrm{M} \times \mathrm{M}$ matrix and $\mathbf{f}$ is a vector function of length M. The matrix B must be nonsingular. The elements of B and $\mathbf{f}$ at a given point T are calculated by the subroutine EQN. Apart from this, it must also calculate the Jacobian $\partial f_i/\partial y_j$ (stored in the array A). BCS is the name of the subroutine, which specifies the boundary conditions. The boundary conditions are defined by functions $g_i(t_1, \mathbf{y_1}) = \mathbf{0}$ for $i = 1, \ldots, \mathrm{ML}$ at the first point $t = \mathrm{T}(1)$, and $g_i(t_N, \mathbf{y_N}) = \mathbf{0}$ for $i = \mathrm{ML} + 1, \ldots, \mathrm{M}$ at the last point $t = \mathrm{T(N)}$. Apart from the functions g_i, this subroutine should also calculate the Jacobian matrix $\partial g_i/\partial y_j$ (stored in the array BC). The general form of these subroutines is as follows:

```
SUBROUTINE EQN(J,M,ML,PAR,A,B,Y,F,T)
IMPLICIT REAL*8(A-H,O-Z)
DIMENSION A(M+ML,M),B(M+ML,M),Y(M),PAR(*),F(M)

DO I=1,M
  F(I)=$f_I(\mathrm{T}, \mathrm{PAR}, \mathrm{Y})$
  DO K=1,M
    A(K,I)=$\partial f_K/\partial \mathrm{Y(I)}$
    B(K,I)=$b_{KI}(\mathrm{T}, \mathrm{PAR})$
  ENDDO
ENDDO
END

SUBROUTINE BCS(M,ML,PAR,BC,G,T1,TN,Y1,YN)
IMPLICIT REAL*8(A-H,O-Z)
DIMENSION PAR(*),BC(M+ML,M),G(M),Y1(M),YN(M)

DO I=1,M
  IF(I.LE.ML) THEN
    G(I)=$g_I(\mathrm{T1}, \mathrm{PAR}, \mathrm{Y1})$
  ELSE
    G(I)=$g_I(\mathrm{TN}, \mathrm{PAR}, \mathrm{YN})$
  ENDIF
  DO J=1,M
    BC(I,J)=$\partial g_I/\partial \mathrm{Y(J)}$
  ENDDO
ENDDO
END
```

It should be noted that the first dimension of A, B and BC is $\mathrm{M} + \mathrm{ML}$ instead of M. Here T, T1 and TN are the values of t at relevant points, while the real arrays Y, Y1 and YN contain the values of $\mathbf{y}$ at these points. In subroutine EQN the parameter J identifies the interval at which the functions are evaluated (i.e., $\mathrm{T} = (t_J + t_{J+1})/2$). This index is useful in cases, where the coefficients are read in the form of a table.

IWK is an integer array of length M $\times$ N, which is used as a scratch space. WK is a real array of length (M + ML) $\times$ 2M $\times$ (N + 1), which is used as a scratch space. IFLAG is an integer variable which is used as a flag. If IFLAG = 0 or 1, then the equation is treated as nonlinear and solution will be calculated iteratively. If IFLAG = 2 or 3, then the equations are treated as linear and calculation will be terminated after the first iteration. If IFLAG = 0 or 2, then the first-order correction will be calculated using the method of deferred correction. For other values of IFLAG, the deferred correction will not be calculated. The deferred correction can be calculated only if the mesh spacing is uniform and the number of points N is at least five. REPS is the specified accuracy. This parameter is only used to check for the convergence of Newton's method, for solution of finite difference equations (only for nonlinear equations i.e., IFLAG = 0, 1). This parameter has no effect on truncation error, which is determined by the mesh spacing. The truncation error can be estimated by the difference between X and XC, if deferred correction is used. IER is the error parameter. IER = 704 implies that N < 3, or M $\leq$ ML, or ML $\leq$ 0, in which case, no calculations are performed. IER = 734 implies that N $\leq$ 4 and deferred correction is requested. In this case, the deferred correction is not calculated. IER = 735 implies that the finite difference matrix is singular and the solution cannot be obtained. This problem may be due to some error in specifying the equation matrix or boundary conditions. IER = 736 implies that the mesh spacing is not uniform, in which case, the deferred correction will not be calculated. IER = 737 implies that the Newton's iteration for solving the finite difference equations fails to converge. This subroutine requires subroutine SETMAT for setting up the matrix of finite difference equations, and subroutine GAUBLK to solve the system of linear equations defined by the block matrix as explained in Section 12.8. Apart from these, the subroutines EQN and BCS specifying the problem must be supplied by the user.

201. GEVP Subroutine to solve a generalised eigenvalue problem for a system of differential equations, with separable boundary conditions, using finite difference method as explained in Section 12.9. If a uniform mesh is used, then it is also possible to estimate the first-order correction to the eigenvalue, using the method of deferred correction. N is the number of mesh points to be used. M is the number of first-order differential equations in the system. ML is the number of boundary conditions at the first boundary (i.e., T(1)). PAR is a real array which is passed on to the subroutines EQN, EQND, BCS and BCSD for calculating the required information about the equation and the boundary conditions. The first element of PAR is used to pass the eigenvalue and hence it should not be used for any other purpose. After execution, PAR(1) will contain the calculated eigenvalue. Other elements of this array can be used to pass any parameter that may be required by these subroutines. Other elements of the array are not used by subroutine GEVP or any other routine called by it except EQN, BCS, EQND and BCSD, and its size can be arbitrary depending on the requirements. X is a real array of length M $\times$ N, which will contain the computed eigenfunction. Even if the eigenfunction is not required, this array should be provided. The subroutine GEVP treats it as a two-dimensional array of dimension (M, N), with X(I, J) containing the Ith component at Jth mesh point. Unfortunately, there is no provision to pass the first dimension of the array X as specified in the calling program. Hence, to obtain meaningful results, the first dimension of the array X in the calling program must also be exactly equal to M. This problem arises, because while solving the finite difference equations it is convenient to assume that the array elements are stored in consecutive positions without any gaps. XC is also a real array of length $\mathrm{M} \times \mathrm{N}$, which will contain the left eigenvector of the finite difference matrix, which is required for calculating the deferred correction. This array is also stored in the same format as the array X and the same remarks apply here too. Even if deferred correction is not requested, this array should be provided, since it is also used as a scratch space during the computations. T is a real array of length N, containing the mesh points t_j for $j = 1, \ldots, \mathrm{N}$ to be used in the finite difference approximation. This array must be supplied by the user. If deferred correction is required, then the mesh must be uniformly spaced, otherwise spacing can be arbitrary. In any case, the mesh points t_j must be in either ascending or descending order. E0 is the initial guess for the eigenvalue. After execution, E0 will contain the corrected eigenvalue, provided deferred correction is calculated. EQN is the name of the subroutine which defines the differential equation. The differential equation is assumed to be in the form $B\mathbf{y}' = A\mathbf{y}$, where A and B are $\mathrm{M} \times \mathrm{M}$ matrices. The matrix B must be nonsingular. The elements of A and B which could be arbitrary functions of the eigenvalue λ and t, are calculated by the subroutine EQN. BCS is the name of the subroutine which specifies the boundary conditions. The boundary conditions are defined by $C_1\mathbf{y_1} = \mathbf{0}$, and $C_N\mathbf{y_N} = \mathbf{0}$, where C_1 and C_N are $\mathrm{ML} \times \mathrm{M}$ and $(\mathrm{M} - \mathrm{ML}) \times \mathrm{M}$ matrices, respectively. These matrices are combined in one $\mathrm{M} \times \mathrm{M}$ matrix BC, the first ML rows of which correspond to the boundary conditions at the first boundary ($t = \mathrm{T}(1)$), while the remaining rows refer to boundary conditions at the last point ($t = \mathrm{T}(\mathrm{N})$). EQND is the name of the subroutine to calculate the derivatives of matrices A and B (as defined in EQN) with respect to the eigenvalue λ. Similarly, BCSD is the name of the subroutine to calculate the derivatives of matrix BC (as defined in BCS) with respect to the eigenvalue λ. If deferred correction is not required, then the subroutines EQND and BCSD are not called, but in order to satisfy the compiler and the linker, dummy subroutines containing only the dimension statement may be required. The general form of these subroutines is as follows:

```
SUBROUTINE EQN(J,M,ML,PAR,A,B,Y,F,T)
IMPLICIT REAL*8(A-H,O-Z)
DIMENSION A(M+ML,M),B(M+ML,M),Y(M),PAR(*),F(M)

DO I=1,M
  F(I)=0.0 set the right-hand side to zero for consistency
  DO K=1,M
    A(K,I)=$a_{KI}$(T, $\lambda$ = PAR(1), PAR(2), ...)
```

```
      B(K,I)=$b_{KI}(\mathrm{T},\lambda=\mathrm{PAR}(1),\mathrm{PAR}(2),\ldots)$
    ENDDO
  ENDDO
  END

  SUBROUTINE BCS(M,ML,PAR,BC,G,T1,TN,Y1,YN)
  IMPLICIT REAL*8(A-H,O-Z)
  DIMENSION PAR(*),BC(M+ML,M),G(M),Y1(M),YN(M)

  DO I=1,M
    G(I)=0  set the right-hand side to zero for consistency
    DO J=1,M
      IF(I.LE.ML) THEN
        BC(I,J)=$c_{IJ}(\mathrm{T1},\lambda,\ldots)$
      ELSE
        BC(I,J)=$c_{IJ}(\mathrm{TN},\lambda,\ldots)$
      ENDIF
    ENDDO
  ENDDO
  END

  SUBROUTINE EQND(J,M,ML,PAR,A,B,T)
  IMPLICIT REAL*8(A-H,O-Z)
  DIMENSION A(M+ML,M),B(M+ML,M),PAR(*)

  DO I=1,M
    DO K=1,M
      A(K,I)=$\partial a_{KI}/\partial\lambda$
      B(K,I)=$\partial b_{KI}/\partial\lambda$
    ENDDO
  ENDDO
  END

  SUBROUTINE BCSD(M,ML,PAR,BC,T1,TN)
  IMPLICIT REAL*8(A-H,O-Z)
  DIMENSION PAR(*),BC(M+ML,M+1)

  DO I=1,M
    DO J=1,M
      BC(I,J)=$\partial c_{IJ}/\partial\lambda$
    ENDDO
  ENDDO
  END
```

