

# Contents

<b>Appendix C</b>	<b>C Programs</b>	<b>1007</b>
C.1	Introduction . . . . .	1007
C.2	Roundoff Error . . . . .	1011
1	CASSUM . . . . .	1011
2	ROUND . . . . .	1011
3	CDIV . . . . .	1012
4	CABS . . . . .	1012
5	CSQRT . . . . .	1012
C.3	Linear Algebraic Equations . . . . .	1012
6	GAUELM . . . . .	1012
7	MATINV . . . . .	1013
8	CROUT . . . . .	1013
9	CROUTH . . . . .	1014
10	CHOLSK . . . . .	1016
11	GAUBND . . . . .	1017
12	SVD . . . . .	1018
13	SVDEVL . . . . .	1018
C.4	Interpolation . . . . .	1019
14	DIVDIF . . . . .	1019
15	DIVDIF0 . . . . .	1020
16	NEARST . . . . .	1020
17	SPLINE . . . . .	1021
18	SPLEVL . . . . .	1021
19	SMOOTH . . . . .	1021
20	BSPLIN . . . . .	1022
21	BSPINT . . . . .	1022
22	BSPEVL . . . . .	1023
23	RATNAL . . . . .	1024
24	POLY2 . . . . .	1024
25	LINRN . . . . .	1025
26	LOCATE . . . . .	1025
27	BSPINT2 . . . . .	1025
28	BSPEV2 . . . . .	1026
29	BSPINTN . . . . .	1027

30	BSPEVN . . . . .	1028
31	BSPEVN1 . . . . .	1029
32	BSPEVN2 . . . . .	1029
C.5	Differentiation . . . . .	1029
33	DRVT . . . . .	1029
C.6	Integration . . . . .	1030
34	SIMSON . . . . .	1030
35	SPLINT . . . . .	1030
36	BSPQD . . . . .	1031
37	ROMBRG . . . . .	1031
38	EPSILN . . . . .	1032
39	GAUSS . . . . .	1032
40	GAUCBY . . . . .	1033
41	GAUCB1 . . . . .	1033
42	GAUCB2 . . . . .	1034
43	GAUSQ2 . . . . .	1034
44	GAUSQ . . . . .	1035
45	GAULAG . . . . .	1036
46	LAGURE . . . . .	1037
47	HERMIT . . . . .	1037
48	GAULG2 . . . . .	1038
49	GAULOG . . . . .	1039
50	GAUSRC . . . . .	1039
51	GAULEG . . . . .	1040
52	GAUJAC . . . . .	1040
53	LAGURW . . . . .	1041
54	GAUHER . . . . .	1041
55	GAUSWT . . . . .	1041
56	FILON . . . . .	1042
57	ADPINT . . . . .	1043
58	KRONRD . . . . .	1044
59	GAUS16 . . . . .	1044
60	CAUCHY . . . . .	1044
61	EULER . . . . .	1045
62	BSPQD2 . . . . .	1046
63	BSPQDN . . . . .	1047
64	MULINT . . . . .	1047
65	NGAUSS . . . . .	1048
66	SPHND . . . . .	1049
67	STRINT . . . . .	1050
68	STROUD . . . . .	1051
69	MCARLO . . . . .	1051
70	RAN1 . . . . .	1052
71	RANF . . . . .	1052
72	EQUIDS . . . . .	1052

C.7	Nonlinear Algebraic Equations . . . . .	1053
73	BISECT . . . . .	1053
74	SECANT . . . . .	1053
75	SECANI . . . . .	1054
76	NEWRAP . . . . .	1055
77	BRENT . . . . .	1056
78	SEARCH . . . . .	1056
79	ZROOT . . . . .	1057
80	ZROOT2 . . . . .	1057
81	MULLER . . . . .	1058
82	MULER2 . . . . .	1059
83	POLYR . . . . .	1060
84	LAGITR . . . . .	1061
85	DAVIDN . . . . .	1061
86	NEWTON . . . . .	1063
87	BROYDN . . . . .	1063
C.8	Optimisation . . . . .	1064
88	BRACKM . . . . .	1064
89	GOLDEN . . . . .	1064
90	BRENTM . . . . .	1064
91	DAVIDM . . . . .	1065
92	BFGS . . . . .	1066
93	LINMIN . . . . .	1066
94	FLNM . . . . .	1067
95	NMINF . . . . .	1068
96	LINMNF . . . . .	1068
97	FLN . . . . .	1069
98	SIMPLX . . . . .	1069
99	SIMPX . . . . .	1070
C.9	Statistical Inferences . . . . .	1071
100	SHSORT . . . . .	1071
101	GAMMAP . . . . .	1071
102	BETAP . . . . .	1071
103	BETSER . . . . .	1072
104	BETCON1 . . . . .	1072
105	BETCON . . . . .	1073
106	BETAI . . . . .	1073
107	FBETA . . . . .	1073
108	RANGAU . . . . .	1073
109	IRANBIN . . . . .	1073
110	IRANPOI . . . . .	1074
111	PCOR . . . . .	1074
C.10	Functional Approximations . . . . .	1074
112	POLFIT . . . . .	1074
113	POLEVL . . . . .	1075

114	POLFIT1 . . . . .	1075
115	POLORT . . . . .	1076
116	POLFIT2 . . . . .	1076
117	POLEV2 . . . . .	1077
118	POLFITN . . . . .	1077
119	POLEVN . . . . .	1078
120	POLEVN1 . . . . .	1078
121	POLEVN2 . . . . .	1079
122	LLSQ . . . . .	1079
123	BSPFIT . . . . .	1080
124	BSPFIT2 . . . . .	1082
125	BSPFITW2 . . . . .	1083
126	BSPFITN . . . . .	1084
127	BSPFITWN . . . . .	1086
128	LINFITXY . . . . .	1087
129	NLLSQ . . . . .	1087
130	DFT . . . . .	1088
131	FFT . . . . .	1088
132	FFTR . . . . .	1089
133	FFTN . . . . .	1089
134	LAPINV . . . . .	1090
135	POLD . . . . .	1090
136	RMK . . . . .	1090
137	RMK1 . . . . .	1091
138	RMKD . . . . .	1091
139	RMKD1 . . . . .	1091
140	PADE . . . . .	1091
141	CHEBCF . . . . .	1092
142	CHEBEX . . . . .	1092
143	CHEBAP . . . . .	1092
144	REMES . . . . .	1093
145	FM . . . . .	1095
146	GAMMA . . . . .	1095
147	GAMMAL . . . . .	1095
148	ERF . . . . .	1095
149	ERFC . . . . .	1096
150	BJ0 . . . . .	1096
151	BJ1 . . . . .	1096
152	BJN . . . . .	1097
153	BY0 . . . . .	1098
154	BJY0 . . . . .	1098
155	BY1 . . . . .	1098
156	BJY1 . . . . .	1099
157	BYN . . . . .	1099
158	SPHBJN . . . . .	1100

159	BI0	1101
160	BI1	1101
161	BIN	1101
162	BK0	1102
163	BK1	1103
164	BKN	1103
165	DAWSON	1103
166	FERMM05	1103
167	FERM05	1104
168	FERM15	1104
169	FERM25	1104
170	PLEG	1104
171	PLM	1104
172	YLM	1104
173	MINMAX	1105
174	POLYL1	1105
175	LINL1	1106
176	SIMPL1	1106
C.11	Algebraic Eigenvalue Problem	1107
177	INVIT	1107
178	TRED2	1108
179	TRBAK	1108
180	TQL2	1109
181	TRIDIA	1109
182	STURM	1110
183	TINVIT	1110
184	HEREVP	1111
185	BALANC	1111
186	BALBAK	1112
187	ELMHES	1112
188	HQR	1113
C.12	Ordinary Differential Equations	1113
189	RKM	1113
190	RK4	1114
191	RK2	1114
192	MSTEP	1114
193	ADAMS	1116
194	STRT4	1117
195	GEAR	1117
196	EXTP	1118
197	FDM	1119
198	GEVP	1121
199	GAUBLK	1125
200	SETMAT	1126
201	BSPODE	1126

C.13	Integral Equations . . . . .	1128
202	FRED . . . . .	1128
203	FREDCO . . . . .	1130
204	FUNK . . . . .	1130
205	RLS . . . . .	1131
206	FORW . . . . .	1134
207	VOLT . . . . .	1135
208	VOLT2 . . . . .	1135
C.14	Partial Differential Equations . . . . .	1136
209	CRANK . . . . .	1136
210	LINES . . . . .	1137
211	ADM . . . . .	1138
212	LAX . . . . .	1140
213	SOR . . . . .	1141
214	ADI . . . . .	1142
	Bibliography . . . . .	1144

# Appendix C

## C Programs

### C.1 Introduction

All functions in this appendix are written in ANSI C, and have been checked to some extent, but the author cannot guarantee their correctness. **Readers are welcome to use these functions at their own risk.** These functions are straightforward translation of corresponding Fortran subprograms in Appendix B. It should be noted that, these functions 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. Readers are expected to use these functions with ‘reasonable’ inputs. Passing on arbitrary or invalid input parameters may give unpredictable results with or without any warning. These functions can be effectively used to understand the working as well as the limitations of various numerical algorithms, on different problems. Apart from 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 functions in this Appendix are all written in ANSI C, but may require some changes before running them on a new system. We leave it to the readers to figure out the changes needed to run the programs on their machines. Since these routines are simple translation of Fortran programs, they preserve the same variable names and arguments in most cases. Nevertheless, since there are some basic differences between C and Fortran this appendix provides an independent summary of usage for these functions. In this section we outline the major differences between the C and Fortran programs.

While in Fortran variable names are not case sensitive, C variable names are case sensitive. Most of the variables in the C functions in Appendix C use only lower case characters. However, for constants and global variables, which may be accessible to more than one functions, we have used upper case letters to avoid any possible conflict with local variables in functions. Nevertheless, in most of comments in programs and in this Appendix, the variables appear in upper case. There are two main reasons for this, first use of upper case clearly identifies that the string of characters is a variable name and helps in understanding and, second it makes it easier to translate the document from Fortran to C. We hope that this will not confuse the readers.

In C all subprogram units are treated as functions which can return any required value. Hence, we have used this facility to return the error parameter (IER in Fortran subprograms) as far as possible. Only in those functions which are supposed to return some other value the error parameter occurs in argument list. But in almost all cases the significance and values returned by C functions are same as those for IER in the corresponding Fortran subprograms. The argument IER appears explicitly only in functions BSPEVL, BSPEV2, BSPEVN, BSPEVN1, BSPEVN2, BSPQD, BSPQD2, BSPQDN, DRVT, MULER2, SECANI, SPLEVL. The argument IER or the returned value of the function is used as an error flag. After execution of the functions this value should be zero if no error has occurred. Nonzero values indicate some error condition. Values of IER (or the function) 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 functions the returned value is set to less than 100 even when it fails to converge to satisfactory accuracy. This is mainly because in most cases these functions 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 error flag does not necessarily imply that the result is correct as all possible errors are not detected by any of these functions. Thus additional checks are required in all cases. But in general one can expect that the result is likely to be correct if error flag is zero. A value greater than 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 functions (e.g., BISECT, NEWRAP, FRED, SECANI) return a negative value of error flag under some special circumstances, even though the execution is successful. It may be noted that some functions call other functions. Hence, if error flag is nonzero the error may have occurred in the secondary functions. In such cases, only the final value of error flag 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 functions has been modified since the first edition was written, to make them more or less unique between different functions. In the present version, when  $IER > 0$ , in most cases the first digit is determined by the topic covered by the function. Thus the values 1x or 1xx are for linear algebra functions (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).

Apart from this C allocates memory dynamically and hence the variables are not generally preserved between two successive calls to the same function. In those functions, where the variables need to be preserved the required variables are declared to be static. Since C also allows additional memory to be allocated during execution, there is no need to pass on scratch arrays as required by Fortran 77. Thus the arguments corresponding to scratch arrays may be missing in C version of the Fortran subprograms. Similarly, in some cases where the Fortran subprogram uses an array of fixed size, which limits the parameter range where the solution can be calculated. This restriction is not necessary in C functions as the array of required size can be allocated at run time.

The main source of confusion between Fortran and C subprograms arises from the treatment of arrays. While Fortran by default numbers the array elements of array A(N) from 1 to N, in C functions the array index varies from 0 to  $N - 1$ . Besides C does not have much facility to treat multi-dimensional arrays and these are essentially treated as array of array. But the main difference arises in order in which elements are stored in the computer memory. Thus an array of array A[3][3] is stored in the order A[0][0], A[0][1], A[0][2], A[1][0], A[1][1], A[1][2], A[2][0], A[2][1], A[2][2]. Which is different from what Fortran will use to store a two dimensional array. Basically, in Fortran the first dimension varies most rapidly, while in C the last dimension varies most rapidly in memory locations. To compound the problems, C does not allow the dimensions to be passed on as argument to the function since the dimensions have to be constant. On the other hand, Fortran allows variable dimensions in subprograms, provided both the array name as well as the variable defining the dimension also occur in the argument list. For one-dimensional arrays the dimension information is not important as it is not required to calculate the memory location of the required array element. But for two-dimensional arrays it is not possible to calculate the memory location unless the second dimension is known. In Fortran we need to know the first dimension. Unfortunately, there is no straightforward way to pass on this second dimension to a C function. Thus in all functions in this appendix we treat a multi-dimensional array as one dimensional array. For example, in the array A[M][N] the element A[i][j] would be in memory location  $j + iN$  with respect to the first element. We can refer to this element as A[j + iN] if A is simply declared as a one-dimensional array or just the address of its first element is passed as the argument to function.

We have followed this practice in all functions, where the second dimension  $N$  can be variable. To calculate this position we need the actual value of the second dimension  $N$ . While in Fortran we will need the first dimension of array  $A(M, N)$  for the same purpose. As a result, the argument to be passed on to Fortran program will not be same as that for the corresponding C function. To circumvent this problem in most cases we have reversed the order of indices in the arrays, thus an array element  $A[I][J][K]$  in C function will correspond to  $A(K, J, I)$  in Fortran subprogram. The main reason for this choice is that in many functions it is convenient to assume that elements are stored in the order determined by Fortran subprogram, because the order of indices was chosen to satisfy this requirement. If a different order is used then we may need frequent copying of elements in another temporary array. Another advantage of this convention is that the same number has to be passed on to the Fortran and C programs as array dimension information.

Only exception we have made is for the situation where a two dimensional array represents a matrix stored in the normal format for a linear algebra problem. In that case  $A[i][j]$  still represents  $j$ th column of  $i$ th row, which in Fortran is referred to as  $A(i, j)$ . For all other arrays the indices are interchanged to follow the order in which memory is allocated. The main reason for this exception is that the matrix elements are conventionally written in this order. Of course, any two-dimensional array can be treated as a matrix, but we have broken our normal convention of interchanging indices only when the two dimensional array is supposed to represent a matrix arising in linear algebra problem like solving a system of linear equations or the eigenvalue problem. In the documentation for each function, the order of indices is spelt out, so hopefully, there would be no confusion on this account.

Another major shortcoming of C is lack of variables of type complex. Hence all complex variables need to be treated as arrays of length 2 or as structures and more importantly the arithmetic operations between these variables need to be defined separately. Most functions in this Appendix treat complex variables as arrays of length 2, while in some cases a complex array is treated as two separate arrays containing the real and imaginary parts. The addition, subtraction and multiplication of complex variables is explicitly coded in the functions as corresponding operation on real and imaginary parts. But other operations like division, taking absolute value or the square root require more care to avoid possible problems with roundoff errors or overflows and underflows. For these operations separate functions CDIV, CABS and CSQRT are provided. A few of the Fortran subprograms requiring complex arithmetic, namely, CONTUR, DELVES, LAGITC, NEWRAC, POLYC, SECANC have not been translated to C. Apart from CONTUR and DELVES all these are complex versions of corresponding routines for real functions. Apart from these, complex versions of other routines like those for solution of a system of linear equations are also not translated. The main reason for this is that C is probably not designed for complex arithmetic and these routines are not likely to be required by C programmers. In principle, it is possible to call Fortran subprograms directly from

a C function and the Fortran version for any of these routines can be used from a C function. For calling a Fortran subprogram from C function all arguments must be passed through address. Further, in general the Fortran compiler adds some prefix or suffix of underline character, so the name has to be modified accordingly.

All C functions in this appendix assume that variables of type int are 32 bit integers and all real variables are assumed to be of type double which is expected to be 64 bits. In most functions 16 bit integers may be enough, but there are a few functions, where this will not be sufficient. Similarly, 32 bit real variables (type float) may be sufficient in many cases but for uniformity all functions use real variables of type double. On most modern computers the execution time is not significantly different between 32 bit and 64 bit real variables, but the memory requirements will be doubled when 64 bits are used instead of 32 bits.

Many of the functions have a parameter REPS and/or AEPS, which specify the required accuracy. Depending on the function 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 function included in the online material and its usage. A brief description of each variable in the call statement also appears in the program files. Some programs giving actual examples of usage are also included in the online material. Functions with an underscore in the names are simple variants of other functions with the suffix dropped and are not described separately in this Appendix. For example CASSUM\_A is a variant of CASSUM. These functions are also included in the list which can be found in the online material.

## C.2 Roundoff Error

**1. CASSUM** Function 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 function to sum the terms in an array TERM. In that case TERM should be an array containing the terms to be summed. Function CASSUM\_A implements this variation.

**2. ROUND** Function to round a real number X to N digits using base B. It is assumed that  $hB^N < 1$ , since otherwise, the accuracy of computer arithmetic is not sufficient to give N digits. The returned value of the function should be the rounded value of X. This function can be used to simulate calculations

using any required number of digits by rounding intermediate result after every arithmetic operation. Since it is fairly expensive to use this function, it should be used for small problems only.

**3. CDIV** Function to perform division for complex numbers. C1, C2 and CR are all arrays of length 2 containing the real and imaginary parts of the complex numbers. C1 is the numerator, C2 the denominator and  $CR = C1/C2$  is the computed ratio. CR can be same as either C1 or C2, that is the result can be overwritten on one of the operands. To avoid overflow or underflow in some cases the calculations are rearranged as explained in Eq. (2.34).

**4. CABS** Function to calculate absolute value of a complex number. The calculations are rearranged to avoid overflow or underflow by writing

$$CABS(x + iy) = \sqrt{x^2 + y^2} = |w| \sqrt{\left(\frac{x}{w}\right)^2 + \left(\frac{y}{w}\right)^2}, \quad (C.1)$$

where  $w = \max(|x|, |y|)$ . CX is an array of length 2 containing the real and imaginary parts of the complex number and CABS is the calculated absolute magnitude.

**5. CSQRT** Function to calculate the square root of a complex number. The calculations are rearranged as explained above to avoid overflow or underflow and in addition the correct branch of the square root function has to be selected. CX and CR are arrays of length 2 containing the real and imaginary parts of the complex numbers. CX is the input complex number while CR is the computed value of  $\sqrt{CX}$ . CR can be same as CX, that is, the result can be overwritten on the input number.

## C.3 Linear Algebraic Equations

**6. GAUELM** Function 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 an 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 function. Hence, if it is required afterwards, a separate copy should be saved. X is an 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 rows of X.  $X[J][I]$  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 function 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 function is executed. LJ is the actual

value of the second dimension of arrays A and X, as declared in the calling function ( $LJ \geq N$ ). 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 function 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. Error status can be obtained from the returned value of GAUELM. A value of 101 indicates that  $N \leq 0$ , or  $N > LJ$ , in which case, no calculations will be performed. A value of 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 second dimension of the matrix is not specified correctly. Note that order of subscripts in X is reverse of that in the Fortran version, since X is not considered as a matrix. This is to ensure that same value of LJ can serve as second dimension for both arrays A and X. To calculate the inverse of a matrix, the function can be called with  $NUM = N$  and array X set to a unit matrix of order N. In this case, the transposed inverse will be returned in the array X. Thus after calculation we will need to transpose the array X to get the inverse matrix. This is required because of the order in which matrices are assumed to be stored in memory. This function 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 function may not give any indication of the problem. SVD should be used to check for ill-conditioning.

**7. MATINV** Function to calculate the inverse of a square matrix using Gaussian elimination, with partial pivoting. N is the order of matrix, IA is the second dimension of arrays A and AI as specified in the calling function. A is an array of length  $IA \times N$  containing the matrix, which must be supplied at the time of calling. AI is an array of length  $IA \times N$  which will contain the inverse of A as calculated by the function. Error status can be obtained from the returned value of MATINV. Nonzero values may be set by function GAUELM, which is used to calculate the inverse. Instead of GAUELM it is possible to use CROUT for calculating the triangular decomposition.

**8. CROUT** Function 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 an array of length  $LJ \times N$ , containing the matrix of coefficients.  $A[i][j]$  is the coefficient of  $x_j$  in  $i$ th equation. This matrix will be overwritten by the function. 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 an 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 rows of X.  $X[J][I]$  should contain the  $I$ th component for the  $J$ th 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 function is executed. LJ is the actual value of the second dimension of arrays A and X, as declared in the calling function ( $LJ \geq N$ ). 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 function 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. Error status can be obtained from the returned value of CROUT. A value of 102 indicates that  $N \leq 0$ , or  $N > LJ$ , in which case, no calculations will be performed. A value of 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. To calculate the inverse of a matrix, the function can be called with  $NUM = N$  and array X set to a unit matrix of order N. In this case, the transposed inverse will be returned in the array X. Thus after calculation we will need to transpose the array X to get the inverse matrix. This is required because of the order in which matrices are assumed to be stored in memory. This function 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 function may not give any indication of the problem. SVD should be used to check for ill-conditioning.

**9. CROUTH** Function 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 an array of length  $LJ \times N$ , containing the matrix of coefficients.  $A[I][J]$  is the coefficient of  $x_J$  in  $I$ th equation. The matrix will be preserved by the function. B is an array of the same dimensions as A. After execution, B will contain the triangular decomposition of the original matrix. X is an 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 rows of X.  $X[J][I]$  should contain the  $I$ th component for the  $J$ th 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 function is executed. LJ is the actual value of the second dimension of arrays A, B and X, as declared in the calling function ( $LJ \geq N$ ). 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  $J$ th right-hand side. However, for extremely ill-conditioned matrices, it may underestimate the error. 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 function 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 an array of length NUM. After execution, WK[I] will contain the estimated error for the  $I$ th right-hand side. Error status can be obtained from the returned value of CROUTH. A value of 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 long double arithmetic, D1 should be declared so. If the matrix is ill-conditioned, increasing NITR may cause the iteration to converge, but the result is unlikely to be more reliable. Returned value of 102 indicates that  $N \leq 0$  or  $N > LJ$ , in which case, no calculations will be performed. Returned value of 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. This function requires 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 function can be called with  $\text{NUM} = \text{N}$  and array *X* set to a unit matrix of order *N*. In this case, the transposed inverse will be returned in the array *X*. If the system of equations is extremely ill-conditioned, the results could be unreliable and the function may not give any indication of problem. However, in most cases,  $\text{WK}[\text{I}]$  will give a reasonable estimate of the actual error. This function 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.

**10. CHOLSK** Function 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 an array of length  $\text{ND} \times \text{N}$ , containing the matrix of coefficients.  $\text{A}[\text{I}][\text{J}]$  is the coefficient of  $x_J$  in *I*th equation. Only the lower triangular part of the matrix may be used by the function, though the memory is reserved for the full array. This matrix will be overwritten by the function. 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 an array of length  $\text{ND} \times \text{NUM}$ , containing the right-hand sides of the linear systems. The system will be solved for each right-hand side vector given by the rows of *X*.  $\text{X}[\text{J}][\text{I}]$  should contain the *I*th component for the *J*th 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 second dimension of arrays *A* and *X*, as declared in the calling function ( $\text{ND} \geq \text{N}$ ). *IFLG* is an integer parameter which determines the kind of calculations required. If  $\text{IFLG} \leq 1$ , 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  $\text{IFLG} \leq 0$ , both the decomposition as well as the solution for the required right-hand sides are obtained. If  $\text{IFLG} = 1$ , then only decomposition is performed and the value of determinant is calculated. If  $\text{IFLG} \geq 2$ , then only the solution for the required right-hand sides are obtained. It may be noted that, the function 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. Error status can be obtained



from the returned value of CHOLSK. A value of 103 indicates that  $N \leq 0$ , or  $N > ND$ , in which case, no calculations will be performed. CHOLSK = 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. To calculate the inverse of a matrix, the function can be called with NUM = N and array X set to a unit matrix of order N. In this case, the transposed inverse will be returned in the array X. If the system of equations is ill-conditioned, the results could be unreliable and the function may not give any indication of the problem.

**11. GAUBND** Function 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  $|i - j| > KB$ . NUM is the number of different right-hand side vectors, for which the equations are to be solved. A is an array of length  $LJ \times (3KB + 1)$  containing the matrix of coefficients which is stored in the band form.  $A[J - I + KB][I]$  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 function instead of GAUELM. The matrix A will be overwritten by the function. Hence, if it is required afterwards, a separate copy should be saved. X is an 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 rows of X.  $X[J][I]$  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  $DET \times 2^{IDET}$ . INC is an integer array of length N, which will contain the pivoting information after the function is executed. LJ is the actual value of the second dimension of arrays A and X, as declared in the calling function ( $LJ \geq N$ ). IFLG is an integer parameter which determines the kind of calculations required. If  $IFLG \leq 1$ , 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. For  $IFLG = -1$ , 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 function 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. Error

status can be obtained from the returned value of GAUBND. A value of 104 indicates that  $N \leq 0$ , or  $N > LJ$  or  $KB > N$ , in which case, no calculations will be performed. GAUBND = 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. To calculate the inverse of a matrix, the function can be called with NUM = N and array X set to a unit matrix of order N. In this case, the transposed 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 function 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 function may not give any indication of the problem.

**12. SVD** Function to perform the singular value decomposition of a  $M \times N$  matrix,  $A = U\Sigma V^T$  ( $M \geq N$ ). This function is based on the procedure *svd* in Wilkinson and Reinsch (1971). A is an array of length  $LA \times M$  containing the matrix. After execution, the matrix  $U$  of SVD will be overwritten on A. V is an array of length  $LV \times N$ , which will contain the matrix  $V$  (not  $V^T$ ) of SVD. SIGMA is an array of length N, which will contain the singular values of A, i.e., the diagonal elements of the diagonal matrix  $\Sigma$ . The singular values may not be arranged in a descending order. LA is the integer variable specifying the actual value of the second dimension of array A in the calling function ( $LA \geq N$ ). Similarly, LV is the actual value of the second dimension of array V in the calling function ( $LV \geq N$ ). The matrices A and V are stored in normal form with first index denoting the row. In principle, LA and LV could be equal and one of them would be enough, but for compatibility with Fortran version two different values are provided. Error status can be obtained from the returned value of SVD. A value of 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 EPS is too low for the arithmetic used. The specified value is suitable for 53-bit arithmetic. For use with lower precision arithmetic EPS may need to be increased. If a reasonable value of EPS 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. SVD = 105 denotes that  $N \leq 0$ , or  $N > LV$ , or  $N \leq 0$ , or  $M > LA$ , or  $N > M$ , in which case, no calculations are performed.

