

# **NAG Fortran Library Manual**

**Mark 18**

**Volume 12**

**H – X05**

- H – Operations Research
- M01 – Sorting
- P01 – Error Trapping
- S – Approximations of Special Functions
- X01 – Mathematical Constants
- X02 – Machine Constants
- X03 – Innerproducts
- X04 – Input/Output Utilities
- X05 – Date and Time Utilities



**NAG Fortran Library Manual, Mark 18**

©The Numerical Algorithms Group Limited, 1997

All rights reserved. No part of this manual may be reproduced, transcribed, stored in a retrieval system, translated into any language or computer language or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the copyright owner.

The copyright owner gives no warranties and makes no representations about the contents of this manual and specifically disclaims any implied warranties or merchantability or fitness for any purpose.

The copyright owner reserves the right to revise this manual and to make changes from time to time in its contents without notifying any person of such revisions or changes.

Printed and produced by NAG

1st Edition – September 1997

ISBN 1-85206-147-2

NAG is a registered trademark of:

The Numerical Algorithms Group Limited  
The Numerical Algorithms Group Inc  
The Numerical Algorithms Group (Deutschland) GmbH

NAG Ltd  
Wilkinson House  
Jordan Hill Road  
OXFORD  
United Kingdom OX2 8DR

Tel: +44 (0)1865 511245  
Fax: +44 (0)1865 310139

NAG GmbH  
Schleißheimerstraße 5  
D-85748 Garching  
Deutschland

Tel: +49 (0)89 3207395  
Fax: +49 (0)89 3207396

NAG Inc  
1400 Opus Place, Suite 200  
Downers Grove, IL 60515-5702  
USA

Tel: +1 630 971 2337  
Fax: +1 630 971 2706

NAG also has a number of distributors throughout the world. Please contact NAG for further details.

## Chapter H – Operations Research

**Note.** Please refer to the Users' Note for your implementation to check that a routine is available.

<b>Routine Name</b>	<b>Mark of Introduction</b>	<b>Purpose</b>
H02BBF	14	Integer programming problem, branch and bound method
H02BFF	16	Interpret MPSX data file defining IP or LP problem, optimize and print solution
H02BUF	16	Converts MPSX data file defining IP or LP problem to format required by H02BBF or E04MFF
H02BVF	16	Prints IP or LP solutions with user specified names for rows and columns
H02BZF	15	Integer programming solution, supplies further information on solution obtained by H02BBF
H03ABF	4	Transportation problem, modified 'stepping stone' method
H03ADF	18	Shortest path problem, Dijkstra's algorithm

---





# **Chapter H**

## **Operations Research**

### **Contents**

<b>1</b>	<b>Scope of the Chapter</b>	<b>2</b>
<b>2</b>	<b>Background to the Problems</b>	<b>2</b>
<b>3</b>	<b>Recommendations on Choice and Use of Available Routines</b>	<b>4</b>
<b>4</b>	<b>Routines Withdrawn or Scheduled for Withdrawal</b>	<b>4</b>
<b>5</b>	<b>References</b>	<b>5</b>

## 1 Scope of the Chapter

This chapter provides routines to solve certain integer programming, transportation and shortest path problems.

## 2 Background to the Problems

General **linear programming** (LP) problems (see Dantzig [2]) are of the form:

$$\text{find } x = (x_1, x_2, \dots, x_n)^T \text{ to maximize } F(x) = \sum_{j=1}^n c_j x_j$$

subject to linear constraints which may have the forms:

$$\sum_{j=1}^n a_{ij} x_j = b_i, \quad i = 1, 2, \dots, m_1 \quad (\text{equality})$$

$$\sum_{j=1}^n a_{ij} x_j \leq b_i, \quad i = m_1 + 1, \dots, m_2 \quad (\text{inequality})$$

$$\sum_{j=1}^n a_{ij} x_j \geq b_i, \quad i = m_2 + 1, \dots, m \quad (\text{inequality})$$

$$x_j \geq l_j, \quad j = 1, 2, \dots, n \quad (\text{simple bound})$$

$$x_j \leq u_j, \quad j = 1, 2, \dots, n \quad (\text{simple bound})$$

This chapter deals with **integer programming** (IP) problems in which some or all the elements of the solution vector  $x$  are further constrained to be **integers**. For general LP problems where  $x$  takes only real (i.e., non-integer) values, refer to Chapter E04.

IP problems may or may not have a solution, which may or may not be unique.

Consider for example the following problem:

$$\begin{aligned} & \text{minimize} && 3x_1 + 2x_2 \\ & \text{subject to} && 4x_1 + 2x_2 \geq 5 \\ & && 2x_2 \leq 5 \\ & && x_1 - x_2 \leq 2 \\ & \text{and} && x_1 \geq 0, x_2 \geq 0. \end{aligned}$$

The shaded region in Figure 1 is the **feasible region**, the region where all the constraints are satisfied, and the points within it which have integer co-ordinates are circled. The lines of shading are in fact contours of decreasing values of the objective function  $3x_1 + 2x_2$ , and it is clear from Figure 1 that the optimum IP solution is at the point (1,1). For this problem the solution is unique.

However, there are other possible situations:

- there may be more than one solution; e.g. if the objective function in the above problem were changed to  $x_1 + x_2$ , both (1,1) and (2,0) would be IP solutions.
- the feasible region may contain no points with integer co-ordinates, e.g. if an additional constraint

$$3x_1 \leq 2$$

were added to the above problem.

- there may be no feasible region, e.g. if an additional constraint

$$x_1 + x_2 \leq 1$$

were added to the above problem.

- the objective function may have no finite minimum within the feasible region; this means that the feasible region is unbounded in the direction of decreasing values of the objective function, e.g. if the constraints

$$4x_1 + 2x_2 \geq 5, \quad x_1 \geq 0, \quad x_2 \geq 0,$$

were deleted from the above problem.

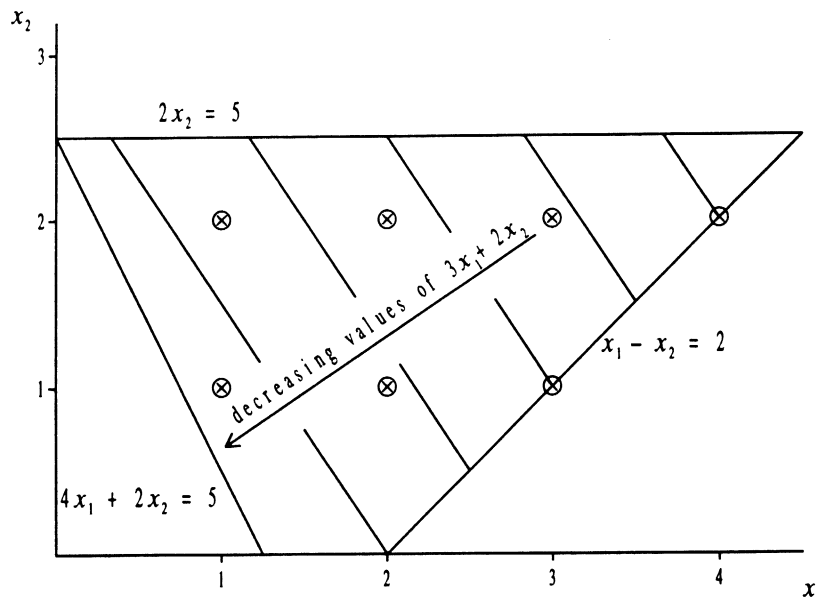


Figure 1

Algorithms for IP problems are usually based on algorithms for general LP problems, together with some procedure for constructing additional constraints which exclude non-integer solutions (see Beale [1]).

The Branch and Bound (B&B) method is a well-known and widely used technique for solving IP problems (see Beale [1] or Mitra [3]). It involves subdividing the optimum solution to the original LP problem into two mutually exclusive sub-problems by branching an integer variable that currently has a fractional optimal value. Each sub-problem can now be solved as an LP problem, using the objective function of the original problem. The process of branching continues until a solution for one of the sub-problems is feasible with respect to the integer problem. In order to prove the optimality of this solution, the rest of the sub-problems in the B&B tree must also be solved. Naturally, if a better integer feasible solution is found for any sub-problem, it should replace the one at hand.

A common method for specifying IP and LP problems in general is the use of the MPSX file format (see [4]). A full description of this file format is provided in the routine documents for H02BUF and E04MZF.

The efficiency in computations is enhanced by discarding inferior sub-problems. These are problems in the B&B search tree whose LP solutions are lower than (in the case of maximization) the best integer solution at hand.

A special type of linear programming problem is the **transportation** problem in which there are  $p \times q$  variables  $y_{kl}$  which represent quantities of goods to be transported from each of  $p$  sources to each of  $q$  destinations.

The problem is to minimize

$$\sum_{k=1}^p \sum_{l=1}^q c_{kl} y_{kl}$$

where  $c_{kl}$  is the unit cost of transporting from source  $k$  to destination  $l$ . The constraints are:

$$\sum_{l=1}^q y_{kl} = A_k \quad (\text{availabilities})$$

$$\sum_{k=1}^p y_{kl} = B_l \quad (\text{requirements})$$

$$y_{kl} \geq 0.$$

Note that the availabilities must equal the requirements:

$$\sum_{k=1}^p A_k = \sum_{l=1}^q B_l = \sum_{k=1}^p \sum_{l=1}^q y_{kl}$$

and if all the  $A_k$  and  $B_l$  are integers, then so are the optimal  $y_{kl}$ .

The **shortest path** problem is that of finding a path of minimum length between two distinct vertices  $n_s$  and  $n_e$  through a network. Suppose the vertices in the network are labelled by the integers  $1, 2, \dots, n$ . Let  $(i, j)$  denote an ordered pair of vertices in the network (where  $i$  is the origin vertex and  $j$  the destination vertex of the arc),  $x_{ij}$  the amount of flow in arc  $(i, j)$  and  $d_{ij}$  the length of the arc  $(i, j)$ . The LP formulation of the problem is thus given as

$$\text{minimize } \sum \sum d_{ij} x_{ij} \text{ subject to } Ax = b, \quad 0 \leq x \leq 1, \quad (1)$$

where

$$a_{ij} = \begin{cases} +1 & \text{if arc } j \text{ is directed away from vertex } i, \\ -1 & \text{if arc } j \text{ is directed towards vertex } i, \\ 0 & \text{otherwise} \end{cases}$$

and

$$b_i = \begin{cases} +1 & \text{for } i = n_s, \\ -1 & \text{for } i = n_e, \\ 0 & \text{otherwise.} \end{cases}$$

The above formulation only yields a meaningful solution if  $x_{ij} = 0$  or  $1$ ; that is, arc  $(i, j)$  forms part of the shortest route only if  $x_{ij} = 1$ . In fact since the optimal LP solution will (in theory) always yield  $x_{ij} = 0$  or  $1$ , (1) can also be solved as an IP problem. Note that the problem may also be solved directly (and more efficiently) using a variant of Dijkstra's algorithm (see [6]).

The **travelling salesman** problem is that of finding a minimum distance route round a given set of cities. The salesperson must visit each city only once before returning to his or her city of origin. It can be formulated as an IP problem in a number of ways. One such formulation is described in Williams [5]. There are currently no routines in the Library for solving such problems.

### 3 Recommendations on Choice and Use of Available Routines

**Note.** Refer to the Users' Note for your implementation to check that a routine is available.

H02BBF solves integer programming problems using a branch and bound method.

H02BFF solves integer or linear programming problems defined by a MPSX data file.

H02BUF converts an MPSX data file defining an integer or a linear programming problem to the form required by H02BBF or E04MFF.

H02BVF prints the solution to an integer or a linear programming problem using specified names for rows and columns.

H02BZF supplies further information on the optimum solution obtained by H02BBF.

H03ABF solves transportation problems. It uses integer arithmetic throughout and so produces exact results. On a few machines, however, there is a risk of integer overflow without warning, so the integer values in the data should be kept as small as possible by dividing out any common factors from the coefficients of the constraint or objective functions.