It should be noted that the first dimension of A, B and BC is $\mathrm{M}+\mathrm{ML}$ instead of M. Here T, T1 and TN are the values of t at relevant points. The real arrays Y, Y1, YN, F and G are not really required for eigenvalue problem, but are provided for consistency. Thus, in EQN and BCS, F(I) and G(I) should be set to zero in order to be consistent with subroutine FDM, since the same subroutine SETMAT is used in both cases. In subroutine EQN and EQND, the parameter J identifies the interval at which the functions are evaluated (i.e., $T = (t_J + t_{J+1})/2$). This index is useful in cases, where the coefficients are read in the form of a table.

IWK is an integer array of length $M \times N$, which is used as a scratch space. WK is a real array of length $(M + ML) \times 2M \times (N + 1)$ used as a scratch space. IFLAG is an integer variable which is used as a flag. If IFLAG = 0, then only the eigenvalue is calculated. If IFLAG = 1, then eigenvector is also calculated, using the method of inverse iteration. If IFLAG = 2, then the first-order correction to the eigenvalue is also calculated, using the method of deferred correction. The deferred correction can be calculated only if the mesh spacing is uniform and the number of mesh points N is at least five. REPS is the specified accuracy. This parameter is only used to check for the convergence of Muller's method (or secant method) for finding zeros of the determinant, and the inverse iteration method for finding eigenvectors. This parameter has no effect on the truncation error, which is determined by the mesh spacing. If deferred correction is used, then the difference E0 − PAR(1) will give an estimate of the truncation error in PAR(1). RMX is a real variable, which is used to specify the region in which the eigenvalues are to be searched. This parameter is passed on to subroutine MULER2, (or SECANI) which terminates the iteration if at any stage $|\lambda| >$ RMX. IER is the error parameter. IER = 704 implies that $N < 3$, or $M \leq ML$ or $ML \leq 0$. In this case, no calculations are performed. IER = 734 implies that $N \leq 4$ and deferred correction is requested. In this case, the deferred correction is not calculated. IER = 735 implies that one of the pivots has vanished during the calculation and hence the eigenvector cannot be calculated. In this case, the eigenvalue will be calculated. This problem can be avoided by perturbing the eigenvalue slightly and recalculating the determinant before calculating the eigenvector. IER = 736 implies that the mesh spacing is not uniform, in which case the deferred correction will not be calculated. IER = 738 implies that the eigenvector vanishes, in which case the calculations are terminated. In this case also the eigenvalue is already calculated and will be stored in PAR(1). IER = 739 implies that the inverse iteration for calculating the eigenvector failed to converge. IER = 740 implies that the inverse iteration for calculating the left eigenvector failed to converge. Apart from these, other values of IER may be set by subroutine MULER2 (or SECANI), which is called to find zeros of the determinant. The subroutine GEVP requires subroutine SETMAT for setting up the matrix of finite difference equations, and subroutine GAUBLK to solve the system of linear equations, or to calculate the determinant of the block matrix as explained in Section 12.8. Subroutine MULER2, SECANI or any other equivalent routine is required for finding zeros of the determinant. It may be noted that subroutine MULER2 requires the function to be complex, but if we want to avoid unnecessary use of complex arithmetic for calculating real eigenvalues, then we can take the real part of CX and calculate the real function DET and pass it back to MULER2 in the complex variable CF. In principle, such a procedure can lead to trouble, if the iterates become complex at some stage and the next iterate has the same real part. This problem can be avoided if SECANI is used. Both SECANI and MULER2 use reverse communication technique and hence the control is passed back to GEVP when a function evaluation is required. Apart from these, the subroutines EQN, EQND, BCS and BCSD specifying the problem must be supplied by the user. For calculating complex eigenvalues, all real variables other that those starting with H, R and T should be declared to be complex in subroutines GEVP, SETMAT, GAUBLK, EQN, EQND, BCS, and BCSD. In this case it will be preferable to use MULER2. This process is implemented in GEVP_C.

202. GAUBLK Subroutine to solve a system of linear equations involving finite difference matrix of the form described in Section 12.8. This subroutine uses Gaussian elimination with partial pivoting. N is the number of mesh points, M is the number of first-order differential equations in the system. ML is the number of boundary conditions at the first boundary. A is a real array of length $(\mathrm{M}+\mathrm{ML}) \times 2\mathrm{M} \times \mathrm{N}$ containing the matrix (Section 12.8). IFLG is an integer variable used as a flag. If IFLG = 0, the subroutine performs elimination as well as the solution of linear equations. If IFLAG = 1, then only the elimination is performed. In both these cases, the determinant is calculated and IFLG is set to 2. If IFLG = 2, then it is assumed that elimination is already performed and the triangular factors are stored in the same array A. In this case, only the linear equations are solved. DET is a real variable specifying the scaled value of the determinant, IDET is an integer variable which contains the exponent part of the determinant. The actual value of the determinant is $\mathrm{DET} \times 2^{\mathrm{IDET}}$. INC is an integer array of length $\mathrm{M} \times \mathrm{N}$, which contains information about row interchanges used during Gaussian elimination. X is a real array of length $\mathrm{M} \times \mathrm{N}$, which should contain the right-hand sides of the linear equations. After execution, the solution will be returned in the same array. IER is the error parameter. IER = 735 implies that one of the pivots vanishes during elimination and hence the equations cannot be solved. In this case, the determinant will vanish, but that does not cause any problem for finding eigenvalues, which are obtained by finding zeros of the determinant. This subroutine is called by subroutines FDM and GEVP. GAUBLK_C is the complex version of GAUBLK, when the matrix is complex.

203. SETMAT Subroutine to setup the finite difference matrix for a system of first-order differential equations with separable boundary conditions, using a central difference approximation. This subroutine is called by subroutines FDM and GEVP. N is the number of mesh points, M is the number of first-order differential equations in the system, ML is the number of boundary conditions at the first boundary. A is a real array of length $(\mathrm{M}+\mathrm{ML}) \times 2\mathrm{M} \times \mathrm{N}$, which will contain the finite difference matrix for the linearised equations. BC is a real array of length $(\mathrm{M}+\mathrm{ML}) \times (\mathrm{M}+1)$, which is used to store information about the boundary conditions. X and XC are real arrays of length $\mathrm{M} \times \mathrm{N}$. X should contain the approximation to solution of the boundary value problem, while after execution, XC will contain the right-hand sides of the finite difference equations. X(I, J) should contain the Ith component at Jth mesh point. T is a real array of length N containing the mesh points. PAR is a real array containing parameters, which are passed on to the subroutines EQN and BCS. EQN and BCS are the names of subroutines for defining the differential equation and the boundary conditions, respectively. These subroutines must be supplied by the user, as explained in the write up for subroutines FDM or GEVP. SETMAT_C is the complex version of SETMAT, when the matrix is complex.