**13. SVDEVL** Function to evaluate the solution of a system of linear equations using the SVD ( $\mathbf{x} = V\Sigma^{-1}U^T\mathbf{b}$ ). This function 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 an array of size  $LU \times M$  containing the left-hand transformation matrix in SVD. V is an array of length  $LV \times N$  containing the right-hand transformation matrix in SVD. SIGMA is an 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 second dimension of arrays U and V respectively, in the calling function. B is an array of length M containing the right-hand side vector. After execution, B will contain the required solution. REPS specifies the required accuracy for zeroing the singular values. If  $\sigma_i < \text{REPS} \times \max(\sigma_j)$ , then the corresponding  $\sigma_i^{-1}$  is set to zero while calculating the solution. For a square matrix, where none of the singular values are reduced to zero, this function 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  $\sigma_i$  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 function gives the least squares solution (with minimum norm if some of the singular values are reduced to zero). Before using this function, the singular value decomposition should be computed using the function SVD. It should be noted that, the function SVD overwrites the matrix U on the original matrix A. The returned value SVDEVL is always zero.

## C.4 Interpolation

**14. DIVDIF** Function 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 function, then it will be reduced to the minimum of these three numbers and the error flag DIVDIF will be set to 22. If NUSE is less than 1, it will be set to  $\text{MIN}(6, \text{NTAB})$  and DIVDIF 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  $\text{NUSE} + 1$  which will contain the interpolated values.  $\text{FB}[I]$  gives the interpolated value using  $I$  points, and  $\text{FB}[\text{NUSE}]$  gives the final value. It may be noted that the first element  $\text{FB}[0]$  is not used to avoid confusion with the index. AEPS specifies the required accuracy. Interpolation is continued, until two successive values differ by less than AEPS. If  $\text{AEPS} \leq 0$ , then this convergence criterion will never be satisfied. DFB and DDFB are output parameters containing the first and second derivative at the same point XB. Error status can be obtained from the returned value of DIVDIF. A value of 21 implies that  $\text{NUSE} < 1$ , in which case it is set to  $\text{MIN}(6, \text{NTAB})$ .  $\text{DIVDIF} = 22$  implies that  $\text{NUSE} > \text{NTAB}$ , or  $\text{NUSE} > \text{NMAX}$ , in which case it is reduced appropriately.  $\text{DIVDIF} = 23$  implies that

the interpolation has not converged to the specified accuracy. 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 function DIVDIF0 may be used. This function requires the function NEARST.

**15. DIVDIF0** Function 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 function 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 function, then it will be reduced to the minimum of these three numbers and the error flag DIVDIF0 will be set to 22. If NUSE is less than 1, it will be set to  $\text{MIN}(6, \text{NTAB})$  and DIVDIF0 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  $\text{NUSE} + 1$  which will contain the interpolated values.  $\text{FB}[I]$  gives the interpolated value using  $I$  points, and  $\text{FB}(\text{NUSE})$  gives the final value. The first element  $\text{FB}[0]$  is not used to avoid confusion. AEPS specifies the required accuracy. Interpolation is continued, until two successive values differ by less than AEPS. If  $\text{AEPS} \leq 0$ , then this convergence criterion will never be satisfied. IFLG is an integer variable used as a flag to decide the first point to be used for interpolation. If  $\text{IFLG} = 0$  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 ( $0 \leq \text{IF1} \leq \text{NTAB} - 1$ ). IF1 specifies the first point to be used for interpolation when  $\text{IFLG} \neq 0$ . For  $\text{IFLG} = 0$  the function 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 is required at same XB with different F values. This situation arises if this function is used for interpolation in more than one variables. Error status can be obtained from the returned value of DIVDIF0. A value of 21 implies that  $\text{NUSE} < 1$ , in which case it is set to  $\text{MIN}(6, \text{NTAB})$ .  $\text{DIVDIF0} = 22$  implies that  $\text{NUSE} > \text{NTAB}$ , or  $\text{NUSE} > \text{NMAX}$ , in which case it is reduced appropriately.  $\text{DIVDIF0} = 23$  implies that the interpolation has not converged to the specified accuracy. This function requires the function NEARST.

**16. NEARST** Function to locate the nearest value in an ordered table using a bisection algorithm. X is an array containing the table with NTAB entries in either ascending or descending order. After execution, X[NEARST] will be the point nearest to XB.

**17. SPLINE** Function 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], ( $0 \leq I < N - 1$ ) the interpolant is given by  $F[I] + DX * (C[I][0] + DX * (C[I][1] + DX * C[I][2]))$ , where  $DX = XB - X[I]$ . Error status can be obtained from the returned value of SPLINE. A value of 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.

**18. SPLEVL** Function for evaluating the cubic spline interpolant at XB, using the coefficients of cubic spline which have been calculated by the function SPLINE. This function 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 an 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 function DIVDIF, though it also has limitations.  $IER = 201$  implies that  $N < 2$ . SPLEVL will return the interpolated value of the function at XB. Before using this function, the coefficients of cubic spline must be calculated using function SPLINE or any other equivalent function.

**19. SMOOTH** Function 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 an array of length  $3 \times NTAB$ , which will contain the coefficients of cubic spline. The array X must be in either ascending or descending order. This function 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 function does not actually plot the curve 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 function BSPFIT instead of SPLINE. Error status can be obtained from the returned value of SMOOTH. A value of 202 implies that  $NP < 2$ . Other values may be set by function SPLINE which is called to calculate the spline coefficients. This function requires functions SPLINE and SPLEVL.

**20. BSPLIN** Function to calculate the B-spline basis functions at a specified point. X is an 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  $NDERIV \leq 0$  only the B-splines will be calculated. For  $NDERIV = 1$ , the first derivative of B-splines will also be calculated. For  $NDERIV > 1$ , the second derivative will also be calculated. Higher derivatives are not calculated, but the function can be modified to include calculation of higher derivatives. B, DB and DDB are arrays of length  $NX + K - 2$  which will contain the values of B-splines, its first and second derivative respectively. All  $NX + K - 2$  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 function 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.,  $X[LEFT] \leq XB \leq X[LEFT + 1]$ . Error status can be obtained from the returned value of BSPLIN. A value of 26 implies that the required point is outside the table on higher side. BSPLIN = 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. BSPLIN = 203 implies that  $NX < 2$  or  $K < 1$ , in which case no calculations are done. The parameter KMAX in the function can be trivially increased, but such high order splines may not be meaningful.

**21. BSPINT** Function 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 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.  $K = 4$  implies cubic B-splines, while  $K = 2$  gives linear B-splines, etc. A is an array of length  $LA \times 3K$ , 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  $K - 1$ , 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 second dimension of  $A$  as specified in the calling function ( $LA \geq N$ ).  $C$  is an array of length  $N$ , which will contain the coefficients of B-spline basis functions in expansion for interpolation.  $XF$  is an 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  $NO + K - 2$ , some tabular points may have to be dropped to ensure that  $N = NO + K - 2$ . 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,  $K = 2$  all points are used.  $NO$  is the number of knots used for B-spline interpolation, this would be equal to  $N + 2 - K$ .  $IFLG$  is an integer variable that specifies what calculation is to be done. For  $IFLG \leq 1$  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  $IFLG \leq 0$  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. Error status can be obtained from the returned value of  $BSPINT$ . A value of 204 implies  $N < K$  or  $K < 2$ , in which case no calculations are done. Other values may be set by functions  $BSPLIN$  or  $GAUBND$ , which are called. This function requires function  $BSPLIN$  to calculate the B-spline basis functions and function  $GAUBND$  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.

**22. BSPEVL** To calculate the value of function using available coefficients of B-spline expansion. The calculated value is given by:

$$BSPEVL = \sum_{i=0}^{N+K-3} WT[i] \phi_i(X0), \quad (C.2)$$

where  $\phi_i$  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 an array of length  $N$  containing the knots for B-splines.  $X[0]$  should contain the first knot and the knots must be distinct and in ascending order.  $K$  is the order of B-splines,  $K = 2$  gives linear B-splines, while  $K = 4$  yields cubic B-splines, etc.  $NDERIV$  specifies the number of derivatives to be calculated. For  $NDERIV \leq 0$  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 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 function 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. IER is the error parameter, which should be zero after successful execution. Nonzero values of IER may be set by function BSPLIN which is called to calculate the B-spline basis functions. This function requires function BSPLIN.

**23. RATNAL** To calculate 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 function, then it will be reduced to the minimum of these three numbers and the error flag RATNAL may be set to 22. If NUSE is less than 1, it will be set to  $\text{MIN}(6, \text{NTAB})$  and RATNAL 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 function. FB is the output parameter containing the interpolated value. AEPS is a parameter specifying the required accuracy. Interpolation will be continued until two successive values differ by less than AEPS. If  $\text{AEPS} \leq 0$ , then this convergence criterion will never be satisfied. Error status can be obtained from the returned value of RATNAL. A value of 21 implies that  $\text{NUSE} < 1$ , in which case it is set to  $\text{MIN}(6, \text{NTAB})$ .  $\text{RATNAL} = 22$  implies that  $\text{NUSE} > \text{NTAB}$ , or  $\text{NUSE} > \text{NMAX}$ , in which case it is reduced appropriately.  $\text{RATNAL} = 23$  implies that, the interpolation has not converged to the specified accuracy.  $\text{RATNAL} = 205$  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 function 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 function with successively increasing value of NUSE, until it encounters the singularity. If these values are converging, then the result may be acceptable. This function requires function NEARST.

**24. POLY2** Function for polynomial interpolation in two dimensions. (XB1, XB2) is the point at which interpolated value is required. X1 and X2 are arrays of length N1 and N2 respectively, containing the abscissas. F is an array of length  $\text{NDIM} \times \text{N2}$  containing the function values  $F[J][I] = f(X1[I], X2[J])$ . NDIM should be set to the actual value of the second dimension of the array F in the calling function ( $\text{NDIM} \geq \text{N1}$ ). NP1 and NP2 are the number of points to be used along the two axes for interpolation. The function may adjust these values if required. FB is the output parameter containing the interpolated



value. Error status can be obtained from the returned value of POLY2. A value of 206 implies that  $N1 > \text{NDIM}$ , in which case no calculations are done. This function 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 function will require functions DIVDIF0 and NEARST.

**25. LINRN** Function to perform linear interpolation in N variables. XB is an array of length N containing the coordinates of the point at which interpolated value is required. X is an array of length  $\text{NXD} \times N$  containing the abscissas.  $X[i][j]$  is the  $j$ th point along  $i$ th dimension. F is an N-dimensional array of dimension  $F[\text{NDIM}[N-1]] \dots [\text{NDIM}[1]][\text{NDIM}[0]]$  containing the function values

$$F[I_{N-1}] \dots [I_1][I_0] = f(X[0][I_0], X[1][I_1], \dots, X[N-1][I_{N-1}]), \quad (\text{C.3})$$

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 function. NXD is the actual value of the second dimension of X, and it should be greater than the maximum element in NP. 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

```
double f[8][6][5][5], x[5][10], xb[5], fb;
int n=4, nxd=10, np[4]={4, 5, 6, 7}, ndim[4]={5, 5, 6, 8};
. . . . .
ier= linrn(n, xb, &x[0][0], &f[0][0][0][0], np, &fb, ndim, nxd);
```

Error status can be obtained from the returned value of LINRN. A value of 207 implies 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 function requires function LOCATE.

**26. LOCATE** Function 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  $X[\text{LOCATE}]$  and  $X[\text{LOCATE}+1]$ , unless it is outside the range of the table. If XB is before the first point, then LOCATE = 0, while if XB is beyond the last point, then LOCATE = NP - 2.

**27. BSPINT2** Function 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 arrays of length NX and NY containing the abscissas. The abscissas must be distinct and in ascending order. F is an array of size  $\text{LA} \times \text{NY}$  containing the function values.  $F[J][I]$  should contain  $f(X[I], Y[J])$ . 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 an array of length  $LA \times 3K$  which will contain the triangular decomposition of matrix for interpolation along X. AY is an array of length  $LA \times 3K$  which will contain the triangular decomposition of matrix for interpolation along Y. LA is the second dimension of arrays AX, AY, C, F as declared in the calling function. For simplicity, the second 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 an 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 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  $IFLG \leq 1$  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. Error status can be obtained from the returned value of BSPINT2. A zero value implies successful execution of the function. Nonzero values may be set by BSPINT, BSPLIN or GAUBND which are called. This function requires BSPINT, BSPLIN and GAUBND. The interpolated value can be calculated using function BSPEV2 with the coefficients computed by BSPINT2.

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

$$BSPEV2 = \sum_{i=0}^{NX+K-3} \sum_{j=0}^{NY+K-3} WT[j][i] \phi_i(X0) \psi_j(Y0), \quad (C.4)$$

where  $\phi_i(x)$  are the B-spline basis functions along  $x$  and  $\psi_j(y)$  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 arrays of length NX, NY containing the knots for B-splines. The knots must be in ascending order with X[0] and Y[0] 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  $NDERIV \leq 0$  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 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 function 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. IER is the error parameter, which should be zero after successful execution. Nonzero values of IER may be set by function BSPLIN which is called to calculate the B-spline basis functions. This function requires function BSPLIN.

**29. BSPINTN** Function 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 an array of length  $NXD \times N$  specifying the abscissas along each dimension. X[J][I] is the Ith abscissa along the Jth dimension. For each dimension, the abscissas must be unique and in ascending order. NXD is the second dimension of arrays X, XF and INTX as specified in the calling function,  $NXD \geq \max(NK[0], NK[1], \dots, NK[N - 1])$ . F is an array of length  $NK[0] \times NK[1] \times \dots \times NK[N - 1]$  containing the table of values. Since BSPINTN treats this array as a one dimensional array, the dimensions of F in the calling function must exactly match the size of the table so that there are no gaps in memory allocation. For example, the dimension could be  $F[NK[N - 1]] \dots [NK[1]][NK[0]]$ . 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 an 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  $K - 1$ , it is stored in band form. C is an array of length  $NK[0] \times NK[1] \times \dots \times NK[N - 1]$ , 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 function must exactly match the size of the table. XF is an 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  $MK[I] + K - 2$ , some tabular points may have to be dropped to ensure that  $NK[I] = MK[I] + K - 2$ . 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. 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  $NK[I] + 2 - K$ . INTX is an integer array of length  $NXD \times N$ , which will contain

the pivoting information for solution of system of linear equations. Error status can be obtained from the returned value of BSPINTN. A zero value implies successful execution of function. Nonzero values may be set by functions BSPINT, BSPLIN or GAUBND, which are called. It may be noted that for simplicity, this function 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 function requires BSPINT to perform interpolation in one dimension, BSPLIN to calculate the B-spline basis functions and 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.

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

$$\text{BSPEVN} = \sum_{i_0=0}^{\text{NK}[0]+K-3} \cdots \sum_{i_{n-1}=0}^{\text{NK}[n-1]+K-3} \text{WT}[i_{n-1}] \cdots [i_0] \quad (\text{C.5})$$

$$\times \phi_{i_0}^{(0)}(\text{X0}[0]) \cdots \phi_{i_{n-1}}^{(n-1)}(\text{X0}[n-1]),$$

where  $\phi_{i_j}^{(j)}$  are the B-spline basis functions along  $j$ th 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 an array of length NXD  $\times$  N containing the knots for B-splines along each dimension. The knots must be distinct and in ascending order with X[J][I] containing Ith knot along Jth dimension. NXD is the second dimension of array X in the calling function. 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 array of length (NK[0] + K - 2)(NK[1] + K - 2)  $\cdots$  (NK[N - 1] + 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 function must exactly match the table size, so that there are no gaps in memory allocation. For example, the dimension could be WT[NK[N - 1] + K - 2]  $\cdots$  [NK[1] + K - 2][NK[0] + K - 2]. These coefficients WT must be calculated beforehand using BSPINTN or any other equivalent function for B-spline approximations in n-dimensions. X0 is an array of length N, specifying the coordinates of the point where the function value needs to be calculated. IER is the error parameter, which should be zero after successful execution. Nonzero value of IER may be set by function BSPLIN which is called to calculate the B-spline basis functions. This function does not calculate the

derivatives of expansion. Since calculating derivatives could require considerable extra time separate versions of this function BSPEVN1 and BSPEVN2 are provided to calculate first and second derivatives. These functions can be used instead of BSPEVN when derivatives are required. This function requires function BSPLIN.

**31. BSPEVN1** To calculate the value of function and its first derivatives using available coefficients of B-spline expansion in n-dimensions. This function 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. If only function value is required then BSPEVN should be used, while if second derivative is also required then BSPEVN2 should be used.

**32. BSPEVN2** To calculate the value of function and its first and second derivatives using available coefficients of B-spline expansion in n-dimensions. This function 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 an array of length  $N \times N$ , 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. If second derivatives are not required then it will be better to use BSPEVN1 (for first derivatives) or BSPEVN (for only function value).

## C.5 Differentiation

**33. DRVT** Function to calculate derivative of a function which can be evaluated at any required point using a user supplied function F(X). This function uses  $h \rightarrow 0$  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 function 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 function, while if it is too small, then roundoff error will dominate. AEPS and REPS specify the required absolute and relative accuracy. This function exits when the difference between successive values is less than  $\max(\text{AEPS}, \text{REPS} \times |\text{DRVT}|)$ , 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 which calculates

the given function at any required point. This function 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.

## C.6 Integration

**34. SIMSON** 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  $\max(AEPS, REPS \times |RI|)$ . DIF is an output parameter containing an estimate of (absolute) error calculated by the function. N is an output parameter which will contain the number of function evaluations used. FUN is the name of the function to calculate the integrand. Error status can be obtained from the returned value of SIMSON. A value of 30 implies that the function 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 function 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. Function FUN(X) must be supplied by the user.

**35. SPLINT** To compute integral of a function supplied in the form of a table of values by integrating the interpolating cubic spline. Before calling this function, the coefficients of cubic spline must be calculated by a call to function 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 arrays of length N, containing the abscissas and function values. The abscissas must be supplied in ascending order. C is an array of length  $3 \times N$  containing the coefficients of cubic spline, which may be calculated using the function SPLINE. Error status can be obtained from the returned value of SPLINT. A value of 31 implies that the lower limit A is outside the limits of table, while SPLINT = 32 denotes that the upper limit B is outside the limits of table. In such cases, the accuracy of integration may be questionable. SPLINT = 301 implies that  $A > B$  or  $X[0] > X[N-1]$ . This

function requires function SPLEVL to evaluate the cubic spline, while function SPLINE will be required to calculate the coefficients of cubic spline before calling this function.

**36. BSPQD** Function 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 function, the coefficients of B-spline expansion must be calculated by a call to function BSPINT.

$$\text{BSPQD} = \int_{\text{XL}}^{\text{XU}} \sum_{i=0}^{N+K-3} \text{WT}[i] \phi_i(x) dx, \quad (\text{C.6})$$

where  $\phi_i(x)$  are the B-spline basis functions of order  $K$ .  $N$  is the number of knots,  $X$  is an array of length  $N$  containing the knots for B-splines. The knots must be in ascending order with  $X[0]$  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.  $\text{WT}$  is an array of length  $N + K - 2$  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.  $\text{XL}$  and  $\text{XU}$  are the lower and upper limits of the integral.  $\text{IER}$  is the error parameter.  $\text{IER} = 31$  implies that the lower limit  $\text{XL}$  is outside the limits of table, while  $\text{IER} = 32$  denotes that the upper limit  $\text{XU}$  is outside the limits of table. In such cases, the accuracy of integration may be questionable. Other values may be set by function BSPLIN which is called to calculate B-spline basis functions. This function requires function BSPLIN to evaluate the B-spline basis functions, while function BSPINT will be required to calculate the coefficients of B-spline before calling this function.

**37. ROMBRG** Function to compute integral over a finite interval, using Romberg integration or the  $h \rightarrow 0$  extrapolation.  $\text{RI}$  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  $A < B$ .  $\text{GI}$  is an array of dimension  $\text{NMAX}$  ( $= 13$ ), containing the expected values of successive exponents in the error expansion for the trapezoidal rule. If  $\text{GI}[I] \leq 0$ , then it will be set to  $2I+2$ , which is the correct value for smooth functions.  $\text{AEPS}$  and  $\text{REPS}$  specify the required absolute and relative accuracy. The calculations are terminated when the successive values differ by less than  $\max(\text{AEPS}, \text{REPS} \times |\text{RI}|)$ .  $\text{DIF}$  is an output parameter giving the error as estimated by the function.  $N$  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 function. Hence,  $N$  will have to be reset after every call to the function. If  $N < 2$  it will be set to a default value of 2. On the other hand, if  $N > \text{NPT}$  ( $= 100$ ), then it will be set to a default value of 2 and the error flag ROMBRG will be set to 33. In particular, this situation may arise if  $N$  is not reset after the previous call to the function. Error status can be obtained from the returned value of ROMBRG. A value of 30 implies that the function 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 function 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 used to calculate the value of the integrand. Function FUN(X) must be supplied by the user. This function 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.

**38. EPSILN** To compute integral of a function over a finite interval, using  $\epsilon$ -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  $\max(AEPS, REPS \times |RI|)$ . DIF is an output parameter giving the error as estimated by the function. 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 function. Hence, N will have to be reset after every call to the function. 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 EPSILN will be set to 33. In particular, this situation may arise if N is not reset after the previous call to the function. Error status can be obtained from the returned value of EPSILN. A value of 30 implies that the function failed to converge to the specified accuracy.  $EPSILN = 34$  denotes that at some stage while constructing the  $\epsilon$ -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.  $EPSILN = 35$  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 function could be increased to allow for larger number of function evaluations, but that is not recommended. FUN is the name of the function used to calculate the value of the integrand. Function FUN(X) must be supplied by the user. This function could be used for singular integrands, if only moderate accuracy is required. It will fail to converge to high accuracy because of roundoff error.

**39. GAUSS** Function to compute integral 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 function 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 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  $\max(\text{AEPS}, \text{REPS} \times |\text{RINT}|)$ . DIF is an output parameter giving the error as estimated by the function. NPT is an output parameter containing the number of function evaluations actually used. Error status can be obtained from the returned value of GAUSS. A value of 30 implies that the function 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,  $\text{NP} = 32$  is the maximum value allowed by this function. The parameter NMAX inside the function 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.  $\text{GAUSS} = 36$  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 used to calculate the value of the integrand. Function FUN(X) must be supplied by the user. This function should be used for smooth integrands. 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.

**40. GAUCBY** Function to compute integral using Gauss-Chebyshev formulae with weight function  $1/\sqrt{(x-A)(B-x)}$ . 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  $\max(\text{AEPS}, \text{REPS} \times |\text{RINT}|)$ . DIF is an output parameter giving the error as estimated by the function. NPT is an output parameter containing the number of function evaluations actually used. Error status can be obtained from the returned value of GAUCBY. A value of 30 implies that the function 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 function 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 used to calculate the value of the integrand multiplied by  $\sqrt{(x-A)(B-x)}$ . Function FUN(X) must be supplied by the user. This function should be used for integrands with square root singularity at both ends. It will evaluate the integral

$$\int_A^B \frac{\text{FUN}(x)}{\sqrt{(x-A)(B-x)}} dx. \quad (\text{C.7})$$

**41. GAUCB1** Function to compute integral using Gauss-Chebyshev formulae with weight function  $\sqrt{(x-A)/(B-x)}$ . 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  $\max(\text{AEPS}, \text{REPS} \times |\text{RINT}|)$ . DIF is an output parameter giving the error as estimated by the function. NPT is an output parameter containing the number of function evaluations actually used. Error status can be obtained from the returned value of GAUCB1. A value of 30 implies that the function 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 function 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 used to calculate the value of the integrand multiplied by  $\sqrt{(B-x)/(x-A)}$ . Function FUN(X) must be supplied by the user. This function should be used for integrands with square root singularity at both ends. It will evaluate the integral

$$\int_A^B \frac{\text{FUN}(x)\sqrt{x-A}}{\sqrt{B-x}} dx. \quad (\text{C.8})$$

**42. GAUCB2** Function to compute integral using Gauss-Chebyshev formulae with weight function  $\sqrt{(x-A)(B-x)}$ . 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  $\max(\text{AEPS}, \text{REPS} \times |\text{RINT}|)$ . DIF is an output parameter giving the error as estimated by the function. NPT is an output parameter containing the number of function evaluations actually used. Error status can be obtained from the returned value of GAUCB2. A value of 30 implies that the function 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 function 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 used to calculate the value of the integrand divided by  $\sqrt{(B-x)(x-A)}$ . Function FUN(X) must be supplied by the user. This function should be used for integrands with square root singularity at both ends. It will evaluate the integral

$$\int_A^B \text{FUN}(x)\sqrt{(x-A)(B-x)} dx. \quad (\text{C.9})$$

**43. GAUSQ2** 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 function to achieve the required accuracy. After execution,  $A1$  will contain the final value used by the function.  $A1$  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  $\max(\text{AEPS}, \text{REPS} \times |\text{RINT}|)$ . Convergence is checked separately for the two integrals. DIF is an output parameter giving the error as estimated by the function. NP is the output parameter containing the number of function evaluations actually used. This function calls the functions GAUSS and GAUSQ to perform the integration over required subintervals. Error status can be obtained from the returned value of GAUSQ2. A value of 30 implies that the function GAUSS failed to converge to the specified accuracy over the interval  $[A1, A]$ . GAUSQ2 = 37 denotes that the function 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 function 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. GAUSQ2 = 38 denotes that both functions 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 used to calculate the value of the integrand. F2 is the name of the function used to calculate  $F(X)*\text{SQRT}(X)$ , as required by the Gaussian formula. Note that the function will find the integral

$$\int_0^A F(x) dx = \int_{A1}^A F(x) dx + \int_0^{A1} \frac{F2(x)}{\sqrt{x}} dx. \quad (\text{C.10})$$

Functions  $F(X)$ , and  $F2(X)$  must be supplied by the user. The requirement of two separate functions is to avoid possible problems caused by singularity in integrand. If  $A - A1 < \text{AEPS}$ , 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 function requires functions GAUSS and GAUSQ. It should be used only for those integrands 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  $f(x)\sqrt{x} = (xf(x))/\sqrt{x}$ .

**44. GAUSQ** Function to compute integral

$$\int_0^A \frac{\text{FUN}(x)}{\sqrt{x}} dx, \quad (\text{C.11})$$

using Gaussian formulae with  $1/\sqrt{x}$  weight function. The function 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 function. The convergence criterion used is  $|DIF| < \max(\text{REPS}|RINT|, \text{AEPS})$ . NPT is an output parameter containing the number of function evaluations actually used. FUN is the name of the function used to calculate the value of the integrand (multiplied by  $\sqrt{x}$ ). Error status can be obtained from the returned value of GAUSQ. A value of 30 implies that the function 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. Function FUN(X) must be supplied by the user. This function should be used for integrands which have a square root singularity (functions of form  $f(x)/\sqrt{x}$  or  $f(x)\sqrt{x}$ ) at  $x = 0$ . This function should be preferably used through GAUSQ2, in which case the upper limit A may be adjusted if the required accuracy is not achieved.

**45. GAULAG** Function 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  $[A, A1]$  being evaluated using Gauss-Legendre formula, while that over  $[A1, \infty)$  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 function to achieve the required accuracy. After execution, A1 will contain the final value used by the function. 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  $\max(\text{AEPS}, \text{REPS} \times |RINT|)$ . Convergence is checked separately for the two integrals. DIF is an output parameter giving the error as estimated by the function. NP is the output parameter containing the number of function evaluations actually used. This function calls the functions GAUSS and LAGURE to perform the integration over required subintervals. Error status can be obtained from the returned value of GAULAG. A value of 30 implies that the function GAUSS failed to converge to the specified accuracy over the interval  $[A, A1]$ . GAULAG = 37 denotes that the function 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 function 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. GAULAG = 38 denotes that both functions 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 used to calculate the value of the integrand. F2 is the name of the function used to calculate  $F(X)*\text{EXP}(X)$ , as required by the Gauss-Laguerre formula.