H03ADF solves shortest path problems using Dijkstra's algorithm.

H02BBF, H02BFF and H03ABF treat all matrices as dense and hence are not intended for large sparse problems. For solving large sparse LP problems, use E04NKF.

### 4 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have been withdrawn. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

H02BAF

## 5 References

- [1] Beale E M (1977) Integer Programming *The State of the Art in Numerical Analysis* (ed D A H Jacobs) Academic Press
  - [2] Dantzig G B (1963) *Linear Programming and Extensions* Princeton University Press
  - [3] Mitra G (1973) Investigation of some branch and bound strategies for the solution of mixed integer linear programs *Math. Programming* 4 155-170
  - [4] (1971) MPSX - Mathematical programming system *Program Number 5734 XM4* IBM Trade Corporation, New York
  - [5] Williams H P (1990) *Model Building in Mathematical Programming* (3rd Edition) Wiley
  - [6] Ahuja R K, Magnanti T L and Orlin J B (1993) *Network Flows: Theory, Algorithms, and Applications* Prentice Hall
-



## H02BBF – NAG Fortran Library Routine Document

**Note.** Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

**Warning.** The specification of the parameters BIGBND and LIWORK changed at Mark 16: the 'default' value of the parameter BIGBND has been increased to  $10^{20}$  and the minimum dimension of the array IWORK has been increased by  $N + 3$ .

### 1 Purpose

H02BBF solves 'zero-one', 'general', 'mixed' or 'all' integer programming problems using a branch and bound method. The routine may also be used to find either the first integer solution or the optimum integer solution. It is not intended for large sparse problems.

### 2 Specification

```

SUBROUTINE H02BBF(ITMAX, MSGLVL, N, M, A, LDA, BL, BU, INTVAR,
1          CVEC, MAXNOD, INTFST, MAXDPT, TOLIV, TOLFES,
2          BIGBND, X, OBJMIP, IWORK, LIWORK, RWORK, LRWORK,
3          IFAIL)
  INTEGER  ITMAX, MSGLVL, N, M, LDA, INTVAR(N), MAXNOD,
1          INTFST, MAXDPT, IWORK(LIWORK), LIWORK, LRWORK,
2          IFAIL
  real    A(LDA,*), BL(N+M), BU(N+M), CVEC(N), TOLIV,
1          TOLFES, BIGBND, X(N), OBJMIP, RWORK(LRWORK)

```

### 3 Description

H02BBF is capable of solving certain types of integer programming (IP) problems using a branch and bound (B&B) method, see Taha [1]. In order to describe these types of integer programs and to briefly state the B&B method, we define the following linear programming (LP) problem:

Minimize

$$F(x) = c_1x_1 + c_2x_2 + \dots + c_nx_n$$

subject to

$$\sum_{j=1}^n a_{ij}x_j \begin{cases} = \\ \leq \\ \geq \end{cases} b_i, \quad i = 1, 2, \dots, m$$

$$l_j \leq x_j \leq u_j, \quad j = 1, 2, \dots, n \quad (1)$$

If, in (1), it is required that (some or) all the variables take integer values, then the integer program is of type (*mixed* or) *all* general IP problem. If additionally, the integer variables are restricted to take only 0-1 values (i.e.,  $l_j = 0$  and  $u_j = 1$ ) then the integer program is of type (mixed or all) *zero-one* IP problem.

The B&B method applies directly to these integer programs. The general idea of B&B (for a full description see Dakin [2] or Mitra [3]) is to solve the problem without the integral restrictions as an LP problem (first *node*). If in the optimal solution an integer variable  $x_k$  takes a non-integer value  $x_k^*$ , two LP sub-problems are created by *branching*, imposing  $x_k \leq [x_k^*]$  and  $x_k \geq [x_k^*] + 1$  respectively, where  $[x_k^*]$  denotes the integer part of  $x_k^*$ . This method of branching continues until the first integer solution (*bound*) is obtained. The hanging nodes are then solved and investigated in order to prove the optimality of the solution. At each node, an LP problem is solved using E04MFF.

## 4 References

- [1] Taha H A (1987) *Operations Research: An Introduction* Macmillan, New York
- [2] Dakin R J (1965) A tree search algorithm for mixed integer programming problems *Comput. J.* **8** 250-255
- [3] Mitra G (1973) Investigation of some branch and bound strategies for the solution of mixed integer linear programs *Math. Programming* **4** 155-170

## 5 Parameters

- 1:** ITMAX — INTEGER *Input/Output*  
*On entry:* an upper bound on the number of iterations for each LP problem.  
*On exit:* unchanged if on entry ITMAX > 0, else ITMAX = max(50, 5 × (N+M)).
- 2:** MSGLVL — INTEGER *Input*  
*On entry:* the amount of printout produced by H02BBF, as indicated below (see Section 5.1 for a description of the printed output). All output is written to the current advisory message unit (as defined by X04ABF).
- | Value | Definition  |
|-------|---|
| 0     | No output.  |
| 1     | The final IP solution only.   |
| 5     | One line of output for each node investigated and the final IP solution.  |
| 10    | The original LP solution (first node), one line of output for each node investigated and the final IP solution. |
- 3:** N — INTEGER *Input*  
*On entry:*  $n$ , the number of variables.  
*Constraint:*  $N > 0$ .
- 4:** M — INTEGER *Input*  
*On entry:*  $m$ , the number of general linear constraints.  
*Constraint:*  $M \geq 0$ .
- 5:** A(LDA,\*) — *real* array *Input*  
**Note:** the second dimension of the array A must be at least N when  $M > 0$ , and at least 1 when  $M = 0$ .  
*On entry:* the  $i$ th row of A must contain the coefficients of the  $i$ th general constraint, for  $i = 1, 2, \dots, m$ .  
 If  $M = 0$  then the array A is not referenced.
- 6:** LDA — INTEGER *Input*  
*On entry:* the first dimension of the array A as declared in the (sub)program from which H02BBF is called.  
*Constraint:*  $LDA \geq \max(1, M)$ .



- 7: BL(N+M) — *real* array *Input*
- 8: BU(N+M) — *real* array *Input*
- On entry:* BL must contain the lower bounds and BU the upper bounds, for all the constraints in the following order. The first  $n$  elements of each array must contain the bounds on the variables, and the next  $m$  elements the bounds for the general linear constraints (if any). To specify a non-existent lower bound (i.e.,  $l_j = -\infty$ ), set  $BL(j) \leq -BIGBND$  and to specify a non-existent upper bound (i.e.,  $u_j = +\infty$ ), set  $BU(j) \geq BIGBND$ . To specify the  $j$ th constraint as an equality, set  $BL(j) = BU(j) = \beta$ , say, where  $|\beta| < BIGBND$ .
- Constraints:*
- $$BL(j) \leq BU(j), \text{ for } j = 1, 2, \dots, N + M,$$
- $$|\beta| < BIGBND \text{ when } BL(j) = BU(j) = \beta.$$
- 9: INTVAR(N) — INTEGER array *Input*
- On entry:* indicates which are the integer variables in the problem. For example, if  $x_j$  is an integer variable then  $INTVAR(j)$  must be set to 1, and 0 otherwise.
- Constraints:*
- $$INTVAR(j) = 0 \text{ or } 1 \text{ for } j = 1, 2, \dots, N, \text{ and}$$
- $$INTVAR(j) = 1 \text{ for at least one value of } j.$$
- 10: CVEC(N) — *real* array *Input*
- On entry:* the coefficients  $c_j$  of the objective function  $F(\mathbf{x}) = c_1x_1 + c_2x_2 + \dots + c_nx_n$ . The routine attempts to find a minimum of  $F(\mathbf{x})$ . If a maximum of  $F(\mathbf{x})$  is desired,  $CVEC(j)$  should be set to  $-c_j$ , for  $j = 1, 2, \dots, n$ , so that the routine will find a minimum of  $-F(\mathbf{x})$ .
- 11: MAXNOD — INTEGER *Input*
- On entry:* the maximum number of nodes that are to be searched in order to find a solution (optimum integer solution). If  $MAXNOD \leq 0$  and  $INTFST \leq 0$ , then the B&B tree search is continued until all the nodes have been investigated.
- 12: INTFST — INTEGER *Input*
- On entry:* specifies whether to terminate the B&B tree search after the first integer solution (if any) is obtained. If  $INTFST > 0$  then the B&B tree search is terminated at node  $k$  say, which contains the first integer solution. For  $MAXNOD > 0$  this applies only if  $k \leq MAXNOD$ .
- 13: MAXDPT — INTEGER *Input*
- On entry:* the maximum depth of the B&B tree used for branch and bound.
- Suggested value:*  $MAXDPT = 3 \times N/2$ .
- Constraint:*  $MAXDPT \geq 2$ .
- 14: TOLIV — *real* *Input/Output*
- On entry:* the integer feasibility tolerance; i.e., an integer variable is considered to take an integer value if its violation does not exceed TOLIV. For example, if the integer variable  $x_j$  is of order unity then  $x_j$  is considered to be integer only if  $(1-TOLIV) \leq x_j \leq (1+TOLIV)$ .
- On exit:* unchanged if on entry  $TOLIV > 0.0$ , else  $TOLIV = 10^{-5}$ .
- 15: TOLFES — *real* *Input/Output*
- On entry:* the maximum acceptable absolute violation in each constraint at a 'feasible' point (feasibility tolerance); i.e., a constraint is considered satisfied if its violation does not exceed TOLFES.
- On exit:* unchanged if on entry  $TOLFES > 0.0$ , else  $TOLFES = \sqrt{\epsilon}$  (where  $\epsilon$  is the *machine precision*).

- 16:** BIGBND — *real* *Input/Output*  
*On entry:* the ‘infinite’ bound size in the definition of the problem constraints. More precisely, any upper bound greater than or equal to BIGBND will be regarded as  $+\infty$  and any lower bound less than or equal to  $-\text{BIGBND}$  will be regarded as  $-\infty$ . ~  
*On exit:* unchanged if on entry  $\text{BIGBND} > 0.0$ , else  $\text{BIGBND} = 10^{20}$ .
- 17:** X(N) — *real* array *Input/Output*  
*On entry:* an initial estimate of the original LP solution.  
*On exit:* with IFAIL = 0, X contains a solution which will be an estimate of either the optimum integer solution or the first integer solution, depending on the value of INTFST. If IFAIL = 9, then X contains a solution which will be an estimate of the best integer solution that was obtained by searching MAXNOD nodes.
- 18:** OBJMIP — *real* *Output*  
*On exit:* with IFAIL = 0 or 9, OBJMIP contains the value of the objective function for the IP solution.
- 19:** IWORK(LIWORK) — INTEGER array *Workspace*
- 20:** LIWORK — INTEGER *Input*  
*On entry:* the dimension of the array IWORK as declared in the (sub)program from which H02BBF is called.  
*Constraint:*  $\text{LIWORK} \geq (25+N+M) \times \text{MAXDPT} + 5 \times N + M + 4$ .
- 21:** RWORK(LRWORK) — *real* array *Workspace*
- 22:** LRWORK — INTEGER *Input*  
*On entry:* the dimension of the array RWORK as declared in the (sub)program from which H02BBF is called.  
*Constraint:*  $\text{LRWORK} \geq \text{MAXDPT} \times (N+1) + 2 \times \text{MIN}(N,M+1)^2 + 14 \times N + 12 \times M$ .
- If MSGLVL > 0, the amounts of workspace provided and required (with  $\text{MAXDPT} = 3 \times N/2$ ) are printed. As an alternative to computing MAXDPT, LIWORK and LRWORK from the formulas given above, the user may prefer to obtain appropriate values from the output of a preliminary run with the values of MAXDPT, LIWORK and LRWORK set to 1. If however only LIWORK and LRWORK are set to 1, then the appropriate values of these parameters for the given value of MAXDPT will be computed and printed unless  $\text{MAXDPT} < 2$ . In both cases H02BBF will then terminate with IFAIL = 6.
- 23:** IFAIL — INTEGER *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. Users who are unfamiliar with this parameter should refer to Chapter P01 for details.  
*On exit:* IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).
- For this routine,** because the values of output parameters may be useful even if IFAIL  $\neq$  0 on exit, users are recommended to set IFAIL to -1 before entry. **It is then essential to test the value of IFAIL on exit.**

## 5.1 Description of Printed Output

The level of printed output from H02BBF is controlled by the user (see the description of MSGLVL in Section 5).