204. BSPODE Subroutine to solve a two-point boundary value problem with separable boundary conditions, using expansion method with B-spline basis functions as explained in Section 12.10. NK is the number of knots to be used. K is the order of B-splines to be used. K = 4 corresponds to cubic B-splines, while K = 2 gives linear B-splines, etc. The order may need to be increased to get higher accuracy. M is the number of first-order differential equations in the system. ML is the number of boundary conditions at the first boundary (i.e., T(1)). PAR is a real array which is passed on to the subroutine EQN and BCS for calculating the required information about the equation and the boundary conditions. This array can be used to pass any parameters that may be required by these subroutines. This array is not used by subroutine BSPODE, or any other routine called by it except EQN and BCS, and its size can be arbitrary depending on the requirements. X is a real array of length $\mathrm{M} \times \mathrm{N}$, which will contain the computed solution. X(I, J) will contain the Ith component of solution at TX(J). The first dimension of X must be equal to M in the calling program. A is a real array of length $(\mathrm{NK} + \mathrm{K} - 2) \times \mathrm{M}$ containing the coefficients of expansion in terms of B-splines. At the time of calling it should contain the initial guess to the coefficients, while after execution it will contain the calculated coefficients. If the problem is linear, then there is no need for initial approximation and this array could be initialised to zero or any other convenient value. The subroutine BSPODE treats it as a two-dimensional array of dimension $(\mathrm{NK} + \mathrm{K} - 2, \mathrm{M})$ with A(I, J) containing the coefficient of Ith basis function in Jth component of the solution.

$$\mathrm{X}_j(t) = \sum_{i=1}^{\mathrm{NK}+\mathrm{K}-2} \mathrm{A}(i,j)\phi_i(t), \qquad j = 1, \ldots, \mathrm{M}. \tag{B.93}$$

Unfortunately, there is no provision to pass the first dimension of the array X or A, as specified in the calling program. Hence, to obtain meaningful results, the first dimension of the array A in the calling program must also be exactly equal to $\mathrm{NK} + \mathrm{K} - 2$. This problem arises because, while solving the system of equations, it is convenient to assume that the array elements are stored in consecutive positions without any gaps. T is a real array of length NK, containing the knots t_j, $(j = 1, \ldots, \mathrm{NK})$, to be used for calculating the B-spline basis functions. This array must be supplied by the user. The knots must be in ascending order with T(1) and T(NK) as the two boundaries. N is the number of mesh points to be used for obtaining the equations connecting the coefficients of expansion, $\mathrm{N} \geq \mathrm{NK} + \mathrm{K} - 2$. TX is a real array of length N containing the mesh points to be used for calculating the coefficients. This array must be supplied by the user if IFLAG > 1, otherwise the subroutine computes the elements of this array assuming uniform spacing and covering the interval T(1) to T(NK). The solution will be calculated at all these points. If the solution is required at any other point then function BSPEVL can be used with the calculated coefficients. EQN is the name of the subroutine, which defines the differential equations. The differential equation is assumed to be in the form $B\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$, where B is a $\mathrm{M} \times \mathrm{M}$ matrix and $\mathbf{f}$ is a vector function of length M. The matrix B must be nonsingular. The elements of B and $\mathbf{f}$ at a given point T are calculated by the subroutine EQN. Apart from this, it must also calculate the Jacobian $\partial f_i/\partial y_j$ (stored in the array A). BCS is the name of the subroutine, which specifies the boundary conditions. The boundary conditions are defined by functions $g_i(t_1, \mathbf{y_1}) = \mathbf{0}$ for $i = 1, \ldots, \mathrm{ML}$ at the first point $t = \mathrm{T}(1)$, and $g_i(t_{NK}, \mathbf{y_{NK}}) = \mathbf{0}$ for $i = \mathrm{ML} + 1, \ldots, \mathrm{M}$ at the last point $t = \mathrm{T}(\mathrm{NK})$. Apart from the functions g_i, this subroutine should also calculate the Jacobian matrix $\partial g_i/\partial y_j$ (stored in the array BC). The general form of these subroutines is as follows:

```
SUBROUTINE EQN(J,M,ML,PAR,A,B,Y,F,T)
IMPLICIT REAL*8(A-H,O-Z)
DIMENSION A(M,M),B(M,M),Y(M),PAR(*),F(M)

DO I=1,M
  F(I)=$f_I(\mathrm{T}, \mathrm{PAR}, \mathrm{Y})$
  DO K=1,M
    A(K,I)=$\partial f_K/\partial \mathrm{Y}(\mathrm{I})$
    B(K,I)=$b_{KI}(\mathrm{T}, \mathrm{PAR})$
  ENDDO
ENDDO
END

SUBROUTINE BCS(M,ML,PAR,BC,G,T1,TN,Y1,YN)
IMPLICIT REAL*8(A-H,O-Z)
DIMENSION PAR(*),BC(M,M),G(M),Y1(M),YN(M)

DO I=1,M
  IF(I.LE.ML) THEN
    G(I)=$g_I(\mathrm{T1}, \mathrm{PAR}, \mathrm{Y1})$
  ELSE
    G(I)=$g_I(\mathrm{TN}, \mathrm{PAR}, \mathrm{YN})$
  ENDIF
  DO J=1,M
    BC(I,J)=$\partial g_I/\partial \mathrm{Y}(\mathrm{J})$
  ENDDO
ENDDO
END
```

It should be noted that these subroutines are same as those required by FDM except that the first dimension of arrays A, B and BC is M instead of M + ML. Here T, T1 and TN are the values of t at relevant points, while the real arrays Y, Y1 and YN contain the values of $\mathbf{y}$ at these points. In subroutine EQN the parameter J identifies the interval at which the functions are evaluated (i.e., TX(j)). This index is useful in cases, where the coefficients are read in the form of a table. WK is a real array of length

$$\mathrm{M} \times (\mathrm{N}+1) \times (\mathrm{M} \times (\mathrm{NK}+\mathrm{K}-2)+7) + (\mathrm{M} \times (\mathrm{NK}+\mathrm{K}-2))^2, \qquad \text{(B.94)}$$

which is used as a scratch space. IFLAG is an integer variable which is used as a flag. If IFLAG = 0 or 2, then the equation is treated as nonlinear and solution will be calculated iteratively. If IFLAG = 1 or 3, then the equations are treated as linear and calculation will be terminated after the first iteration. If IFLAG = 0 or 1, then the array TX need not be initialised at the time of calling as the subroutine will compute the elements assuming uniform spacing. For other values of IFLAG, the array TX must be supplied by the user. REPS is the specified accuracy. This parameter is only used to check for the convergence of Newton's method, for solution of system of equations (only for nonlinear equations i.e., IFLAG = $0, 2$). This parameter has no effect on truncation error, which is determined by the knots and the order of B-splines. IER is the error parameter. IER = 705 implies that N < 3, or M $\leq$ ML, or ML $\leq$ 0, or N < NK + K − 2, in which case, no calculations are performed. IER = 741 implies that the Newton's iteration for solving the system of equations fails to converge. This subroutine requires subroutine BSPLIN, BSPEVL, SVD and SVDEVL. Apart from these, the subroutines EQN and BCS specifying the problem must be supplied by the user.

B.13 Integral Equations

205. FRED Subroutine to solve a linear Fredholm equation as defined by Eq. (13.1) using quadrature methods. M is the number of abscissas to be used in the quadrature formula. A and B are real variables specifying respectively, the lower and the upper limit of the integral. WT and X are real arrays of length M, which will return the weights and abscissas used by the quadrature formula. F and FC are real arrays of length M, which will contain the calculated solution. F(I) will contain the computed solution at the abscissa X(I), while if the trapezoidal rule is used, then FC(I) will contain the value obtained after applying deferred correction, using the Gregory's formula including second-order differences. Array FC is not required for other quadrature formulae. Hence, in that case, a dummy array of length one may be provided. FG and FKER are the names of the function routines provided to calculate the functions $g(x)$ and $K(x,t)$. For Fredholm equation of the third kind $g(x)$ is not required, but a dummy routine giving some nonzero value (e.g., $g(x) = 1$) should be provided. In fact, this function supplies the starting vector for the inverse iteration and if necessary a reasonable guess for the eigenfunction may be provided via this function. However, in most cases, the inverse iteration method does converge to the eigenvalue, even when initial vector is far from the eigenvector and such a guess is not essential, but it can be used to improve the efficiency. EI is a real variable, which gives the initial guess for the eigenvalue of the Fredholm equation of the third kind. After execution, EI will contain the calculated eigenvalue. For equations of other kinds, EI will be ignored. WK is a real array used as a scratch space. The size of this array must be M^2 for Fredholm equations of the first and second kind, while it should be $2\mathrm{M}^2+\mathrm{M}$ for equations of the third kind. IWK is an integer array of length M used as a scratch space. IQ is an integer variable used to specify the quadrature formula to be used. If $\mathrm{IQ}=1$, then trapezoidal rule is used, for $\mathrm{IQ}=2$, Simpson's 1/3 rule is used, for $\mathrm{IQ}=4,8,16,32$ a composite rule using 4, 8, 16 or 32-point Gauss-Legendre formula is used. If IQ is negative, then it is assumed that weights and abscissas are supplied in the arrays WT and X. Other values of IQ will cause an error return. For $\mathrm{IQ}>1$ if the number of points are not compatible with the composite quadrature formula, then the next lower value of M is used. For example, if $\mathrm{IQ}=4$ and $\mathrm{M}=11$, then computations will be performed using only 8 points, but the value of M remains unchanged. In this case, a warning is issued by setting $\mathrm{IER}=-11$. IT is an integer variable, which specifies the kind of Fredholm equation to be solved. $\mathrm{IT}=1,2$ and 3 corresponds to the Fredholm equations of the first, second and third kind, respectively. This subroutine can only find a real eigenvalue, but it can be easily modified to find complex eigenvalues also. REPS is a real parameter specifying the required (relative) accuracy in calculating the eigenvalues. This parameter is used only if $\mathrm{IT}=3$. It is only passed on to the subroutine INVIT for calculating the eigenvalue. IER is the error parameter. $\mathrm{IER}=706$ implies that $\mathrm{IT}>3$, or $\mathrm{IT}\leq 0$, or the number of points M are not sufficient for application of the specified quadrature formula. In this case, no calculations are performed. $\mathrm{IER}=707$ implies that the value of IQ is not proper, in which case also no calculations are performed. $\mathrm{IER}=-11$ implies that the number of points have been adjusted. In this case, the calculations are performed with a smaller value of M, which is applicable for the quadrature formula requested. Apart from these, other values may be set by subroutines GAUELM and INVIT, which are called by this subroutine. Subroutine FRED requires subroutine GAUELM to solve the system of linear equations, and subroutine INVIT (for $\mathrm{IT}=3$) to calculate the eigenvalue and the eigenvector. It should be noted that subroutine INVIT is called with $\mathrm{IFLG}=0$, which keeps the shift P constant, to ensure convergence to the nearest eigenvalue. Consequently, the convergence could be rather slow in some cases. It may be more efficient to use other values of IFLG, but in that case, the inverse iteration will not necessarily converge to the nearest eigenvalue. FUNCTION FG(X) and FUNCTION FKER(X, T) must be supplied by the user.