Note that the function will find the integral

$$\int_A^\infty F(x) dx = \int_A^{A1} F(x) dx + \int_{A1}^\infty e^{-x} F2(x) dx. \quad (\text{C.12})$$

Both Functions  $F(X)$ , and  $F2(X)$  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 function. If  $A1 - A < \text{AEPS}$ , 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 function requires functions `GAUSS` and `LAGURE`. It should be used only for those functions which fall off exponentially at large  $X$ . Similar function can be written to handle singularities by combining the Gauss-Legendre formula with another Gaussian formula for singular weight functions.

**46. LAGURE** Function to compute integral  $\int_A^\infty e^{-x} F(x) dx$  using Gauss-Laguerre formulae. The function 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 function. The convergence criterion used is  $|\text{DIF}| < \max(\text{REPS}|\text{RINT}|, \text{AEPS})$ . `NPT` is an output parameter containing the number of function evaluations actually used. Error status can be obtained from the returned value of `LAGURE`. A value of 30 implies that the function 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 used to calculate the value of the integrand (multiplied by  $e^x$ ). Function  $F(X)$  must be supplied by the user. This function should be used for integrands which fall off exponentially at large  $x$ . It will be better to use this function through `GAULAG`, which adjusts the lower limit `A` to achieve the required accuracy.

**47. HERMIT** Function to compute integral  $\int_{-\infty}^\infty e^{-x^2} F(x) dx$  using Gauss-Hermite formulae. The function 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 function. The convergence criterion used is  $|\text{DIF}| < \max(\text{REPS}|\text{RINT}|, \text{AEPS})$ . `NPT` is an output parameter containing the number of function evaluations actually used. Error status can be obtained from the returned value of `HERMIT`. A value of 30 implies that the function 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 used to calculate the value of the integrand (multiplied by  $e^{x^2}$ ). Function F(X) must be supplied by the user. This function should be used for integrands 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.

**48. GAULG2** Function 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  $[A1, A]$  being evaluated using Gauss-Legendre formula, while that over  $(0, A1]$  is evaluated using a Gaussian formula with  $\log(1/x)$  weight function. If  $A < A1$ , then A1 is set equal to A to start with. A1 is adjusted by the function to achieve the required accuracy. After execution, A1 will contain the final value used by the function. 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  $\max(\text{AEPS}, \text{REPS} \times |\text{RINT}|)$ . This criterion is applied separately to each part of the integral. DIF is an output parameter giving the error as estimated by the function. NP is the output parameter containing the number of function evaluations actually used. This function calls the functions GAUSS and GAULOG to perform the integration over required subintervals. Error status can be obtained from the returned value of GAULG2. A value of 31 implies that the function GAUSS failed to converge to the specified accuracy over the interval  $[A1, A]$ . GAULG2 = 32 denotes that the function 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 function 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. GAULG2 = 34 denotes that function GAUSS failed while evaluating the integral over  $[0, A1]$ . In case of multiple failures, the second digit of GAULG2 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 used to calculate the value of the integrand. F2 is the name of the function used to calculate  $F(X)/\text{LOG}(1/X)$ , as required by the Gaussian formula. Note that the function will find the integral

$$\int_0^A F(x) dx = \int_{A1}^A F(x) dx - \log A1 \int_0^{A1} F2(x) dx + \int_0^{A1} F2(x) \log(A1/x) dx . \quad (\text{C.13})$$

Both the functions  $F(X)$ , and  $F2(X)$  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  $A - A1 < AEPS$ , 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 function requires functions GAUSS and GAULOG. It should be used only for those functions which have logarithmic singularity at  $x = 0$ .

**49. GAULOG** Function to compute integral

$$\int_0^A F(x) \log(A/x) dx, \quad (C.14)$$

using Gaussian formulae with  $\log(A/x)$  weight function. The function 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 function. The convergence criterion used is  $|DIF| < \max(\text{REPS}|RINT|, \text{AEPS})$ . NPT is an output parameter containing the number of function evaluations actually used. F is the name of the function used to calculate the value of the integrand (divided by  $\log A/x$ ). Error status can be obtained from the returned value of GAULOG. A value of 30 implies that the function 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. Function  $F(X)$  must be supplied by the user. This function should be used for integrands which have a logarithmic singularity at  $x = 0$ . It would be better to use this function through GAULG2 which adjusts the value of A to achieve required accuracy.

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

$$P_j(x) = (a_j x + b_j)P_{j-1}(x) - c_j P_{j-2}(x), \quad (C.15)$$

where  $P_j(x)$  is the orthogonal polynomial of degree  $j$  in  $x$ . The coefficients  $a_j, b_j, c_j$  for  $j = 1, 2, \dots, n$  must be supplied. Here N is the number of abscissas in the required Gaussian formula. Output parameters W and AB are arrays of length N, containing the weights and corresponding abscissas. COF is an array of length  $3 \times N$  containing the coefficients in the recurrence relation for the polynomials.  $\text{COF}[i-1][0]$ ,  $\text{COF}[i-1][1]$  and  $\text{COF}[i-1][2]$  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 function to calculate the weights and abscissas. Error status can be obtained from the returned value of GAUSRC. A value of 302 implies that  $N \leq 0$ , in which case no calculations are performed. GAUSRC = 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. There is no error check on the output and the accuracy can be tested by integrating the functions  $w(x)x^j$  for  $j = 0, 1, \dots, 2N - 1$ , using the calculated weights and abscissas. This function is in general better conditioned than GAUSWT, which uses only the moments. This function requires function TQL2.

**51. GAULEG** Function to calculate weights and abscissas of a Gauss-Legendre quadrature formula. This function uses the recurrence relation for Legendre polynomials

$$nP_n(x) = (2n - 1)xP_{n-1}(x) - (n - 1)P_{n-2}(x), \quad (\text{C.16})$$

to calculate the weights and abscissas using function GAUSRC. Here N is the number of abscissas in the required Gaussian formula. Output parameters W and A are arrays of length N, containing the weights and corresponding abscissas. Error status can be obtained from the returned value of GAULEG. Zero value implies successful execution. Nonzero values may be set by function GAUSRC. There is no error check on the output and the accuracy can be tested by integrating the functions  $x^j$  for  $j = 0, 1, \dots, 2N - 1$ , using the calculated weights and abscissas. This function requires functions GAUSRC and TQL2.

**52. GAUJAC** Function to calculate weights and abscissas of a Gauss-Jacobi quadrature formula. This function uses the recurrence relation for Jacobi polynomials

$$\begin{aligned} 2n(n + \alpha + \beta)(2n - 2 + \alpha + \beta)P_n^{\alpha, \beta}(x) = \\ - 2(n - 1 + \alpha)(n - 1 + \beta)(2n + \alpha + \beta)P_{n-2}^{\alpha, \beta}(x) \\ + (2n - 1 + \alpha + \beta)(\alpha^2 - \beta^2 + (2n + \alpha + \beta)(2n - 2 + \alpha + \beta)x)P_{n-1}^{\alpha, \beta}(x), \end{aligned} \quad (\text{C.17})$$

to calculate the weights and abscissas using function GAUSRC. Here N is the number of abscissas in the required Gaussian formula while ALP and BETA are the indices  $\alpha$  and  $\beta$  in (C.17). The corresponding weight function is  $w(x) = (1 - x)^\alpha(1 + x)^\beta$  on the interval  $(-1, 1)$ . Output parameters W and A are arrays of length N, containing the weights and corresponding abscissas. Error status can be obtained from the returned value of GAUJAC. Zero value implies successful execution. GAUJAC = 313 implies  $\text{ALP} \leq -1$  or  $\text{BETA} \leq -1$ . Other values may be set by function GAUSRC. 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  $j = 0, 1, \dots, 2N-1$ , using the calculated weights and abscissas. Setting  $ALP = 0$  and  $BETA = 0$  in this function 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  $t^\beta$ , by transforming the lower limit to  $t = 0$ .

$$\int_{-1}^1 (1-x)^\alpha (1+x)^\beta f(x) dx = 2^{\alpha+\beta+1} \int_0^1 (1-t)^\alpha t^\beta f(2t-1) dt. \quad (C.18)$$

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 function requires functions GAMMA, GAUSRC and TQL2.

**53. LAGURW** Function to calculate weights and abscissas of a Gauss-Laguerre quadrature formula. This function uses the recurrence relation for associated Laguerre polynomials

$$nL_n^\alpha(x) = (n+\alpha)((2n-1+\alpha)-x)L_{n-1}^\alpha(x) - (n-1+\alpha)^2(n+\alpha)L_{n-2}^\alpha(x), \quad (C.19)$$

to calculate the weights and abscissas using function GAUSRC. Here  $N$  is the number of abscissas in the required Gaussian formula and  $ALP$  is the index  $\alpha$  in (C.19). The corresponding weight function is  $w(x) = x^\alpha e^{-x}$  on the interval  $(0, \infty)$ . Output parameters  $W$  and  $A$  are arrays of length  $N$ , containing the weights and corresponding abscissas. Error status can be obtained from the returned value of LAGURW. Zero value implies successful execution.  $LAGURW = 313$  implies  $ALP \leq -1$ . Other values may be set by function GAUSRC. There is no error check on the output and the accuracy can be tested by integrating the functions  $e^{-x}x^{j+\alpha}$  for  $j = 0, 1, \dots, 2N-1$ , using the calculated weights and abscissas. For  $ALP = 0$  it will yield the standard Gauss-Laguerre quadrature formula. This function requires functions GAMMA, GAUSRC and TQL2.

**54. GAUHER** Function to calculate weights and abscissas of a Gauss-Hermite quadrature formula. This function uses the recurrence relation for Hermite polynomials

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x), \quad (C.20)$$

to calculate the weights and abscissas using function 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 arrays of length  $N$ , containing the weights and corresponding abscissas. Error status can be obtained from the returned value of GAUHER. Zero value implies successful execution. Nonzero values may be set by function GAUSRC. There is no error check on the output and the accuracy can be tested by integrating the functions  $e^{-x^2}x^j$  for  $j = 0, 1, \dots, 2N-1$ , using the calculated weights and abscissas. This function requires functions GAUSRC and TQL2.

**55. GAUSWT** Function to calculate weights and abscissas of a Gaussian quadrature formula, with arbitrary weight function. This function requires the values of moments i.e., the integrals  $\text{FMOM}(m) = \int_a^b w(x)x^m dx$  for  $m = 0, 1, \dots, 2N - 1$ . Existence of such formulae should be ensured before trying to use this function. 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 arrays of length  $N$ , containing the weights and corresponding abscissas.  $\text{FMOM}$  is the name of the function which calculates the moments as defined above. Function  $\text{FMOM}(M)$  must be supplied by the user.  $\text{QGAUS}$  is a parameter which specifies the type of formula required. If  $\text{QGAUS} = 1$ , then a Gaussian formula is found, otherwise the function calculates the weights for an interpolatory formula with given weight function. In this case, the array  $AB$  must contain the required abscissas. Error status can be obtained from the returned value of  $\text{GAUSWT}$ . A value of 303 implies that  $N \leq 0$ , in which case no calculations are performed.  $\text{GAUSWT} = 322$  implies that the function  $\text{GAUELM}$  failed to find the coefficients of the required polynomial.  $\text{GAUSWT} = 323$  implies that the function  $\text{POLYR}$  failed to find zeros of the polynomial. Both these errors can occur only for  $\text{QGAUS} = 1$  and in both these cases, no further calculations are performed. Similarly,  $\text{GAUSWT} = 324$  implies that function  $\text{GAUELM}$  failed to calculate the weights. In this case, the abscissas are already calculated, but the values may not be reliable.  $\text{GAUSWT} = 322, 323, 324$  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  $\text{GAUSWT} = 0$ , 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 function requires functions  $\text{POLYR}$  and  $\text{LAGITR}$  for solving the required polynomial, and function  $\text{GAUELM}$  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  $n = 0, 1, \dots, 2N - 1$ , using the calculated weights and abscissas. If the recurrence relation for the corresponding orthogonal polynomial are known then  $\text{GAUSRC}$  should be used to calculate the weights and abscissas.

**56. FILON** Function to evaluate integrals of the form

$$\int_a^b f(x) \sin kx \, dx \quad \text{or} \quad \int_a^b f(x) \cos kx \, dx, \quad (\text{C.21})$$

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 parameter to determine the type of

integral,  $QSIN = 1$ , implies that the integrand involves  $\sin kx$  otherwise 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  $\max(AEPS, REPS \times |RI|)$ . DIF is an output parameter which should contain an estimate of (absolute) error calculated by the function. N is an output parameter containing the number of function evaluations used. Error status can be obtained from the returned value of FILON. A value of 30 implies that the function 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 function could be increased if larger number of function evaluations are to be allowed. FUN is the name of the function to calculate the non-oscillatory part  $f(x)$  in the integrand. Function FUN(X) must be supplied by the user. The parameter THC is the critical value of  $\theta$ , 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  $(100h)^{1/6}$ . The value here is appropriate for 53-bit arithmetic. If other values are used, the errors could be larger.

**57. ADPINT** Function to compute integral using adaptive technique 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 function. The function will attempt to ensure that  $DIF < \max(|RINT| \times REPS, AEPS)$ . 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. NMAX is the maximum number of function evaluations that user is prepared to allow. If  $NMAX \leq 0$ , then NMAX will be set to a default value of MAXPT (= 100000). Error status can be obtained from the returned value of ADPINT. A value of 31 implies that the function failed to converge to the specified accuracy on at least one subinterval.  $ADPINT = 32$  denotes that this failure occurred more than IFMAX (= 5) times. In this case the accuracy requirement is adjusted by effectively increasing AEPS within the function. 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 function 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  $x = 0$ .  $ADPINT = 325$  denotes that the function 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 used to calculate the value of the integrand. Function F(X) must be supplied by the user. This function also needs the function KRONRD (or GAUS16). This function 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 function 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.

**58. KRONRD** Function to compute integral 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 function. The function 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. F is the name of the function to calculate the integrand. Function F(X) must be supplied by the user. The returned value of KRONRD is always zero. This function is used by ADPINT.

**59. GAUS16** Function to compute integral using 16 point Gauss-Legendre formula. This function 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 function 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 functions. Thus in general KRONRD should be preferred but this function 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 function. The function 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. F is the name of the function to calculate the integrand. Function F(X) must be supplied by the user. The returned value of GAUS16 is always zero. This function is used by ADPINT.

**60. CAUCHY** Function 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 function. The function will attempt to ensure that  $DIF < \max(|RINT| \times REPS, AEPS)$ . F is the name of the function used to calculate the value of the integrand. FUNP is the name of the function used to calculate  $F(C + X) + F(C - X)$ . Error status can be obtained from the returned value of CAUCHY. A value of 304 implies that  $A > B$ ,  $A > C$  or  $C > B$ . In this case no calculations are done. Other values may be set by function ADPINT, which is called twice to perform the integration. The returned value is set to  $IER1 + 2IER2$ , where IER1 and IER2 are the values returned by the 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. This function needs functions ADPINT and KRONRD. Functions F(X) and FUNP(X) must be supplied by the user. The value of C is available through global variable CAU\_CC. 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  $F(C + X) + F(C - X)$  can be simplified to remove singularity, the roundoff errors will be reduced.

**61. EULER** Function to compute the sum of an alternating series using Euler's transformation.

$$SUM = \sum_{i=0}^{N-1} A0 \times (-1)^i \text{TERM}(i). \quad (\text{C.22})$$

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 function sums the first M1 terms and the last M2 terms separately and applies the Euler transform to the remaining terms from M1 to  $N - M2 - 1$ . 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  $N = 0$ , 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  $\max(\text{AEPS}, \text{REPS} \times |\text{SUM}|)$ . 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 function. M2 and N2 are relevant only for finite series. SUM is the calculated value of the sum. TERM is the name of the function to calculate the terms of the series. Error status can be obtained from the returned value of EULER. A value of 31 implies that  $M1 + M2 + 2NMAX \geq N$ , 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. EULER = 32 implies that the series obtained after Euler transform did not converge to required accuracy at the lower end. EULER = 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. EULER = 36 implies that the series obtained after Euler transform did not converge to required accuracy at both ends. Function TERM(I) must be supplied by the user to calculate the Ith term of the series apart from the sign as in Eq. (C.22). Note that in keeping with the C convention the terms are numbered from 0 to  $N - 1$ . This must be accounted for in writing function TERM(I).

**62. BSPQD2** Function 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 function, the coefficients of B-spline expansion must be calculated by a call to the function BSPINT2.

$$\text{BSPQD2} = \int_{\text{XL}}^{\text{XU}} dx \int_{\text{YL}}^{\text{YU}} dy \sum_{i=0}^{\text{NX}+K-3} \sum_{j=0}^{\text{NY}+K-3} \text{WT}[j][i] \phi_i(x) \psi_j(y), \quad (\text{C.23})$$

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. NX, NY are the number of knots along  $x$  and  $y$  respectively. X and Y are arrays of length NX and NY containing the knots for B-splines. The knots must be in ascending order with X[0], Y[0] 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 an array of length  $\text{IW} \times (\text{NY} + K - 2)$  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 second dimension of array WT as declared in the calling function. 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  $y$ . IER is the error parameter.  $\text{IER} = 0$  implies

successful execution of the function. Nonzero values may be set by BSPQD which is called to perform integration in one dimension. This function requires function BSPLIN to evaluate the B-spline basis functions, and function BSPQD for integration in one dimension, while function BSPINT2 will be required to calculate the coefficients of B-spline before calling this function.

**63. BSPQDN** Function 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 function, the coefficients of B-spline expansion must be calculated by a call to the function BSPINTN. This function evaluates the integral

$$\int_{XL[0]}^{XU[0]} dx_0 \cdots \int_{XL[N-1]}^{XU[N-1]} dx_{N-1} \sum_{i_0=0}^{NK[0]+K-3} \cdots \sum_{i_{N-1}=0}^{NK[N-1]+K-3} \text{WT}[i_{N-1}] \cdots [i_0] \phi_{i_0}^{(0)}(x_0) \cdots \phi_{i_{N-1}}^{(N-1)}(x_{N-1}) \quad (\text{C.24})$$

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  $x_0$  is evaluated for each value of other indices and then the integral over  $x_1$  is calculated and so on. It may be noted that the function BSPQD to evaluate the integral in one dimension is not called recursively.  $N$  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  $I$ th dimension.  $X$  is an array of length  $NXD \times N$  containing the knots for B-splines along each dimension. The knots must be in ascending order.  $X[J][I]$  is the  $I$ th knot along  $J$ th dimension. It may be noted that the function BSPINTN gives the list of knots in the same format and that array can be directly used in this function.  $NXD$  is the second dimension of the array  $X$  as specified in the calling function,  $NXD \geq \max(NK[i])$ .  $K$  is the order of B-splines,  $K = 2$  gives linear B-splines, while  $K = 4$  yields cubic B-splines, etc.  $WT$  is an array of length  $(NK[0] + K - 2)(NK[1] + K - 2) \cdots (NK[N - 1] + K - 2)$  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 function the array should have dimension  $WT[NK[N-1]+K-2] \cdots [NK[1]+K-2][NK[0]+K-2]$ . Alternately, it can be treated as a one dimensional array of required length.  $XL$  and  $XU$  are 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  $I$ th dimension.  $IER$  is the error parameter.  $IER = 0$  implies successful execution of the function. Nonzero values may be set by BSPQD which is called to perform integration in one dimension. This function requires function BSPLIN to evaluate the B-spline basis functions, and function BSPQD for integration in one dimension, while function BSPINTN will be required to calculate the coefficients of B-splines before calling this function.

**64. MULINT** Function for integration over a hyper-rectangle in N dimensions using a series of product Gauss-Legendre formulae. A and B are 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 function is concerned, but the number of function evaluations required to evaluate the integral for large N will put some limit. M is an integer array of length N, specifying which Gaussian formula is to be used along each axis in the first approximation. The function 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 function. 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 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 an 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 function will attempt to ensure that  $DIF < \max(|RINT| \times REPS, AEPS)$ . NUM is an output parameter containing the number of function evaluations actually used. 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. Error status can be obtained from the returned value of MULINT. A value of 39 implies that the integral failed to converge to the required accuracy in MAXPT function evaluations. 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. MULINT = 305 implies that  $N < 1$ . MULINT = 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 function requires function NGAUSS.

**65. NGAUSS** Function for integration over a hyper-rectangle in N dimensions using a product Gauss-Legendre formula. A and B are 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.  $M$  is an integer array of length  $N$ , specifying the Gaussian formula to be used along each axis. The function attempts to use  $M[I]$ -point Gauss-Legendre formula along the  $I$ th axis. Formulae with 2, 4, 8, 16 and 32 points are incorporated in the function. If any other value is specified for  $M[I]$ , then 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 no calculations will be performed.  $F$  is the name of the function 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 an 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.  $MAXPT$  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. Error status can be obtained from the returned value of  $NGAUSS$ . A value of 305 implies that  $N < 1$ .  $NGAUSS = 306$  implies that either  $M[J]$  is not 2, 4, 8, 16 or 32 or  $INT[J] < 1$  for some  $J$ .  $NGAUSS = 307$  implies that the number of points exceeded  $MAXPT$ . In all these cases no calculations are performed. This function is called by `MULINT` which adjusts the number of abscissas to be used along each dimension depending on the required accuracy. It can also be used directly to calculate the integral using quadrature formulae with specified number of points in each dimension.

**66. SPHND** Function 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 function `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 to calculate the integrand. The limits  $A[I]$  and  $B[I]$  of integration along  $I$ th dimension should be  $[0, \pi]$  for  $I = 1, \dots, N - 2$ ,  $[0, 2\pi]$  for  $I = N - 1$  and  $[r_l, r_u]$  for  $I = 0$ . 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 function gives the transformation between hyper-spherical coordinates  $x_i$  and Cartesian coordinates  $y_i$

$$\begin{aligned}
 y_0 &= x_0 \cos(x_1), \\
 y_1 &= x_0 \sin(x_1) \cos(x_2), \\
 y_2 &= x_0 \sin(x_1) \sin(x_2) \cos(x_3), \\
 &\dots \\
 y_{n-2} &= x_0 \sin(x_1) \sin(x_2) \cdots \sin(x_{n-2}) \cos(x_{n-1}), \\
 y_{n-1} &= x_0 \sin(x_1) \sin(x_2) \cdots \sin(x_{n-2}) \sin(x_{n-1}).
 \end{aligned} \tag{C.25}$$

The integral can be written as

$$\begin{aligned} \int_{S_n} f(x) dV \\ = \int_{r_l}^{r_u} dx_0 \int_0^\pi dx_1 \cdots \int_0^\pi dx_{n-2} \int_0^{2\pi} dx_{n-1} f(x) x_0^{n-1} \prod_{i=1}^{n-2} \sin^{n-i-1} x_i . \end{aligned} \quad (\text{C.26})$$

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 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.

**67. STRINT** Function for integration over a hyper-rectangle in N dimensions using compound monomial rules of degree 1, 3 or 5. A and B are 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 function is concerned, but the function STROUD which is invoked to actually evaluate the integrals will have some limit. If this limit is exceeded, error flag will be set. 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 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 an 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 function will attempt to ensure that  $DIF < \max(|RINT| \times REPS, AEPS)$ . NUM is an output parameter containing the number of function evaluations actually used. 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.

Error status can be obtained from the returned value of STRINT. A value of 39 implies that the integral failed to converge to the required accuracy in MAXPT function evaluations. 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 value of IND[I] should not be reset, since that will save some function evaluations in computing initial approximations, which are known to be unsatisfactory. STRINT = 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. STRINT = 309 implies that N is outside the limits accepted by STROUD. This function requires function STROUD.

**68. STROUD** Function for integration over a hyper-rectangle in N dimensions using a compound monomial rule of degree 1, 3 or 5. A and B are 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 error flag will be set. 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  $(2N^2 + 1)$ -point formula of degree 5. If any other value is specified for M, then 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 no calculations will be performed. F is the name of the function 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 an 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. 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. Error status can be obtained from the returned value of STROUD. A value of 308 implies that the number of points exceeded MAXPT. STROUD = 309 implies that  $N < 1$  or  $N > NMAX$ . STROUD = 310 implies that M is not 1, 3 or 5, or  $IND[I] < 1$  for some I. In all these cases and no calculations are performed. This function is called by STRINT which adjusts the number of abscissas to be used along each dimension depending on the required accuracy.

**69. MCARLO** Function for integration over a hyper-rectangle in N dimensions using Monte Carlo method. A and B are 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.  $NPT$  is the maximum number of function evaluations to be used for integration.  $F$  is the name of the function 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 an 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  $\sigma$  as defined in Section 6.11.  $ERR$  is set to  $2.576\sigma/\sqrt{NP}$  corresponding to a confidence level of 99%. The function attempts to ensure that  $ERR < \max(|RI| \times REPS, AEPS)$ .  $NP$  is the number of function evaluations actually used. Error status can be obtained from the returned value of `MCARLO`. A value of 39 implies that the function failed to converge to the specified accuracy. In this case,  $RI$  will contain the best estimate and  $ERR$  the corresponding error estimate. `MCARLO = 311` implies that  $N < 1$  and no calculations are done. This function requires function `RANF(ISEED)` to generate random numbers. If a different function is used for this purpose, then the value of seed may have to be changed.

**70. RAN1** Function 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 function. After the first call, this variable should not be changed in any other function. It may be noted that this function 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 function.

**71. RANF** Function to generate a sequence of random numbers using a combination of three linear congruential generators. This function is based on the function `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 function. After the first call, this variable should not be changed in any other function.

**72. EQUIDS** Function for integration over a hyper-rectangle in  $N$  dimensions using the method based on equidistributed sequences.  $A$  and  $B$  are arrays of length  $N$ , with  $A[I]$  and  $B[I]$  specifying the lower and upper limits for the  $I$ th 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 error flag will be set.  $NPT$  is the maximum number of function evaluations to be used in computing the integral.  $F$  is the name of the function 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 an 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 function attempts to ensure that  $DIF < \max(|S2| \times REPS, AEPS)$ .  $NP$  is an output parameter containing the number of function evaluations actually used. Error status can be obtained from the returned value of  $EQUIDS$ . A value of 39 implies that the integral failed to converge to the required accuracy in  $NPT$  function evaluations. 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 function with double precision.  $EQUIDS = 312$  implies that  $N < 1$  or  $N > NMAX$ .

## C.7 Nonlinear Algebraic Equations

**73. BISECT** Function 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. Error status can be obtained from the returned value of  $BISECT$ . A value of 401 implies that  $NB \leq 0$ , in which case, no calculations are performed.  $BISECT = 421$  implies that the function has the same sign at both the end points and hence bisection cannot be performed.  $BISECT = -1$  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 function, 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.

**74. SECANT** 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  $\max(AEPS, REPS|X|)$ . FUN is the name of the external routine to calculate the required function. Function FUN(X) must be supplied by the user. Error status can be obtained from the returned value of SECANT. A value of 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. SECANT = 402 implies that the starting value was outside the given interval, in which case, no calculations are performed. SECANT = 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. SECANT = 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 function 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  $FUN \times 2^{JF}$  to avoid overflows and underflows, then use the function SECAN\_2 instead of SECANT. The call statement is identical for the two functions.