When  $MSGLVL > 0$ , the summary printout at the end of execution of H02BBF includes a listing of the status of every variable and constraint. Note that default names are assigned to all variables and constraints. The following describes the printout for each variable.

<b>Varbl</b>	gives the name ( <b>V</b> ) and index $j$ , for $j = 1, 2, \dots, n$ of the variable.
<b>State</b>	gives the state of the variable ( <b>FR</b> if neither bound is in the working set, <b>EQ</b> if a fixed variable, <b>LL</b> if on its lower bound, <b>UL</b> if on its upper bound, <b>TF</b> if temporarily fixed at its current value). If <b>Value</b> lies outside the upper or lower bounds by more than the feasibility tolerance, <b>State</b> will be <b>++</b> or <b>--</b> respectively.
<b>Value</b>	is the value of the variable at the final iteration.
<b>Lower Bound</b>	is the lower bound specified for the variable. ( <b>None</b> indicates that $BL(j) \leq -BIGBND$ .) Note that if $INTVAR(j) = 1$ , then the printed value of <b>Lower Bound</b> for the $j$ th variable may not be the same as that originally supplied in $BL(j)$ .
<b>Upper Bound</b>	is the upper bound specified for the variable. ( <b>None</b> indicates that $BU(j) \geq BIGBND$ .) Note that if $INTVAR(j) = 1$ , then the printed value of <b>Upper Bound</b> for the $j$ th variable may not be the same as that originally supplied in $BU(j)$ .
<b>Lagr Mult</b>	is the value of the Lagrange multiplier for the associated bound constraint. This will be zero if <b>State</b> is <b>FR</b> or <b>TF</b> . If $x$ is optimal, the multiplier should be non-negative if <b>State</b> is <b>LL</b> , and non-positive if <b>State</b> is <b>UL</b> .
<b>Residual</b>	is the difference between the variable <b>Value</b> and the nearer of its bounds $BL(j)$ and $BU(j)$ .

The meaning of the printout for general constraints is the same as that given above for variables, except that ‘variable’ is replaced by ‘constraint’,  $BL(j)$  and  $BU(j)$  are replaced by  $BL(n + j)$  and  $BU(n + j)$  respectively, and with the following change in the heading.

<b>L Con</b>	gives the name ( <b>L</b> ) and index $j$ , for $j = 1, 2, \dots, m$ of the constraint.
--------------	---

When  $MSGLVL > 1$ , the summary printout at the end of every node during the execution of H02BBF is a listing of the outcome of forcing an integer variable with a non-integer value to take a value within its specified lower and upper bounds.

<b>Node No</b>	is the current node number of the B&B tree being investigated.
<b>Parent Node</b>	is the parent node number of the current node.
<b>Obj Value</b>	is the final objective function value. If a node does not have a feasible solution then <b>No Feas Soln</b> is printed instead of the objective function value. If a node whose optimum solution exceeds the best integer solution so far is encountered (i.e., it does not pay to explore the sub-problem any further), then its objective function value is printed together with a <b>CO</b> (Cut Off).
<b>Varbl Chosen</b>	is the index of the integer variable chosen for branching.
<b>Value Before</b>	is the non-integer value of the integer variable chosen.
<b>Lower Bound</b>	is the lower bound value that the integer variable is allowed to take.
<b>Upper Bound</b>	is the upper bound value that the integer variable is allowed to take.
<b>Value After</b>	is the value of the integer variable after the current optimization.
<b>Depth</b>	is the depth of the B&B tree at the current node.

## 6 Error Indicators and Warnings

If on entry  $IFAIL = 0$  or  $-1$ , explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings specified by the routine:

## IFAIL = 1

No feasible integer point was found, i.e., it was not possible to satisfy all the integer variables to within the integer feasibility tolerance (determined by TOLIV). Increase TOLIV and rerun H02BBF.

## IFAIL = 2

The original LP solution appears to be unbounded. This value of IFAIL implies that a step as large as BIGBND would have to be taken in order to continue the algorithm (see Section 8).

## IFAIL = 3

No feasible point was found for the original LP problem, i.e., it was not possible to satisfy all the constraints to within the feasibility tolerance (determined by TOLFES). If the data for the constraints are accurate only to the absolute precision  $\sigma$ , the user should ensure that the value of the feasibility tolerance is greater than  $\sigma$ . For example, if all elements of  $A$  are of order unity and are accurate only to three decimal places, the feasibility tolerance should be at least  $10^{-3}$  (see Section 8).

## IFAIL = 4

The maximum number of iterations (determined by ITMAX) was reached before normal termination occurred for the original LP problem (see Section 8).

## IFAIL = 5

Not used by this routine.

## IFAIL = 6

An input parameter is invalid.

## IFAIL = 7

The IP solution reported is not the optimum IP solution. In other words, the B&B tree search for at least one of the branches had to be terminated since an LP sub-problem in the branch did not have a solution (see Section 8).

## IFAIL = 8

The maximum depth of the B&B tree used for branch and bound (determined by MAXDPT) is too small. Increase MAXDPT (along with LIWORK and/or LRWORK if appropriate) and rerun H02BBF.

## IFAIL = 9

The IP solution reported is the best IP solution for the number of nodes (determined by MAXNOD) investigated in the B&B tree.

## IFAIL = 10

No feasible integer point was found for the number of nodes (determined by MAXNOD) investigated in the B&B tree.

## Overflow

It may be possible to avoid the difficulty by increasing the magnitude of the feasibility tolerance (TOLFES) and rerunning the program. If the message recurs even after this change, see Section 8.

## 7 Accuracy

The routine implements a numerically stable active set strategy and returns solutions that are as accurate as the condition of the problem warrants on the machine.

## 8 Further Comments

The original LP problem may not have an optimum solution, i.e., H02BBF terminates with IFAIL = 2,3,4 or overflow may occur. In this case, the user is recommended to relax the integer restrictions of the problem and try to find the optimum LP solution by using E04MFF instead.

In the B&B method, it is possible for an LP sub-problem to terminate without finding a solution. This may occur due to the number of iterations exceeding the maximum allowed. Therefore the B&B tree search for that particular branch cannot be continued. Thus the final IP solution reported is not the optimum IP solution (IFAIL = 7). For the second and unlikely case, a solution could not be found despite a second attempt at an LP solution.

## 9 Example

To solve the integer programming problem:

maximize

$$F(x) = 3x_1 + 4x_2$$

subject to the bounds

$$\begin{aligned} x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

and to the general constraints

$$\begin{aligned} 2x_1 + 5x_2 &\leq 15 \\ 2x_1 - 2x_2 &\leq 5 \\ 3x_1 + 2x_2 &\geq 5 \end{aligned}$$

where  $x_1$  and  $x_2$  are integer variables.

The initial point, which is feasible, is

$$x_0 = (1, 1)^T,$$

and  $F(x_0) = 7$ .

The optimal solution is

$$x^* = (2, 2)^T,$$

and  $F(x^*) = 14$ .

Note that maximizing  $F(x)$  is equivalent to minimizing  $-F(x)$ .

### 9.1 Program Text

**Note.** The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      H02BBF Example Program Text
*      Mark 16 Revised. NAG Copyright 1993.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
      INTEGER          NMAX, MMAX
      PARAMETER       (NMAX=10,MMAX=10)
      INTEGER          LDA
      PARAMETER       (LDA=MMAX)
      INTEGER          LIWORK, LRWORK
      PARAMETER       (LIWORK=1000,LRWORK=1000)
*      .. Local Scalars ..
      real            BIGBND, OBJMIP, TOLFES, TOLIV
      INTEGER          I, IFAIL, INTFST, ITMAX, J, M, MAXDPT, MAXNOD,
+                    MSGlvl, N

```

```

*   .. Local Arrays ..
  real          A(LDA,NMAX), BL(MMAX+NMAX), BU(MMAX+NMAX),
+             CVEC(NMAX), RWORK(LRWORK), X(NMAX)
  INTEGER       INTVAR(NMAX), IWORK(LIWORK)
*   .. External Subroutines ..
  EXTERNAL      H02BBF
*   .. Executable Statements ..
  WRITE (NOUT,*) 'H02BBF Example Program Results'
*   Skip heading in data file
  READ (NIN,*)
  READ (NIN,*) N, M
  IF (N.LE.NMAX .AND. M.LE.MMAX) THEN
*
*       Read ITMAX, MSGLVL, MAXNOD, INTFST, MAXDPT, TOLFES, TOLIV,
*       CVEC, A, BIGBND, BL, BU, INTVAR and X from data file
*
  READ (NIN,*) ITMAX, MSGLVL
  READ (NIN,*) MAXNOD
  READ (NIN,*) INTFST, MAXDPT
  READ (NIN,*) TOLFES, TOLIV
  READ (NIN,*) (CVEC(I),I=1,N)
  READ (NIN,*) ((A(I,J),J=1,N),I=1,M)
  READ (NIN,*) BIGBND
  READ (NIN,*) (BL(I),I=1,N+M)
  READ (NIN,*) (BU(I),I=1,N+M)
  READ (NIN,*) (INTVAR(I),I=1,N)
  READ (NIN,*) (X(I),I=1,N)
*
*       Solve the IP problem
*
  IFAIL = -1
*
  CALL H02BBF(ITMAX,MSGLVL,N,M,A,LDA,BL,BU,INTVAR,CVEC,MAXNOD,
+           INTFST,MAXDPT,TOLIV,TOLFES,BIGBND,X,OBJMIP,IWORK,
+           LIWORK,RWORK,LRWORK,IFAIL)
*
  END IF
  STOP
  END

```

## 9.2 Program Data

### H02BBF Example Program Data

2	3										:Values of N and M
0	10										:Values of ITMAX and MSGLVL
0											:Value of MAXNOD
0	4										:Values of INTFST and MAXDPT
0.0	0.0										:Values of TOLFES and TOLIV
-3.0	-4.0										:End of CVEC
2.0	5.0										
2.0	-2.0										
3.0	2.0										:End of matrix A
1.0E+20											:Value of BIGBND
0.0	0.0	-1.0E+20	-1.0E+20	5.0							:End of BL
1.0E+20	1.0E+20	15.0	5.0	1.0E+20							:End of BU
1	1										:End of INTVAR
1.0	1.0										:End of X

### 9.3 Program Results

H02BBF Example Program Results

\*\*\* H02BBF  
 \*\*\* Start of NAG Library implementation details \*\*\*

Implementation title: Generalised Base Version  
 Precision: FORTRAN double precision  
 Product Code: FLBAS18D  
 Mark: 18A

\*\*\* End of NAG Library implementation details \*\*\*

Parameters  
 -----

Linear constraints.....	3	First integer solution..	OFF
Variables.....	2	Max depth of the tree...	4
Feasibility tolerance...	1.05D-08	Print level.....	10
Infinite bound size.....	1.00D+20	EPS (machine precision).	1.11D-16
Integer feasibility tol.	1.00D-05	Iteration limit.....	50
Max number of nodes.....	NONE		

\*\* Workspace provided with MAXDPT = 4: LRWORK = 1000 LIWORK = 1000  
 \*\* Workspace required with MAXDPT = 4: LRWORK = 84 LIWORK = 137

\*\*\* Optimum LP solution \*\*\* -17.50000

Varbl	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
V 1	FR	3.92857	0.000000E+00	None	0.0000E+00	3.929
V 2	FR	1.42857	0.000000E+00	None	0.0000E+00	1.429

L Con	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
L 1	UL	15.0000	None	15.0000	-1.0000	0.0000E+00
L 2	UL	5.00000	None	5.00000	-0.5000	-8.8818E-16
L 3	FR	14.6429	5.00000	None	0.0000E+00	9.643

\*\*\* Start of tree search \*\*\*

Node No	Parent Node	Obj Value	Varbl Chosen	Value Before	Lower Bound	Upper Bound	Value After	Depth
2	1	No Feas Soln	1	3.93	4.00	None	4.00	1
3	1	-16.2	1	3.93	0.000E+00	3.00	3.00	1
4	3	-15.5	2	1.80	2.00	None	2.00	2
5	3	-13.0	2	1.80	0.000E+00	1.00	1.00	2

\*\*\* Integer solution \*\*\*

Node No	Parent Node	Obj Value	Varbl Chosen	Value Before	Lower Bound	Upper Bound	Value After	Depth
6	4	No Feas Soln	1	2.50	3.00	3.00	3.00	3
7	4	-14.8	1	2.50	0.000E+00	2.00	2.00	3
8	7	-12.0	CO 2	2.20	3.00	None	3.00	4
9	7	-14.0	2	2.20	2.00	2.00	2.00	4

\*\*\* Integer solution \*\*\*

\*\*\* End of tree search \*\*\*

Total of 9 nodes investigated.

Exit H02BBF - Optimum IP solution found.

Final IP objective value = -14.00000

Varbl	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
V 1	UL	2.00000	0.000000E+00	2.00000	-3.000	0.0000E+00
V 2	EQ	2.00000	2.00000	2.00000	-4.000	0.0000E+00

L Con	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
L 1	FR	14.0000	None	15.0000	0.0000E+00	1.000
L 2	FR	0.000000E+00	None	5.00000	0.0000E+00	5.000
L 3	FR	10.0000	5.00000	None	0.0000E+00	5.000



## H02BFF – NAG Fortran Library Routine Document

Note. Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

### 1 Purpose

H02BFF solves linear or integer programming problems specified in MPSX input format. It is not intended for large sparse problems.

### 2 Specification