206. FREDCO Subroutine to solve a linear Fredholm equation of the first or second kind using the collocation method. The function is approximated by an expansion of the form $f(x)=\sum_{i=1}^{N} a_i$ PHI(i,x). The basis functions PHI(I,

X) must be supplied by the user. N is the number of points (which equals the number of basis functions) to be used in the collocation method. A and B are real variables specifying respectively, the lower and the upper limit of the integral. F is a real array of length N, which will contain the calculated coefficients of the expansion. F(I) contains the coefficient (a_i) of PHI(I, X) in the expansion for the solution. X is a real array of length N, which specifies the points to be used for collocation. This array must be supplied by the user. REPS and AEPS specify the accuracy with which the integrals are to be calculated. If IQ $= 0$, then these parameters are passed on to the subroutine ADPINT, while in other cases, these parameters are ignored. WK is a real array of length N^2 used as a scratch space. IWK is an integer array of length N used as a scratch space. IQ is an integer variable used to specify the treatment of integrals required to calculate

$$\mathrm{PSI(I,\ X)} = \int_A^B \mathrm{FKER(X,\ T)\ PHI(I,\ T)}\ d\mathrm{T}\,. \tag{B.95}$$

If IQ $\neq 0$, then it is assumed that the function PSI(I, X) is supplied separately, while if IQ $= 0$, then the integrals are calculated using the subroutine ADPINT. This subroutine uses external function routine FUNK to calculate the integrand, which in turn requires the functions PHI(I, X) and FKER(X, T). IT is an integer variable which specifies the kind of Fredholm equation to be solved. IT $= 1$ and 2 corresponds to the Fredholm equations of the first and second kind, respectively. IER is the error parameter. IER $= 708$ implies that IT > 2, or IT ≤ 0, or N < 1, in which case, no calculations are performed. Apart from these, other values may be set by the subroutines GAUELM or ADPINT, which are called by this subroutine. This subroutine requires subroutine GAUELM to solve the system of linear equations, subroutines ADPINT and KRONRD (for IQ $= 0$) to evaluate the integrals and FUNCTION FUNK(X) to evaluate the required integrand. Further, the functions FG(X) ($= g(x)$), PHI(I, X) ($= \phi_i(x)$) and FUNCTION FKER(X, T) ($= K(x,t)$, for IQ $= 0$) or FUNCTION PSI(I, X) (for IQ $\neq 0$) must be supplied by the user. The names of these functions routines are fixed and cannot be passed on to the subroutine FREDCO, because the functions FKER and PHI are called by the function FUNK, and there is no simple way of passing on these names to FUNK. In order to maintain consistency, the names of other function routines are also not passed on to the subroutine. Common block ZZFRED is used to pass on variables to FUNK, XI and II in the common block are respectively the values of x and i for evaluating the integrand. This subroutine can also be used to solve linear Volterra equations, by defining the kernel to be zero for $t > x$.

207. FUNK Function routine to calculate the integrand for calculating $\psi_i(x)$ as required by subroutine FREDCO. Function FKER(X, T) is the kernel $K(x,t)$, while PHI(I, T) are the basis functions $\phi_i(t)$ used for the expansion. Names of these routines have to be the same as there is no provision to pass on the names to FUNK. The common block ZZFRED is used to pass on the values of X and I.

208. RLS Subroutine to solve a linear inverse problem in one dimension using Regularised Least Squares (RLS) technique with B-spline basis functions. The inverse problem is defined as

$$d_i = \int_a^b K_i(t) f(t)\, dt, \qquad i = 1, 2, \ldots, \mathrm{NM}, \tag{B.96}$$

where the unknown function $f(t)$ is to be obtained using given data d_i and kernels $K_i(t)$ for all i. The corresponding forward problem where d_i are calculated for a given $f(t)$ is solved by the subroutine FORW. The kernel is given in the form of table of values at a grid of points in t covering the required interval. Alternately, it is possible to provide directly the coefficients of matrix which are

$$a_{ij} = \int_a^b K_i(t)\phi_j(t)\, dt\,, \tag{B.97}$$

where $\phi_j(t)$ is the jth basis function. This form is more efficient to use and can be utilised when many options are to be tried with the same data points and set of knots. The subroutine itself calculates these integrals and hence during second and subsequent calls this option can be used, provided the data points and knots have not been modified. Using the coefficients a_{ij} instead of kernels also saves storage space. It may be noted that for this to be useful, d_i need not have the same values between different sets, only the set of NM kernels should be the same between different inversion problems. This option is useful as the solution will need to be calculated with different values of regularisation parameter before the optimum value is found. This routine can also estimate the errors in computed solution using Monte Carlo simulation.

NK is the number of knots defining the B-spline basis functions. The number of basis functions would be $\mathrm{NK}+\mathrm{K}-2$. XO is a real array of length at least NK which should contain the knots for defining the B-splines. The knots must be in ascending order with XO(1) containing the first knot. K is the order of B-splines required, $\mathrm{K} = 2$ for linear B-splines while $\mathrm{K} = 4$ for cubic B-splines, etc. NR is the number of points used for defining the kernels. R is a real array of length NR, which should contain the points r_i at which kernel values are available. This array should be in ascending order. The same set of points are used to calculate the solution after the coefficients of expansion are obtained. Hence, this array will be required even if matrix coefficients are directly available (unless $\mathrm{NR} = 0$). RKER is a real array of length $\mathrm{IK} \times \mathrm{NR}$ containing the kernels for the inverse problem. RKER(i, j) should contain $K_i(r_j)$. This array must be supplied if $\mathrm{IFLG} < 2$, otherwise it is not required. The mesh r_j need not be uniformly spaced as the integrals are evaluated using the trapezoidal rule. Since the accuracy of trapezoidal rule is low and also because in realistic problems the kernels are generally highly oscillatory, a large number of mesh points will be required to define the integrals to a good accuracy. IK is the first dimension of arrays RKER, AC and A, as specified in the calling program. IK must be at least $\mathrm{NM} + \mathrm{NS}$. AC is a real array of length $\mathrm{IK} \times (\mathrm{NK} + \mathrm{K} - 2)$

containing the coefficients of the matrix defining the inversion problem. AC(i, j) should contain the coefficient a_{ij} defined by (B.97). If IFLG < 2, these coefficients are calculated by evaluating the integrals in (B.97) using the trapezoidal rule. These calculations require substantial computing and hence on subsequent attempts the calculations can be suppressed by using IFLG $= 2, 3$. The matrix coefficients depend on the kernels (and the number of data points) and the knots and order of B-splines. Thus if these are not modified between two calls, it is preferable to use IFLG $= 2$ to find the solution using pre-calculated coefficients. If either the kernels or basis functions are modified then IFLG should be reset to 0 to force calculations of the coefficients once again. NM is the number of data points (and kernels) in the inversion problem. For each data point, the corresponding kernel must be supplied. NS is the number of points to be used for calculating the regularisation term. The subroutine chooses a uniform mesh covering the full interval for this purpose. ALP is the regularisation parameter, which must be positive. IDE is an integer parameter, which specifies the order of derivative to be used for regularisation. IDE should be 1 or 2 for first or second derivative smoothing. DI is a real array of length NM, which should contain the data d_i, for inversion. DE is a real array of length NM, containing the estimated errors in d_i. These errors must be positive. DF is a real array of length NM, which will contain the normalised residuals obtained by fit. If DS(I) is the computed value of DI(I) using the solution, then the residual DF(I)=(DI(I)−DS(I))/DE(I). Ideally, these residuals should have a Gaussian distribution with zero mean and unit variance. A significant trend in the residual would generally imply that ALP should be reduced or number of knots is not sufficient to represent the solution. F is a real array of length NR which will contain the calculated solution at R(I). B is a real array of length NM + NS, which will contain the coefficients of expansion. The computed solution is

$$f(t) = \sum_{i=1}^{\mathrm{NK}+\mathrm{K}-2} \mathrm{B}(i)\phi_i(t). \tag{B.98}$$

These coefficients can be used to compute the solution at any required point using FUNCTION BSPEVL. Although the number of coefficients is only NK + K − 2, the rest of the array is used as scratch space.