**75. SECANI** To calculate a real zero of a given function using the secant iteration. The function is calculated in the form  $F \times 2^{JF}$  to avoid overflows and underflows. It also uses the reverse communication technique for passing function values. This function will return control to the calling function 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.  $IER \geq 0$  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  $\max(AEPS, REPS|X|)$ . 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 function 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.

**76. NEWRAP** To calculate a real zero of a given function using the Newton-Raphson method. This function 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  $\max(AEPS, REPS|X|)$ . 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. Error status can be obtained from the returned value of NEWRAP. A value of 403 implies that the starting value was outside the given range, in which case, no calculations are performed.

NEWRAP = 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. NEWRAP = 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. NEWRAP = 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 NEWRAP is set to  $126 - 100 \times \text{multiplicity}$ . In this case, the computed value of the zero may be reasonable. On successful completion of the function, the returned value of NEWRAP will be zero if the root is simple and  $\text{NEWRAP} = -k$ , if the root is detected to be multiple with multiplicity  $k$ . Function FUN(X, DF) must be supplied by the user.

**77. BRENT** To calculate a real zero of a given function using the Brent's method. This function is based on the procedure 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 function 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  $\max(\text{AEPS}, \text{REPS}|X|)$ . F is the name of the external routine to calculate the required function. Function F(X) must be supplied by the user. Error status can be obtained from the returned value of BRENT. A value of 427 implies that the function has the same sign at both the end points and hence the Brent's method cannot be applied. BRENT = 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 function 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 function, 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.

**78. SEARCH** Function 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  $CFUN(Z, F)$  must be supplied by the user. Here  $Z$  and  $F$  are both arrays of length 2 containing the real and imaginary parts of the complex values.  $F$  is the function value at  $Z$ . The output is displayed on the screen. The returned value of the function  $SEARCH$  is always zero.

**79. ZROOT** Function to calculate complex zeros of a given function using Muller's method. This function 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 function calls `MULLER` (or `MULER2`) to find the zeros.  $N$  specifies the number of zeros to be determined.  $CX$  is an array of length  $2N$ , containing the real and imaginary parts of the  $N$  complex starting values for the iteration.  $NZ$  is the number of known zeros of the given function.  $CZERO$  is a array of length  $2(NZ + N)$  containing the real and imaginary parts of zeros. At the time of calling, the first  $2NZ$  elements should contain the known zeros of the function. All zeros found by the function 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  $\max(AEPS, REPS|X|)$ .  $RMAX$  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  $CF(Z, F)$  (or Function  $CF(Z, F, JF)$  for `MULER2`) must be supplied by the user. Here both  $F$  and  $Z$  are assumed to be arrays of length 2 containing the real and imaginary parts of the complex variables.  $F$  (or  $F \times 2^{JF}$ ) is the value of function at  $Z$ . Error status can be obtained from the returned value of `ZROOT`. A value of 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 function `MULLER` has returned a nonzero value, which is less than 100. `ZROOT = 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 function was called. Apart from `CF`, the function also requires functions `MULLER` or `MULER2`, `CABS`, `CSQRT`, `CDIV`. 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`.

**80. ZROOT2** To calculate complex zeros of a given function using Muller's method. This function 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 function calls MULER2 (or MULLER) to find the zeros. N specifies the number of zeros to be determined. CX is an array of length 2N, containing the real and imaginary parts of the N starting values for the iteration. NZ is the number of known zeros of the given function. CZERO is an array of length 2(NZ+N) containing the real and imaginary parts of zeros. At the time of calling, the first 2NZ elements should contain the known zeros of the function. All zeros found by the function 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  $\max(\text{AEPS}, \text{REPS}|X|)$ . RMAX is a parameter to specify the approximate range of  $z$  values, where zeros are required. If the iteration goes outside the region  $|Z| \leq \text{RMAX}$ , 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, F, IX) must be supplied by the user. Here both F and Z are assumed to be arrays of length 2 containing the real and imaginary parts of the complex argument, and IX is an integer variable such that the function value is given by  $f(Z) = F \times 2^{IX}$ . This form is useful for functions which will otherwise result in overflow or underflow. Error status can be obtained from the returned value of ZROOT2. A value of 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 function MULER2 has returned with a nonzero value, which is less than 100. ZROOT2 = 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 function was called. Apart from CF, the function also requires functions MULER2 (or MULLER), CABS, CSQRT, CDIV. Function MULER2 uses *reverse communication* technique to pass the function value. This function returns the control to the calling function whenever a function evaluation is required. This technique is useful when a large number of parameters, including some external function 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 function, it should check the value of IER to decide the action. If  $\text{IER} < 0$ , then a function evaluation is required and MULER2 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.

**81. MULLER** 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. All these variables are treated as arrays of length 2 containing the real and imaginary parts. 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  $\max(\text{AEPS}, \text{REPS}|CX3|)$ . CF is the name of the external routine to calculate the required function. Function CF(Z,F) must be supplied by the user. Here F and Z are both arrays of length 2 containing the real and imaginary parts of complex arguments and F is the function value at Z. NZ is the number of known zeros of the function. CZERO is an array of length 2NZ containing the real and imaginary parts of the known complex zeros. RMAX is a parameter to specify the approximate range of  $z$  values, where zeros are required. If the iteration gives  $|Z| > \text{RMAX}$ , 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. Error status can be obtained from the returned value of MULLER. A value of 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  $|CX2 - CX3|$  should give an estimate of accuracy achieved. MULLER = 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. MULLER = 404 implies that the three starting values are not distinct, in which case no calculations are performed. MULLER = 431 implies that iteration has gone outside the specified limits i.e.,  $|CX3| > \text{RMAX}$ . MULLER = 432 implies that the iteration has failed to converge to any reasonable accuracy. MULLER = 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. Apart from CF this function also requires CABS, CDIV and CSQRT to handle complex arithmetic.

**82. MULER2** Function to calculate a complex zero of a given function using Muller's method. This function is essentially identical to the function MULLER described earlier, except for the fact that here the given function is calculated in the form  $f(CX) = CF \times 2^{IX}$ , which is suitable for those functions which will normally overflow or underflow on the computer. For example, this function 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 function will return

control to the calling function 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 arrays of length 2 containing the real and imaginary parts of 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.  $IER \geq 0$  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, except that in this case the parameter IER gives the error parameter instead of the returned value of function.  $IER = 42$  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  $|CX2 - CX3|$  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.,  $|CX3| > RMAX$ .  $IER = 432$  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. Negative values of IER imply that the execution is not over and MULR2 should be called back after calculating the function value at the required point. Variables other than CF and IX should not be modified during this function calculations. For an example of the usage of this function see function ZROOT2 above. Apart from CF this function also requires CABS, CDIV and CSQRT to handle complex arithmetic.

**83. POLYR** Function 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 an array of length N+1 containing the coefficients of the polynomial. A[I] is the coefficient of  $x^I$  in the polynomial i.e., A[0] is the constant term while A[N] is the coefficient of  $x^N$ . A[N] should be nonzero. RR is an array of length N, which will contain the real part of zeros of the polynomial after the execution is complete.

The roots are sorted in the order of increasing real part. However, if the function is terminated abnormally, then the roots may not be sorted. RI is an array of length N, which will contain the imaginary part of zeros of the polynomial after the execution is complete. QREFIN is a parameter to specify if refinement of roots is needed. QREFIN = 1 specifies that the computed zeros should be “refined” by iterating with the original polynomial, otherwise 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. Error status can be obtained from the returned value of POLYR. A value of 406 implies that  $N \leq 0$ , while POLYR = 408 implies that  $A[N] = 0$ . In such cases, no calculations are performed. POLYR = 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 arrays RR and RI. POLYR =  $n \times 11$  implies that the iteration for refining the roots did not converge for  $n$  of the roots. The parameter EPS inside the function should be of the order of  $h$  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  $|y| \leq 10\text{EPS} \times |x|$ , then the root is assumed to be real. Depending on the outcome of this test, the function 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 function, unless there are roots with very small, but nonzero imaginary parts. This function requires function LAGITR to perform the Laguerre’s iteration, and CABS, CDIV and CSQRT to handle complex arithmetic.

**84. LAGITR** Function 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 array of length N + 1 containing the coefficients. A[I] is the coefficient of  $x^I$ . CXI is an array of length 2 containing the real and imaginary parts of the starting value for the Laguerre’s iteration. After execution, CXI will contain the computed root. Error status can be obtained from the returned value of LAGITR. A value of 439 implies that the iteration has failed to converge to any reasonable accuracy. LAGITR = 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. This function requires CABS, CDIV and CSQRT to handle complex arithmetic.

**85. DAVIDN** Function to solve a system of nonlinear equations using Davidenko’s method. This method can be coupled with any function for solving a system of nonlinear equations to improve its chances of convergence. It can be used with function NEWTON for Newton’s method, when the Jacobian can be easily calculated, or with function BROYDN for Broyden’s method, when the Jacobian is too complicated to be calculated explicitly. FCN is the name of the function to calculate the vector function, i.e., the left-hand sides of the equations (the right-hand sides are assumed to be zero). This function should

also calculate the Jacobian if Newton's method is to be used. The parameters THETA, X0[NMAXD], F0[NMAXD] are global variables used to pass on parameters to function FCN. Here, X0 and F0 are arrays of length NMAXD containing the initial guess and corresponding function values, which may be required to calculate the function required for the Davidenko's method. The function can be parametrised in any convenient way, but  $\text{THETA} = \theta_0$  should have the solution X0 and  $\text{THETA} = 0$  should correspond to the required function. The function FCN should be of the form

```
double THETA, X0[NMAXD], F0[NMAXD];
void fcn(int np, double x[], double f[], double *df)
{
    f[0]=...; ... ; F[NP-1]=...;
    .....
    for(i=0; i<np; ++i) f[i]=f[i]-THETA*F0[i];
}
```

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 function NEWTON. For use with BROYDN, the variable DF should be removed from the list of arguments. The last statement 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 function. The function 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 an 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 scanf statement. Apart from the starting values, the function will also ask for successive values of THETA, the parameter introduced into the equations. The value of THETA should be changed from  $\theta_0$  to zero gradually, where  $\theta_0$  is the value for which X0 is the exact solution of the system of equations. The value of  $\theta_0$  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 an 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  $\max(\text{AEPS}, \text{REPS}|X[I]|)$ . The function 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  $\text{AEPS0} = 10^{-4}$  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. Error status can be obtained from the returned value of DAVIDN. A value of 407 implies that the value of NP is greater than NMAXD ( $= 200$ ), or  $NP < 1$ . In this case, the value of NMAXD can be increased. DAVIDN = 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 may be set by the function NEWTON or BROYDN. This function requires function NEWTON or BROYDN to solve the system of nonlinear equations and function GAUELM to solve intermediate systems of linear equations. To use BROYDN instead of NEWTON, the call statement should be changed as done in DAVIDN\_B, which is the version for use with BROYDN.

**86. NEWTON** Function to solve a system of nonlinear equations using Newton's method. This method requires the calculation of the full Jacobian at every step. This function can be used directly, if a good approximation to the root is known. Otherwise, it may be better to use it through function DAVIDN to improve the chances of convergence. NP is the number of equations in the system. X is an 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 an array of length NP containing the value of vector function  $\mathbf{f}$  at the point specified by the array X. FCN is the name of the function used to calculate the left-hand sides of the system of equations (the right-hand sides are assumed to be zero). The function 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 arrays of length NP containing the values of  $\mathbf{x}$  and the vector function  $\mathbf{f}$ , respectively. DF is a two-dimensional array of length  $NP \times NP$  containing the Jacobian with  $DF[J][I] = \partial F[I]/\partial X[J]$ . It should be noted that in function FCN, the second 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  $\max(\text{AEPS}, \text{REPS}|X[I]|)$ . Error status can be obtained from the returned value of NEWTON. A value of 442 implies that the iteration did not converge to the specified accuracy. NEWTON = 441 implies that the function 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.

**87. BROYDN** Function to solve a system of nonlinear equations using Broyden's method. This method does not require the calculation of derivatives. This function can be used directly if a good approximation to the root is known. Otherwise, it may be better to use it through function DAVIDN\_B to improve the chances of convergence. NP is the number of equations in the system. X is an 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 an array of length NP containing the value of vector function  $\mathbf{f}$  at the point specified by the array X. FCN is the name of the function used to calculate the left-hand sides of the system of equations (the right-hand sides are assumed to

be zero). The function FCN(NP, X, F) must be supplied by the user. (Here NP is the number of equations in the system. X and F are arrays of length NP containing the values of  $\mathbf{x}$  and the vector function  $\mathbf{f}(\mathbf{x})$ , respectively.) REPS and AEPS specify the relative and absolute convergence criterion. All components of the root should be calculated to an accuracy of  $\max(\text{AEPS}, \text{REPS}|X[I]|)$ . Error status can be obtained from the returned value of BROYDN. A value of 442 implies that the iteration did not converge to the specified accuracy. BROYDN = 441 implies that the function 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.

## C.8 Optimisation

**88. BRACKM** Function 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. Error status can be obtained from the returned value of BRACKM. A value of 501 implies that  $A = B$  and no calculations are performed. BRACKM = 521 implies that the function 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 function to jump over the minimum, or if the minimum is too far off as compared to the initial step size.

**89. GOLDEN** To minimise a function in one dimension using the golden section search. Before calling the function, 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  $|B - A| < \max(\text{REPS}|X|, \text{AEPS})$ . 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. Error status can be obtained from the returned value of GOLDEN. A value of 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. GOLDEN = 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. GOLDEN = 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.



**90. BRENTM** To minimise a function in one dimension using the Brent's method. This function is based on the procedure given by Brent (1973). Before calling this function, 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  $\max(\text{REPS}|X|, \text{AEPS})$ . 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. Error status can be obtained from the returned value of BRENTM. A value of 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. BRENTM = 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 function and the result may not be accurate to the specified accuracy, even if the interval has been reduced to the required size.

**91. DAVIDM** 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  $\max(\text{REPS}|X2|, \text{AEPS})$ . F is the name of the function routine used to calculate the function to be minimised and its derivative. Error status can be obtained from the returned value of DAVIDM. A value of 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. DAVIDM = 502 implies that  $X1 = X2$ , in which case, no calculations will be performed. DAVIDM = 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. DAVIDM = 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. Function  $F(X, DF)$  must be supplied by the user. Here  $DF$  is the first derivative of  $F$  at  $X$ .

**92. BFGS** 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 an array of length  $N$ , containing the starting values for the independent variables. After execution, the minimiser will be returned in the same array  $X$ .  $F$  gives the value of the function at  $X$ .  $G$  is an array of length  $N$ , which will contain the value of the gradient vector at  $X$  ( $G[I] = \frac{\partial F}{\partial X[I]}$ ).  $H$  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 second 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. 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  $\max(REPS|X[I]|, AEPS)$ .  $FCN$  is the name of the function which calculates the function  $F$  and the gradient vector  $G$ . Error status can be obtained from the returned value of  $BFGS$ . A value of 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.  $BFGS = 503$  implies that  $N < 1$ , in which case no calculations are performed.  $BFGS = 526$  implies that the iteration failed to converge to a satisfactory accuracy. Other values may be set by the function  $LINMIN$ , which is called to perform the line searches. Function  $FCN(N, X, F, G)$  must be supplied by the user. Here  $N$  is the number of variables,  $F$  is the function value, while  $X$  and  $G$  are 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 function requires the function  $LINMIN$  to perform the line searches and the function  $FLNM$  to calculate the function and its derivative along the given line, as required by  $LINMIN$ .

**93. LINMIN** Function to perform line search as required by the function  $BFGS$ . 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 function requires the first derivative of the function in addition to the function value. This function 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 function 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 function may fail to find acceptable points. The parameters F, V, XI, N and NUM are not used by this function, but are simply passed on to the function FLNM, for calculating the function value and the first derivative. F is the name of the function used to calculate the function of N variables, V is an 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 an 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. Error status can be obtained from the returned value of LINMIN. A value of 527 implies that the LINMIN 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. LINMIN = 55 or 528 implies that the function 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 (LINMIN = 55), otherwise LINMIN = 528. LINMIN = 54 implies that the function 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 function BFGS. This function requires the function FLNM and the function F to calculate the function. Function 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 arrays of length N, containing the coordinates of the point X, where the function FX and the gradient vector G are to be evaluated.

**94. FLNM** To calculate the function as required for the line search routine LINMIN. FCN is the name of the function 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 an 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 an 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  $XI[I] = X0[I] + X \times V[I]$ . These coordinates are returned in N to

$2N - 1$  element of array V. The Function FCN(N, XI, F, G) must be supplied by the user. Here XI is an array of length N, containing the coordinates of the required point. F is the function value at this point. G is an array of length N containing the gradient vector at XI.

**95. NMINF** 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 an 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 gives the value of the function at X. NUM is an output parameter containing the number of function evaluations required. The parameters REPS and AEPS specify the relative and absolute convergence criterion. The iteration is continued until the change in function value  $\delta F$  during one complete cycle consisting of N iterations, satisfies  $|\delta F| < \max(\text{REPS}|F|, \text{AEPS})$ . It may be noted that unlike function 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. FCN is the name of the function which calculates the function F. Error status can be obtained from the returned value of NMINF. A value of 504 implies that  $N \leq 1$ , in which case, no calculations are performed. NMINF = 529 implies that the iteration failed to converge to a satisfactory accuracy. Other values may be set by the function LINMNF, which is called to perform the line searches. Function 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. Apart from FCN, this function requires function LINMNF to perform the line searches and function FLN to calculate the function as required by LINMNF, and function SVD to perform the singular value decomposition.

**96. LINMNF** Function to perform line search as required by the function 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 function does not require any derivative. Any point, where the function value is less than that at the starting point is acceptable. This function 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 function 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 function may fail to find an acceptable point. The parameters F, V, XI, N and NUM are not used by this function, but are simply passed on to the

function FLN, to calculate the function value. F is the name of the function used to calculate the function of N variables, V is an array of length N, containing the direction along which the line search is to be carried out. XI is an 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. Error status can be obtained from the returned value of LINMNF. A value of 56 implies that the function failed to find an acceptable point and in this case, the starting point itself is accepted. The function LINMNF requires the functions FLN and F to calculate the function. Function 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.

**97. FLN** To calculate the function as required for the line search routine LINMNF. FCN is the name of the function 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 an array of length N, containing the direction along which the search is being carried out. X0 is an 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 Function FCN(N, XI, F) must be supplied by the user. Here XI is an array of length N, containing the coordinates of the required point and F is the function value at this point.

**98. SIMPLX** Function 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 an array of length  $IA \times (N + M2 + 1)$ . IA specifies the second dimension of A, exactly as in the calling function ( $IA \geq M1 + M2 + M3 + 2$ ). The array A contains the tableau as explained in Section 8.7. It may be noted that in the C function the dimensions of array A have been interchanges so effectively, the rows are treated as columns and vice versa. At input the first column of A contains the cost coefficients  $A[i + 1][0] = c_i$ , where  $c_0x_0 + c_1x_1 + \dots + c_{N-1}x_{N-1}$  is the function to be minimised. The next  $M = M1 + M2 + M3$  columns contain the coefficients for the constraints, with the columns 1 to M1 containing the constraints of the first type, columns M1 + 1 to M1 + M2 containing the constraints of the second type and the remaining columns containing the constraints of the third type.  $A[0][i] = b_i$  and  $A[j + 1][i] = a_j$  for the  $i$ th constraint. If  $M2 + M3 > 0$ , then the column M + 1 will be used by the function 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 column will not

be required. After the execution is complete, the optimal feasible vector will be returned in the array X of length N. F is an output parameter containing the optimum value of the objective function. Error status can be obtained from the returned value of SIMPLX. A value of 57 implies that the objective function is unbounded from below and no optimal feasible vector exists. SIMPLX = 58 implies that, there is no basic feasible vector. Hence, the constraints are inconsistent or the problem is not properly formulated. SIMPLX = 505 implies that at least one of the variables N, M1, M2 and M3 is negative or  $IA < M + 2$ . SIMPLX = 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. SIMPLX = 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. SIMPLX = 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 parameter to control roundoff error, any quantity less than AEPS in magnitude is assumed to be zero. This function requires SIMPX to actually perform the simplex minimisation.

**99. SIMPX** Function 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 array A of length  $IA \times (N - M + 1)$  with  $IA \geq M + 2$ . IA is the second dimension of A, exactly as specified in the calling function. 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 parameter which determines the column from which cost coefficients are used. If  $QF = 1$ , then the main objective function specified by the first column of the tableau is to be minimised. Otherwise, the auxiliary objective function specified by the last column 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  $M + 1$  and  $N - M + 1$  respectively, used to store permutations of the original variables. AEPS is a parameter used to control roundoff error. Any quantity less than AEPS in magnitude is assumed to be zero. Error status can be obtained from the returned value of SIMPX. A value of 57 implies that the objective function is unbounded from below and the optimal feasible vector does not exist. SIMPX = 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 function.

## C.9 Statistical Inferences

**100. SHSORT** Function 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.

**101. GAMMAP** Function to calculate the incomplete Gamma function

$$P(a, x) = \frac{\gamma(a, x)}{\Gamma(a)} = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt. \quad (\text{C.27})$$

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  $A \leq 0$  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

$$P(a, x) = \frac{x^a}{\Gamma(a+1)} \left( 1 + a \sum_{n=1}^{\infty} \frac{(-x)^n}{(a+n)n!} \right). \quad (\text{C.28})$$

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

$$Q(a, x) = 1 - P(a, x) = \frac{\frac{e^{-x} x^a}{\Gamma(a)}}{x + \frac{1-a}{1 + \frac{1}{x + \frac{2-a}{1 + \frac{2}{x+\dots}}}}} \quad (\text{C.29})$$

is used. For other values of parameter the power series

$$P(a, x) = \frac{e^{-x} x^a}{\Gamma(a+1)} \left( 1 + \sum_{n=1}^{\infty} \frac{x^n}{(a+1)(a+2)\cdots(a+n)} \right) \quad (\text{C.30})$$

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

**102. BETAP** Function to calculate the incomplete Beta function

$$I_x(a, b) = \frac{B_x(a, b)}{B(a, b)} = \frac{1}{B(a, b)} \int_0^x t^{a-1} (1-t)^{b-1} dt. \quad (C.31)$$

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  $A \leq 0$  or  $B \leq 0$  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.

**103. BETSER** Function routine to calculate the incomplete Beta function (C.31) using the infinite series (DiDonato & Morris 1992)

$$I_x(a, b) = \frac{x^a}{aB(a, b)} \left( 1 + a \sum_{j=1}^{\infty} \frac{(1-b)(2-b) \cdots (j-b)}{j!(a+j)} x^j \right) \quad (C.32)$$

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. (C.31). This function needs Function GAMMAL to calculate the logarithm of Gamma function.

**104. BETCON1** Function routine to calculate the incomplete Beta function (C.31) using a continued fraction (DiDonato & Morris 1992) approximation

$$I_x(a, b) = \frac{\frac{x^a(1-x)^b}{aB(a, b)}}{1 + \frac{d_1}{1 + \frac{d_2}{1 + \cdots}}} \quad (C.33)$$

with

$$d_{2n} = \frac{n(b-n)x}{(a+2n-1)(a+2n)}, \quad d_{2n+1} = -\frac{(a+n)(a+b+n)x}{(a+2n)(a+2n+1)}. \quad (C.34)$$

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. (C.31). This function needs Function GAMMAL to calculate the logarithm of Gamma function.

**105. BETCON** Function routine to calculate the incomplete Beta function (C.31) using a continued fraction (DiDonato & Morris 1992) approximation

$$I_x(a, b) = \frac{\frac{x^a(1-x)^b}{aB(a, b)} a_1}{b_1 + \frac{a_2}{b_2 + \frac{a_3}{b_3 + \cdots}}} \quad (\text{C.35})$$

where, coefficients  $a_i, b_i$  are defined using Eq. (C.34)

$$a_1 = 1, \quad b_1 = 1 + d_1, \quad a_{n+1} = -d_{2n-1}d_{2n}, \quad b_{n+1} = 1 + d_{2n} + d_{2n+1}, \quad (n > 1). \quad (\text{C.36})$$

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. (C.31). This function needs Function GAMMAL to calculate the logarithm of Gamma function.

**106. BETAI** Function routine to calculate the incomplete Beta function (C.31) 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. (C.31). This function needs Function GAMMAL to calculate the logarithm of Gamma function, sub-routines ADPINT and KRONRD to evaluate the integral. Further, the function FBETA is needed to define the integrand for the integral.

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

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

**109. 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] will be the probability of having I or less events.

**110. 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] will be the probability of having I or less counts.

**111. 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.,  $N + 2$  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.

## C.10 Functional Approximations

**112. POLFIT** Function 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 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. If error estimates are not available then all SIG[I] can be set to one. A is an array of length  $M + 1$ , which will contain the coefficients of the orthogonal polynomials in the calculated fit. ALP and BETA are arrays of length  $M + 1$ , which will contain the coefficients  $\alpha_i$  and  $\beta_i$  as defined in recurrence relation for orthogonal polynomials.  $ALP[I] = \alpha_{I+1}$  and  $BETA[I] = \beta_I$ . Y is an array of length N, Y[I] will contain the calculated value of function at X[I] using the least squares fit. H is an array of length  $M + 1$  containing the  $\chi^2$  values. H[I] will contain the

$\chi^2$  using polynomial of degree I,

$$H[M] = \sum_{j=0}^{N-1} \left( \frac{Y[j] - F[j]}{\text{SIG}[j]} \right)^2. \quad (\text{C.37})$$

As explained in Section 10.2.2, this estimate of  $H[I]$  may have a significant roundoff error. GAM is an array of length  $M+1$ , which will contain the quantity  $\gamma_i$  for the orthogonal polynomials as defined in Section 10.2.2. Error status can be obtained from the returned value of POLFIT. A value of 601 implies that  $M \geq N$  or  $M < 0$ , in which case, no calculations are performed. POLFIT = 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 function POLEVL.

**113. POLEVL** Function to evaluate the approximating polynomial, and its derivatives using Clenshaw's recurrences. This function can be used to calculate the value of approximating polynomial at any point X, after the required coefficients  $a_j$ ,  $\alpha_j$  and  $\beta_j$  have been calculated by function POLFIT. M ( $> 0$ ) is the degree of polynomial. A, ALP and BETA are arrays of length  $M+1$ , with  $A[j] = a_j$ ,  $\text{ALP}[j] = \alpha_{j+1}$  and  $\text{BETA}[j] = \beta_j$ . These coefficients must be calculated before calling the function 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. The returned value of POLEVL is always zero.

**114. POLFIT1** Function 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 arrays of length N containing the data points and errors. F is an array of length  $N \times \text{NUM}$  containing the function values for each set of points.  $F[J][I]$  is the value of function in Jth set at  $X[I]$  and  $\text{SIG}[I]$  is the corresponding error. If error estimates are not available then all  $\text{SIG}[I]$  can be set to one. The second dimension of F must be N in the calling program. A is an array of length  $(M+1) \times \text{NUM}$ , which will contain the coefficients of the orthogonal polynomials in the calculated fit for each set. The second dimension of A must be  $M+1$  in the calling program. ALP and BETA are arrays of length  $M+1$ , which will contain the coefficients  $\alpha_i$  and  $\beta_i$  as defined in recurrence relation for orthogonal polynomials. GAM is an array of length  $M+1$ , which will contain the quantity  $\gamma_i$  for the orthogonal polynomials as defined in Section 10.2.2. Error status can be obtained from the returned value of POLFIT1. A value of 601 implies that  $M \geq N$  or  $M < 0$ , in which case, no calculations are performed. POLFIT1 = 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 function POLEVL.