```

SUBROUTINE H02BFF(INFILE, MAXN, MAXM, OPTIM, XBLDEF, XBUDEF,
1             MAXDPT, MSGLVL, N, M, X, CRNAME, IWORK, LIWORK,
2             RWORK, LRWORK, IFAIL)
INTEGER      INFILE, MAXN, MAXM, MAXDPT, MSGLVL, N, M,
1             IWORK(LIWORK), LIWORK, LRWORK, IFAIL
real        XBLDEF, XBUDEF, X(MAXN), RWORK(LRWORK)
CHARACTER*3  OPTIM
CHARACTER*8  CRNAME(MAXN+MAXM)

```

### 3 Description

H02BFF solves linear programming (LP) or integer programming (IP) problems specified in MPSX [1] input format. It calls either E04MFF (to solve an LP problem) or H02BBF and H02BZF (to solve an IP problem); these routines are designed to solve problems of the form

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad c^T x \quad \text{subject to} \quad l \leq \begin{pmatrix} x \\ Ax \end{pmatrix} \leq u$$

where  $c$  is an  $n$  element vector and  $A$  is an  $m$  by  $n$  matrix (i.e., there are  $n$  variables and  $m$  general linear constraints). H02BBF is used if at least one of the variables is restricted to take an integer value at the optimum solution. The document for H02BUF should be consulted for a detailed description of the MPSX format.

In the MPSX data file the first free row, that is, a row defined with the row type N, is taken as the objective row. Similarly, if there are more than one RHS, RANGES or BOUNDS sets, then the first set is used for the optimization. H02BFF also prints the solution to the problem using the row and column names specified in the MPSX data file (by calling H02BVF).

### 4 References

- [1] (1971) MPSX – Mathematical programming system *Program Number 5734 XM4* IBM Trade Corporation, New York

### 5 Parameters

- 1: INFILE — INTEGER *Input*  
*On entry:* the unit number associated with the MPSX data file.  
*Constraint:*  $0 \leq \text{INFILE} \leq 99$ .
- 2: MAXN — INTEGER *Input*  
*On entry:* an upper limit for the number of variables in the problem.  
*Constraint:*  $\text{MAXN} \geq 1$ .

- 3: MAXM — INTEGER** *Input*  
*On entry:* an upper limit for the number of constraints (including the objective) in the problem.  
*Constraint:* MAXM  $\geq$  1.
- 4: OPTIM — CHARACTER\*3** *Input*  
*On entry:* specifies the direction of the optimization. OPTIM must be set to 'MIN' for minimization and to 'MAX' for maximization.  
*Constraint:* OPTIM = 'MIN' or 'MAX'.
- 5: XBLDEF — real** *Input*  
*On entry:* the default lower bound to be used for the variables in the problem, when none is specified in the BOUNDS section of the MPSX data file. For a standard LP or IP problem XBLDEF would normally be set to zero.
- 6: XBUDEF — real** *Input*  
*On entry:* the default upper bound to be used for the variables in the problem, when none is specified in the BOUNDS section of the MPSX data file. For a standard LP or IP problem XBUDEF would normally be set to 'infinity' (i.e., XBUDEF  $\geq 10^{20}$ ).  
*Constraint:* XBUDEF  $\geq$  XBLDEF.
- 7: MAXDPT — INTEGER** *Input*  
*On entry:* for an IP problem, MAXDPT must specify the maximum depth of the branch and bound tree.  
*Constraint:* MAXDPT  $\geq$  2.  
 For an LP problem, MAXDPT is not referenced.
- 8: MSGLVL — INTEGER** *Input*  
*On entry:* the amount of printout produced by E04MFF or H02BBF, as indicated below. For a description of the printed output see Section 8.2 of the document for E04MFF or Section 5.1 of the document for H02BBF (as appropriate). All output is written to the current advisory message unit (as defined by X04ABF).  
 For an LP problem (E04MFF):
- | <b>Value</b> | <b>Definition</b>  |
|--------------|--|
| 0            | No output.   |
| 1            | The final solution only.   |
| 5            | One line of output for each iteration (no printout of the final solution). |
| 10           | The final solution and one line of output for each iteration.              |
- For an IP problem (H02BBF):
- | <b>Value</b> | <b>Definition</b>  |
|--------------|--|
| 0            | No output.   |
| 1            | The final IP solution only.  |
| 5            | One line of output for each node investigated and the final IP solution.   |
| 10           | The original LP solution (first node) with dummy names for the rows and columns, one line of output for each node investigated and the final IP solution with MPSX names for the rows and columns. |
- 9: N — INTEGER** *Output*  
*On exit:* n, the actual number of variables in the problem.

- 10: M — INTEGER *Output*  
*On exit:* m, the actual number of general linear constraints in the problem.
- 11: X(MAXN) — *real* array *Output*  
*On exit:* the solution to the problem, stored in X(1),X(2),...X(N). X(i) is the value of the variable whose MPSX name is stored in CRNAME(i), for  $i = 1, 2, \dots, N$ .
- 12: CRNAME(MAXN+MAXM) — CHARACTER\*8 array *Output*  
*On exit:* the first N elements contain the MPSX names for the variables in the problem.
- 13: IWORK(LIWORK) — INTEGER array *Output*  
*On exit:* the first (N+M) elements contain ISTATE (the status of the constraints in the working set at the solution). Further details can be found in Section 5 of the document for E04MFF or Section 5 of the document for H02BZF (as appropriate).
- 14: LIWORK — INTEGER *Input*  
*On entry:* the dimension of the array IWORK as declared in the (sub)program from which H02BFF is called.  
*Constraints:*  
 For an LP problem,  $LIWORK \geq 4 \times MAXN + MAXM + 3$ .  
 For an IP problem,  $LIWORK \geq (25+MAXN+MAXM) \times MAXDPT + 7 \times MAXN + 2 \times MAXM + 4$ .
- 15: RWORK(LRWORK) — *real* array *Output*  
*On exit:* the first (N+M) elements contain BL (the lower bounds), the next (N+M) elements contain BU (the upper bounds) and the next (N+M) elements contain CLAMDA (the Lagrange multipliers). Further details can be found in Section 5 of the document for E04MFF or Section 5 of the document for H02BZF (as appropriate). Note that for an IP problem the contents of BL and BU may not be the same as those originally specified by the user in the MPSX data file and/or via the parameters XBLDEF and XBUDEF.
- 16: LRWORK — INTEGER *Input*  
*On entry:* the dimension of the array RWORK as declared in the (sub)program from which H02BFF is called.  
*Constraints:*  
 For an LP problem,  $LRWORK \geq 2 \times \min(MAXN, MAXM+1)^2 + MAXM \times MAXN + 12 \times MAXN + 9 \times MAXM$ .  
 For an IP problem,  $LRWORK \geq MAXDPT \times (MAXN+1) + 2 \times \min(MAXN, MAXM+1)^2 + MAXM \times MAXN + 19 \times MAXN + 15 \times MAXM$ .
- 17: IFAIL — INTEGER *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6 Error Indicators and Warnings

If on entry  $IFAIL = 0$  or  $-1$ , explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors detected by the routine:

$IFAIL = i < 0$

Either MAXM and/or MAXN are too small or the MPSX data file is non-standard and/or corrupt. This corresponds to  $IFAIL = -i$  in Section 6 of the document for H02BUF.

$IFAIL = 1$

X is a weak local minimum. This means that the solution is not unique.

$IFAIL = 2$

The solution appears to be unbounded. This value of IFAIL implies that a step as large as XBUDEF would have to be taken in order to continue the algorithm. See Section 8.

$IFAIL = 3$

No feasible point was found, i.e., it was not possible to satisfy all the constraints to within the feasibility tolerance (defined internally as  $\sqrt{\text{machine precision}}$ ). See Section 8.

$IFAIL = 4$

The maximum number of iterations (defined internally as  $\max(50, 5(n + m))$ ) was reached before normal termination occurred. See Section 8.

$IFAIL = 5$

An input parameter is invalid. Refer to the printed output to determine which parameter must be re-defined.

$IFAIL = 6$

A serious error has occurred in an internal call to either E04MFF or H02BBF (as appropriate). Check all subroutine calls and array dimensions.

For an IP problem only:

$IFAIL = 7$

The solution reported is not the optimum solution. See Section 8.

$IFAIL = 8$

MAXDPT is too small. Try increasing its value (along with that of LIWORK and/or LRWORK if appropriate) and rerun H02BFF.

$IFAIL = 9$

No feasible integer point was found, i.e., it was not possible to satisfy all the integer variables to within the integer feasibility tolerance (defined internally as  $10^{-5}$ ). See Section 8.

## 7 Accuracy

The routine implements a numerically stable active set strategy and returns solutions that are as accurate as the condition of the problem warrants on the machine.

## 8 Further Comments

For an LP problem only:

if IFAIL = 2 on exit, users can obtain more information by making separate calls to H02BUF, E04MFF and H02BVF (in that order). Note that this will (by default) cause the final LP solution to be printed twice on the current advisory message unit (see X04ABF), once with dummy names for the rows and columns and once with user supplied names. To suppress the printout of the final LP solution with dummy names for the rows and columns, include the statement

```
CALL E04MHF(' Print Level = 5 ')
```

prior to calling E04MFF.

if IFAIL = 3 on exit, users are recommended to reset the value of the feasibility tolerance and rerun H02BFF. (Further advice is given under the description of IFAIL = 3 in Section 6 of the document for E04MFF). For example, to reset the value of the feasibility tolerance to 0.01, include the statement

```
CALL E04MHF(' Feasibility Tolerance = 0.01 ')
```

prior to calling H02BFF.

if IFAIL = 4 on exit, users are recommended to increase the maximum number of iterations allowed before termination and rerun H02BFF. For example, to increase the maximum number of iterations to 500, include the statement

```
CALL E04MHF(' Iteration Limit = 500 ')
```

prior to calling H02BFF.

Note that H02BUF uses an 'infinite' bound size of  $10^{20}$  in the definition of  $l$  and  $u$ . In other words, any element of  $u$  greater than or equal to  $10^{20}$  will be regarded as  $+\infty$  (and similarly any element of  $l$  less than or equal to  $-10^{20}$  will be regarded as  $-\infty$ ). If this value is deemed to be 'inappropriate', users are recommended to reset the value of the 'infinite' bound size and make any necessary changes to BL and/or BU prior to calling E04MFF. For example, to reset the value of the 'infinite' bound size to 10000, include the statement