IFLG is an integer parameter, which specifies the type of calculation required. If IFLG $= 0$, the matrix coefficients are computed using the kernels. The system of equations is solved to find the solution and IFLG is set to 4 after calculations. The calculation of integrals to find matrix coefficients take significant time and hence this option should be used only when the coefficients are not available. After the first call to RLS, the coefficients are computed and returned in array AC. These coefficients should be used again if another solution is required with same set of kernels but with different regularisation (ALP or IDE) or with different data points DI for the same set of kernels. If IFLG $= 1$, the matrix coefficients are computed using the kernels. The system of equations is setup and the SVD of the matrix is computed, but the solution for given DI is not computed. In this case also IFLG is set to 4 after calculations. Thus during subsequent calls solution for different DI's can be computed using SVDEVL. If IFLG = 2, the matrix coefficients are assumed to be known and available in array AC. These coefficients are used to setup the matrix and find the solution and IFLG is set to 4 after calculations. This option is more efficient than IFLG = 0, and should be used when the coefficients have already been computed. If IFLG = 3, the matrix coefficients are assumed to be known and available in array AC. These coefficients are used to setup the matrix and its SVD is computed, but the solution for given DI is not computed. In this case also IFLG is set to 4 after calculations. Thus during subsequent calls solution for different DI's can be computed using the subroutine SVDEVL. If IFLG = 4, the SVD of equation matrix is assumed to be available in arrays, A, AV and SIGMA. These can be used to compute the solution for a given data set DI very efficiently. It may be noted that the matrix of equations depends only on the knots, kernels, regularisation and errors DE, but not on DI. So if only DI is changed there is no need to recompute the SVD. This is very useful, since on many applications several different data sets have to be inverted. Another application comes in estimating errors in computed inverse solution. For this purpose we need to solve the problem with different DI, which differ only in addition of different realisation of random errors to same basic data. Thus for estimating the errors in inversion although several different solutions are required, the net effort is not much since the SVD, which takes much more time, is computed only once. It should be noted that in all cases the subroutine sets IFLG to 4 so if during subsequent call the matrix needs to be changed, IFLG must be set to 0 or 2 before calling. This would be needed if either the knots, kernels or errors DE are changed. If only errors are modified, then IFLG can be set to 2, to avoid computation of coefficients. While if kernels or knots are modified then IFLG should be set to 0, so that the coefficients are recomputed.

IER is the error parameter. IER = 709 implies that $\mathrm{NM} \leq \mathrm{NK}+\mathrm{K}-2$ or $\mathrm{IK} < \mathrm{NM}+\mathrm{NS}$ or $\mathrm{IV} < \mathrm{NK}+\mathrm{K}-2$. IER = 710 implies that $\mathrm{ALP} < 0$ or $\mathrm{IDE} < 1$ or $\mathrm{IDE} > 2$. In all these cases no calculations are done. Other values may be set by subroutines BSPLIN or SVD which are called. It may be noted that the subroutine allows solution to be computed even when ALP = 0, although the solution would be unacceptable in this case. REPS is the required accuracy for solution of equations using SVD. All singular values less than REPS times the largest singular value will be set to zero during solution. CHISQ is the computed value of χ^2 define by

$$\chi^2 = \sum_{i=1}^{\mathrm{NM}} \mathrm{DF}(i)^2 = \sum_{i=1}^{\mathrm{NM}} \left(\frac{1}{\mathrm{DE}(i)}\right)^2 \left(\mathrm{DI}(i) - \sum_{j=1}^{\mathrm{NK+K-2}} a_{ij}\mathrm{B}(j)\right)^2 , \qquad \text{(B.99)}$$

SUMD is the value of regularisation term computed at the solution, for IDE = 2, it is given by

$$\mathrm{SUMD} = \left(\frac{\mathrm{R(NR)} - \mathrm{R}(1)}{\mathrm{NS}-1}\right) \sum_{i=1}^{\mathrm{NS}} \left(\frac{d^2 f}{dt^2}(t_i)\right)^2 , \qquad \text{(B.100)}$$

where t_i are the points at which smoothing is applied. For IDE $= 1$ the second derivative is replaced by first derivative. The values of CHISQ and SUMD for different values of ALP can be used to infer the optimal value of ALP using L-curve. A is a real array of length IK $\times$ (NK $+$ K $-$ 2) containing the matrix U of SVD of the matrix of equations. AV is a real array of length IV $\times$ (NK $+$ K $-$ 2) containing the matrix V of SVD of the matrix of equations. IV is the first dimension of array V as declared in the calling program. SIGMA is a real array of length (NK $+$ K $-$ 2) containing the singular values of the matrix. If IFLG < 4, the arrays A, AV and SIGMA will be calculated, otherwise they must be supplied. NSIM is the number of data sets to be used for Monte Carlo simulation for estimating the errors in solution. If NSIM < 2, error estimates are not calculated. FE is a real array of length NR which will contain the estimated errors. FE(I) would contain the estimated error in F(I). This is calculated only if NSIM > 1. WK is a real array of length NR$\times$NSIM$+$NM $+$ NS$+$4NK$+$5K$+$2, used as scratch space for simulations if NSIM > 1. This subroutine requires subroutines BSPLIN, BSPEVL, SVD, SVDEVL and RANGAU.

209. FORW To solve the forward problem corresponding to the inverse problem solved by RLS. This routine evaluates the required integrals using the trapezoidal rule. NP is the number of points used in defining the kernels. NM is the number of data points, d_i to be calculated. This should be the same as the number of kernels that are supplied in RKER. R is a real array of length NP containing the points at which values of kernels are available. RKER is a real array of length IK $\times$ NP containing the kernels for the inverse problem, RKER$(i,j) = K_i(\mathrm{R}(j))$. These kernels are the same as what are used by RLS for inversion. IK is the first dimension of RKER as declared in the calling program, IK $\geq$ NM. DI is a real array of length NM, which will contain the calculated data points d_i, using the kernels. F is a real array of length NP containing the function values, F$(i) = f(\mathrm{R}(i))$. If IFLG $= 0$, the function values are calculated using user supplied function routine FUN, otherwise, these values must be supplied while calling the subroutine. FUN is the name of the function routine to calculate the given function. This is used only if IFLG $= 0$, otherwise the function values are to be supplied in the array F. In the latter case a dummy routine will still be required to satisfy the compiler or linker. IER is the error parameter, IER $= 711$ implies that IK $<$ NM and no calculations are done. IFLG is an integer parameter that is used as a flag to decide the type of computation required. If IFLG $= 0$, then the function values are calculated using a user supplied routine FUN. These values are returned in the array F and IFLG is set to 1, so that next time the values need not be calculated. If data points corresponding to different kernels are required then IFLG need not be reset. If a different FUN is used next time then IFLG must be reset to 0. For other values of IFLG the function values must be supplied in the array F. FUNCTION FUN(X) must be supplied by the user to calculate $f(x)$. If the data points DI calculated by this routine are supplied to RLS without adding any errors and using the same kernels with very small RLM, then the inverted function should match the function FUN used by FORW to a very good accuracy.

210. VOLT Subroutine to solve linear Volterra equations using quadrature method based on the trapezoidal rule. N is the number of points at which the function value needs to be estimated. A is a real variable specifying the lower limit of the integral. It is also the initial point, from which the solution is required. H is a real variable specifying the step size to be used in the quadrature method. The points are assumed to have a uniform spacing of H. F and X are real arrays of length N, which will contain the calculated value of the solution at a set of uniformly spaced points. F(I) is the calculated solution at X(I). FG and FKER are the names of the function routines used to calculate the functions $g(x)$ and the kernel $K(x,t)$, respectively. IT is an integer variable specifying the kind of integral equation. If IT = 2, Volterra equation of the second kind are solved, while for other values of IT a Volterra equation of the first kind is solved. If IT = −1, then the computed values are smoothed as explained in Section 13.8. This option can be used only for equations of the first kind. It should be noted that smoothing cannot be applied to the first and the last point. IER is the error parameter. IER = 712 implies that N < 3, in which case no calculations are performed. IER = 751 implies that the denominator is zero at some stage, in which case, no further calculations can be performed. This failure usually occurs for equations of the first kind when either H = 0 or $K(x,x) = 0$. FUNCTION FG(X) and FUNCTION FKER(X, T) must be supplied by the user.

211. VOLT2 Subroutine to solve nonlinear Volterra equation of the second kind

$$\int_{\mathrm{A}}^{x} K(x,t,f(t))\,dt = f(x) + g(x), \tag{B.101}$$

using quadrature method based on the Simpson's rule. N is the number of points at which the function value needs to be estimated. A is a real variable specifying the lower limit of the integral. It is also the initial point from which the solution is required. H is a real variable specifying the step size to be used in the quadrature method. The points are assumed to have a uniform spacing of H. F and X are real arrays of length N, which will contain the calculated value of the solution at a set of uniformly spaced points. F(I) is the calculated solution at X(I). FG and FKER are the names of the function routines used to calculate the functions $g(x)$ and the kernel $K(x,t,f)$, respectively. REPS is a real parameter, which specifies the (relative) accuracy to which the resulting nonlinear algebraic equations are solved. This parameter does not control the truncation error, which is determined by H. It is only used to decide the termination criterion for the fixed-point iteration, while solving the nonlinear equations for F(I). IER is the error parameter. IER = 712 implies that N < 3, in which case no calculations are performed. IER = 752 implies that the fixed-point iteration fails to converge at some point, in which case, the calculations are aborted at that point. FUNCTION FG(X) and FUNCTION FKER(X, T, F) must be supplied by the user. It may be noted that FKER is not the kernel in the usual sense since it also includes the unknown function $f(t)$ as it appears inside the integral in Eq. (B.101).