**115. POLORT** Function to evaluate the orthogonal polynomials, and its derivatives. It may be noted that this function 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 function can be used to calculate the value of orthogonal polynomial basis functions at any point X, after the required coefficients  $\alpha_j$  and  $\beta_j$  have been calculated by function POLFIT or POLFIT1. M ( $> 0$ ) is the degree of polynomial. ALP and BETA are arrays of length M + 1, with  $ALP[j] = \alpha_{j+1}$  and  $BETA[j] = \beta_j$ . These coefficients must be calculated before calling the function POLORT. X is the value of the independent variable, at which the polynomials have to be evaluated. F, DF and DDF are arrays of length M + 1, which will contain the calculated values of the M + 1 orthogonal polynomials and its first and second derivatives, respectively at X.

**116. POLFIT2** Function 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 an array of length NX containing the points along X axis, while Y is an array of length NY containing the points along the Y axis. F is an array of length LA  $\times$  NY containing the function values. F[J][I] is the value at X[I], Y[J]. AX is an array of length IC  $\times$  3 containing information about the fit along X direction. AX[0][I], AX[1][I], AX[2][I] will respectively contain the coefficients  $\alpha_{i+1}, \beta_i, \gamma_i$  for the orthogonal polynomials. AY is an array of length IC  $\times$  3 containing information about the fit along Y direction. AY[0][I], AY[1][I], AY[2][I] will respectively contain the coefficients  $\alpha_{i+1}, \beta_i, \gamma_i$  for the orthogonal polynomials. The arrays AX and AY will be calculated by the function. LA is the second dimension of arrays F and FY as declared in the calling function, LA  $\geq \max(NX, NY)$ . C is an array of length IC  $\times$  (MY + 1) containing the fitted coefficients for the polynomial fit in two dimensions. The fitted polynomial would be

$$\sum_{i=0}^{MX} \sum_{j=0}^{MY} C[j][i] \phi_i(x) \psi_j(y), \quad (C.38)$$

where  $\phi_i(x)$  and  $\psi_j(y)$  are the orthogonal polynomials in  $x$  and  $y$  respectively. IC is the second dimension of arrays AX, AY and C as declared in the calling function, IC  $> \max(MX, MY)$ . MX is the required degree of polynomial in X. MY is the required degree of polynomial in Y. FY is an array of length LA  $\times$  NY containing the fitted values of the function at each of the tabular points. CHISQ is calculated value of  $\chi^2$  for the fit. Error status can be obtained from the returned value of POLFIT2. A value of 602 implies that IC  $< MX + 1$  or LA  $< \max(NX, NY)$ . POLFIT2 = 603 implies that NX  $\leq MX$ , NY  $\leq MY$ , MX  $< 0$  or 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 function POLEV2. This function requires functions POLFIT1, POLEV2 and POLORT.

**117. POLEV2** Function to evaluate the approximating polynomial, and its derivatives using expansion in orthogonal polynomials in 2 dimensions. This function 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 function POLFIT2. NX is the degree of polynomial in X, NY is the degree of polynomial in Y. AX is an array of length  $LA \times 2$  containing the coefficients  $\alpha_i$  and  $\beta_i$  for orthogonal polynomials in X. AX[0][I] contains  $\alpha_{i+1}$  and AX[1][I] contains  $\beta_i$ . AY is an array of length  $LA \times 2$  containing the coefficients  $\alpha_i$  and  $\beta_i$  for orthogonal polynomials in Y. AY[0][I] contains  $\alpha_{i+1}$  and AY[1][I] contains  $\beta_i$ . These coefficients must be calculated before calling the function POLEV2 using POLFIT2. LA is the second dimension of arrays AX, AY and WT in the calling function,  $LA > \max(NX, NY)$ . WT is an 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). Output parameters, DFX and DFY are respectively,  $\partial F/\partial x$  and  $\partial F/\partial y$ , while DFXX, DFX Y, DFYY are the second derivative  $\partial^2 F/\partial x^2$ ,  $\partial^2 F/\partial x \partial y$ ,  $\partial^2 F/\partial y^2$ . Error status can be obtained from the returned value of POLEV2. A value of 604 implies that  $\max(NX, NY) \geq LA$ , in which case no calculations are done. The function requires POLORT to calculate the orthogonal basis functions in one dimensions.

**118. POLFITN** Function 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 an array of length  $LA \times N$  containing the points along each axis, X[J][I] is the Ith point along the Jth dimension. F is an array of length  $NK[0] \times NK[1] \times \dots \times NK[N - 1]$  containing the function values. F is treated as a one dimensional array in this function and hence the dimensions of array in the calling function must exactly match the number of points in each dimension, e.g., F[NK[N-1]]...[NK[1]][NK[0]]. AX is an array of length  $LA \times (3N + 3)$  containing information about the fit along each direction. AX[3J][I], AX[3J+1][I], AX[3J+2][I] will respectively contain the coefficients  $\alpha_{i+1}, \beta_i, \gamma_i$  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 second dimension of arrays X and AX as declared in the calling function,  $LA \geq \max(NK[I])$ . C is an array of length  $(MK[0] + 1)(MK[1] + 1) \dots (MK[N - 1] + 1)$  containing the fitted coefficients for the polynomial fit in N dimensions. The fitted polynomial

would be

$$\sum_{i_0=0}^{MK[0]} \sum_{i_1=0}^{MK[1]} \cdots \sum_{i_{N-1}=0}^{MK[N-1]} C[i_{N-1}] \cdots [i_1][i_0] \phi_{i_0}(x_0) \phi_{i_1}(x_1) \cdots \phi_{i_{N-1}}(x_{N-1}), \quad (C.39)$$

where  $\phi_{i_j}(x_j)$  are the orthogonal polynomials along  $j$ th dimension. C is treated as a one dimensional array in this function and hence the dimensions of array in the calling function must exactly match the number of polynomials in each dimension, e.g.  $C[MK[N-1]+1] \cdots [MK[1]+1][MK[0]+1]$ . MK is an integer array of length N containing the required degree of polynomial in each direction. FY is an array of the same length and shape as F which will contain the fitted value of the function at each of the tabular points. CHISQ is calculated value of  $\chi^2$  for the fit. Error status can be obtained from the returned value of POLFITN. A value of 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 function POLEVN, POLEVN1 or POLEVN2. This function requires functions POLFIT1, POLEVN, POLORT.

**119. POLEVN** Function to evaluate the approximating polynomial using expansion in orthogonal polynomials in N dimensions. This function 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 function 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 an array of length  $LA \times (3N + 3)$  containing the coefficients  $\alpha_i$  and  $\beta_i$  for orthogonal polynomials in X.  $AX[3J][I]$  contains  $\alpha_{I+1}$  and  $AX[3J+1][I]$  contains  $\beta_I$  for orthogonal polynomials along Jth dimension. These coefficients must be calculated before calling the function POLEVN using POLFITN. LA is the second dimension of array AX in the calling function. This must be the same as what was used in call to POLFITN while calculating the coefficients. WT is an array of length  $(MK[0] + 1)(MK[1] + 1) \cdots (MK[N - 1] + 1)$ , containing the coefficients of the fit. This array is treated as one-dimensional array in the function and hence its dimensions in the calling function must exactly match the number of orthogonal polynomials in each dimension, for example,  $WT[MK[N-1]+1] \cdots [MK[1]+1][MK[0]+1]$ . X0 is an 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. The returned value is always zero. This function requires POLORT to calculate the orthogonal basis functions in one dimensions. This function does not calculate the derivatives of F. If first derivatives are required then one can use POLEVN1, while for second derivatives use POLEVN2.

**120. POLEVN1** Function to evaluate the approximating polynomial and its first derivative using expansion in orthogonal polynomials in N dimensions. This function 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 function POLFITN. This is the version of POLEVN which also calculates the first derivatives. The arguments are the same as those for POLEVN, except for array DF of length N, which will contain the first derivatives of F at X0. DF[i] will contain  $\partial F/\partial x_i$ . This function 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.

**121. POLEVN2** Function to evaluate the approximating polynomial and its first and second derivatives using expansion in orthogonal polynomials in N dimensions. This function 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 function POLFITN. This is the version of POLEVN which also calculates the first and second derivatives. The arguments are same as those of POLEVN, except for arrays DF and DDF. DF is an 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 an array of length  $N^2$  which will contain the second derivatives of F at X0. DDF[j][i] will contain  $\partial^2 F/\partial x_i \partial x_j$ . The second dimension of DDF in the calling function must be equal to N. This function 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.

**122. LLSQ** Function 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 an array of length  $IX \times N$  containing the coordinates of tabular points. X[J][I] is the Ith coordinate of Jth tabular point. IX is the second dimension of X in the calling function ( $IX \geq K$ ). 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 an array of length N, containing the function values. F[I] is the function value at  $(X[I][0], X[I][1], \dots, X[I][K-1])$ . This is treated as one dimensional array in the function and hence should not have any gaps in storage. This allows the function to be used for a general distribution of points not necessarily along a hyper-rectangular mesh. EF is an 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

$$\sum_{i=0}^{M-1} \frac{A[i]}{EF[j]} \phi_i(X[j][0], \dots, X[j][K-1]) = \frac{F[j]}{EF[j]}, \quad j = 0, 1, \dots, N-1; \quad (C.40)$$

using SVD. A is an 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 an array of length  $IU \times N$  which will contain the matrix  $U$  from SVD of the design matrix. V is an 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 second dimension of U in the calling function ( $IU \geq M$ ). IV is the second dimension of V and COV in the calling function ( $IV \geq M$ ). SIGMA is an array of length M, which will contain the singular values of the design matrix. Y is an array of length N which will contain the fitted values of the function at the tabular points. PHI is the name of the function 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  $\chi^2$  defined by

$$\chi^2 = \sum_{i=0}^{N-1} \left( \frac{F[i]}{EF[i]} - \sum_{j=0}^{M-1} \frac{A[j]}{EF[i]} \phi_j(X[i][0], \dots, X[i][K-1]) \right)^2. \quad (C.41)$$

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. Error status can be obtained from the returned value of LLSQ. A value of 606 implies that  $M > N$ ,  $M \leq 0$ ,  $N \leq 0$  or  $K > IX$ . LLSQ = 607 implies that EF[I] are not all positive. In both these cases no calculations are done. The function 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 an array of length K containing the coordinates of the point at which the basis functions need to be calculated. FX is an array of length M containing the calculated basis functions at X. FX[I] should give  $\phi_I(X)$ . For polynomial fits in one dimension  $\phi_i(x) = x^i$ , but in that case it may be better to use POLFIT to calculate the fit. Apart from PHI it also needs functions SVD and SVDEVL.

**123. BSPFIT** Function 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 an array of length N containing the coordinates of tabular points. F is an array of length N, containing the function values. F[I] is the function value at X[I]. EF is an 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

$$\sum_{i=0}^{NO+K-3} \frac{C[i]}{EF[j]} \phi_i(X[j]) = \frac{F[j]}{EF[j]}, \quad j = 0, 1, \dots, N-1; \quad (C.42)$$

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 an array of length  $LA \times 2N$  (when  $RLM > 0$ ) which will contain the matrix  $U$  from SVD of the design matrix. If  $RLM \leq 0$  then A can be an array of length  $LA \times N$ . V is an 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 second dimension of A in the calling function,  $LA \geq NO + K - 2$ . IV is the second dimension of V and COV in the calling function ( $IV \geq NO + K - 2$ ). SIGMA is an array of length  $NO + K - 2$ , which will contain the singular values of the design matrix. C is an 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 an 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[0] containing the first knot. This will yield  $(NO + K - 2)$  B-spline basis functions for fitting. Y is an 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  $IFLG \leq 1$  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  $IFLG \leq 0$  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. 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  $\lambda$  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  $\chi^2$  defined by

$$\chi^2 = \sum_{i=0}^{N-1} \left( \frac{F[i]}{EF[i]} - \sum_{j=0}^{NO+K-3} \frac{A[j]}{EF[i]} \phi_j(X[i]) \right)^2. \quad (C.43)$$

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 C[I] and C[J]. The diagonal elements are the variance in fitted parameters. Error status can be obtained from the returned value of BSPFIT. A value of 608 implies that  $NO + K - 2 > N$ , or  $K < 2$ . BSPFIT = 609 implies that  $RLM > 0$  but IDE is

not 1 or 2. BSPFIT = 610 implies that  $EF[I]$  are not all positive. In all these cases no calculations are done. Other values may be set by SVD or BSPLIN. This function requires functions BSPLIN, BSPEVL, SVD, and SVDEVL.

**124. BSPFIT2** Function 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 function 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 arrays of length NX, NY containing the coordinates of tabular points. F is an array of length LA  $\times$  NY, containing the function values.  $F[J][I]$  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 an array of length IV  $\times$  2NX which will contain the matrix  $U$  from SVD of the design matrix for fit along x-axis. AY is an array of length IV  $\times$  2NY which will contain the matrix  $U$  from SVD of the design matrix for fit along y-axis. These dimensions are when  $RLM > 0$ . For  $RLM \leq 0$  the required dimensions are IV  $\times$  NX and IV  $\times$  NY for AX and AY respectively. LA is the second dimension of arrays F, C and FY as declared in the calling function. All these arrays should have the same second dimension. LA must be at least  $2 \times \max(NX, NY)$  when  $RLM > 0$ , otherwise LA must be at least  $\max(NX, NY)$ . VX is an 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 an 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 second dimension of AX, AY, VX and VY in the calling function ( $IV \geq \max(MX, MY) + K - 2$ ). SIGMAX is an array of length MX + K - 2, which will contain the singular values of the design matrix for fit along the x-axis. SIGMAY is an array of length MY + K - 2, which will contain the singular values of the design matrix for fit along the y-axis. C is an array of length LA  $\times$  NY, which will contain the fitted coefficients. Although, there are only (MX + K - 2)(MY + K - 2) coefficients the array must have a larger length, since the remaining elements are used as scratch space. XF is an array of length MX containing the knots required to define the B-spline basis functions along  $x$ . YF is an 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[0], YF[0] containing the first knot. FY is an array of length LA  $\times$  NY which will contain the fitted values of the function at the tabular points. 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  $\lambda$  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  $\chi^2$  defined by

$$\chi^2 = \sum_{i=0}^{NX-1} \sum_{j=0}^{NY-1} \left( F[j][i] - \sum_{j1=0}^{MX+K-3} \sum_{j2=0}^{MY+K-3} A[j2][j1] \phi_{j1}(X[i]) \psi_{j2}(Y[j]) \right)^2. \quad (C.44)$$

Here  $\phi_j(x)$  are the basis functions in  $x$  and  $\psi_j(y)$  are those in  $y$ . Error status can be obtained from the returned value of BSPFIT2. A value of 608 implies that  $MX + K - 2 > NX$ ,  $MY + K - 2 > NY$ , or  $K < 2$ . BSPFIT2 = 609 implies that  $RLM > 0$  but IDE is not 1 or 2. In all these cases no calculations are done. Other values may be set by BSPFIT, SVD or BSPLIN. This function requires functions BSPFIT, BSPLIN, BSPEVL, BSPEV2, SVD and SVDEVL.

**125. BSPFITW2** Function 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 function 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 arrays of length NX, NY containing the coordinates of tabular points. F is an array of length IC  $\times$  NY, containing the function values. F[J][I] is the function value at (X[I], Y[J]). EF is an array of length IC  $\times$  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 an array of length LA  $\times$  3NX  $\times$  NY which will contain the matrix  $U$  from SVD of the design matrix. If  $RLM \leq 0$  then the size of array A needs to be only LA  $\times$  NX  $\times$  NY. LA is the second dimension of array A as declared in the calling function. LA must be at least  $(MX + K - 2)(MY + K - 2)$ . V is an array of length IV  $\times$   $(MX + K - 2)(MY + K - 2)$  which will contain the matrix  $V$  from SVD of the design matrix. IV is the second dimension of V in the calling function,  $IV \geq (MX + K - 2)(MY + K - 2)$ . SIGMA is an array of length  $(MX + K - 2)(MY + K - 2)$ , which will contain the singular values of the design matrix. C is an array of length IC  $\times$   $(MY + K - 2)$ , which will contain the fitted coefficients. IC is the second dimension of arrays C, F, EF, FY as declared in the calling function, IC must be at least NX. XF is an array of length MX containing the knots required to define the B-spline basis functions

along  $x$ . YF is an 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[0], YF[0] containing the first knot. FY is an array of length IC  $\times$  NY which will contain the fitted values of the function at the tabular points. 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  $\lambda$  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 3 times larger. CHISQ is the minimum value of  $\chi^2$  defined by

$$\sum_{i=0}^{NX-1} \sum_{j=0}^{NY-1} \left( \frac{F[j][i] - \sum_{j1=0}^{MX+K-3} \sum_{j2=0}^{MY+K-3} A[j2][j1] \phi_{j1}(X[i]) \psi_{j2}(Y[j])}{EF[j][i]} \right)^2. \quad (C.45)$$

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. Error status can be obtained from the returned value of BSPFITW2. A value of 608 implies that  $MX + K - 2 > NX$ ,  $MY + K - 2 > NY$ , or  $K < 2$ . BSPFITW2 = 609 implies that  $RLM > 0$  but IDE is not 1 or 2. BSPFITW2 = 610 implies that  $EF[J][I]$  are not all positive. In all these cases no calculations are done. Other values may be set by SVD or BSPLIN. This function requires functions BSPLIN, BSPEV2, SVD and SVDEVL.

**126. BSPFITN** Function 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 function 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 an array of length  $LA \times N$  containing the coordinates of tabular points. X[J][I] is the  $I$ th point along  $J$ th axis. F is an array of length  $NK[0] \times NK[1] \times \dots \times NK[N-1]$ , containing the function values.  $F[i_{N-1}] \dots [i_1][i_0]$  is the function value at  $(X[0][i_0], X[1][i_1], \dots, X[N-1][i_{N-1}])$ . This is treated as a one-dimensional array and hence its dimensions in the calling function must exactly match the size in each dimension, e.g., the dimensions could be

$F[NK[N-1]] \dots [NK[1]][NK[0]]$ .  $K$  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 an array of length  $IV \times LA \times N$  which will contain the matrix  $U$  from SVD of the design matrix for fit along each axis.  $LA$  is the second dimension of array  $X$ , as declared in the calling function. The second dimension of  $A$  should be  $LA \times IV$ .  $LA$  must be at least  $2 \times \max(NK[I])$  when  $RLM > 0$ , otherwise  $LA$  must be at least  $\max(NK[I])$ .  $V$  is an array of length  $IV^2 \times N$  which will contain the matrix  $V$  from SVD of the design matrix for fit along each axis.  $IV$  is the second dimension of  $XF$  and  $SIGMA$  in the calling function ( $IV \geq K - 2 + \max MK[I]$ ). The second dimension of  $V$  in calling function is  $IV^2$ .  $SIGMA$  is an array of length  $IV \times N$ , which will contain the singular values of the design matrix for fit along each axis.  $C$  is an array of length  $NK[0] \times NK[1] \times \dots \times NK[N-1]$ , which will contain the fitted coefficients. If  $RLM > 0$  this length must be 2 times larger. Note that although there are only  $(MK[0] + K - 2)(MK[1] + K - 2) \dots (MK[N-1] + K - 2)$  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 function must exactly match the size in each dimension, e.g., the dimensions could be

$$C[NX][MK[N-2] + K - 2] \dots [MK[1] + K - 2][MK[0] + K - 2]. \quad (C.46)$$

Here the first dimension has to be increased suitably to accommodate the scratch space required. The first 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 functions to calculate the function value at any required point, then the exact dimensions in calling function are immaterial as long as the total length is larger than the required value.  $XF$  is an 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.  $XF[J][I]$  is the  $I$ th knot along  $J$ th dimension.  $MK$  is an integer array of length  $N$  containing the number of knots for defining B-splines along each axis.  $FY$  is an array of same size and shape as  $F$  which will contain the fitted values of the function at the tabular points.  $REPS$  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  $\lambda$  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  $\chi^2$  obtained using the fitted coefficients. Error status can be obtained from the returned value of `BSPFITN`. A value of 609 implies

that  $RLM > 0$  but IDE is not 1 or 2. In this case no calculations are done. Other values may be set by BSPFIT, SVD or BSPLIN. This function requires functions BSPFIT, BSPLIN, BSPEVL, BSPEVN, SVD and SVDEVL.

**127. BSPFITWN** Function 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 an array of length  $LA \times N$  containing the coordinates of tabular points.  $X[J][I]$  is the  $I$ th point along  $J$ th axis.  $F$  is an array of length  $NK[0] \times NK[1] \times \dots \times NK[N-1]$ , containing the function values.  $F[i_{N-1}] \dots [i_1][i_0]$  is the function value at  $(X[0][i_0], X[1][i_1], \dots, X[N-1][i_{N-1}])$ . This is treated as an one-dimensional array and hence its dimensions in the calling function must exactly match the size in each dimension, e.g., the dimensions could be  $F[NK[N-1]] \dots [NK[1]][NK[0]]$ .  $EF$  is an 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 an array of length

$$NK[0]NK[1] \dots NK[N-1] \\ \times (MK[0] + K - 2)(MK[1] + K - 2) \dots (MK[N-1] + K - 2), \quad (C.47)$$

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 second dimension of array  $X$ , as declared in the calling function.  $LA$  must be at least  $\max(NK[I])$ .  $V$  is an array of length

$$[(MK[0] + K - 2)(MK[1] + K - 2) \dots (MK[N-1] + K - 2)]^2, \quad (C.48)$$

which will contain the matrix  $V$  from SVD of the design matrix.  $IV$  is the second dimension of  $XF$  in the calling function ( $IV \geq \max(MK[I])$ ).  $SIGMA$  is an array of length  $(MK[0] + K - 2)(MK[1] + K - 2) \dots (MK[N-1] + K - 2)$ , which will contain the singular values of the design matrix.  $C$  is an array of length  $NK[0] \times NK[1] \times \dots \times NK[N-1]$ , which will contain the fitted coefficients. If  $RLM > 0$  this length must be  $N + 1$  times larger. Note that although there are only  $(MK[0] + K - 2)(MK[1] + K - 2) \dots (MK[N-1] + K - 2)$  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 function must exactly match the size in each dimension, e.g., the dimensions could be

$$C[NX][MK[N-2] + K - 2] \dots [MK[1] + K - 2][MK[0] + K - 2].$$

Here the first dimension has to be increased suitably to accommodate the scratch space required. The first 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 functions to calculate the function value at any required point, then the exact dimensions in calling function are immaterial as long as the total length is larger than the required value. XF is an 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.  $XF[J][I]$  is the  $I$ th knot along  $J$ th dimension. MK is an integer array of length  $N$  containing the number of knots for defining B-splines along each axis. FY is an array of same size and shape as F which will contain the fitted values of the function at the tabular points. 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  $\lambda$  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  $\chi^2$  obtained using the fitted coefficients. Error status can be obtained from the returned value of BSPFITWN. A value of 608 implies that LA or IV are not large enough to store the required quantities. BSPFITWN = 609 implies that RLM > 0 but IDE is not 1 or 2. In these cases no calculations are done. Other values may be set by SVD or BSPLIN. This function requires functions BSPLIN, BSPEVN, SVD and SVDEVL.

**128. 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  $\chi^2$  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.

**129. NLLSQ** Function to calculate the  $\chi^2$  function for a nonlinear least squares fit for use with function BFGS. By suppressing the gradient calculations it can also be used with function NMINF. This is the function to be minimised by BFGS. N is the number of parameters to be fitted, A is an 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 an array of length N, containing the calculated derivatives. G[i] will contain  $\frac{\partial F}{\partial a_i}$ . The data points are passed through global variables NNLSQ, FXLSQ, XLSQ, EFLSQ, FX1LSQ, which must be initialised in the calling function before calling BFGS. Here, XLSQ, FXLSQ, EFLSQ are arrays of length NPL containing respectively, the values of abscissas, function values and estimated errors in function values at each point. NNLSQ is the number of data points in the table,  $NNLSQ \leq NPL$ . FX1LSQ is an array of length NP which will contain the fitted value of the function at all points. Although EFLSQ[I] should contain the estimated error in FXLSQ[I], in many cases it is found that multiplying all elements of EFLSQ by suitable constant improves the convergence of BFGS significantly without changing the minimum. Only the value of  $\chi^2$  will need to be scaled suitably. This function 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 function to this function and hence the name has to be explicitly changed in the function. Function FCN(N, A, X, F, DF) must be supplied by the user. Here N is the number of parameters, A is an array of length N containing the values of parameters. X specifies the point where the function value needs to be calculated. F should contain the calculated function value F(X, A), while DF is an 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 function NMINF. The corresponding version may be found in NLLSQ\_F.

**130. DFT** Function to calculate the discrete Fourier transform (DFT) using normal sum. This function 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 an array of length 2N, which should contain the data points. CF is an array of length 2N which will contain the Fourier transform of CG. Both CF and CG are complex and hence the arrays should contain the real and imaginary parts stored consecutively. IFLG is a flag, if  $IFLG \geq 0$  the DFT is calculated, while if  $IFLG < 0$  the inverse DFT will be calculated. Error status can be obtained from the returned value of DFT. A value of 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 function 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.

**131. FFT** Function 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 an array of length 2N, which should contain the data points when calling the function. 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 function. CG is complex and hence both real and



imaginary parts must be stored in the array in consecutive locations. IFLG is a flag, if  $IFLG \geq 0$  the DFT is calculated, while if  $IFLG < 0$  the inverse DFT will be calculated. Error status can be obtained from the returned value of FFT. A value of 611 implies that  $N < 2$ , in which case, no calculations are performed.  $DFT = 631$  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 function does not perform the division.

**132. FFTR** Function 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 an array of length  $2 \times (N/2)$ , which should contain the data points when calling the function. In the calling function it may be treated as a real array of length  $N$  stored in normal order. After execution, the DFT will be overwritten on the same array CG as explained in Section 10.6. The DFT will be complex and it will contain the real and imaginary parts stored in consecutive memory locations. If necessary, another copy of the original data should be preserved for later use before calling the function. IFLG is a flag. If  $IFLG \geq 0$  the DFT is calculated, while if  $IFLG < 0$  the inverse DFT will be calculated. Error status can be obtained from the returned value of FFTR. A value of 611 implies that  $N < 4$  and no calculations are done.  $FFTR = 631$  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 function does not perform the division. This function requires function FFT for calculating the DFT of complex data and function CDIV to perform complex division.