```
CALL E04MHF(' Infinite Bound Size = 1.0E+4 ')
```

prior to calling E04MFF.

For an IP problem only:

if IFAIL = 2,3,4,7 or 9 on exit, users can obtain more information by making separate calls to H02BUF, H02BBF, H02BZF and H02BVF (in that order).

Note that H02BUF uses an 'infinite' bound size of  $10^{20}$  in the definition of  $l$  and  $u$ . In other words, any element of  $u$  greater than or equal to  $10^{20}$  will be regarded as  $+\infty$  (and similarly any element of  $l$  less than or equal to  $-10^{20}$  will be regarded as  $-\infty$ ). If this value is deemed to be 'inappropriate', users are recommended to reset the value of the parameter BIGBND (as described in Section 5 of the document for H02BBF) and make any necessary changes to BL and/or BU prior to calling H02BBF.

## 9 Example

This example solves the same problem as the example for H02BUF, except that it treats it as an IP problem.

One of the applications of integer programming is to the so-called diet problem. Given the nutritional content of a selection of foods, the cost of each food, the amount available of each food and the consumer's minimum daily energy requirements, the problem is to find the cheapest combination. This gives rise to the following problem:

minimize

$$c^T x$$

subject to

$$\begin{aligned} Ax &\geq b, \\ 0 &\leq x \leq u, \end{aligned}$$

where

$$c = (3 \ 24 \ 13 \ 9 \ 20 \ 19)^T, \quad x = (x_1, x_2, x_3, x_4, x_5, x_6)^T,$$

$x_1, x_2$  and  $x_6$  are real,

$x_3, x_4$  and  $x_5$  are integer,

$$A = \begin{pmatrix} 110 & 205 & 160 & 160 & 420 & 260 \\ 4 & 32 & 13 & 8 & 4 & 14 \\ 2 & 12 & 54 & 285 & 22 & 80 \end{pmatrix}, \quad b = \begin{pmatrix} 2000 \\ 55 \\ 800 \end{pmatrix} \text{ and}$$

$$u = (4 \ 3 \ 2 \ 8 \ 2 \ 2)^T.$$

The rows of  $A$  correspond to energy, protein and calcium and the columns of  $A$  correspond to oatmeal, chicken, eggs, milk, pie and bacon respectively.

The MPSX data representation of this problem is given in Section 9.2.

## 9.1 Program Text

**Note.** The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      H02BFF Example Program Text
*      Mark 18 Revised.  NAG Copyright 1997.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
INTEGER          MAXN, MAXM
PARAMETER       (MAXN=50,MAXM=50)
real           XBLDEF, XBUDEF
PARAMETER       (XBLDEF=0.0e0,XBUDEF=1.0e+20)
INTEGER          MAXDPT
PARAMETER       (MAXDPT=3*MAXN/2)
INTEGER          MSGLVL
PARAMETER       (MSGLVL=5)
INTEGER          LIWORK
PARAMETER       (LIWORK=(25+MAXN+MAXM)*MAXDPT+2*MAXM+7*MAXN+4)
INTEGER          LRWORK
PARAMETER       (LRWORK=MAXDPT*(MAXN+1)
+               +2*MAXN**2+MAXM*MAXN+19*MAXN+15*MAXM)
CHARACTER*3     OPTIM
PARAMETER       (OPTIM='MIN')
*      .. Local Scalars ..
INTEGER          IFAIL, INFILE, M, N
*      .. Local Arrays ..
real           RWORK(LRWORK), X(MAXN)
INTEGER          IWORK(LIWORK)
CHARACTER*8     CRNAME(MAXN+MAXM)
*      .. External Subroutines ..
EXTERNAL        H02BFF
*      .. Executable Statements ..
WRITE (NOUT,*) 'H02BFF Example Program Results'

```

```

*   Skip heading in data file
    READ (NIN,*)
*
*   Solve the problem
*
    INFILE = NIN
    IFAIL = 0
*
    CALL H02BFF(INFILE,MAXN,MAXM,OPTIM,XBLDEF,XBUDEF,MAXDPT,MSGLVL,N,
+             M,X,CRNAME,IWORK,LIWORK,RWORK,LRWORK,IFAIL)
*
    STOP
    END

```

## 9.2 Program Data

### H02BFF Example Program Data

```

NAME          DIET
ROWS
G  ENERGY
G  PROTEIN
G  CALCIUM
N  COST
COLUMNS
OATMEAL  ENERGY  110.0
OATMEAL  PROTEIN   4.0
OATMEAL  CALCIUM   2.0
OATMEAL  COST      3.0
CHICKEN  ENERGY  205.0
CHICKEN  PROTEIN   32.0
CHICKEN  CALCIUM   12.0
CHICKEN  COST      24.0
INTEGER  'MARKER'          'INTORG'
EGGS     ENERGY  160.0
EGGS     PROTEIN   13.0
EGGS     CALCIUM   54.0
EGGS     COST      13.0
MILK     ENERGY  160.0
MILK     PROTEIN   8.0
MILK     CALCIUM  285.0
MILK     COST      9.0
PIE      ENERGY  420.0
PIE      PROTEIN   4.0
PIE      CALCIUM   22.0
PIE      COST      20.0
INTEGER  'MARKER'          'INTEND'
BACON    ENERGY  260.0
BACON    PROTEIN   14.0
BACON    CALCIUM   80.0
BACON    COST      19.0
RHS
DEMANDS  ENERGY  2000.0
DEMANDS  PROTEIN   55.0
DEMANDS  CALCIUM   800.0

```

```

BOUNDS
UI SERVINGS OATMEAL 4.0
UI SERVINGS CHICKEN 3.0
UP SERVINGS EGGS 2.0
UP SERVINGS MILK 8.0
UP SERVINGS PIE 2.0
UI SERVINGS BACON 2.0
ENDATA
    
```

### 9.3 Program Results

#### H02BFF Example Program Results

```

*** H02BBF
*** Start of NAG Library implementation details ***
    
```

```

Implementation title: Generalised Base Version
Precision: FORTRAN double precision
Product Code: FLBAS18D
Mark: 18A
    
```

```

*** End of NAG Library implementation details ***
    
```

#### Parameters

-----

```

Linear constraints..... 3      First integer solution..  OFF
Variables..... 6      Max depth of the tree...  75

Feasibility tolerance... 1.05E-08      Print level..... 5
Infinite bound size..... 1.00E+20      EPS (machine precision). 1.11E-16

Integer feasibility tol. 1.00E-05      Iteration limit..... 50
Max number of nodes..... NONE
    
```

```

** Workspace provided with MAXDPT = 75: LRWORK = 10075 LIWORK = 9679
** Workspace required with MAXDPT = 75: LRWORK = 677 LIWORK = 2587
    
```

```

*** Optimum LP solution *** 92.50000
    
```

```

*** Start of tree search ***
    
```

Node No	Parent Node	Obj Value	Varbl Chosen	Value Before	Lower Bound	Upper Bound	Value After	Depth
2	1	93.2	4	4.50	5.00	8.00	5.00	1
3	1	93.8	4	4.50	0.000E+00	4.00	4.00	1
4	2	94.8	5	1.81	2.00	2.00	2.00	2
5	2	96.1	5	1.81	0.000E+00	1.00	1.00	2
6	3	96.9	6	0.308	1.00	2.00	1.00	2
7	3	94.5	6	0.308	0.000E+00	0.000E+00	0.000E+00	2
8	7	96.5	3	0.500	1.00	2.00	1.00	3
9	7	97.4	3	0.500	0.000E+00	0.000E+00	0.000E+00	3
10	4	97.0	1	3.27	4.00	4.00	4.00	3

```

*** Integer solution ***
    
```



Node No	Parent Node	Obj Value	Varbl Chosen	Value Before	Lower Bound	Upper Bound	Value After	Depth
11	4	95.7		1 3.27	0.000E+00	3.00	3.00	3
12	11	99.5	CD	4 5.19	6.00	8.00	6.00	4
13	11	96.2		4 5.19	5.00	5.00	5.00	4
14	5	97.3	CD	4 7.13	8.00	8.00	8.00	3
15	5	96.5		4 7.13	5.00	7.00	7.00	3
16	13	107.	CD	6 0.115	1.00	2.00	1.00	5
17	13	96.4		6 0.115	0.000E+00	0.000E+00	0.000E+00	5
18	17	103.	CD	3 0.188	1.00	2.00	1.00	6
19	17	97.5	CD	3 0.188	0.000E+00	0.000E+00	0.000E+00	6
20	15	101.	CD	6 0.769E-01	1.00	2.00	1.00	4
21	15	96.6		6 0.769E-01	0.000E+00	0.000E+00	0.000E+00	4
22	8	97.2	CD	4 3.50	4.00	4.00	4.00	4
23	8	98.5	CD	4 3.50	0.000E+00	3.00	3.00	4
24	21	100.	CD	3 0.125	1.00	2.00	1.00	5
25	21	97.3	CD	3 0.125	0.000E+00	0.000E+00	0.000E+00	5
26	6	97.0	CD	4 2.88	3.00	4.00	3.00	3
27	6	105.	CD	4 2.88	0.000E+00	2.00	2.00	3

\*\*\* End of tree search \*\*\*

Total of 27 nodes investigated.

Exit H02BBF - Optimum IP solution found.

Final IP objective value = 97.00000

Varbl	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
OATMEAL	EQ	4.00000	4.00000	4.00000	3.000	0.0000E+00
CHICKEN	LL	0.000000E+00	0.000000E+00	3.00000	24.00	0.0000E+00
EGGS	LL	0.000000E+00	0.000000E+00	2.00000	13.00	0.0000E+00
MILK	LL	5.00000	5.00000	8.00000	9.000	0.0000E+00
PIE	EQ	2.00000	2.00000	2.00000	20.00	0.0000E+00
BACON	LL	0.000000E+00	0.000000E+00	2.00000	19.00	0.0000E+00

L Con	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
ENERGY	FR	2080.00	2000.00	None	0.0000E+00	80.00
PROTEIN	FR	64.0000	55.0000	None	0.0000E+00	9.000
CALCIUM	FR	1477.00	800.000	None	0.0000E+00	677.0



## H02BUF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

H02BUF reads data for a linear or integer programming problem from an external file which is in standard or compatible MPSX input format.

### 2. Specification

```

SUBROUTINE H02BUF (INFILE, MAXN, MAXM, OPTIM, XBLDEF, XBUDEF, NMOBJ,
1             NMRHS, NMRNG, NMBND, MPSLST, N, M, A, BL, BU, CVEC,
2             X, INTVAR, CRNAME, NMPROB, IWORK, IFAIL)

INTEGER      INFILE, MAXN, MAXM, N, M, INTVAR(MAXN),
1           IWORK(MAXN+MAXM), IFAIL
real       XBLDEF, XBUDEF, A(MAXM,MAXN), BL(MAXN+MAXM),
1           BU(MAXN+MAXM), CVEC(MAXN), X(MAXN)
CHARACTER*3  OPTIM
CHARACTER*8  NMOBJ, NMRHS, NMRNG, NMBND, CRNAME(MAXN+MAXM), NMPROB
LOGICAL     MPSLST

```

### 3. Description

H02BUF reads linear programming (LP) or integer programming (IP) problem data from an external file which is prepared in standard or compatible MPSX [1] input format and then initializes  $n$  (the number of variables),  $m$  (the number of general linear constraints), the vectors  $c$ ,  $l$  and  $u$  and the  $m$  by  $n$  matrix  $A$  for use with E04MFF or H02BBF, which are designed to solve problems of the form

$$\underset{x \in R^n}{\text{minimize}} \quad c^T x \quad \text{subject to} \quad l \leq \begin{pmatrix} x \\ Ax \end{pmatrix} \leq u.$$

This routine may be followed by calls to either E04MFF (to solve an LP problem) or H02BBF and H02BZF (to solve an IP problem), possibly followed by a call to H02BVF (to print the solution using MPSX names).