B.14 Partial Differential Equations

All subroutines in this chapter are simple implementations of finite difference methods. Although these subroutines may accept equations in somewhat general form with variable coefficients, the result may not be reliable in all cases. For example, if the coefficient A of the parabolic equation in CRANK is negative in some range of (x, t) values, no reliable solution can be computed. The subroutine does not check for such inconsistencies and no error message will be issued. Similarly, some subroutines allow for rather general boundary conditions, but the accuracy may be only first order in Δt, if the boundary conditions depend on time explicitly. Further, for special differential equations, it is possible to improve the efficiency significantly by writing the difference equations directly without calculating the coefficients. Similarly, the boundary conditions can be implemented directly to improve efficiency in some of the subroutines (e.g., ADI). On the other hand, those subroutines which accept boundary condition in simple Dirichlet form only, can be modified to incorporate more general boundary conditions. Hence, these subroutines can only be treated as concrete (and straightforward) examples of implementing the algorithms covered in Chapter 14. They are not expected to be particularly efficient or robust. Users are expected to modify these routines to suit their problems.

212. CRANK Subroutine to solve a linear parabolic equation of the form

$$\frac{\partial u}{\partial t} = A(x,t)\frac{\partial^2 u}{\partial x^2} + B(x,t)\frac{\partial u}{\partial x} + C(x,t)u + D(x,t), \tag{B.102}$$

subject to boundary conditions

$$\begin{aligned} A_0(t)u(\mathrm{X0},t) + B_0(t)\frac{\partial}{\partial x}u(\mathrm{X0},t) &= F_0(t),\\ A_n(t)u(\mathrm{XN},t) + B_n(t)\frac{\partial}{\partial x}u(\mathrm{XN},t) &= F_n(t), \end{aligned} \tag{B.103}$$

using the Crank-Nicolson difference scheme (14.29). Although the coefficients in the boundary conditions are functions of time, if $B_0 = 0$ at some time step, then it must be so at every time step. The same is true for the coefficient B_n. This essentially means that if boundary condition is of Dirichlet type, it should remain so at all times. T is the initial value of time. After execution, it is replaced by the value of t at the last point. DT is the time step to be used in the computations. $(\mathrm{X0}, \mathrm{XN})$ is the range of x values over which the equation needs to be solved. NT is the number of time steps each of length DT, over which the equation is to be integrated. NX is the number of mesh points in the X direction. X is a real array of length NX, which contains the mesh points in X direction. This array need not be initialised at input, since the subroutine assumes a uniform mesh spacing and calculates the required mesh points. U is a real array of length NX, containing the solution at some time step. This array can be used to supply the initial values, while after execution, it will contain the solution at the required time. COF and BC are the names of the subroutines used to calculate the coefficients in the differential equation and the boundary conditions, respectively. SUBROUTINE COF(X, T, A, B, C, D) and SUBROUTINE BC(T, X0, XN, A0, B0, F0, AN, BN, FN) must be supplied by the user. Subroutine COF should calculate the coefficients A, B, C, D at a given value of X and T. Subroutine BC should calculate the coefficients A0, B0, F0, AN, BN and FN for a given value of T at the end points X0 and XN. FIC is the name of the function routine, which may be used to calculate the initial values for the solution, when IFLG = 0. FUNCTION FIC(X, T) must be supplied by the user if IFLG = 0. This routine calculates the solution $u(x,t)$ at the initial time T. For other values of IFLG a dummy function routine FIC may be required to satisfy the Fortran compiler or the linker. IER is the error parameter. IER = 713 implies that DT = 0, or XN = X0, or NX $\leq$ 2, in which case, no calculations are performed. IER = 761 implies that the difference equations are singular and the solution cannot be calculated. This problem may arise due to some error in specifying the equation or the boundary conditions. IFLG is an integer variable used as a flag. If IFLG = 0, the initial values are calculated using the function routine FIC supplied by the user. For other values of IFLG the initial values must be supplied in the array U at the time of calling. In any case, the subroutine sets IFLG to 1, so that during subsequent calls to the subroutine, the initial values are taken from the array U. This subroutine can be used to calculate the solution at several values of T by calling it repeatedly, after changing DT and NT appropriately. Other variables should not be changed. WK is a real array of length 8 $\times$ NX used as a scratch space by the subroutine.

213. LINES Subroutine to solve a system of nonlinear parabolic equations of the form

$$\frac{\partial u_i}{\partial t} = f_i\left(x, t, \mathbf{u}, \frac{\partial \mathbf{u}}{\partial x}, \frac{\partial^2 \mathbf{u}}{\partial x^2}\right), \qquad (i = 1, 2, \ldots, \mathrm{NE}), \tag{B.104}$$

subject to Dirichlet boundary conditions $\mathbf{u}(\mathrm{X0}, t) = \mathbf{g_1}(t)$ and $\mathbf{u}(\mathrm{XN}, t) = \mathbf{g_2}(t)$, using the method of lines. This subroutine is actually used through the subroutines MSTEP or RKM of Chapter 12. Since the system of resulting ordinary differential equations are expected to be stiff, it is preferable to use the subroutine MSTEP with GEAR. Subroutine LINES specifies the system of ordinary differential equations, as required by the subroutine MSTEP (subroutine DIF in the arguments for MSTEP). The parameters for the partial differential equations are passed via the common block ZZLINE. The relevant variables in this common block must be initialised before calling the subroutine MSTEP. NX is the number of mesh points in the x direction. NE is the number of parabolic equations in the system. DX is the step size in x direction. $(\mathrm{X0}, \mathrm{XN})$ is the interval over which the solution is to be calculated. Since the boundary conditions are assumed to be in the Dirichlet form, the solution at the end points need not be calculated. Hence, the number of ordinary differential equations N = (NX − 2)NE. X is a real array of length NX − 2, containing the mesh points x_i, excluding the end points. This array must be initialised before calling the subroutine MSTEP. The dimension of this and other arrays in common block must match that declared in the calling program. U0 and UN are real arrays of length NE, containing the boundary values (at X0 and XN) of the solution at the current time. These arrays need not be initialised before calling MSTEP, since the boundary values are calculated by subroutine LINES using a user supplied subroutine BC. UX is a real array of length $2 \times$ NE, used as a scratch space to store intermediate numbers by subroutine LINES. The arguments of LINES have the same definition as specified for subroutine DIF as required by MSTEP or RKM. T is the value of t at which the derivatives of solution are required. N is the number of ordinary differential equations (which would be (NX − 2)NE). U and DU are real arrays of length N. U will specify the values of solution at required point and DU will contain the time derivative of all these functions as calculated by LINES.

This subroutine requires subroutines FCN and BC to supply the information about the equations and the boundary conditions. SUBROUTINE FCN(NE, X, T, U, UX, UXX, DU) calculates the derivatives

$$\mathrm{DU(I)} = \frac{\partial u_i}{\partial t} = f_i(\mathrm{X,T,U,UX,UXX}), \qquad (\mathrm{I} = 1,\ldots,\mathrm{NE}), \tag{B.105}$$

for a given value of X, T, U(J), UX(J), UXX(J), (J = 1, . . . , NE), as specified by the differential equation. Here U, UX, UXX and DU are real arrays of length NE, containing $\mathbf{u}$, $\partial\mathbf{u}/\partial x$, $\partial^2\mathbf{u}/\partial x^2$, $\partial\mathbf{u}/\partial t$ respectively. SUBROUTINE BC(NE, T, X0, XN, U0, UN) calculates the boundary values U0(I), UN(I), (I = 1, . . . , NE) for a given value of T, X0 and XN. The names of the subroutines FCN and BC are fixed and cannot be passed on as arguments. The initial values should be supplied to the subroutine MSTEP at the time of calling. The NE × (NX − 2) components of the initial vector are arranged in the normal Fortran order, with NE components of $\mathbf{u}$ at each of the NX − 2 mesh points stored consecutively. Thus, $u_j(x_k)$ occupies the $(k-1)\mathrm{NE}+j$ position in the initial vector. The final result at the required value of time will also be returned in the same order.