**133. FFTN** Function 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  $I$ th coordinate, which must be equal to a power of 2. CG is a complex array of length  $NN[0] \times NN[1] \times \dots \times NN[ND-1]$ , which should contain the data points when calling the function. The data should be stored in the normal Fortran order with

$$\begin{aligned} &CG[j_0 + j_1 NN[0] + j_2 NN[0] NN[1] + \dots + j_{ND-1} NN[0] \dots NN[ND-2]] \\ &= g_{j_0, j_1, \dots, j_{ND-1}}, \end{aligned} \tag{C.49}$$

for  $0 \leq j_r < NN[r] - 1$  and  $0 \leq r < ND$ . In fact, in the calling function CG can be treated as a  $(ND + 1)$ -dimensional array with dimension  $CG[NN[ND-1]] \dots [NN[1]][NN[0]][2]$ . Both real and imaginary parts of data needs to be stored in consecutive locations. 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 function. IFLG is a flag. If  $\text{IFLG} \geq 0$  the DFT is calculated, while if  $\text{IFLG} < 0$  the inverse DFT will be calculated. Error status can be obtained from the returned value of FFTN. A value of 631 implies that at least one of the  $\text{NN}[\text{I}]$ , ( $\text{I} = 0, 1, \dots, \text{ND}-1$ ) 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  $\text{NN}[0] \times \text{NN}[1] \times \dots \times \text{NN}[\text{ND}-1]$  to match the original data. The function does not perform the division.

**134. LAPINV** Function 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 an 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  $t_N$  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 an array of length N, which will contain the value of the required function at  $t_i$  after execution of the function. 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 function, or by removing the discontinuity which is causing the slow convergence. The second alternative will be more effective. Error status can be obtained from the returned value of LAPINV. A value of 61 implies that the  $\epsilon$ -algorithm failed to converge for at least one of the points. LAPINV = 62 implies that the  $\epsilon$ -algorithm encountered a zero denominator at some stage. Since only the last value will be retained, the error flag may be misleading in those cases, where failure has occurred at more than one points. Function CFS(CS,CF) must be supplied by the user. Here both CS and CF are arrays of length 2 containing the real and imaginary parts of the arguments. CF is  $f(\text{CS})$ .

**135. POLD** Function to evaluate a polynomial and its derivatives at any point. N is the degree of polynomial. A is an array of length  $N + 1$  containing the coefficients of the polynomial. A[0] should contain the constant term and A[N] should be the coefficient of  $X^N$ . 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 an array of length ND, which will contain the computed values of the derivatives. PD[I-1] will contain the Ith derivative of polynomial. The value of polynomial is returned as POLD.

**136. RMK** To evaluate a rational function at any point. M is the degree of numerator, while K is the degree of denominator. A is an array of length  $M + 1$  containing the coefficients of the polynomial in the numerator. A[0] should contain the constant term and A[M] should be the coefficient of  $X^M$ .

B is an array of length  $K + 1$  containing the coefficients of the polynomial in the denominator. B[0] should contain the constant term and B[K] 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.

**137. RMK1** 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 an array of length  $M + 1$  containing the coefficients of the polynomial in the numerator. A[0] should contain the constant term and A[M] should be the coefficient of  $X^M$ . B is an array of length K containing the coefficients of the polynomial in the denominator. B[I-1] 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.

**138. RMKD** 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 an array of length  $M + 1$  containing the coefficients of the polynomial in the numerator. A[0] should contain the constant term and A[M] should be the coefficient of  $X^M$ . B is an array of length  $K + 1$  containing the coefficients of the polynomial in the denominator. B[0] should contain the constant term and B[K] 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.

**139. RMKD1** 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 an array of length  $M + 1$  containing the coefficients of the polynomial in the numerator. A[0] should contain the constant term and A[M] should be the coefficient of  $X^M$ . B is an array of length K containing the coefficients of the polynomial in the denominator. B[I-1] 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.

**140. PADE** Function 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 an array of length  $M + K + 1$ , which will contain the coefficients of the Padé approximation. A[i], ( $i = 0, \dots, K - 1$ ) is the coefficient of  $x^{i+1}$  in the denominator, the constant term being assumed to be unity, A[K + i], ( $i = 0, 1, \dots, M$ ) is the coefficient of  $x^i$  in the numerator, which gives the Padé approximation

$$R_{MK}(x) = \frac{A[K] + A[K+1]x + A[K+2]x^2 + \dots + A[K+M]x^M}{1 + A[0]x + A[1]x^2 + \dots + A[K-1]x^K}, \quad (C.50)$$

C is an array of length  $M + K + 1$  containing the coefficients of the Maclaurin series for the required function. C[i] should contain the coefficient of  $x^i$  in the

Maclaurin series. These coefficients must be supplied by the user. Error status can be obtained from the returned value of PADE. A value of 612 implies that either  $M < 0$  or  $K < 0$ , in which case no calculations are performed. Other values may be set by the function GAUELM, which is called to solve the system of linear equations. This function requires function GAUELM to solve the system of linear equations.

**141. CHEBCF** Function to convert a power series into a series of Chebyshev polynomials and vice versa.  $N$  is the degree of polynomial.  $C$  and  $P$  are arrays of length  $N + 2$  containing respectively, the coefficients of the Chebyshev and the power series expansions.  $C[i]$  is the coefficient of  $T_i(x)$  in Chebyshev expansion, while  $P[i]$  is the coefficient of  $x^i$  in the power series. It should be noted that  $C[0]$  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 function. The returned value of the function is always zero.

**142. CHEBEX** Function 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.

$$FUN(x) = \frac{1}{2}c_0 + \sum_{i=1}^{N-1} c_i T_i(x). \quad (C.51)$$

$N$  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 an array of length  $N$ , which will contain the computed coefficients.  $FUN$  is the name of the function routine supplied to calculate the required function. Error status can be obtained from the returned value of CHEBEX. A value of 613 implies that  $N < 10$ , in which case no calculations are done. Function  $FUN(X)$  must be supplied by the user.

**143. CHEBAP** Function 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 function 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 an array of length  $M + K + 1$ , which will contain the coefficients of the rational function approximation.  $A[i - 1]$ , ( $i = 1, \dots, K$ ) is the coefficient of  $T_i(x)$  in the denominator, the constant term being assumed to be unity,  $A[K + i]$ , ( $i = 0, 1, \dots, M$ ) is the coefficient of  $T_i(x)$  in the numerator, which gives the approximation

$$R_{MK}(x) = \frac{A[K] + A[K + 1]T_1(x) + A[K + 2]T_2(x) + \dots + A[K + M]T_M(x)}{1 + A[0]T_1(x) + A[1]T_2(x) + \dots + A[K - 1]T_K(x)} \quad (\text{C.52})$$

C is an array of length  $M + 2K + 1$ , containing coefficients of the Chebyshev series for the required function.  $C[i]$  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  $C[0]/2$ . Error status can be obtained from the returned value of CHEBAP. A value of 612 implies that either  $M < 0$  or  $K < 0$ , in which case, no calculations are performed. Other values may be set by the function GAUELM, which is called to solve the system of linear equations. This function requires function GAUELM to solve the system of linear equations.

**144. REMES** Function 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  $3(M + K + 1)$ , 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 an array of length  $M + K + 2$ , which will contain the coefficients in the same form as that given by the function PADE.  $A[i - 1]$ , ( $i = 1, 2, \dots, K$ ) is the coefficient of  $x^i$  in the denominator, the constant term being unity, while  $A[K + i]$ , ( $i = 0, 1, \dots, M$ ) 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 function, since the function itself selects a uniform mesh and calculates the value of the function at the required points. EX is an array of length  $M + K + 5$  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  $M + K + 2$ , if the error curve is of the standard form. Some types of nonstandard error curves can be handled by this function (Example 10.11). If IFLG = 2, then the value of IE must be supplied. EMAX is the maximum error in the final approximation calculated by the function. 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  $T_{M+K+1}(x)$ . 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 ( $\geq M + K + 2$ ) 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  $M + K + 2$  extrema, where the error is expected to be the largest and is alternating in sign. Error status can be obtained from the returned value of REMES. A value of 614 implies that either  $M < 0$ , or  $K < 0$ , or  $XU \leq XL$ , or  $M + K + 2 > NMAXR$ , in which case, no calculations are performed. The last requirement arises because of the dimensions of global array AA, which is used to transfer the coefficients to an auxiliary function routine for finding extrema of the function. REMES = 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. REMES = 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 may be set by the function BRENTM which is called to find the extrema, or the function GAUELM which is called to solve the system of linear equations. This function requires the functions BRENTM, GAUELM, FM, FUN and FUND. Function FUN(X) calculates the required function, while Function FUND(X) calculates the weight function. REMES generates approximation of the form  $FUN(X) \approx FUND(X)R_{MK}(X)$ . If  $FUND(x) = 1$  and  $FUN(x) = f(x)$ , then the function will calculate minimax approximation to  $f(x)$  with respect to the absolute error, while if  $FUND(x) = 1/f(x)$  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 functions FUN and FUND are fixed and cannot be changed. The functions FUN and FUND must be supplied by the user. The global variables MM, KK, AA, SI are used to pass on parameters to function FM(X), which calculates the function to be minimised. Here AA is an array of length NMAXR which contains the coefficients of rational function approximation. SI is a 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  $\text{RMK1}(\text{M}, \text{K}, \&\text{A}[\text{K}], \text{A}, \text{X})$ .

**145. FM** Function to be used with REMES. This function 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  $\text{SI} \geq 10$ , in which case, function value is not calculated, since it is already known.

**146. GAMMA** Function to calculate the Gamma function for a real argument

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt, \quad (x > 0). \quad (\text{C.53})$$

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

$$-x\Gamma(-x)\Gamma(x) = \frac{\pi}{\sin(\pi x)}. \quad (\text{C.54})$$

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

$$\log \Gamma(x) = (x - \frac{1}{2}) \log x - x + \log(\sqrt{2\pi}) + \frac{1}{x} R_{mk}(1/x^2), \quad (\text{C.55})$$

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 function 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 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  $|\Gamma(x)|$ .

**147. GAMMAL** Function 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.

**148. ERF** Function to calculate the Error function

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt, \quad (\text{C.56})$$

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

$$\begin{aligned} \text{erf}(x) &\approx x R_{mk}(x^2), & (0 \leq x < 2); \\ \text{erf}(x) &\approx 1 - \frac{e^{-x^2}}{x} R'_{m'k'}(\frac{1}{x^2}), & (2 \leq x < \infty). \end{aligned} \quad (\text{C.57})$$

While for  $x < 0$  it uses  $\operatorname{erf}(-x) = -\operatorname{erf}(x)$  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.

**149. ERFC** Function to calculate the complementary Error function

$$\operatorname{erfc}(x) = 1 - \operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt, \quad (\text{C.58})$$

for a real argument. This function 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.

**150. BJ0** Function 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

$$J_0(x) = \sqrt{\frac{2}{\pi x}} \left( P_0(x) \cos\left(x - \frac{\pi}{4}\right) - Q_0(x) \sin\left(x - \frac{\pi}{4}\right) \right). \quad (\text{C.59})$$

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

$$P_0(x) \approx R_{mk}(1/x^2); \quad Q_0(x) \approx \frac{1}{x} R_{m'k'}(1/x^2). \quad (\text{C.60})$$

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  $J_0(x)$ .

**151. BJ1** Function 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  $J_1(x) \approx x R_{mk}(x^2)$ , while for larger arguments it uses the asymptotic form

$$J_1(x) = \sqrt{\frac{2}{\pi x}} \left( P_1(x) \cos\left(x - \frac{3\pi}{4}\right) - Q_1(x) \sin\left(x - \frac{3\pi}{4}\right) \right). \quad (\text{C.61})$$

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

$$P_1(x) \approx R_{mk}(1/x^2); \quad Q_1(x) \approx \frac{1}{x} R_{m'k'}(1/x^2). \quad (\text{C.62})$$

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  $J_1(x)$ .

**152. BJN** Function to calculate the Bessel function of the first kind of integral order,  $J_n(x)$  for a real argument. This function 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  $k = 0, 1, \dots, N$  will be calculated. For negative  $N$ ,  $J_k(x)$  with negative  $k = 0, -1, \dots, -N$  are calculated.  $XB$  is the argument at which the function values are required.  $BJ$  is an array which will contain the computed values of the Bessel functions.  $BJ[i] = J_i(XB)$ . Since this array is also used as scratch space the length of the array must be at least,  $|N| + 16 + \max(25, 5\sqrt{|N|})$ . For  $|N| < |XB|$  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

$$J_{k-1}(x) = \frac{2k}{x} J_k(x) - J_{k+1}(x), \quad k = n_1 - 1, \dots, 2, 1. \quad (C.63)$$

The normalisation is then found using the relation

$$J_0(x) + 2 \sum_{k=1}^{\infty} J_{2k}(x) = 1. \quad (C.64)$$

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  $N_1$  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  $J_1(x)$ .

**153. BY0** Function 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 function 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

$$Y_0(x) = \frac{2}{\pi} \{ J_0(x)(\log(x/2) + \gamma) + x^2 R_{mk}(x^2) \}, \quad (\text{C.65})$$

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

$$Y_0(x) = \sqrt{\frac{2}{\pi x}} \left( P_0(x) \sin\left(x - \frac{\pi}{4}\right) + Q_0(x) \cos\left(x - \frac{\pi}{4}\right) \right). \quad (\text{C.66})$$

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

$$P_0(x) \approx R_{mk}(1/x^2); \quad Q_0(x) \approx \frac{1}{x} R_{m'k'}(1/x^2). \quad (\text{C.67})$$

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  $Y_0(x)$ .

**154. BJY0** Function 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 function 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 function 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 function is essentially same as BY0, except that in this case the function values have to appear as arguments while calling the function. 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  $J_0(\text{XB})$  and  $Y_0(\text{XB})$ .

**155. BY1** Function 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 function 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

$$Y_1(x) = \frac{2}{\pi} \left\{ J_1(x)(\log(x/2) + \gamma) - \frac{1}{x} - xR_{mk}(x^2) \right\}, \quad (\text{C.68})$$

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

$$Y_1(x) = \sqrt{\frac{2}{\pi x}} \left( P_1(x) \sin\left(x - \frac{3\pi}{4}\right) + Q_1(x) \cos\left(x - \frac{3\pi}{4}\right) \right). \quad (\text{C.69})$$

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

$$P_1(x) \approx R_{mk}(1/x^2); \quad Q_1(x) \approx \frac{1}{x} R_{m'k'}(1/x^2). \quad (\text{C.70})$$

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  $Y_1(x)$ .

**156. BJY1** Function 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 function 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 function 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 function is essentially same as BY1, except that in this case the function values have to appear as arguments while calling the function. 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  $J_1(\text{XB})$  and  $Y_1(\text{XB})$ .

**157. BYN** Function to calculate the Bessel function of the second kind of integral order,  $Y_n(x)$  for a real argument. This function calculates  $Y_k(x)$  for  $k$  between 0 and  $n$ . This function is not defined for  $x \leq 0$ , and the function 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  $k = 0, 1, \dots, N$  will be calculated. For negative  $N$ ,  $Y_k(x)$  with negative  $k = 0, -1, \dots, -N$  are calculated.  $XB$  is the argument at which the function values are required.  $BY$  is an array of length  $|N| + 1$  which will contain the computed values of the Bessel functions.  $BY[i] = Y_i(XB)$ . Since the recurrence relation for  $Y_n(x)$

$$Y_{n+1}(x) = \frac{2n}{x}Y_n(x) - Y_{n-1}(x), \quad (C.71)$$

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 function needs  $BY0$  and  $BY1$  to calculate  $Y_0(x)$  and  $Y_1(x)$ .

**158. SPHBJN** Function to calculate the spherical Bessel function of integral order,

$$j_n(x) = \sqrt{\frac{\pi}{2x}} J_{n+1/2}(x), \quad (C.72)$$

for a real argument. This function 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  $k = 0, 1, \dots, N$  will be calculated. For negative  $N$ ,  $j_k(x)$  with negative  $k = 0, -1, \dots, -N$  are calculated.  $XB$  is the argument at which the function values are required.  $BJ$  is an array which will contain the computed values of the Bessel functions.  $BJ[i] = j_i(XB)$ . Since this array is also used as scratch space the length of the array must be at least,  $|N| + 16 + \max(25, 5\sqrt{|N|})$ . 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

$$j_0(x) = \frac{\sin x}{x}; \quad j_1(x) = \frac{\sin x}{x^2} - \frac{\cos x}{x} \quad \text{or} \quad j_{-1}(x) = \frac{\cos x}{x}. \quad (C.73)$$

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

$$j_{k-1}(x) = \frac{2k+1}{x}j_k(x) - j_{k+1}(x), \quad k = n_1 - 1, \dots, 2, 1. \quad (C.74)$$

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.

**159. BI0** Function 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

$$I_0(x) = \frac{e^{|x|}}{\sqrt{|x|}} R_{mk}(1/|x|). \quad (\text{C.75})$$

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.

**160. BI1** Function 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  $I_1(x) \approx x R_{mk}(x^2)$ , while for larger arguments it uses the asymptotic form

$$I_1(x) = \frac{e^{|x|}}{\sqrt{|x|}} R_{mk}(1/|x|). \quad (\text{C.76})$$

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.

**161. BIN** Function to calculate the modified Bessel function of the first kind of positive integral order,  $I_n(x)$  for a real argument. This function 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  $k = 0, 1, \dots, N$  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 an array which will contain the computed values of the Bessel functions.  $\text{BI}[i] = I_i(\text{XB})$ . Since this array is also used as scratch

space the length of the array must be at least,  $N + 16 + \max(25, 5\sqrt{N})$ . For  $N < XB - 10$  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

$$I_{k-1}(x) = \frac{2k}{x} I_k(x) + I_{k+1}(x), \quad k = n_1 - 1, \dots, 2, 1. \quad (\text{C.77})$$

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  $I_1(x)$ .

**162. BK0** Function 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 function 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

$$K_0(x) = -I_0(x)(\gamma + \log(x/2)) + x^2 R_{mk}(x^2), \quad (\text{C.78})$$

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

$$K_0(x) = \frac{e^{-x}}{\sqrt{x}} R_{mk}(1/x). \quad (\text{C.79})$$

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 when different terms are combined.

**163. BK1** Function 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 function 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

$$K_1(x) = I_1(x)(\gamma + \log(x/2)) + \frac{1}{x} - xR_{mk}(x^2), \quad (\text{C.80})$$

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

$$K_1(x) = \frac{e^{-x}}{\sqrt{x}} R_{mk}(1/x). \quad (\text{C.81})$$

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.

**164. BKN** Function to calculate the modified Bessel function of the second kind of positive integral order,  $K_n(x)$  for a real argument. This function calculates  $K_j(x)$  for  $j$  between 0 and  $n$ . This function is not defined for  $x \leq 0$ , and no calculations are done but no warning is issued. 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  $j = 0, 1, \dots, N$  will be calculated. N must be positive, otherwise |N| will be used. X is the argument at which the function values are required. BK is an array of length N + 1 which will contain the computed values of the Bessel functions.  $BK[i] = K_i(XB)$ . 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 function needs BK0 and BK1 to calculate  $K_0(x)$  and  $K_1(x)$ .

**165. DAWSON** Function to calculate the Dawson's integral

$$D(x) = e^{-x^2} \int_0^x e^{t^2} dt, \quad (\text{C.82})$$

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  $10^{-15}$ .

**166. FERMM05** Function to calculate the Fermi integrals

$$F_k(x) = \int_0^\infty \frac{t^k dt}{e^{t-x} + 1}, \quad (\text{C.83})$$

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  $10^{-15}$ .

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

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

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

**170. PLEG** Function 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 an array of length L + 1, which will contain the computed values of the Legendre polynomials for all orders up to L,  $P[j] = P_j(X)$ . The recurrence relation is used to compute the values.

**171. PLM** Function to calculate associated Legendre functions,  $P_l^m(x)$  for a specified value of  $x$ . L, M define the order of Legendre functions,  $L > 0$  and  $|M| \leq L$ . 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 an array of length L + 1, which will contain the computed values of the associated Legendre functions.  $P[j] = P_j^M(X)$  for  $j \geq M$ . The function first computes the value of  $P_M^M$ . For positive  $m$ ,

$$P_m^m = \prod_{i=1}^m (2i-1)(1-x^2)^{m/2}. \quad (\text{C.84})$$

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

$$(n-m)P_n^m(x) = (2n-1)xP_{n-1}^m(x) - (n-1+m)P_{n-2}^m(x), \quad (\text{C.85})$$

is used to compute the value of  $P_L^M$ .



**172. YLM** Function to calculate the Spherical harmonic,

$$Y_l^m(\theta, \phi) = (-1)^m c_{lm} P_l^m(\cos \theta) e^{im\phi}, \quad (\text{C.86})$$

where

$$c_{lm}^2 = \frac{(2l+1)(l-m)!}{4\pi(l+m)!}, \quad (\text{C.87})$$

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.  $L > 0$ ,  $|M| \leq L$ .  $\theta$  and  $\phi$  are real variables specifying the angular coordinates  $\theta$  and  $\phi$  respectively.  $Y$  is an array of length 2 containing the real and imaginary parts of spherical harmonic. It is possible to use the argument  $x = \cos \theta$  instead of  $\theta$ . This function requires PLM to compute the associated Legendre functions.  $YLM\_X$  is the version of  $YLM$  with  $x = \cos \theta$  as the argument.

**173. MINMAX** Function 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 an array of length  $M + K + 2$ , which will contain the coefficients in the same form as that given by the function PADE.  $A[i-1]$ , ( $i = 1, 2, \dots, K$ ) is the coefficient of  $x^i$  in the denominator, the constant term being unity, while  $A[K+i]$ , ( $i = 0, 1, \dots, M$ ) is the coefficient of  $x^i$  in the numerator. At the time of calling the function, 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 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 function.  $EPS$  is the required tolerance, the iteration is continued until the change in  $EMAX$  is less than  $EPS$ . Error status can be obtained from the returned value of  $MINMAX$ . A value of 615 implies that either  $M < 0$ , or  $K < 0$ , or  $M + K + 2 > N$ , in which case, no calculations are performed.  $MINMAX = 634$  implies that the iteration failed to converge to the specified accuracy. Other values may be set by the function  $SIMPX$ . This function requires the function  $SIMPX$  to solve the LP problem.

**174. POLYL1** Function 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 an array of length  $M + 2$ , which will contain the coefficients.  $A[i]$ , ( $i = 0, 1, 2, \dots, M$ ) is the coefficient of  $x^i$ .  $X$  and  $F$  are 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 function.  $EPS$  is the expected level of

roundoff error, this parameter is passed on to the function SIMPL1 for simplex iteration. It is used to decide the sign of cost coefficients and pivot elements. Error status can be obtained from the returned value of POLYL1. A value of 616 implies that either  $M < 0$ , or  $M + 2 > N$ , in which case no calculations are performed. Other values may be set by the function SIMPL1. This function requires the function SIMPL1 to solve the LP problem using a slightly modified simplex algorithm.

**175. LINL1** Function to calculate a general linear  $L_1$ -approximation in terms of arbitrary basis functions for discrete data. Approximation of the form

$$f(x) \approx \sum_{i=0}^{M-1} a_i \phi_i(x), \quad (\text{C.88})$$

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 function 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 an array of length  $M + 1$ , which will contain the coefficients of approximation.  $A[i]$ , ( $i = 0, 1, \dots, M - 1$ ) is the coefficient of  $\phi_i$ .  $F$  is an array of length  $N$ , containing the input data,  $F[I]$  should contain the value of the function at  $I$ th point. These values must be supplied by the user.  $G$  is an array of length  $IG \times N$  containing the values of basis functions at each point in the table,  $G[j][i] = \phi_i(x_j)$ . 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 function 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 second dimension of  $G$  as specified in the calling function ( $IG \geq M$ ).  $ESUM$  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 function.  $EPS$  is the expected level of roundoff error, this parameter is passed on to the function SIMPL1 for simplex iteration. It is used to decide the sign of cost coefficients and pivot elements. Error status can be obtained from the returned value of LINL1. A value of 616 implies that  $M \leq 0$ , or  $M + 1 > N$ , in which case no calculations are performed. Other values may be set by the function SIMPL1. This function requires the function SIMPL1 to solve the LP problem using a slightly modified simplex algorithm.

**176. SIMPL1** Function to solve LP problems using a modified version of the simplex algorithm, specially suitable for LP problems arising in  $L_1$ -approximations. This function is called by function POLYL1 or LINL1 and is similar to function SIMPX. The LP problem is assumed to be in the standard form and the initial tableau is supplied in the array  $A$  of length  $IA \times (N - M + 1)$  with  $IA \geq M + 2$ .  $IA$  is the second dimension of  $A$ , exactly as specified in the calling function.  $N$  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. AEPS is a parameter used to control roundoff error. Any quantity less than AEPS in magnitude is assumed to be zero. Error status can be obtained from the returned value of SIMPL1. A value of 63 implies that the objective function is unbounded from below and the optimal feasible vector does not exist. SIMPL1 = 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 function. Since the problems arising out of  $L_1$ -approximation have very high degree of degeneracy, this function may fail to converge in some cases, because of cycling as explained in Section 8.7.

## C.11 Algebraic Eigenvalue Problem

**177. INVIT** Function 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 an array of length  $IA \times M$  containing the input matrix. IA is the second dimension of A as specified in the calling function. P is a variable specifying the initial shift to be used. This shift should be close to the required eigenvalue. U is an 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 eigen-

vector is less than REPS. NIT is an integer variable specifying the maximum number of iterations required to be performed. If  $\text{NIT} \leq 0$ , then a default value of NIT0 (= 100) will be used. Error status can be obtained from the returned value of INVIT. A value of 106 implies that  $M \leq 1$  or  $M > \text{IA}$ , in which case, no calculations are performed.  $\text{INVIT} = 141$  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.  $\text{INVIT} = 142$  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 may be set by function GAUELM, which is called to solve the system of linear equations. In particular,  $\text{INVIT} = 121$  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 function, such that zero pivots are replaced by a suitably chosen small number. To find an eigenvector of a generalised eigenvalue problem  $A(\lambda)\mathbf{x} = \mathbf{0}$ , use  $P = 0$ ,  $\text{IFLG} = 0$ , and the matrix A should be set to  $A(\lambda)$  with  $\lambda$  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

**178. TRED2** Function to reduce a real symmetric matrix to tridiagonal form using Householder's method. This function is based on the procedure *tred2* in the *Handbook*. A is an array of length  $\text{IA} \times \text{N}$  containing the matrix. After execution, the array A will be overwritten by the transformation matrix Q, such that  $Q^T A Q$  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 second dimension of the array A, as specified in the calling function,  $\text{IA} \geq \text{N}$ . D and E are 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  $\eta$  is the smallest positive number that is representable in the computer and  $\hbar$  is the machine accuracy. For single precision arithmetic (type float) use  $\text{REPS} = 10^{-30}$ . For double precision arithmetic use  $\text{REPS} = 10^{-300}$ . Error status can be obtained from the returned value of TRED2. A value of 107 implies that  $\text{N} \leq 1$  or  $\text{N} > \text{IA}$ , in which case, no calculations are performed.

**179. TRBAK** Function 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 function TQL2, but will be required if the eigenvectors are calculated by the function TRIDIA. A is an array of length  $\text{IA} \times \text{N}$  containing the transformation matrix Q generated by the function TRED2. The last column of A is used as a scratch space by the function

and hence its contents will be destroyed during execution. IA is the second dimension of A, as specified in the calling function. N is the order of the matrix. Z is an array of length  $IZ \times N$  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 second dimension of array Z, as specified in the calling function. NZ is the number of eigenvectors. Thus,  $Z[i][j]$  should contain the  $i$ th component of the  $j$ th eigenvector. This function simply performs the matrix multiplication  $AZ$  to obtain the required eigenvectors. The returned value of the function is always zero.