Note that H02BUF uses an 'infinite' bound size of  $10^{20}$  in the definition of  $l$  and  $u$ . In other words, any element of  $u$  greater than or equal to  $10^{20}$  will be regarded as  $+\infty$  (and similarly any element of  $l$  less than or equal to  $-10^{20}$  will be regarded as  $-\infty$ ). If this value is deemed to be 'inappropriate', users are recommended to reset the value of either the optional parameter **Infinite Bound Size** (if an LP problem is being solved) or the parameter **BIGBND** (if an IP problem is being solved) and make any necessary changes to **BL** and/or **BU** prior to calling E04MFF or H02BBF (as appropriate).

The documents for H02BVF, E04MFF and/or H02BBF and H02BZF should be consulted for further details.

#### MPSX input format

The input file of data may only contain two types of lines:

1. Indicator lines (specifying the type of data which is to follow).
2. Data lines (specifying the actual data).

The input file must not contain any blank lines. Any characters beyond column 80 are ignored. Indicator lines must not contain leading blank characters (in other words they must begin in column 1). The following displays the order in which the indicator lines must appear in the file:

```

NAME      user-given name
ROWS
    data line(s)
COLUMNS
    data line(s)

```

```

RHS
  data line(s)
RANGES (optional)
  data line(s)
BOUNDS (optional)
  data line(s)
ENDATA

```

The 'user-given name' specifies a name for the problem and must occupy columns 15-22. The name can either be blank or up to a maximum of 8 characters.

A data line follows the same fixed format made up of fields defined below. The contents of the fields may have different significance depending upon the section of data in which they appear.

	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
Columns	2-3	5-12	15-22	25-36	40-47	50-61
Contents	Code	Name	Name	Value	Name	Value

The names and codes consist of 'alphanumeric' characters (i.e. a-z, A-Z, 0-9, +, -, asterisk (\*), blank ( ), colon (:), dollar sign (\$) or fullstop (.) only) and the names must not contain leading blank characters. Values are read using Fortran format E12.0. This allows values to be entered in several equivalent forms. For example, 1.2345678, 1.2345678E+0, 123.45678E-2 and 12345678E-07 all represent the same number. It is safest to include an explicit decimal point.

Note that in order to ensure numeric values are interpreted as intended, they should be *right-justified* in the 12-character field, with no trailing blanks. This is because in some situations trailing blanks may be interpreted as zeros and this can dramatically affect the interpretation of the value. This is relevant if the value contains an exponent, or if it contains neither an exponent nor an explicit decimal point. For example, the fields

```

% % % % 1 . 23 E - 2 %
% % % % % % % 1 2 3 % %

```

may be interpreted as 1.23E-20 and 12300 respectively (where % is used to denote a blank). The actual behaviour is system-dependent.

Comment lines are allowed in the data file. These must have an asterisk (\*) in column 1 and any characters in columns 2-80. In any data line, a dollar sign (\$) as the first character in field 3 or 5 indicates that the information from that point through column 80 consists of comments.

Columns outside the six fields must be blank, except for columns 72-80, whose contents are ignored by the routine. These columns may be used to enter a sequence number. A non-blank character outside the predefined six fields and columns 72-80 is considered to be a major error (IFAIL = 11; see Section 6), unless it is part of a comment.

### ROWS Data Lines

These lines specify row (constraint) names and their inequality types (i.e. =, ≥ or ≤).

Field 1:	defines the constraint type. It may be in column 2 or column 3.
N	free row, that is no constraint. It may be used to define the objective row.
G	greater than or equal to (i.e. ≥).
L	less than or equal to (i.e. ≤).
E	exactly equal to (i.e. =).
Field 2:	defines the row name.

Row type N stands for 'Not binding', also known as 'Free'. It can be used to define the objective row. The objective row is a free row that specifies the vector  $c$  in the objective function. It is taken to be the first free row, unless some other free row name is specified by the parameter NMOBJ (see Section 5). Note that the objective function must be included in the MPSX data file. Thus the maximum number of constraints (MAXM; see Section 5) in the problem must be  $m+1$ .

**COLUMNS Data Lines**

These lines specify the names to be assigned to the variables (columns) in the constraint matrix  $A$ , and define, in terms of column vectors, the actual values of the corresponding matrix elements.

- Field 1: blank (ignored)
- Field 2: gives the name of the column associated with the elements specified in the following fields.
- Field 3: contains the name of a row.
- Field 4: used in conjunction with field 3 contains the value of the matrix element.
- Field 5: is optional (may be used like field 3).
- Field 6: is optional (may be used like field 4).

Note that only non-zero elements of  $A$  need to be specified in the COLUMNS section, as any unspecified elements are assumed to be zero.

**RHS Data Lines**

This section specifies the right hand side values of the constraint matrix  $A$ . The lines specify the name of the RHS (right hand side) vector given to the problem, the numerical values of the elements of the vector are also defined by the data lines and may appear in any order. The data lines have exactly the same format as the COLUMNS data lines, except that the column name is replaced by the RHS name. Note that any unspecified elements are assumed to be zero.

**RANGES Data Lines (optional)**

Ranges are used for constraints of the form  $l \leq Ax \leq u$ , where  $l$  and  $u$  are finite. The range of the constraint is  $r = u-l$ . Either  $l$  or  $u$  must be specified in the RHS section and  $r$  must be defined in this section.

The data lines have exactly the same format as the COLUMNS data lines, except that the column name is replaced by the RANGE name.

**BOUNDS Data Lines (optional)**

These lines specify limits on the values of the variables ( $l$  and  $u$  in  $l \leq x \leq u$ ). If the variable is not specified in the bound set then it is automatically assumed to lie between default lower and upper bounds (usually 0 and  $+\infty$ ). Like an RHS column which is given a name, the set of variables in one bound set is also given a name.

- Field 1: specifies the type of bound or defines the variable type.
  - LO lower bound
  - UP upper bound
  - FX fixed variable
  - FR free variable ( $-\infty$  to  $+\infty$ )
  - MI lower bound is  $-\infty$
  - PL upper bound is  $+\infty$ . This is the default variable type.
- Field 2: identifies a name for the bound set.
- Field 3: identifies the column name of the variable belonging to this set.
- Field 4: identifies the value of the bound; this has a numerical value only in association with LO, UP, FX in field 1, otherwise it is blank.
- Field 5: is blank and ignored.
- Field 6: is blank and ignored.

Note that if RANGES and BOUNDS sections are both present, the RANGES section must appear first.

### Integer problems

In IP problems there are two common integer variable types: (a) 0-1 integer variables which represent 'on' or 'off' situations and (b) general integer variables which are forced to take an integer value, in a specified range, at the optimal integer solution. Integer variables can be defined in the following compatible and standard MPSX forms.

In the compatible MPSX format, the type of integer variables are defined in field 1 of the BOUNDS section, that is:

- Field 1: specifies the type of the integer variable.
- BV 0-1 integer variable (bound value is 1.0).
- UI general integer variable (bound value is in field 4).

In the standard MPSX format, the integer variables are treated the same as the 'ordinary' bounded variables, in the BOUNDS section. Integer markers are, however, introduced in the COLUMNS section to specify the integer variables. The indicator lines for these markers are:

	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
Columns	2-3	5-12	15-22	25-36	40-47	50-61
Contents		INTEGER	'MARKER'		'INTORG'	

to mark the beginning of the integer variables and

	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
Columns	2-3	5-12	15-22	25-36	40-47	50-61
Contents		INTEGER	'MARKER'		'INTEND'	

to mark the end. That is, any variables between these markers are treated as integer variables. Note that if the (INTEND) indicator line is not specified in the file then all the variables between the (INTORG) indicator line and the end of the COLUMNS section are assumed to be integer variables. The routine accepts both standard and/or compatible MPSX format as a means of specifying integer variables. An example of the standard MPSX format is given in Section 9.2 of the document for H02BFF.

## 4. References

- [1] MPSX – Mathematical Programming System.  
Program number 5734 XM4, IBM Trade Corporation, New York, 1971.

## 5. Parameters

- 1: INFILE – INTEGER. *Input*  
*On entry:* the unit number associated with the MPSX data file.  
*Constraint:*  $0 \leq \text{INFILE} \leq 99$ .
- 2: MAXN – INTEGER. *Input*  
*On entry:* an upper limit for the number of variables in the problem.  
*Constraint:*  $\text{MAXN} \geq 1$ .
- 3: MAXM – INTEGER. *Input*  
*On entry:* an upper limit for the number of constraints (including the objective) in the problem.  
*Constraint:*  $\text{MAXM} \geq 1$ .
- 4: OPTIM – CHARACTER\*3. *Input*  
*On entry:* specifies the direction of the optimization. OPTIM must be set to 'MIN' for minimization and to 'MAX' for maximization.  
*Constraint:* OPTIM = 'MIN' or 'MAX'.

- 5: **XBLDEF – real.** *Input*  
*On entry:* the default lower bound to be used for the variables in the problem when none is specified in the BOUNDS section of the MPSX data file. For a standard LP or IP problem XBLDEF would normally be set to zero.
- 6: **XBUDEF – real.** *Input*  
*On entry:* the default upper bound to be used for the variables in the problem when none is specified in the BOUNDS section of the MPSX data file. For a standard LP or IP problem XBUDEF would normally be set to ‘infinity’ (i.e.  $XBUDEF \geq 10^{20}$ ).  
*Constraint:*  $XBUDEF \geq XBLDEF$ .
- 7: **NMOBJ – CHARACTER\*8.** *Input/Output*  
*On entry:* either the name of the objective function to be used for the optimization, or blank (in which case the first objective (free) row in the file is used).  
*On exit:* the name of the objective row as defined in the MPSX data file.
- 8: **NMRHS – CHARACTER\*8.** *Input/Output*  
*On entry:* either the name of the RHS set to be used for the optimization, or blank (in which case the first RHS set is used).  
*On exit:* the name of the RHS set read in the MPSX data file.
- 9: **NMRNG – CHARACTER\*8.** *Input/Output*  
*On entry:* either the name of the RANGE set to be used for the optimization, or blank (in which case the first RANGE set (if any) is used).  
*On exit:* the name of the RANGE set read in the MPSX data file. This is blank if the MPSX data file does not have a RANGE set.
- 10: **NMBND – CHARACTER\*8.** *Input/Output*  
*On entry:* either the name of the BOUNDS set to be used for the optimization, or blank (in which case the first BOUNDS set (if any) is used).  
*On exit:* the name of the BOUNDS set read in the MPSX data file. This is blank if the MPSX data file does not have a BOUNDS set.
- 11: **MPSLST – LOGICAL.** *Input*  
*On entry:* if  $MPSLST = .TRUE.$ , then a listing of the input data is sent to the current advisory message unit (as defined by X04ABF). This can be useful for debugging the MPSX data file.
- 12: **N – INTEGER.** *Output*  
*On exit:*  $n$ , the actual number of variables in the problem.
- 13: **M – INTEGER.** *Output*  
*On exit:*  $m$ , the actual number of general linear constraints in the problem.
- 14: **A(MAXM,MAXN) – real array.** *Output*  
*On exit:*  $A$ , the matrix of general linear constraints.
- 15: **BL(MAXN+MAXM) – real array.** *Output*  
*On exit:*  $l$ , the lower bounds for all the variables and constraints in the following order. The first  $N$  elements of  $BL$  contain the bounds on the variables and the next  $M$  elements contain the bounds for the general linear constraints (if any). Note that an ‘infinite’ lower bound is indicated by  $BL(j) = -1.0E+20$  and an equality constraint by  $BL(j) = BU(j)$ .