214. ADM Subroutine to solve a parabolic equation in two space variables

$$\begin{aligned}\frac{\partial u}{\partial t} = A_{xx}(x,y,t)\frac{\partial^2 u}{\partial x^2} + A_{yy}(x,y,t)\frac{\partial^2 u}{\partial^2 y} + A_x(x,y,t)\frac{\partial u}{\partial x} + A_y(x,y,t)\frac{\partial u}{\partial y}\\ + A_u(x,y,t)u + A_0(x,y,t)\end{aligned} \tag{B.106}$$

with Dirichlet boundary conditions over a rectangular region, using alternating direction method. This subroutine uses the difference scheme

$$\begin{aligned}
\frac{u_{jl}^{n+1/2}-u_{jl}^{n}}{\frac{1}{2}\Delta t} &= \frac{1}{(\Delta x)^2}A_{xx}^{n+1/2}\delta_x^2 u_{jl}^{n+1/2}+\frac{1}{2\Delta x}A_x^{n+1/2}\delta_x u_{jl}^{n+1/2}\\
&\quad+\frac{1}{(\Delta y)^2}A_{yy}^{n}\delta_y^2 u_{jl}^{n}+\frac{1}{2\Delta y}A_y^{n}\delta_y u_{jl}^{n}\\
&\quad+\frac{1}{2}(A_u^n u_{jl}^n+A_u^{n+1/2}u_{jl}^{n+1/2})+\frac{1}{2}(A_0^{n+1/2}+A_0^n),\\
\frac{u_{jl}^{n+1}-u_{jl}^{n+1/2}}{\frac{1}{2}\Delta t} &= \frac{1}{(\Delta x)^2}A_{xx}^{n+1/2}\delta_x^2 u_{jl}^{n+1/2}+\frac{1}{2\Delta x}A_x^{n+1/2}\delta_x u_{jl}^{n+1/2}\\
&\quad+\frac{1}{(\Delta y)^2}A_{yy}^{n+1}\delta_y^2 u_{jl}^{n+1}+\frac{1}{2\Delta y}A_y^{n+1}\delta_y u_{jl}^{n+1}\\
&\quad+\frac{1}{2}(A_u^{n+1} u_{jl}^{n+1}+A_u^{n+1/2}u_{jl}^{n+1/2})+\frac{1}{2}(A_0^{n+1/2}+A_0^{n+1}),
\end{aligned}\tag{B.107}$$

where all coefficients on the right-hand side are evaluated at $x = x_j$ and $y = y_l$. T is the initial value of time, DT is the time step to be used. After execution, T will be replaced by the current value of time. (X0, XN) and (Y0, YN) are the intervals along x and y axes, in which the solution is required. NT is the number of time steps, NX and NY are the number of mesh points in the x and y directions. X is a real array of length NX, which will contain the mesh points x_j in x direction. Y is a real array of length NY, which will contain the mesh points y_l in y direction. The arrays X and Y need not be initialised before calling the subroutine, since the subroutine assumes a uniform spacing and initialises the arrays before proceeding with the calculations. U is a real array of length IU $\times$ NY, which contains the solution u. This array can also be used to pass the initial values of the solution (if IFLG $\neq$ 0). U(I, J) is the solution $u(x_I, y_J)$. IU ($\geq$ NX) is the first dimension of U as declared in the calling program. COF is the name of the subroutine to calculate the coefficients of the equation. SUBROUTINE COF(X, Y, T, AXX, AYY, AX, AY, AU, A0) should calculate the coefficients AXX, AYY, AX, AY, AU and A0 at a given value of X, Y and T. BC is the name of the function routine to calculate the boundary values. FUNCTION BC(IB, X, Y, T) should calculate the boundary values for the solution. Here IB is an integer variable, which takes the value 1, 2, 3, 4 corresponding to the four boundary lines. The boundary values are given by

$$\begin{aligned}
u(\mathrm{X0},y,t)&=\mathrm{BC}(1,\mathrm{X0},y,t), &\qquad u(\mathrm{XN},y,t)&=\mathrm{BC}(2,\mathrm{XN},y,t),\\
u(x,\mathrm{Y0},t)&=\mathrm{BC}(3,x,\mathrm{Y0},t), &\qquad u(x,\mathrm{YN},t)&=\mathrm{BC}(4,x,\mathrm{YN},t).
\end{aligned}\tag{B.108}$$

If the boundary values depend on time, then the accuracy of difference approximation may be only $O(\Delta t)$. FIC is the name of the function routine, which may be used to calculate the initial values for the solution, when IFLG = 0. FUNCTION FIC(X, Y, T) must be supplied by the user, if IFLG = 0. This routine calculates the solution $u(x, y, t)$ at the initial time T. For other values of IFLG a dummy function routine FIC may be required to satisfy the Fortran compiler, or the linker. IER is the error parameter. IER = 714 implies that DT = 0, or XN = X0, or YN = Y0, or NX $\leq$ 2, or NY $\leq$ 2, or IU < NX, in which case, no calculations are performed. IER = 762 implies that the difference equations are singular and the solution cannot be calculated. This problem may arise due to some error in specifying the equation. IFLG is an integer variable used as a flag. If IFLG = 0, the initial values are calculated using the function routine FIC supplied by the user. For other values of IFLG, the initial values must be supplied in the array U at the time of calling. In any case, the subroutine sets IFLG to 1, so that during subsequent calls to the subroutine, the initial values are taken from the array U. This subroutine can be used to calculate the solution at several values of T, by calling it repeatedly after changing DT and NT appropriately. Other variables should not be changed. WK is a real array of length max(NX, NY) $\times$ (4 + NY), used as a scratch space by the subroutine.

215. LAX Subroutine to solve a system of hyperbolic equations in the conservation law form, using Lax-Wendroff difference scheme. The equations are assumed to be in the form $\partial\mathbf{u}/\partial t + \partial\mathbf{f}/\partial x = 0$, where $\mathbf{f}(x, t, \mathbf{u})$ can be a nonlinear function of $\mathbf{u}$. N is the number of equations in the system, T is the initial value of time. After execution, T will be replaced by the current value of t. DT is the time step to be used. The time step is kept fixed. (X0, XN) is the interval along x-axis, on which the solution is required. NT is the number of time steps over which integration is to be performed, NX is the number of mesh points to be used along the x-axis. X is a real array of length NX, used to store the coordinates of the mesh points x_j. This array need not be initialised before calling the subroutine, since the subroutine assumes a uniform spacing and calculates x_j before performing the calculations. U is a real array of length IU $\times$ NX, which contains the solution u. This array can also be used to pass the initial values of the solution (if IFLG $\neq$ 0). IU(I, J) contains the Ith component of the solution u_i at Jth mesh point x_j. IU ($\geq$ N) is the first dimension of U, as declared in the calling program. FLUX is the name of the subroutine, which is used to calculate the flux $\mathbf{f}$ occurring in the differential equations. SUBROUTINE FLUX(N, X, T, U, F) should calculate the flux F(I), (I = 1, ..., N) for a given value of X, T and U(J), (J = 1, ..., N). BC is the name of the function routine to calculate the boundary values. FUNCTION BC(IB, N, X, T, IW, A) should calculate the coefficients required for the boundary conditions. Here IB is an integer variable which takes the value 1 or 2 corresponding to X = X0 and XN, respectively. IW is an integer array of length N, which contains information about the type of boundary condition on each component of $\mathbf{u}$ or $\mathbf{f}$. If IW(I) = 0, then the boundary condition is of the simple Dirichlet form $a_{i1}(t)u_i = a_{i3}(t)$, otherwise the boundary condition is applied to the corresponding component of the flux

$$a_{i1}(t)f_i + a_{i2}(t)\frac{\partial f_i}{\partial x} = a_{i3}(t). \tag{B.109}$$

The coefficients a_{ij} are returned in the array A, which should be dimensioned as A(N, 3) in the subroutine BC. FIC is the name of the subroutine, which may be used to calculate the initial values for the solution if IFLG = 0. SUBROUTINE FIC(N, X, T, U) must be supplied by the user, when IFLG = 0. This subroutine calculates the solution $u_i(\mathrm{X},\mathrm{T})$, $(i = 1,\ldots,\mathrm{N})$ at the initial time T. For other values of IFLG a dummy subroutine FIC may be required to satisfy the Fortran compiler or the linker. (In subroutine FIC, U is a real array of length N, which returns the value of the solution u_i for $i = 1,\ldots,\mathrm{N}$.) IER is the error parameter. IER = 715 implies that DT = 0, or XN = X0, or NX $\leq$ 2, or IU $<$ N, in which case, no calculations are performed. IER = 763 implies that the denominator vanishes while calculating the boundary values and hence the solution cannot be continued. This problem may arise due to some error in specifying the boundary conditions. IFLG is an integer variable used as a flag. If IFLG = 0, the initial values are calculated using the subroutine FIC supplied by the user. For other values of IFLG, the initial values must be supplied in the array U. In any case, the subroutine sets IFLG to 1, so that during subsequent calls to the subroutine, the initial values are taken from the array U. This subroutine can be used to calculate the solution at several values of T by calling it repeatedly, after changing DT and NT appropriately. Other variables should not be changed. WK is a real array of length N $\times$ (5 + NX), used as a scratch space by the subroutine. IWK is an integer array of length N, used as a scratch space by the subroutine.