**180. TQL2** Function to find eigenvalues and eigenvectors of  $ZZ^T$  using the  $QL$  algorithm, where  $T$  is a symmetric tridiagonal matrix and  $Z$  is an orthogonal matrix. This function 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 function gives the eigenvectors of the original matrix. This function 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 an array of length  $IZ \times N$ , containing the transformation matrix. After execution, this array will be overwritten by the eigenvectors of the original matrix  $ZZ^T$ , with  $i$ th column containing the  $i$ th eigenvector.  $N$  is the order of the matrix.  $IZ$  is the second dimension of  $Z$ , as specified in the calling function.  $D$  is an 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 an 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 function and hence its contents will be destroyed during the execution. REPS is the tolerance which should be equal to  $\hbar$ , the machine accuracy. Error status can be obtained from the returned value of TQL2. A value of 108 implies that  $N \leq 1$  or  $N > IZ$ , in which case, no calculations are performed.  $TQL2 = 143$  implies that the  $QL$  algorithm failed to converge for some eigenvalue, in which case, the calculations are abandoned.

**181. TRIDIA** Function 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 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  $\lambda_i$  are sorted in increasing order, then all eigenvalues from  $\lambda_{M1}$  to  $\lambda_{M2}$  will be determined. If  $M1 > M2$ , then no calculations are performed.  $EI$  is an array of length  $M2 - M1 + 1$ , which will contain the

eigenvalues after execution. EPS1 is a 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 an array of length  $IV \times N$  which will contain the calculated eigenvectors.  $EV[i][j]$  will contain the  $i$ th component of the  $j$ th eigenvector. IV is the second dimension of the array EV as specified in the calling function,  $IV \geq (M2 - M1 + 1)$ . Error status can be obtained from the returned value of TRIDIA. A value of 109 implies that  $N \leq 1$  or  $M2 - M1 + 1 > IV$  or  $M1 < 0$  or  $M2 \geq N$ , in which case, no calculations are performed. Other values may be set by function TINVIT, which is called to calculate the eigenvalues and eigenvectors. Only the last nonzero value of error flag will be retained. This function requires the function STURM to locate the eigenvalues and function TINVIT to find the eigenvalues and eigenvectors of a symmetric tridiagonal matrix and function RAN1 to generate random numbers.

**182. STURM** Function 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 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  $D[i] = a_{ii}$ . M1 and M2 are integers specifying which eigenvalues are to be located. If eigenvalues  $\lambda_i$  are sorted in increasing order, then all eigenvalues from  $\lambda_{M1}$  to  $\lambda_{M2}$  will be located. If  $M1 > M2$ , then no calculations are performed. EL and EU are arrays of length M2. After execution, the  $i$ th eigenvalue should be located in the interval  $(EL[i], EU[i])$ . 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  $|EU[i] - EL[i]| < REPS$  and the eigenvalue has been isolated. At least three bisections are performed after the eigenvalues are isolated, that is  $EU[i-1] < EL[i]$ . 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. Error status can be obtained from the returned value of STURM. A value of 110 implies that  $M1 < 0$  or  $M2 > N - 1$ , in which case no calculations are done.

**183. TINVIT** Function 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 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  $D[i] = a_{ii}$ . EL and EU give the lower and upper limit on the eigenvalue. The iteration will not go beyond these limits. EI will contain the calculated eigenvalue. EV is an array of length N, which will contain the calculated eigenvector. For IFLG  $\neq 0$  it should contain the previous eigenvector determined by the function as input, so that the function 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)  $\max(\mathbf{v}_{j+1})\lambda > 1/\text{REPS}$  (where  $\lambda$  is an estimate for the eigenvalue), (3)  $\|\mathbf{u}_{j+1} - \mathbf{u}_j\|_1 < \text{REPS}$ . 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. NUM is an integer variable, which will contain the number of iterations required by the function. Error status can be obtained from the returned value of TINVIT. A value of 144 implies that the inverse iteration failed to converge to the specified accuracy. This function requires Function RAN1(SEED) to generate random numbers. If a different function is used for this purpose, the seed should be changed appropriately.

**184. HER EVP** Function to find all eigenvalues and eigenvectors of a complex Hermitian matrix. This function 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. AR and AI are arrays of length  $IA \times N$ , containing the real and imaginary parts of the matrix. The function preserves the contents of these arrays. N is the order of the matrix. IA is the second dimension of AR, AI as specified in the calling function. EI is an array of length N, which will contain the eigenvalues of the matrix. The eigenvalues are sorted in the ascending order. VR and VI are arrays of length  $IZ \times N$ , which will contain the real and imaginary parts of eigenvectors of the complex matrix. VR[I][J] and VI[I][J] will contain the Ith component of Jth eigenvector. IZ is the second dimension of arrays VR and VI as declared in the calling function. REPS is the tolerance which should be of the order of  $\hbar$ , the machine accuracy. Error status can be obtained from the returned value of HER EVP. A value of 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 function 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 function picks alternate eigenvectors.

**185. BALANC** Function for reducing the norm of a matrix by exact diagonal similarity transformations. This function is based on the procedure *balance* in the *Handbook*. A is an 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 second dimension of array A, as declared in the calling function. 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 ( $j < \text{LOW}$  or  $i > \text{IGH}$ ). 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 function BALANC. D is an 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. Error status can be obtained from the returned value of BALANC. A value of 112 implies that  $N \leq 1$  or  $N > IA$ , in which case, no calculations are performed.

**186. BALBAK** Function to perform back-transformation of a set of right eigenvectors from those of the balanced matrix to that for the original matrix. This function is based on the procedure *balbak* in the *Handbook*. This function only handles real eigenvectors, but since the transformation is real and linear, the real and imaginary parts of complex eigenvectors can be transformed by two separate calls to this function. N is the order of the matrix. LOW and IGH are integer variables obtained by the function BALANC. CZ is an 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. For complex eigenvectors two calls will be needed for transforming the real and imaginary parts separately. In that case CZ should contain the real (or imaginary) part of the eigenvectors. M is the number of eigenvectors, and IZ is the second dimension of array CZ as declared in the calling function. D is an array of length N, containing the information about transformation as generated by the function BALANC. The returned value of the function is always zero. This function can be modified as explained in the file, to back-transform left eigenvectors and the corresponding version is implemented in BALBAK\_L.

**187. ELMHES** Function for reducing a general real matrix to an upper Hessenberg form using real stabilised elementary similarity transformation. This function is based on the procedure *elmhes* in the *Handbook*. A is an 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 second dimension of array A as declared in the calling function. LOW and IGH are integer variables as given by the function BALANC while balancing the matrix. This function only processes the sub-matrix in rows LOW to IGH. If the matrix is not balanced,



then set  $LOW = 0$  and  $IGH = N - 1$  before calling this function.  $INC$  is an integer array of length  $N$ , containing the information about the row and column interchanges used during the reduction. Error status can be obtained from the returned value of  $ELMHES$ . A value of 113 implies that  $N \leq 1$  or  $N > IA$ , in which case, no calculations are performed.

**188. HQR** Function to find eigenvalues of an upper Hessenberg matrix using  $QR$  algorithm. This function 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 an 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 second dimension of  $H$  as specified in the calling function.  $ER$  and  $EI$  are 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. Error status can be obtained from the returned value of  $HQR$ . A value of 114 implies that  $N \leq 1$  or  $N > IH$ , in which case, no calculations are performed.  $HQR = 145$  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  $EI$ .

## C.12 Ordinary Differential Equations

**189. RKM** Function to solve initial value problems in ordinary differential equations, using a second or fourth-order Runge-Kutta method with adaptive step size control. This function accepts a system of first-order differential equations.  $N$  is the number of first-order equations.  $Y$  is an 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 an 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 function, 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 function 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 function 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 function 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 function is allowed to use. If  $NMAX \leq 0$ , then a default value of NMX (= 10000) will be used. Error status can be obtained from the returned value of RKM. A value of 701 implies that  $N \leq 0$ , in which case, no calculations are performed. RKM = 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 function may be modified. RKM = 722 implies that the step size is too small for the arithmetic used (i.e.,  $H < h|T0|$ ) and the calculations are aborted. RKM = 723 implies that the integration could not be completed in the specified number (NMAX) of steps. The failure of this function can arise either because of singularity, or stiffness, or because the specified accuracy is too high. This function requires function RK4 or RK2 to perform one step of Runge-Kutta integration. Function 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 parameters in the beginning of the function should be changed as indicated. RKM\_2 is the version of RKM for second-order method. Further, the function 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 array containing the dependent variables at time  $t = T$ , where the derivatives need to be evaluated. DY is the array which will contain the calculated value of the derivatives. DY[i] should contain  $Y'[i] = f_i(T, Y)$ .

**190. RK4** Function 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 function. H is the step size to be used. Y0, DY0 and Y1 are 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 function. DIF is the name of the external function used to calculate the right-hand sides of the differential equations. The function DIF as described in the write up for subroutine RKM must be supplied by the user. This function is called by function RKM, but it can be used independently to integrate an equation with constant step size. This can be achieved by repeatedly calling the function after updating the value of T ( $= T + H$ ), Y0 ( $= Y1$ ) and DY0 (using DIF).

**191. RK2** Function to perform one step of integration using a second-order Runge-Kutta method. All arguments have the same meaning as that for the function RK4.

**192. MSTEP** Function to solve initial value problems in ordinary differential equations, using a fourth-order multistep method with adaptive step size control. This function accepts a system of first-order differential equations. It can be used with either function ADAMS for Adams-Bashforth-Moulton predictor-corrector method, or function GEAR for stiffly stable fourth-order method. The change can be affected by choice of a flag. By setting the flag appropriately, this function can also be used to integrate the equation using a fixed step size. This function 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 an 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 function is required for continuing the integration further. DY is an 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 first 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 function, 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 function 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 function can be called repeatedly after changing TN between every call. Other variables (including the scratch arrays) should not be changed, because the function uses the previous values of the solution as the starting values for the new problem. YF is an 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 function 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 function. 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 function is allowed to use. If  $NMAX \leq 0$ , then a default value of  $NMX (= 100000)$  will be used.  $IFLG$  is an integer variable used as a flag by the function. If  $IFLG = 0$  or  $1$ , then the integration is started by generating fresh starting values using function `STRT4`. For higher values of  $IFLG$  it is assumed that the starting values are already available in the arrays  $Y$  and  $DY$ . The function 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 error flag 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 an array, which is used as a scratch space by the function. The length of this array should be  $2N$  for Adams method and  $3N + N^2$  for stiffly stable method. Error status can be obtained from the returned value of  $MSTEP$ . A value of  $702$  implies that  $N \leq 0$ , in which case, no calculations are performed.  $MSTEP = 724$  or  $725$  implies that function `STRT4` failed to generate the required starting values.  $MSTEP = 726$  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 function may be modified.  $MSTEP = 727$  implies that the step size is too small for the arithmetic used (i.e.,  $H < h|T0|$ ) and the calculations are aborted.  $MSTEP = 728$  implies that the integration could not be completed in the specified number ( $NMAX$ ) of function evaluations. The failure of this function can arise either because of singularity, or stiffness (with ADAMS), or because the specified accuracy is too high. For integration with fixed step size, error flag may be set to  $729$  if the corrector fails to converge. This function requires functions `ADAMS`, `GEAR` and `GAUELM` to perform the integration and function `STRT4` and `RK4` to generate the starting values. In addition the function `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 array containing the dependent variables at time  $t = T$ , where the derivatives need to be evaluated.  $DY$  is the array which will contain the calculated value of the derivatives.  $DY[i]$  should contain  $Y'[i] = f_i(T, Y)$ .

**193. ADAMS** Function to perform one step of solution of an initial value problem, using a fourth-order Adams-Bashforth-Moulton predictor-corrector method. This function can be easily modified to use any other predictor-corrector formula. This function is called by function `MSTEP` to perform one step of integration.  $N$  is the number of first-order differential equations,  $Y$  and  $DY$  are arrays of length  $7N$ , as used by function `MSTEP`. `DIF` is the name of the function 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 function 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 function. 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. WK is an array of length  $2N$ , which is used to transmit the predicted value to function MSTEP. Error status can be obtained from the returned value of ADAMS. A value of 729 implies that the iteration on corrector has failed to converge. Function DIF as described in the writeup for function MSTEP must be supplied by the user.

**194. STRT4** Function 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 function is called by the function MSTEP.  $N$  is the number of first-order differential equations to be solved.  $Y$  and  $DY$  are arrays of length  $4N$ , containing the values of  $\mathbf{y}$  and  $\mathbf{y}'$  at the four starting points. While calling the function, 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 function. DIF is the name of the function 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 function.  $T$  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 function. WK is an array of length  $2N$  used to pass on the value for modifier. Error status can be obtained from the returned value of STRT4. A value of 724 implies that the function failed to find the starting values, because the step size became too small.  $STRT4 = 725$  implies that the routine failed to find the starting values in the specified number ( $NIT = 10$ ) of attempts. This function requires functions RK4 and DIF. Function DIF as described in the writeup for function MSTEP must be supplied by the user.

**195. GEAR** Function to perform one step of solution of initial value problem using a fourth-order stiffly stable method based on the backward differentiation formula. This function uses the Broyden's method to solve the implicit corrector formula. This function is called by the function MSTEP to perform one step of integration.  $N$  is the number of first-order differential equations,  $Y$  and  $DY$  are arrays of length  $7N$  as used by function MSTEP. DIF is the name of the function 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 function 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 function. 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. WK1 is an array of length  $3N$  used to pass predicted values. WK is an array of length  $N^2$  used to store the inverse of the Jacobian. Error status can be obtained from the returned value of GEAR. A value of 729 implies that the corrector iteration fails to converge. Function DIF as described in write-up for MSTEP must be supplied by the user. This function also requires function GAUELM to solve systems of linear equations. The parameter CFAC in the function may need to be reduced if the function MSTEP needs to adjust the step size too often.

**196. EXTP** Function to solve initial value problems in ordinary differential equations using extrapolation method. This function accepts a system of first-order differential equations. N is the number of first-order equations. Y is an 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 an 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 function, 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 function 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 function 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 function 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 function. 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 function is allowed to use. If  $NMAX \leq 0$ , then a default value of NMX (= 100000) will be used. 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. Error status can be obtained from the returned value of EXTP. A value of 703 implies that  $N \leq 0$ , in which case, no calculations are performed. EXTP = 730 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 function may be modified. EXTP = 731 implies that the step size is too small for the arithmetic used (i.e.,  $H < h|T0|$ ) and the calculations are aborted. EXTP = 732 implies that the integration could not be completed in the specified number (NMAX) of steps. EXTP = 733 implies that the denominator in the rational function extrapolation is zero. The failure of this function 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. The function 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 array containing the dependent variables at time  $t = T$ , where the derivatives need to be evaluated. DY is the array which will contain the calculated value of the derivatives. DY[i] should contain  $Y'[i] = f_i(T, Y)$ .

**197. FDM** Function 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[0]). PAR is an array which is passed on to the function 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 functions. This array is not used by function FDM, or any other function called by it except EQN and BCS, and its size can be arbitrary depending on the requirements. X is an array of length  $M \times N$ , 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 function FDM treats it as a two-dimensional array of dimension X[N][M] with X[J][I] containing the Ith component at Jth mesh point. Unfortunately, there is no provision to pass the second dimension of the array X, as specified in the calling function. Hence, to obtain meaningful results, the second dimension of the array X in the calling function 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 an array of length  $M \times N$ , 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  $XC[J][I] - X[J][I]$  gives an estimate of the truncation error in  $X[J][I]$  provided the deferred correction is calculated. T is an array of length N, containing the mesh points  $t_j$ , ( $j = 0, \dots, N-1$ ), 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 function, which defines the differential equations. The differential equation is assumed to be in the form  $By' = f(t, y)$ , where  $B$  is a  $M \times M$  matrix and  $f$  is a vector function of length M. The matrix  $B$  must be nonsingular. The elements of  $B$  and  $f$  at a given point T are calculated by the function 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 function, which specifies the boundary conditions. The boundary conditions are defined by functions  $g_i(t_0, y_0) = 0$  for  $i = 0, \dots, ML-1$  at the first point  $t = T[0]$ , and  $g_i(t_{N-1}, y_{N-1}) = 0$  for  $i = ML, \dots, M-1$  at the last point  $t = T[N-1]$ . Apart from the functions  $g_i$ , this function should also calculate the Jacobian matrix  $\partial g_i / \partial y_j$  (stored in the array BC). The general form of these functions is as follows:

```
void eqn(int j, int m, int ml, double par[], double *a,
double *b, double y[], double f[], double t)
{
    int i,k,m1;

    m1=m+ml;
    for(i=0; i<m; ++i) {
        f[i]=f_i(t, par, y);
        for(k=0; k<m; ++k) {
            a[k+i*m1]=∂f_k/∂y[i];
            b[k+i*m1]=b_{ki}(t, par);
        }
    }
    return;
}

void bcs(int m, int ml, double par[], double *bc, double g[],
double t1, double tn, double y1[], double yn[])
{
    int i,k,m1;

    m1=m+ml;
    for(i=0; i<m; ++i) {
        if(i<ml) g[i]=g_i(t1, par, y1);
        else g[i]=g_i(tn, par, yn);

        for(k=0; k<m; ++k) bc[i+k*m1]=∂g_i/∂y[k];
    }
}
```



```

    return;
}

```

It should be noted that the second dimension of A, B and BC is M+ML instead of M. Here T, T1 (= T[0]) and TN (= T[N - 1]) are the values of  $t$  at relevant points, while the arrays Y, Y1 and YN contain the values of  $\mathbf{y}$  at these points. In function EQN 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.

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. Error status can be obtained from the returned value of FDM. A value of 704 implies that  $N < 3$ , or  $M \leq ML$ , or  $ML \leq 0$ , in which case, no calculations are performed. FDM = 734 implies that  $N \leq 4$  and deferred correction is requested. In this case, the deferred correction is not calculated. FDM = 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. FDM = 736 implies that the mesh spacing is not uniform, in which case, the deferred correction will not be calculated. FDM = 737 implies that the Newton's iteration for solving the finite difference equations fails to converge. This function requires function SETMAT for setting up the matrix of finite difference equations, and function GAUBLK to solve the system of linear equations defined by the block matrix as explained in Section 12.8. Apart from these, the functions EQN and BCS specifying the problem must be supplied by the user.

**198. GEVP** Function 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[0]). PAR is an array which is passed on to the functions 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[0] will contain the calculated eigenvalue without applying the deferred correction. Other elements of this array can be used to pass any parameter that may be required by these functions. Other elements of the array are not used by function GEVP or any other function called by it except EQN, BCS, EQND and BCSD, and its size can be arbitrary depending on the requirements. X is an 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 function GEVP treats it as a two-dimensional array of dimension X[N][M], with X[J][I] containing the Ith component at Jth mesh point. Unfortunately, there is no provision to pass the second dimension of the array X as specified in the calling function. Hence, to obtain meaningful results, the second dimension of the array X in the calling function 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 an array of length  $M \times N$ , 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 an array of length N, containing the mesh points  $t_j$  for  $j = 0, \dots, N-1$  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 function which defines the differential equation. The differential equation is assumed to be in the form  $By' = Ay$ , where  $A$  and  $B$  are  $M \times M$  matrices. The matrix  $B$  must be nonsingular. The elements of  $A$  and  $B$  which could be arbitrary functions of the eigenvalue  $\lambda$  and  $t$ , are calculated by the function EQN. BCS is the name of the function which specifies the boundary conditions. The boundary conditions are defined by  $C_1 y_0 = 0$ , and  $C_N y_{N-1} = 0$ , where  $C_1$  and  $C_N$  are  $ML \times M$  and  $(M - ML) \times M$  matrices, respectively. These matrices are combined in one  $M \times M$  matrix BC, the first ML rows of which correspond to the boundary conditions at the first boundary ( $t = T[0]$ ), while the remaining rows refer to boundary conditions at the last point ( $t = T[N-1]$ ). EQND is the name of the function to calculate the derivatives of matrices  $A$  and  $B$  (as defined in EQN) with respect to the eigenvalue  $\lambda$ . Similarly, BCSD is the name of the function to calculate the derivatives of matrix BC (as defined in BCS) with respect to the eigenvalue  $\lambda$ . Following the C convention the indices of arrays  $A$ ,  $B$  and  $BC$  are interchanged as compared to the Fortran subprogram, thus the rows are really columns. The order of index should be clear from the form of functions given below. If deferred correction is not required, then the functions EQND and BCSD are not called, but in order

to satisfy the compiler and the linker, dummy functions may be required. The general form of these functions is as follows:

```

void eqn(int j, int m, int ml, double par[], double *a,
double *b, double y[], double f[], double t)
{
    int i,k,m1;

    m1=m+ml;
    for(i=0; i<m; ++i) {
        f[i]=0.0;
        for(k=0; k<m; ++k) {
            a[k+i*m1]=aki(t,par);
            b[k+i*m1]=bki(t,par);
        }
    }
    return;
}

void bcs(int m, int ml, double par[], double *bc, double g[],
double t1, double tn, double y1[], double yn[])
{
    int i,k,m1;

    m1=m+ml;
    for(i=0; i<m; ++i) {
        g[i]=0.0;
        for(k=0; k<m; ++k) {
            if(i<ml) bc[i+k*m1]=bcik(t1,par);
            else bc[i+k*m1]=bcik(tn,par);
        }
    }
    return;
}

void eqnd(int j, int m, int ml, double par[], double *a,
double *b, double t)
{
    int i,k,m1;

    m1=m+ml;
    for(i=0; i<m; ++i) {
        for(k=0; k<m; ++k) {
            a[k+i*m1]=∂aki(t,par)/∂par[0];
            b[k+i*m1]=∂bki(t,par)/∂par[0];
        }
    }
    return;
}

```

```

    }

    void bcsd(int m, int ml, double par[], double *bc,
    double t1, double tn)
    {
        int i,k,m1;

        m1=m+ml;
        for(i=0; i<m; ++i) {
            for(k=0; k<m; ++k) {
                if(i<m1) bc[i+k*m1]=∂bcik(t1,par)/∂par[0];
                else bc[i+k*m1]=∂bcik(tn,par)/∂par[0];
            }
        }
        return;
    }

```

It should be noted that the second dimension of A, B and BC is M+ML instead of M. Here T, T1 (= T[0]) and TN (= T[N − 1]) are the values of  $t$  at relevant points. The 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 function FDM, since the same function SETMAT is used in both cases. In functions 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.

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  $E0 - PAR[0]$  will give an estimate of the truncation error in PAR[0]. EL and EU respectively, specify the lower and upper limit on eigenvalue. These parameters are used to specify the region in which the eigenvalues are to be searched. These parameters are passed on to function SECANI, (or MULER2) which terminates the iteration if at any stage it goes outside the specified limits. Error status can be obtained from the returned value of GEVP. A value of 704 implies that  $N < 3$ , or  $M \leq ML$  or  $ML \leq 0$ . In this case, no calculations are performed. GEVP = 734 implies that  $N \leq 4$  and deferred correction is requested. In this case, the deferred correction is not calculated. GEVP = 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. GEVP = 736 implies that the mesh spacing is not uniform, in which case the deferred correction will not be calculated. GEVP = 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[0]. GEVP = 739 implies that the inverse iteration for calculating the eigenvector failed to converge. GEVP = 740 implies that the inverse iteration for calculating the left eigenvector failed to converge. Apart from these, other values may be set by function SECANI (or MULR2), which is called to find zeros of the determinant. The function GEVP requires function SETMAT for setting up the matrix of finite difference equations, and function GAUBLK to solve the system of linear equations, or to calculate the determinant of the block matrix as explained in Section 12.8. Function MULR2, SECANI or any other equivalent function is required for finding zeros of the determinant. It may be noted that function MULR2 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 MULR2 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 MULR2 use reverse communication technique and hence the control is passed back to GEVP when a function evaluation is required. Apart from these, the functions EQN, EQND, BCS and BCSD specifying the problem must be supplied by the user.

**199. GAUBLK** Function to solve a system of linear equations involving finite difference matrix of the form described in Section 12.8. This function 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 an array of length  $(M + ML) \times 2M \times N$  containing the matrix (Section 12.8). IFLG is an integer variable used as a flag. If IFLG = 0, the function 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 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  $DET \times 2^{IDET}$ . INC is an integer array of length  $M \times N$ , which contains information about row interchanges used during Gaussian elimination. X is an array of length  $M \times N$ , which should contain the right-hand sides of the linear equations. After execution, the solution will be returned in the same array. Error status can be obtained from the returned value of GAUBLK. A value of 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 function is called by functions FDM and GEVP.

**200. SETMAT** Function to setup the finite difference matrix for a system of first-order differential equations with separable boundary conditions, using a central difference approximation. This function is called by functions 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 an array of length  $(M + ML) \times 2M \times N$ , which will contain the finite difference matrix for the linearised equations. BC is an array of length  $(M + ML) \times (M + 1)$ , which is used to store information about the boundary conditions. X and XC are arrays of length  $M \times N$ . 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.  $X[J][I]$  should contain the Ith component at Jth mesh point. T is an array of length N containing the mesh points. PAR is an array containing parameters, which are passed on to the functions EQN and BCS. EQN and BCS are the names of functions for defining the differential equation and the boundary conditions, respectively. These functions must be supplied by the user, as explained in the write up for functions FDM or GEVP. The returned value of the function is always zero.

**201. BSPODE** Function 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.,  $T[0]$ ). PAR is an array which is passed on to the function 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 functions. This array is not used by function BSPODE, or any other function called by it except EQN and BCS, and its size can be arbitrary depending on the requirements. X is an array of length  $M \times N$ , which will contain the computed solution.  $X[J][I]$  will contain the Ith component of solution at  $TX[J]$ . The second dimension of X must be equal to M in the calling function. A is an array of length  $(NK + K - 2) \times M$  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 function BSPODE treats it as a two-dimensional array of dimension  $A[M][NK + K - 2]$  with  $A[J][I]$  containing the coefficient of Ith basis function in Jth component of

the solution.

$$X_j(t) = \sum_{i=0}^{NK+K-3} A[j][i] \phi_i(t), \quad j = 0, \dots, M-1. \quad (\text{C.89})$$

Unfortunately, there is no provision to pass the second dimension of the array  $X$  or  $A$ , as specified in the calling function. Hence, to obtain meaningful results, the second dimension of the array  $A$  in the calling function must also be exactly equal to  $NK + K - 2$ . 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 an array of length  $NK$ , containing the knots  $t_j$ , ( $j = 0, \dots, NK - 1$ ), 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  $T[0]$  and  $T[NK-1]$  as the two boundaries.  $N$  is the number of mesh points to be used for obtaining the equations connecting the coefficients of expansion,  $N \geq NK + K - 2$ .  $TX$  is an 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 function computes the elements of this array assuming uniform spacing and covering the interval  $T[0]$  to  $T[NK-1]$ . 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 function, 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  $M \times M$  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 function `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 function, which specifies the boundary conditions. The boundary conditions are defined by functions  $g_i(t_0, \mathbf{y}_0) = \mathbf{0}$  for  $i = 0, \dots, ML - 1$  at the first point  $t = T[0]$ , and  $g_i(t_{NK-1}, \mathbf{y}_{NK-1}) = \mathbf{0}$  for  $i = ML, \dots, M - 1$  at the last point  $t = T[NK - 1]$ . Apart from the functions  $g_i$ , this function should also calculate the Jacobian matrix  $\partial g_i / \partial y_j$  (stored in the array  $BC$ ). The general form of these functions is as follows:

```
void eqn(int j, int m, int ml, double par[], double *a,
double *b, double y[], double f[], double t)
{
    int i,k;
    for(i=0; i<m; ++i) {
        f[i]=f_i(t, par, y);
        for(k=0; k<m; ++k) {
            a[k+i*m]=∂f_k/∂y[i];
            b[k+i*m]=b_ki(t, par)
        }
    }
    return;
}
```

```

}

void bcs(int m, int ml, double par[], double *bc, double g[],
double t1, double tn, double y1[], double yn[])
{
    int i,k;

    for(i=0; i<m; ++i) {
        if(i<ml) g[i]=gi(t1,par,y1);
        else g[i]=gi(tn,par,yn);
        for(k=0; k<m; ++k) bc[i+k*m]=∂gi/∂y[k];
    }
    return;
}

```

It should be noted that these functions are same as those required by FDM except that the second dimension of arrays A, B and BC is M instead of M+ML. Here T, T1 (= T[0]) and TN (= T[NK − 1]) are the values of  $t$  at relevant points, while the arrays Y, Y1 and YN contain the values of  $\mathbf{y}$  at these points. In function 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.