- 16: BU(MAXN+MAXM) – *real* array. *Output*  
*On exit:*  $u$ , the upper bounds for all the variables and constraints in the following order. The first  $N$  elements of BU contain the bounds on the variables and the next  $M$  elements contain the bounds for the general linear constraints (if any). Note that an ‘infinite’ upper bound is indicated by  $BU(j) = 1.0E+20$  and an equality constraint by  $BU(j) = BL(j)$ .
- 17: CVEC(MAXN) – *real* array. *Output*  
*On exit:*  $c$ , the coefficients of the objective function. The signs of these coefficients are determined by the problem (either LP or IP) and the direction of the optimization (see OPTIM above).
- 18: X(MAXN) – *real* array. *Output*  
*On exit:* an initial estimate of the solution to the problem. More precisely,  $X(j) = 1.0$  if  $j$  is odd and 0.0 otherwise, for  $j = 1, 2, \dots, N$ .
- 19: INTVAR(MAXN) – INTEGER array. *Output*  
*On exit:* indicates which are the integer variables in the problem. More precisely,  $INTVAR(k) = 1$  if  $x_k$  is an integer variable, and 0 otherwise, for  $k = 1, 2, \dots, N$ .
- 20: CRNAME(MAXN+MAXM) – CHARACTER\*8 array. *Output*  
*On exit:* the MPSX names of all the variables and constraints in the problem in the following order. The first  $N$  elements contain the MPSX names for the variables and the next  $M$  elements contain the MPSX names for the general linear constraints (if any).
- 21: NMPROB – CHARACTER\*8. *Output*  
*On exit:* the name of the problem as defined in the MPSX data file.
- 22: IWORK(MAXN+MAXM) – INTEGER array. *Workspace*
- 23: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

There are too many rows present in the data file. Increase MAXM by at least  $(M - \text{MAXM})$  and rerun H02BUF.

IFAIL = 2

There are too many columns present in the data file. Increase MAXN by at least  $(N - \text{MAXN})$  and rerun H02BUF.

The following error exits (apart from IFAIL = 14) are caused by having either a corrupt or a non-standard MPSX data file. Refer to Section 3 for a detailed description of the MPSX format which can be read by H02BUF. If MPSSLST = .TRUE., the last line of printed output refers to the line in the MPSX data file which contains the reported error.



**IFAIL = 3**

The objective function row was not found. There must be at least one row in the ROWS section with row type N for the objective row.

**IFAIL = 4**

There are no rows specified in the ROWS section.

**IFAIL = 5**

An illegal constraint type was detected in the ROWS section. The constraint type must be either N, L, G or E.

**IFAIL = 6**

An illegal row name was detected in the ROWS section. Names must be made up of alphanumeric characters with no leading blanks.

**IFAIL = 7**

An illegal column name was detected in the COLUMNS section. Names must be made up of alphanumeric characters with no leading blanks.

**IFAIL = 8**

An illegal bound type was detected in the BOUNDS section. The bound type must be either LO, UP, FX, FR, MI, PL, BV or UI.

**IFAIL = 9**

An unknown column name was detected in the BOUNDS section. All the column names must be specified in the COLUMNS section.

**IFAIL = 10**

The last line in the file does not contain the ENDDATA line indicator.

**IFAIL = 11**

An illegal data line was detected in the file. This line is neither a comment line nor a valid data line.

**IFAIL = 12**

An unknown row name was detected in COLUMNS or RHS or RANGES section. All the row names must be specified in the ROWS section.

**IFAIL = 13**

There were no columns specified in the COLUMNS section.

**IFAIL = 14**

An input parameter is invalid.

**IFAIL = 15**

Incorrect integer marker. In standard MPSX data format, integer variables should be defined between INTORG and INTEND markers.

**7. Accuracy**

Not applicable.

**8. Further Comments**

None.

## 9. Example

This example solves the same problem as the example for H02BFF, except that it treats it as an LP problem.

One of the applications of linear programming is to the so-called diet problem. Given the nutritional content of a selection of foods, the cost of each food, the amount available of each food and the consumer's minimum daily energy requirements, the problem is to find the cheapest combination. This gives rise to the following problem:

minimize

$$c^T x$$

subject to

$$Ax \geq b,$$

$$0 \leq x \leq u,$$

where

$$c = (3 \ 24 \ 13 \ 9 \ 20 \ 19)^T, \quad x = (x_1, x_2, x_3, x_4, x_5, x_6)^T \text{ is real,}$$

$$A = \begin{pmatrix} 110 & 205 & 160 & 160 & 420 & 260 \\ 4 & 32 & 13 & 8 & 4 & 14 \\ 2 & 12 & 54 & 285 & 22 & 80 \end{pmatrix}, \quad b = \begin{pmatrix} 2000 \\ 55 \\ 800 \end{pmatrix} \text{ and}$$

$$u = (4 \ 3 \ 2 \ 8 \ 2 \ 2)^T.$$

The rows of  $A$  correspond to energy, protein and calcium and the columns of  $A$  correspond to oatmeal, chicken, eggs, milk, pie and bacon respectively.

The MPSX representation of the problem is given in Section 9.2.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      H02BUF Example Program Text
*      Mark 16 Release. NAG Copyright 1993.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5, NOUT=6)
INTEGER          MAXN, MAXM
PARAMETER       (MAXN=50, MAXM=50)
INTEGER          LDA
PARAMETER       (LDA=MAXM)
real            XBUDEF, XBLDEF
PARAMETER       (XBUDEF=1.0e+20, XBLDEF=0.0e0)
INTEGER          LIWORK
PARAMETER       (LIWORK=2*MAXN+3)
INTEGER          LWORK
PARAMETER       (LWORK=2*(MAXM+1)**2+7*MAXN+5*MAXM)
CHARACTER*3     OPTIM
PARAMETER       (OPTIM='MIN')
*      .. Local Scalars ..
real          OBJVAL
INTEGER          IFAIL, INFILE, ITER, M, N
LOGICAL         MPSSLST
CHARACTER*8     KBLANK, NMBND, NMOBJ, NMPROB, NMRHS, NMRNG
*      .. Local Arrays ..
real          A(MAXM,MAXN), AX(MAXM), BL(MAXN+MAXM),
+              BU(MAXN+MAXM), CLAMDA(MAXN+MAXM), CVEC(MAXN),
+              WORK(LWORK), X(MAXN)
INTEGER          INTVAR(MAXN), ISTATE(MAXN+MAXM), IWORK(LIWORK)
CHARACTER*8     CRNAME(MAXN+MAXM)
*      .. External Subroutines ..
EXTERNAL        E04MFF, E04MHF, H02BUF, H02BVF
*      .. Data statements ..
DATA           KBLANK/'          '/
```

```

*      .. Executable Statements ..
WRITE (NOUT,*) 'H02BUF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
*
*      Initialize parameters
*
      INFILE = NIN
      NMPROB = KBLANK
      NMOBJ = KBLANK
      NMRHS = KBLANK
      NMRNG = KBLANK
      NMBND = KBLANK
      MPSLST = .FALSE.
*
      IFAIL = 0
*
*      Convert the MPSX data file for use by E04MFF
*
      CALL H02BUF(INFILE,MAXN,MAXM,OPTIM,XBLDEF,XBUDEF,NMOBJ,NMRHS,
+              NMRNG,NMBND,MPSLST,N,M,A,BL,BU,CVEC,X,INTVAR,CRNAME,
+              NMPROB,ISTATE,IFAIL)
*
*      Solve the problem
*
      IFAIL = -1
*
      CALL E04MHF('Print Level = 5')
*
      CALL E04MFF(N,M,A,LDA,BL,BU,CVEC,ISTATE,X,ITER,OBJVAL,AX,CLAMDA,
+              IWORK,LIWORK,WORK,LWORK,IFAIL)
*
      IF (IFAIL.EQ.0 .OR. IFAIL.EQ.1 .OR. IFAIL.EQ.3) THEN
*
*          Print solution (using MPSX names)
*
          IFAIL = 0
*
          CALL H02BVF(N,M,A,LDA,BL,BU,X,CLAMDA,ISTATE,CRNAME,IFAIL)
*
          ELSE
              WRITE (NOUT,99999) 'E04MFF terminated with IFAIL = ', IFAIL
          END IF
*
          STOP
*
99999 FORMAT (1X,A,I3)
      END

```

## 9.2. Program Data

**Note:** the MPSX data which is read by H02BUF begins with the second record of this data file; the first record is a caption which is read by the example program.

H02BUF Example Program Data

```

NAME          DIET
ROWS
G  ENERGY
G  PROTEIN
G  CALCIUM
N  COST
COLUMNS
OATMEAL  ENERGY  110.0
OATMEAL  PROTEIN   4.0
OATMEAL  CALCIUM   2.0
OATMEAL  COST      3.0
CHICKEN  ENERGY  205.0
CHICKEN  PROTEIN   32.0
CHICKEN  CALCIUM   12.0
CHICKEN  COST      24.0

```

EGGS	ENERGY	160.0
EGGS	PROTEIN	13.0
EGGS	CALCIUM	54.0
EGGS	COST	13.0
MILK	ENERGY	160.0
MILK	PROTEIN	8.0
MILK	CALCIUM	285.0
MILK	COST	9.0
PIE	ENERGY	420.0
PIE	PROTEIN	4.0
PIE	CALCIUM	22.0
PIE	COST	20.0
BACON	ENERGY	260.0
BACON	PROTEIN	14.0
BACON	CALCIUM	80.0
BACON	COST	19.0
RHS		
DEMANDS	ENERGY	2000.0
DEMANDS	PROTEIN	55.0
DEMANDS	CALCIUM	800.0
BOUNDS		
UI SERVINGS	OATMEAL	4.0
UI SERVINGS	CHICKEN	3.0
UP SERVINGS	EGGS	2.0
UP SERVINGS	MILK	8.0
UP SERVINGS	PIE	2.0
UI SERVINGS	BACON	2.0
ENDATA		

### 9.3. Program Results

H02BUF Example Program Results

Calls to E04MHF

Print Level = 5

\*\*\* E04MFF  
 \*\*\* Start of NAG Library implementation details \*\*\*

Implementation title: Generalised Base Version  
 Precision: FORTRAN double precision  
 Product Code: FLBAS16A  
 Mark: 16A

\*\*\* End of NAG Library implementation details \*\*\*

Parameters

Problem type.....	LP		
Linear constraints.....	3	Feasibility tolerance..	1.05E-08
Variables.....	6	Crash tolerance.....	1.00E-02
Infinite bound size....	1.00E+20	COLD start.....	
Infinite step size.....	1.00E+20	EPS (machine precision)	1.11E-16
Check frequency.....	50	Expand frequency.....	10000
Print level.....	5	Iteration limit.....	50
Monitoring file.....	-1		
Workspace provided is	IWORK(	103),	WORK( 5802).
To solve problem we need	IWORK(	15),	WORK( 89).

Itn	Step	Ninf	Sinf/Objective	Norm Gz
0	0.0E+00	3	1.799000E+03	0.0E+00
1	1.5E-02	1	2.550000E+02	0.0E+00
2	1.4E-03	0	1.271429E+02	0.0E+00
3	8.7E-02	0	1.129048E+02	0.0E+00
4	2.1E-01	0	1.062857E+02	0.0E+00
5	1.9E+00	0	9.733333E+01	0.0E+00
6	2.9E+00	0	9.250000E+01	0.0E+00

Exit E04MFF - Optimal LP solution.

Final LP objective value = 92.50000

Exit from LP problem after 6 iterations.

Varbl	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
OATMEAL	UL	4.00000	0.000000	4.00000	-3.188	0.0000
CHICKEN	LL	0.000000	0.000000	3.00000	12.47	0.0000
EGGS	LL	0.000000	0.000000	2.00000	4.000	0.0000
MILK	FR	4.50000	0.000000	8.00000	0.0000	3.500
PIE	UL	2.00000	0.000000	2.00000	-3.625	0.0000
BACON	LL	0.000000	0.000000	2.00000	4.375	0.0000

L Con	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
ENERGY	LL	2000.00	2000.00	None	5.6250E-02	0.0000
PROTEIN	FR	60.0000	55.0000	None	0.0000	5.000
CALCIUM	FR	1334.50	800.000	None	0.0000	534.5

---



## H02BVF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

H02BVF prints the solution to a linear or integer programming problem computed by E04MFF or H02BBF and H02BZF, with user supplied names for the rows and columns.

### 2. Specification