216. SOR Subroutine to solve a linear second-order elliptic equation

$$\begin{aligned} A_{xx}(x,y)\frac{\partial^2 u}{\partial x^2} + A_{xy}(x,y)\frac{\partial^2 u}{\partial x\partial y} + A_{yy}(x,y)\frac{\partial^2 u}{\partial y^2} + A_x(x,y)\frac{\partial u}{\partial x} \\ + A_y(x,y)\frac{\partial u}{\partial y} + A_0(x,y)u + F(x,y) = 0, \end{aligned} \tag{B.110}$$

with Dirichlet boundary conditions on a rectangular region. Although arbitrary coefficients will be accepted by this subroutine, the result may be meaningful only when the resulting equation is elliptic, i.e. $A_{xx}A_{yy} - 4A_{xy}^2 > 0$. This subroutine uses the SOR method to solve the difference equations

$$\begin{aligned} A_{xx}\frac{2u_{jl} - u_{j-1,l} - u_{j+1,l}}{(\Delta x)^2} + A_{xy}\frac{u_{j+1,l-1} + u_{j-1,l+1} - u_{j+1,l+1} - u_{j-1,l-1}}{4\Delta x\Delta y} \\ + A_{yy}\frac{2u_{jl} - u_{j,l-1} - u_{j,l+1}}{(\Delta y)^2} + A_x\frac{u_{j-1,l} - u_{j+1,l}}{2\Delta x} + A_y\frac{u_{j,l-1} - u_{j,l+1}}{2\Delta y} \\ - A_0 u_{jl} = F, \end{aligned} \tag{B.111}$$

where all coefficients are evaluated at (x_j, y_l). The points $(\mathrm{X0},\mathrm{Y0})$ and $(\mathrm{XN},\mathrm{YN})$ define the rectangle over which the solution is required. NX and NY are the number of mesh points along the x and y directions. X and Y are real arrays of lengths NX and NY respectively, which contain the mesh points x_j and y_l. These arrays need not be initialised before calling the subroutine, since the subroutine assumes a uniform spacing and initialises these arrays before proceeding with the calculations. U is a real array of length IU $\times$ NY, which contains the solution. U(I, J) will contain the computed approximation to $u(x_i, y_j)$. Before calling the subroutine, an initial approximation to the solution should be supplied in this array. IU ($\geq$ NX) is the value of the first dimension of array U, as declared in the calling program. COF is the name of the subroutine used to calculate the coefficients in the equation. SUBROUTINE COF(X, Y, AXX, AXY, AYY, AX, AY, A0, F) should calculate the coefficients AXX, AXY, AYY, AX, AY, A0 and F at a given value of X and Y. BC is the name of the function routine to calculate the boundary values. FUNCTION BC(IB, X, Y) should calculate the boundary values for the solution. Here IB is an integer variable which takes the value 1, 2, 3, 4, corresponding to the four boundary lines. The boundary values are given by

$$\begin{aligned} u(\mathrm{X0}, y) &= \mathrm{BC}(1, \mathrm{X0}, y), \qquad & u(\mathrm{XN}, y) &= \mathrm{BC}(2, \mathrm{XN}, y), \\ u(x, \mathrm{Y0}) &= \mathrm{BC}(3, x, \mathrm{Y0}), \qquad & u(x, \mathrm{YN}) &= \mathrm{BC}(4, x, \mathrm{YN}), \end{aligned} \tag{B.112}$$

OMEGA is the value of the relaxation parameter ω to be used in the calculation. For meaningful results $1 < \omega < 2$, but this condition is not checked by the subroutine. If OMEGA ≤ 0, then the subroutine uses the optimum value for the Poisson's equation, based on the number of mesh points in each direction (Section 14.8). IER is the error parameter. IER = 716 implies that XN = X0, or YN = Y0, or NX ≤ 2, or NY ≤ 2, or IU < NX, in which case, no calculations are performed. IER = 764 implies that the diagonal term in the difference equation vanishes and the calculations are abandoned. This failure may arise due to some error in specifying the equation, or the equation may be such that the diagonal term actually vanishes at some mesh point. In the latter case, changing mesh spacing along some axis may make the coefficient nonzero. IER = 765 implies that the SOR iteration failed to converge in MAXIT (= 1000) iterations. AEPS is the convergence parameter, which specifies the accuracy to which the difference equations are to be solved. The iteration is terminated when the maximum change in all elements is less than AEPS. It should be noted that, this parameter does not affect the truncation error, which depends only on the number of mesh points. For optimum results, AEPS should be somewhat smaller than the expected accuracy in the solution. Accuracy of the solution can be checked by repeating the calculations with different number of mesh points. NIT is an output parameter, which gives the number of SOR iterations required by the subroutine to achieve the specified accuracy. WK is a real array of length $9 \times \mathrm{NX} \times \mathrm{NY}$, used as a scratch space by the subroutine.

217. ADI Subroutine to solve a linear second-order elliptic equation

$$\begin{aligned} A_{xx}(x,y)\frac{\partial^2 u}{\partial x^2} + A_{yy}(x,y)\frac{\partial^2 u}{\partial y^2} + A_x(x,y)\frac{\partial u}{\partial x} \\ + A_y(x,y)\frac{\partial u}{\partial y} + A_0(x,y)u + F(x,y) = 0, \end{aligned} \tag{B.113}$$

on a rectangular region using the Alternating Direction Implicit iterative (ADI) method. Although, arbitrary coefficients will be accepted by this subroutine, the result may be meaningful only when the resulting equation is elliptic, i.e. $A_{xx}A_{yy} > 0$. This subroutine accepts more general boundary conditions of the form $A_0u + A_n\partial_n u = F$, where $\partial_n u$ is the normal derivative. The points $(\mathrm{X0}, \mathrm{Y0})$ and $(\mathrm{XN}, \mathrm{YN})$ define the rectangle, over which the solution is required. KN is the parameter k in the ADI algorithm. This subroutine repeats a cycle of 2^k iterations, until the result converges. The convergence criterion is tested after each iteration. Hence, the actual number of iterations may not be a multiple of 2^k. Ideally k should be chosen such that convergence takes place in approximately 2^k iterations, but the exact choice may not be very crucial. NX and NY are the number of mesh points along the x and y directions. X and Y are real arrays of lengths NX and NY, respectively containing the mesh points x_j and y_l. These arrays need not be initialised before calling the subroutine, since the subroutine assumes a uniform spacing and initialises these arrays before proceeding with the calculations. U is a real array of length IU $\times$ NY, which contains the solution. U(I, J) will contain the computed approximation to $u(x_i, y_j)$. Before calling the subroutine an initial approximation to the solution should be supplied in this array. IU ($\geq$ NX) is the value of the first dimension of the array U, as declared in the calling program.

COF is the name of the subroutine used to calculate the coefficients in the equation. SUBROUTINE COF(X, Y, AXX, AYY, AX, AY, A0, F) should calculate the coefficients AXX, AYY, AX, AY, A0 and F at a given value of X and Y. BC is the name of the subroutine to calculate the coefficients for the boundary conditions. SUBROUTINE BC(IB, X, Y, A0, AN, F) should calculate the coefficients A0, AN and F for the boundary conditions. Here IB is an integer variable which takes the value 1, 2, 3, 4, corresponding to the four boundary lines $x = \mathrm{X0}$, $x = \mathrm{XN}$, $y = \mathrm{Y0}$ and $y = \mathrm{YN}$, respectively. EL and EU are respectively, the lower and upper limit on the eigenvalues of the partitions X and Y of the finite difference matrix. If EL ≤ 0, then the value for Poisson's equation will be used. Similarly, if EU $\leq$ EL, then the value for Poisson's equation will be used. IER is the error parameter. IER = 717 implies that XN = X0, or YN = Y0, or NX ≤ 2, or NY ≤ 2, or IU $<$ NX, in which case, no calculations are performed. IER = 766 implies that the matrix for ADI iteration is singular and the calculations are abandoned. This problem may arise due to some error in specifying the equation or the boundary conditions. IER = 767 implies that the ADI iteration failed to converge in MAXIT (= 1000) iterations. AEPS is the convergence parameter, which specifies the accuracy to which the difference equations are to be solved. The iteration is terminated when the maximum change in all elements is less than AEPS. It should be noted that, this parameter does not affect the truncation error, which depends only on the number of mesh points. For optimum results, AEPS should be somewhat smaller than the expected accuracy in the solution. Accuracy of the solution can be checked by repeating the calculations with different number of mesh points. NIT is an output parameter, which gives the number of ADI iterations required by the subroutine to achieve the specified accuracy. WK is a real array of length $\max(\mathrm{NX}, \mathrm{NY}) \times (2 + \mathrm{NY})$, used as a scratch space by the subroutine.

WKA is a real array of length $7 \times \mathrm{NX} \times \mathrm{NY}$, used as a scratch space by the subroutine.

Bibliography

Brent, R. P. (1973): *Algorithms for Minimization Without Derivatives,* Prentice-Hall, Englewood Cliffs, New Jersey.

DiDonato R., Morris, A. H. Jr. (1986) *Computation of the Incomplete Gamma Function Ratios and their Inverse*, ACM Transactions on Mathematical Software (TOMS), **12**, p. 377.

DiDonato R., Morris, A. H. Jr. (1992) *Algorithm 708: Significant Digit Computation of the Incomplete Beta Function Ratios,* ACM Transactions on Mathematical Software (TOMS), **18**, p. 360.

Metcalf, M. and Reid, J. K. (1999): *Fortran 90/95 Explained,* (2nd ed.), Oxford University Press, Oxford

Perry Cole, J. W. (1987): *ANSI Fortran 77, A Structured Problem-Solving Approach,* S. Chand, New Delhi.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (2007): *Numerical Recipes : The Art of Scientific Computing,* (3rd ed.) Cambridge University Press, New York.

Wilkinson, J. H. and Reinsch, C. (1971): *Linear Algebra: Handbook for Automatic Computation,* Vol. 2, Springer-Verlag, Berlin.