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 function 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. Error status can be obtained from the returned value of BSPODE. A value of 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. BSPODE = 741 implies that the Newton's iteration for solving the system of equations fails to converge. This function requires function BSPLIN, BSPEVL, SVD and SVDEVL. Apart from these, the functions EQN and BCS specifying the problem must be supplied by the user.

### C.13 Integral Equations

**202. FRED** Function 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 variables specifying respectively, the lower and the upper limit of the integral. WT and X are arrays of length M, which will



return the weights and abscissas used by the quadrature formula. F and FC are 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 functions 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 function 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 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.

IQ is an integer variable used to specify the quadrature formula to be used. If IQ = 1, then trapezoidal rule is used, for IQ = 2, Simpson's 1/3 rule is used, for 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 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 IQ = 4 and 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 the returned value of FRED to -11. IT is an integer variable, which specifies the kind of Fredholm equation to be solved. IT = 1, 2 and 3 corresponds to the Fredholm equations of the first, second and third kind, respectively. This function can only find a real eigenvalue. REPS is a parameter specifying the required (relative) accuracy in calculating the eigenvalues. This parameter is used only if IT = 3. It is only passed on to the function INVIT for calculating the eigenvalue. Error status can be obtained from the returned value of FRED. A value of 706 implies that IT > 3, or IT ≤ 0, or the number of points M are not sufficient for application of the specified quadrature formula. In this case, no calculations are performed. FRED = 707 implies that the value of IQ is not proper, in which case also no calculations are performed. FRED = -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 functions GAUELM and INVIT, which are called by this function. Function FRED requires function GAUELM to solve the system of linear equations, and function INVIT (for IT = 3) to calculate the eigenvalue and the eigenvector. It should be noted that function INVIT is called with 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. Functions FG(X) and FKER(X, T) must be supplied by the user.

**203. FREDCO** Function 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  $f(x) = \sum_{i=0}^{N-1} a_i \text{PHI}(i, x)$ . The basis functions  $\text{PHI}(I, X)$  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 variables specifying respectively, the lower and the upper limit of the integral. F is an array of length N, which will contain the calculated coefficients of the expansion. F[I] contains the coefficient ( $a_i$ ) of  $\text{PHI}(I, X)$  in the expansion for the solution. X is an 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 function ADPINT, while in other cases, these parameters are ignored. IQ is an integer variable used to specify the treatment of integrals required to calculate

$$\text{PSI}(I, X) = \int_A^B \text{FKER}(X, T) \text{PHI}(I, T) dT. \quad (\text{C.90})$$

If IQ  $\neq$  0, then it is assumed that the function  $\text{PSI}(I, X)$  is supplied separately, while if IQ = 0, then the integrals are calculated using the function ADPINT. This function uses external function FUNK to calculate the integrand, which in turn requires the functions  $\text{PHI}(I, X)$  and  $\text{FKER}(X, T)$ . 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. Error status can be obtained from the returned value of FREDCO. A value of 708 implies that IT > 2, or IT  $\leq$  0, or N < 1, in which case, no calculations are performed. Apart from these, other values may be set by the functions GAUELM or ADPINT, which are called by this function. This function requires function GAUELM to solve the system of linear equations, functions ADPINT and KRONRD (for IQ = 0) to evaluate the integrals and function FUNK(X) to evaluate the required integrand. Further, the functions FG(X) ( $= g(x)$ ),  $\text{PHI}(I, X)$  ( $= \phi_i(x)$ ) and  $\text{FKER}(X, T)$  ( $= K(x, t)$ , for IQ = 0) or  $\text{PSI}(I, X)$  (for IQ  $\neq$  0) must be supplied by the user. Depending on value of IQ, some of these functions may not be required, but even in that case a dummy function to satisfy the compiler or linker may be needed. The names of these functions are fixed and cannot be passed on to the function 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 functions are also not passed on to the function. The global variables XFRED and IFRED are used to pass on the value of  $x$  and  $i$  to FUNK for calculating the integrand. This function can also be used to solve linear Volterra equations, by defining the kernel to be zero for  $t > x$ .

**204. FUNK** Function to calculate the integrand for calculating  $\psi_i(x)$  as required by function 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 functions have to be the same as there is no provision to pass on the names to FUNK. The values of X and I are passed on through the global variables XFRED and IFRED.

**205. RLS** Function 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

$$d_i = \int_a^b K_i(t) f(t) dt, \quad i = 0, 1, \dots, \text{NM} - 1; \quad (\text{C.91})$$

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 function 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

$$a_{ij} = \int_a^b K_i(t) \phi_j(t) dt, \quad (\text{C.92})$$

where  $\phi_j(t)$  is the  $j$ th 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 function 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 function 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  $\text{NK} + \text{K} - 2$ . XO is an 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[0] containing the first knot. K is the order of B-splines required,  $\text{K} = 2$  for linear B-splines while  $\text{K} = 4$  for cubic B-splines, etc. NR is the number of points used for defining the kernels. R is an 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  $\text{NR} = 0$ ). RKER is an array of length  $\text{IK} \times \text{NR}$  containing the kernels for the inverse problem. RKER[j][i] should contain  $K_i(r_j)$ . This array must be supplied if IFLG < 2, otherwise it is not required. A dummy array

may be required to satisfy the compiler, even if  $IFLG \geq 2$ . 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 second dimension of arrays RKER and AC, as specified in the calling function. IK must be at least  $NM + NS$ . AC is an array of length  $IK \times (NK + K - 2)$  containing the coefficients of the matrix defining the inversion problem.  $AC[j][i]$  should contain the coefficient  $a_{ij}$  defined by (C.92). If  $IFLG < 2$ , these coefficients are calculated by evaluating the integrals in (C.92) 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 function 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 an array of length NM, which should contain the data  $d_i$ , for inversion. DE is an array of length NM, containing the estimated errors in  $d_i$ . These errors must be positive. DF is an 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 an array of length NR which will contain the calculated solution at  $R[I]$ . B is an array of length  $NM + NS$ , which will contain the coefficients of expansion. The computed solution is

$$f(t) = \sum_{i=0}^{NK+K-3} B[i] \phi_i(t). \quad (C.93)$$

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. In this case the kernels RKER are not used. 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 function 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 parameters, (ALP and IDE) 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 function 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. It may be noted that the set of kernels may change if we are interested in studying the effect of dropping some of these modes on the computed solution.

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  $\chi^2$  define by

$$\chi^2 = \sum_{i=0}^{NM-1} DF[i]^2 = \sum_{i=0}^{NM-1} \left( \frac{1}{DE[i]} \right)^2 \left( DI[i] - \sum_{j=0}^{NK+K-3} a_{ij} B(j) \right)^2, \quad (C.94)$$

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

$$\text{SUMD} = \left( \frac{R[\text{NR} - 1] - R[0]}{\text{NS} - 1} \right) \sum_{i=0}^{\text{NS}-1} \left( \frac{d^2 f}{dt^2}(t_i) \right)^2, \quad (\text{C.95})$$

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 an array of length  $\text{IV} \times (\text{NM} + \text{NS})$  containing the matrix  $U$  of SVD of the matrix of equations. AV is an array of length  $\text{IV} \times (\text{NK} + \text{K} - 2)$  containing the matrix  $V$  of SVD of the matrix of equations. IV is the second dimension of arrays A and AV as declared in the calling function. SIGMA is an array of length  $(\text{NK} + \text{K} - 2)$  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 an 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. Error status can be obtained from the returned value of RLS. A value of 705 implies that  $\text{NM} \leq \text{NK} + \text{K} - 2$  or  $\text{IK} < \text{NM} + \text{NS}$  or  $\text{IV} < \text{NK} + \text{K} - 2$ . RLS = 710 implies that  $\text{ALP} < 0$  or  $\text{IDE} < 1$  or  $\text{IDE} > 2$ . In all these cases no calculations are done. Other values may be set by functions BSPLIN or SVD which are called. It may be noted that the function allows solution to be computed even when  $\text{ALP} = 0$ , although the solution would be unacceptable in this case. This function requires functions BSPLIN, BSPEVL, SVD, SVDEVL and RANGAU.

**206. FORW** To solve the forward problem corresponding to the inverse problem solved by RLS. This function 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 an array of length NP containing the points at which values of kernels are available. RKER is an array of length  $\text{IK} \times \text{NP}$  containing the kernels for the inverse problem,  $\text{RKER}[j][i] = K_i(R[j])$ . These kernels are the same as what are used by RLS for inversion. IK is the second dimension of RKER as declared in the calling function,  $\text{IK} \geq \text{NM}$ . DI is an array of length NM, which will contain the calculated data points  $d_i$ , using the kernels. F is an array of length NP containing the function values,  $F[i] = f(R[i])$ . If IFLG = 0, the function values are calculated using user supplied function FUN, otherwise, these values must be supplied while calling the function. 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 function will still be required to satisfy the compiler or linker. Error status can be obtained from the returned value of FORW. A value of 711 implies that  $\text{IK} < \text{NM}$  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 function 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 function 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.

**207. VOLT** Function 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 variable specifying the lower limit of the integral. It is also the initial point, from which the solution is required. H is 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 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 functions used to calculate the  $g(x)$  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. Error status can be obtained from the returned value of VOLT. A value of 712 implies that  $N < 3$ , in which case no calculations are performed. VOLT = 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  $K(x, x) = 0$ . Functions FG(X) and FKER(X, T) must be supplied by the user.

**208. VOLT2** Function to solve nonlinear Volterra equation of the second kind

$$\int_A^x K(x, t, f(t)) dt = f(x) + g(x), \quad (\text{C.96})$$

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 the lower limit of the integral. It is also the initial point from which the solution is required. H is 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 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 functions used to calculate the  $g(x)$  and the kernel  $K(x, t, f)$ , respectively. REPS is a 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  $F[I]$ . Error status can be obtained from the returned value of  $VOLT2$ . A value of 712 implies that  $N < 3$ , in which case no calculations are performed.  $VOLT2 = 752$  implies that the fixed-point iteration fails to converge at some point, in which case, the calculations are aborted at that point. In this case the values already calculated so far will be available in arrays  $X$  and  $F$ , but their reliability may need to be checked. Functions  $FG(X)$  and  $FKER(X, T, F)$  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. (C.96).

## C.14 Partial Differential Equations

All functions in this chapter are simple implementations of finite difference methods. Although these functions 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 function does not check for such inconsistencies and no error message will be issued. Similarly, some functions allow for rather general boundary conditions, but the accuracy may be only first order in  $\Delta t$ , 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 functions (e.g., **ADI**). On the other hand, those functions which accept boundary condition in simple Dirichlet form only, can be modified to incorporate more general boundary conditions. Hence, these functions 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 functions to suit their problems.

**209. CRANK** Function to solve a linear parabolic equation of the form

$$\frac{\partial u}{\partial t} = A(x, t) \frac{\partial^2 u}{\partial x^2} + B(x, t) \frac{\partial u}{\partial x} + C(x, t)u + D(x, t), \quad (C.97)$$

subject to boundary conditions

$$\begin{aligned} A_0(t)u(X0, t) + B_0(t) \frac{\partial}{\partial x} u(X0, t) &= F_0(t), \\ A_n(t)u(XN, t) + B_n(t) \frac{\partial}{\partial x} u(XN, t) &= F_n(t), \end{aligned} \quad (C.98)$$



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. (X0, 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 an array of length NX, which contains the mesh points in X direction. This array need not be initialised at input, since the function assumes a uniform mesh spacing and calculates the required mesh points. U is an 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 functions used to calculate the coefficients in the differential equation and the boundary conditions, respectively. Functions COF(X, T, A, B, C, D) and BC(T, X0, XN, A0, B0, F0, AN, BN, FN) must be supplied by the user. Function COF should calculate the coefficients A, B, C, D at a given value of X and T. Function 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, 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 function calculates the solution  $u(x, t)$  at the initial time T. For other values of IFLG a dummy function FIC may be required to satisfy the compiler or the linker. IFLG is an integer variable used as a flag. If IFLG = 0, the initial values are calculated using the function 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 function sets IFLG to 1, so that during subsequent calls to the function, the initial values are taken from the array U. This function 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. Error status can be obtained from the returned value of CRANK. A value of 713 implies that  $DT = 0$ , or  $XN = X0$ , or  $NX \leq 2$ , in which case, no calculations are performed.  $CRANK = 761$  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.

**210. LINES** Function to solve a system of nonlinear parabolic equations of the form

$$\frac{\partial u_i}{\partial t} = f_i\left(x, t, \mathbf{u}, \frac{\partial \mathbf{u}}{\partial x}, \frac{\partial^2 \mathbf{u}}{\partial x^2}\right), \quad (i = 0, 1, \dots, NELIN - 1); \quad (C.99)$$

subject to Dirichlet boundary conditions  $\mathbf{u}(X0, t) = \mathbf{g}_1(t)$  and  $\mathbf{u}(XN, t) = \mathbf{g}_2(t)$ , using the method of lines. This function is actually used through the functions

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 function MSTEP with GEAR. Function LINES specifies the system of ordinary differential equations, as required by the function MSTEP (function DIF in the arguments for MSTEP). The parameters for the partial differential equations are passed via the global variables, DXLIN, X0LIN, XNLIN, XLIN, U0LIN, UNLIN, NXLIN, NELIN. The relevant variables amongst these must be initialised before calling the function MSTEP. NXLIN is the number of mesh points in the  $x$  direction. NELIN is the number of parabolic equations in the system. DXLIN is the step size in  $x$  direction. (X0LIN, XNLIN) 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  $N = (NXLIN - 2)NELIN$ . XLIN is an array of length  $NXLIN - 2$ , containing the mesh points  $x_i$ , excluding the end points. This array must be initialised before calling the function MSTEP. U0LIN and UNLIN are arrays of length NELIN, containing the boundary values (at X0LIN and XNLIN) of the solution at the current time. These arrays need not be initialised before calling MSTEP, since the boundary values are calculated by function LINES using a user supplied function BC. The arguments of LINES have the same definition as specified for function 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  $(NXLIN - 2)NELIN$ ). U and DU are 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 function requires functions FCN and BC to supply the information about the equations and the boundary conditions. Function FCN(NELIN, X, T, U, UX, UXX, DU) calculates the derivatives

$$DU[I] = \frac{\partial u_i}{\partial t} = f_i(X, T, U, UX, UXX), \quad (I = 0, \dots, NELIN - 1), \quad (C.100)$$

for a given value of X, T, U[J], UX[J], UXX[J], ( $J = 0, \dots, NELIN - 1$ ), as specified by the differential equation. Here U, UX, UXX and DU are arrays of length NELIN, containing  $\mathbf{u}$ ,  $\partial \mathbf{u} / \partial x$ ,  $\partial^2 \mathbf{u} / \partial x^2$ ,  $\partial \mathbf{u} / \partial t$  respectively. Function BC(NELIN, T, X0LIN, XNLIN, U0LIN, UNLIN) calculates the boundary values U0LIN[I], UNLIN[I], ( $I = 0, \dots, NELIN - 1$ ) for a given value of T, X0LIN and XNLIN. The names of the functions FCN and BC are fixed and cannot be passed on as arguments. The initial values should be supplied to the function MSTEP at the time of calling. The  $NELIN \times (NXLIN - 2)$  components of the initial vector are arranged in the normal Fortran order, with NELIN components of  $\mathbf{u}$  at each of the  $NXLIN - 2$  mesh points stored consecutively. Thus,  $u_j(x_k)$  occupies the  $k \times NELIN + j$  position in the initial vector. The final result at the required value of time will also be returned in the same order.

**211. ADM** Function to solve a parabolic equation in two space variables

$$\begin{aligned} \frac{\partial u}{\partial t} = & A_{xx}(x, y, t) \frac{\partial^2 u}{\partial x^2} + A_{yy}(x, y, t) \frac{\partial^2 u}{\partial y^2} + A_x(x, y, t) \frac{\partial u}{\partial x} + A_y(x, y, t) \frac{\partial u}{\partial y} \\ & + A_u(x, y, t)u + A_0(x, y, t) \end{aligned} \quad (\text{C.101})$$

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

$$\begin{aligned} \frac{u_{jl}^{n+1/2} - u_{jl}^n}{\frac{1}{2}\Delta t} = & \frac{1}{(\Delta x)^2} A_{xx}^{n+1/2} \delta_x^2 u_{jl}^{n+1/2} + \frac{1}{2\Delta x} A_x^{n+1/2} \delta_x u_{jl}^{n+1/2} \\ & + \frac{1}{(\Delta y)^2} A_{yy}^n \delta_y^2 u_{jl}^n + \frac{1}{2\Delta y} A_y^n \delta_y u_{jl}^n \\ & + \frac{1}{2} (A_u^n u_{jl}^n + A_u^{n+1/2} u_{jl}^{n+1/2}) + \frac{1}{2} (A_0^{n+1/2} + A_0^n), \\ \frac{u_{jl}^{n+1} - u_{jl}^{n+1/2}}{\frac{1}{2}\Delta t} = & \frac{1}{(\Delta x)^2} A_{xx}^{n+1/2} \delta_x^2 u_{jl}^{n+1/2} + \frac{1}{2\Delta x} A_x^{n+1/2} \delta_x u_{jl}^{n+1/2} \\ & + \frac{1}{(\Delta y)^2} A_{yy}^{n+1} \delta_y^2 u_{jl}^{n+1} + \frac{1}{2\Delta y} A_y^{n+1} \delta_y u_{jl}^{n+1} \\ & + \frac{1}{2} (A_u^{n+1} u_{jl}^{n+1} + A_u^{n+1/2} u_{jl}^{n+1/2}) + \frac{1}{2} (A_0^{n+1/2} + A_0^{n+1}), \end{aligned} \quad (\text{C.102})$$

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 an array of length NX, which will contain the mesh points  $x_j$  in  $x$  direction. Y is an 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 function, since the function assumes a uniform spacing and initialises the arrays before proceeding with the calculations. U is an 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[J][I] is the solution  $u(x_I, y_J)$ . IU ( $\geq$  NX) is the second dimension of U as declared in the calling function. COF is the name of the function to calculate the coefficients of the equation. Function 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 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

$$\begin{aligned} u(\text{X0}, y, t) &= \text{BC}(1, \text{X0}, y, t), & u(\text{XN}, y, t) &= \text{BC}(2, \text{XN}, y, t), \\ u(x, \text{Y0}, t) &= \text{BC}(3, x, \text{Y0}, t), & u(x, \text{YN}, t) &= \text{BC}(4, x, \text{YN}, t). \end{aligned} \quad (\text{C.103})$$

If the boundary values depend on time, then the accuracy of difference approximation may be only  $O(\Delta t)$  instead of  $O((\Delta t)^2)$ . FIC is the name of the function, 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 function calculates the solution  $u(x, y, t)$  at the initial time  $T$ . For other values of  $IFLG$  a dummy function  $FIC$  may be required to satisfy the compiler, or the linker.  $IFLG$  is an integer variable used as a flag. If  $IFLG = 0$ , the initial values are calculated using the function  $FIC$  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 function sets  $IFLG$  to 1, so that during subsequent calls to the function, the initial values are taken from the array  $U$ . This function 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. Error status can be obtained from the returned value of  $ADM$ . A value of 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.  $ADM = 762$  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.

**212. LAX** Function 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 an array of length  $NX$ , used to store the coordinates of the mesh points  $x_j$ . This array need not be initialised before calling the function, since the function assumes a uniform spacing and calculates  $x_j$  before performing the calculations.  $U$  is an 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  $IFLG \neq 0$ ).  $IU[J][I]$  contains the  $I$ th component of the solution  $u_i$  at  $J$ th mesh point  $x_j$ .  $IU (\geq N)$  is the second dimension of  $U$ , as declared in the calling function.  $FLUX$  is the name of the function, which is used to calculate the flux  $\mathbf{f}$  occurring in the differential equations. Function  $FLUX(N, X, T, U, F)$  should calculate the flux  $F[I]$ , ( $I = 0, \dots, N-1$ ) for a given value of  $X, T$  and  $U[J]$ , ( $J = 0, \dots, N-1$ ).  $BC$  is the name of the function 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

$$a_{i1}(t)f_i + a_{i2}(t)\frac{\partial f_i}{\partial x} = a_{i3}(t). \quad (\text{C.104})$$

The coefficients  $a_{ij}$  are returned in the array A, which should be dimensioned as A[3][N] in the function BC. FIC is the name of the function, which may be used to calculate the initial values for the solution if IFLG = 0. Function FIC(N, X, T, U) must be supplied by the user, when IFLG = 0. This function calculates the solution  $u_i(X, T)$ , ( $i = 0, \dots, N - 1$ ) at the initial time T. For other values of IFLG a dummy function FIC may be required to satisfy the compiler or the linker. (In function FIC, U is an array of length N, which returns the value of the solution  $u_i$  for  $i = 0, \dots, N - 1$ .) IFLG is an integer variable used as a flag. If IFLG = 0, the initial values are calculated using the function FIC supplied by the user. For other values of IFLG, the initial values must be supplied in the array U. In any case, the function sets IFLG to 1, so that during subsequent calls to the function, the initial values are taken from the array U. This function 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. Error status can be obtained from the returned value of LAX. A value of 715 implies that DT = 0, or XN = X0, or NX ≤ 2, or IU < N, in which case, no calculations are performed. LAX = 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.

**213. SOR** Function to solve a linear second-order elliptic equation

$$\begin{aligned} A_{xx}(x, y)\frac{\partial^2 u}{\partial x^2} + A_{xy}(x, y)\frac{\partial^2 u}{\partial x \partial y} + A_{yy}(x, y)\frac{\partial^2 u}{\partial y^2} + A_x(x, y)\frac{\partial u}{\partial x} \\ + A_y(x, y)\frac{\partial u}{\partial y} + A_0(x, y)u + F(x, y) = 0, \end{aligned} \quad (\text{C.105})$$

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

$$\begin{aligned} A_{xx}\frac{2u_{jl} - u_{j-1,l} - u_{j+1,l}}{(\Delta x)^2} + A_{xy}\frac{u_{j+1,l-1} + u_{j-1,l+1} - u_{j+1,l+1} - u_{j-1,l-1}}{4\Delta x \Delta y} \\ + A_{yy}\frac{2u_{jl} - u_{j,l-1} - u_{j,l+1}}{(\Delta y)^2} + A_x\frac{u_{j-1,l} - u_{j+1,l}}{2\Delta x} + A_y\frac{u_{j,l-1} - u_{j,l+1}}{2\Delta y} \\ - A_0u_{jl} = F, \end{aligned} \quad (\text{C.106})$$

where all coefficients are evaluated at  $(x_j, y_l)$ . The points (X0, Y0) and (XN, 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 arrays of lengths NX and NY respectively, which contain the mesh points  $x_j$  and  $y_j$ . These arrays need not be initialised before calling the function, since the function assumes a uniform spacing and initialises these arrays before proceeding with the calculations. U is an array of length  $IU \times NY$ , which contains the solution. U[J][I] will contain the computed approximation to  $u(x_i, y_j)$ . Before calling the function, an initial approximation to the solution should be supplied in this array. IU ( $\geq NX$ ) is the value of the second dimension of array U, as declared in the calling function. COF is the name of the function used to calculate the coefficients in the equation. Function 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 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

$$\begin{aligned} u(X0, y) &= BC(1, X0, y), & u(XN, y) &= BC(2, XN, y), \\ u(x, Y0) &= BC(3, x, Y0), & u(x, YN) &= BC(4, x, YN). \end{aligned} \quad (C.107)$$

OMEGA is the value of the relaxation parameter  $\omega$  to be used in the calculation. For meaningful results  $1 < \omega < 2$ , but this condition is not checked by the function. If OMEGA  $\leq 0$ , then the function uses the optimum value for the Poisson's equation, based on the number of mesh points in each direction (Section 14.8). 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 function to achieve the specified accuracy. Error status can be obtained from the returned value of SOR. A value of 716 implies that  $XN = X0$ , or  $YN = Y0$ , or  $NX \leq 2$ , or  $NY \leq 2$ , or  $IU < NX$ , in which case, no calculations are performed. SOR = 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. SOR = 765 implies that the SOR iteration failed to converge in MAXIT (= 1000) iterations.

**214. ADI** Function to solve a linear second-order elliptic equation

$$\begin{aligned}
 &A_{xx}(x, y) \frac{\partial^2 u}{\partial x^2} + A_{yy}(x, y) \frac{\partial^2 u}{\partial y^2} + A_x(x, y) \frac{\partial u}{\partial x} \\
 &+ A_y(x, y) \frac{\partial u}{\partial y} + A_0(x, y)u + F(x, y) = 0,
 \end{aligned}
 \tag{C.108}$$

on a rectangular region using the Alternating Direction Implicit iterative (ADI) method. Although arbitrary coefficients will be accepted by this function, the result may be meaningful only when the resulting equation is elliptic, i.e.  $A_{xx}A_{yy} > 0$ . This function accepts more general boundary conditions of the form  $A_0u + A_n\partial_nu = F$ , where  $\partial_nu$  is the normal derivative. The points (X0, Y0) and (XN, YN) define the rectangle, over which the solution is required. KN is the parameter  $k$  in the ADI algorithm. This function 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 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 function, since the function assumes a uniform spacing and initialises these arrays before proceeding with the calculations. U is an array of length IU  $\times$  NY, which contains the solution. U[J][I] will contain the computed approximation to  $u(x_i, y_j)$ . Before calling the function an initial approximation to the solution should be supplied in this array. IU ( $\geq$  NX) is the value of the second dimension of the array U, as declared in the calling function. COF is the name of the function used to calculate the coefficients in the equation. Function 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 function to calculate the coefficients for the boundary conditions. Function 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  $x = X0$ ,  $x = XN$ ,  $y = Y0$  and  $y = YN$ , 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  $EL \leq 0$ , 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. 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 function to achieve the specified accuracy. Error status can be obtained from the returned value of ADI. A value of 717 implies that  $XN = X0$ , or  $YN = Y0$ , or  $NX \leq 2$ , or  $NY \leq 2$ , or  $IU < NX$ , in which case, no calculations are performed. ADI = 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. ADI = 767 implies that the ADI iteration failed to converge in MAXIT (= 1000) iterations.

## 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.
- Kernighan, B. W. and Ritchie, D. M. (1988): *The C Programming Language*, (2nd ed.), Prentice Hall, New Jersey.
- 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.