```

SUBROUTINE H02BVF (N, M, A, LDA, BL, BU, X, CLAMDA, ISTATE, CRNAME,
1                IFAIL)
    INTEGER          N, M, LDA, ISTATE(N+M), IFAIL
    real            A(LDA,N), BL(N+M), BU(N+M), X(N), CLAMDA(N+M)
    CHARACTER*8     CRNAME(N+M)

```

### 3. Description

H02BVF prints the solution to a linear or integer programming problem with user supplied names for the rows and columns. All output is written to the current advisory message unit (as defined by X04ABF). The routine must be preceded in the same program by calls to H02BUF and either E04MFF (if an LP problem has been solved) or H02BBF and H02BZF (if an IP problem has been solved). The documents for H02BUF, E04MFF and/or H02BBF and H02BZF should be consulted for further details.

### 4. References

- [1] MPSX – Mathematical Programming System.  
Program number 5734 XM4, IBM Trade Corporation, New York, 1971.

### 5. Parameters

- 1: N – INTEGER. *Input*  
*On entry:* the number of variables, as returned by H02BUF.  
*Constraint:*  $N > 0$ .
- 2: M – INTEGER. *Input*  
*On entry:* the number of general linear constraints, as returned by H02BUF.  
*Constraint:*  $M \geq 0$ .
- 3: A(LDA,\*) – *real* array. *Input*  
**Note:** the second dimension of the array A must be at least at least N when  $M > 0$ , and at least 1 when  $M = 0$ .  
*On entry:* the matrix of general linear constraints, as returned by H02BUF.
- 4: LDA – INTEGER. *Input*  
*On entry:* this **must** be the same parameter MAXM as supplied to H02BUF.  
*Constraint:*  $LDA \geq \max(1, M)$ .
- 5: BL(N+M) – *real* array. *Input*  
*On entry:* the lower bounds for all the constraints, as returned by E04MFF or H02BZF.
- 6: BU(N+M) – *real* array. *Input*  
*On entry:* the upper bounds for all the constraints, as returned by E04MFF or H02BZF.

- 7:  $X(N)$  – *real* array. *Input*  
*On entry:* the solution to the problem, as returned by E04MFF or H02BBF.
- 8: CLAMDA(N+M) – *real* array. *Input*  
*On entry:* the Lagrange multipliers (reduced costs) for each constraint with respect to the working set, as returned by E04MFF or H02BZF.
- 9: ISTATE(N+M) – INTEGER array. *Input*  
*On entry:* the status of every constraint in the working set at the solution, as returned by E04MFF or H02BZF.
- 10: CRNAME(N+M) – CHARACTER\*8 array. *Input*  
*On entry:* the user defined names for all the variables and constraints, as returned by H02BUF.
- 11: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry,  $N \leq 0$ ,  
 or  $M < 0$ ,  
 or  $LDA < \max(1, M)$ .

## 7. Accuracy

Not applicable.

## 8. Further Comments

None.

## 9. Example

See the example for H02BUF.

---



## H02BZF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

**Warning:** the specification of the parameter LIWORK changed at Mark 16: the minimum dimension of the array IWORK has been increased by  $N + 3$ .

### 1. Purpose

H02BZF extracts more information associated with the solution of an integer programming problem computed by H02BBF.

### 2. Specification

```

SUBROUTINE H02BZF (N, M, BL, BU, CLAMDA, ISTATE, IWORK, LIWORK, RWORK,
1                LRWORK, IFAIL)
INTEGER          N, M, ISTATE(N+M), IWORK(LIWORK), LIWORK, LRWORK, IFAIL
real           BL(N+M), BU(N+M), CLAMDA(N+M), RWORK(LRWORK)

```

### 3. Description

H02BZF extracts the following information associated with the solution of an integer programming problem computed by H02BBF: the upper and lower bounds used for the solution, the Lagrange multipliers (costs), and the status of the variables at the solution.

In the branch and bound method employed by H02BBF, the arrays BL and BU are used to impose restrictions on the values of the integer variables in each sub-problem. That is, if the variable  $x_j$  is restricted to take value  $v_j$  in a particular sub-problem, then  $BL(j) = BU(j) = v_j$  is set in the sub-problem. Thus, on exit from this routine, some of the elements of BL and BU which correspond to integer variables may contain these imposed values, rather than those originally supplied to H02BBF.

### 4. References

None.

### 5. Parameters

- 1: N – INTEGER. *Input*  
*On entry:* this **must** be the same parameter N as supplied to H02BBF.  
*Constraint:*  $N > 0$ .
- 2: M – INTEGER. *Input*  
*On entry:* this **must** be the same parameter M as supplied to H02BBF.  
*Constraint:*  $M \geq 0$ .
- 3: BL(N+M) – *real* array. *Output*  
*On exit:* if H02BBF exits with IFAIL = 0, 7 or 9, the values in the array BL contain the lower bounds imposed on the integer solution for all the constraints. The first N elements contain the lower bounds on the variables, and the next M elements contain the lower bounds for the general linear constraints (if any).
- 4: BU(N+M) – *real* array. *Output*  
*On exit:* if H02BBF exits with IFAIL = 0, 7 or 9, the values in the array BU contain the upper bounds imposed on the integer solution for all the constraints. The first N elements contain the upper bounds on the variables, and the next M elements contain the upper bounds for the general linear constraints (if any).

- 5: CLAMDA(N+M) – *real* array. *Output*

*On exit:* if H02BBF exits with IFAIL = 0, 7 or 9, the values in the array CLAMDA contain the values of the Lagrange multipliers for each constraint with respect to the current working set. The first N elements contain the multipliers (reduced costs) for the bound constraints on the variables, and the next M elements contain the multipliers (shadow costs) for the general linear constraints (if any).

- 6: ISTATE(N+M) – INTEGER array. *Output*

*On exit:* if H02BBF exits with IFAIL = 0, 7 or 9, the values in the array ISTATE indicate the status of the constraints in the working set at an integer solution. Otherwise, ISTATE indicates the composition of the working set at the final iterate. The significance of each possible value of ISTATE(*j*) is as follows:

ISTATE( <i>j</i> )	Meaning
-2	The constraint violates its lower bound by more than TOLFES (the feasibility tolerance, see H02BBF).
-1	The constraint violates its upper bound by more than TOLFES.
0	The constraint is satisfied to within TOLFES, but is not in the working set.
1	This inequality constraint is included in the working set at its lower bound.
2	This inequality constraint is included in the working set at its upper bound.
3	This constraint is included in the working set as an equality. This value of ISTATE can occur only when BL( <i>j</i> ) = BU( <i>j</i> ).
4	This corresponds to an integer solution being declared with $x_j$ being temporarily fixed at its current value. This value of ISTATE can occur only when IFAIL = 0, 7 or 9 on exit from H02BBF.

- 7: IWORK(LIWORK) – INTEGER array. *Workspace*

This **must** be the same parameter IWORK as supplied to H02BBF. It is used to pass information from H02BBF to H02BZF and therefore the contents of this array **must not** be changed before calling H02BZF.

- 8: LIWORK – INTEGER. *Input*

*On entry:* the dimension of the array IWORK as declared in the (sub)program from which H02BZF is called.

- 9: RWORK(LRWORK) – *real* array. *Workspace*

This **must** be the same parameter RWORK as supplied to H02BBF. It is used to pass information from H02BBF to H02BZF and therefore the contents of this array **must not** be changed before calling H02BZF.

- 10: LRWORK – INTEGER. *Input*

*On entry:* the dimension of the array RWORK as declared in the (sub)program from which H02BZF is called.

- 11: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry,  $N \leq 0$ ,  
or  $M < 0$ .

## 7. Accuracy

Not applicable.

## 8. Further Comments

None.

## 9. Example

One of the applications of integer programming is to the so-called diet problem. Given the nutritional content of a selection of foods, the cost of each food, the amount available of each food and the consumer's minimum daily energy requirements, the problem is to find the cheapest combination. This gives rise to the following problem:

minimize

$$c^T x$$

subject to

$$Ax \geq b,$$

$$0 \leq x \leq u,$$

where

$$c = (3 \ 24 \ 13 \ 9 \ 20 \ 19)^T, \quad x = (x_1, x_2, x_3, x_4, x_5, x_6)^T \text{ is integer,}$$

$$A = \begin{pmatrix} 110 & 205 & 160 & 160 & 420 & 260 \\ 4 & 32 & 13 & 8 & 4 & 14 \\ 2 & 12 & 54 & 285 & 22 & 80 \end{pmatrix}, \quad b = \begin{pmatrix} 2000 \\ 55 \\ 800 \end{pmatrix} \text{ and}$$

$$u = (4 \ 3 \ 2 \ 8 \ 2 \ 2)^T.$$

The rows of  $A$  correspond to energy, protein and calcium and the columns of  $A$  correspond to oatmeal, chicken, eggs, milk, pie and bacon respectively.

The following program solves the above problem to obtain the optimal integer solution and then examines the effect of increasing the energy required to 2200 units.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      H02BZF Example Program Text
*      Mark 16 Revised. NAG Copyright 1993.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5, NOUT=6)
      INTEGER          NMAX, MMAX
      PARAMETER        (NMAX=10, MMAX=10)
      INTEGER          LDA
      PARAMETER        (LDA=MMAX)
      INTEGER          LIWORK, LRWORK
      PARAMETER        (LIWORK=1000, LRWORK=1000)
*      .. Local Scalars ..
      real            BIGBND, INIVAL, OBJMIP, TOLFES, TOLIV
      INTEGER          I, IFAIL, INTFST, ITMAX, J, M, MAXDPT, MAXNOD,
+                     MSGLVL, N
*      .. Local Arrays ..
      real            A(LDA, NMAX), BL(NMAX+MMAX), BU(NMAX+MMAX),
+                     CLAMDA(NMAX+MMAX), CVEC(NMAX), RWORK(LRWORK),
+                     X(NMAX)
      INTEGER          INTVAR(NMAX), ISTATE(NMAX+MMAX), IWORK(LIWORK)
      CHARACTER*8     NAMES(NMAX+MMAX)
```

```

* .. External Subroutines ..
EXTERNAL      H02BBF, H02BZF, OUTSOL
* .. Executable Statements ..
WRITE (NOUT,*) 'H02BZF Example Program Results'
* Skip heading in data file
READ (NIN,*)
READ (NIN,*) N, M
IF (N.LE.NMAX .AND. M.LE.MMAX) THEN
*
*   Read ITMAX, MSGGLVL, MAXNOD, INTFST, MAXDPT, TOLFES, TOLIV,
*   CVEC, A, BIGBND, BL, BU, INTVAR and X from data file
*
  READ (NIN,*) ITMAX, MSGGLVL
  READ (NIN,*) MAXNOD
  READ (NIN,*) INTFST, MAXDPT
  READ (NIN,*) TOLFES, TOLIV
  READ (NIN,*) (CVEC(J),J=1,N)
  READ (NIN,*) (NAMES(J),(A(I,J),I=1,M),J=1,N)
  READ (NIN,*) BIGBND
  READ (NIN,*) (BL(I),I=1,N)
  READ (NIN,*) (NAMES(N+I),BL(N+I),I=1,M)
  READ (NIN,*) (BU(I),I=1,N+M)
  READ (NIN,*) (INTVAR(I),I=1,N)
  READ (NIN,*) (X(I),I=1,N)
*
*   Solve the IP problem using H02BBF
*
  IFAIL = -1
*
  CALL H02BBF(ITMAX,MSGGLVL,N,M,A,LDA,BL,BU,INTVAR,CVEC,MAXNOD,
+           INTFST,MAXDPT,TOLIV,TOLFES,BIGBND,X,OBJMIP,IWORK,
+           LIWORK,RWORK,LRWORK,IFAIL)
*
  IF (IFAIL.EQ.0 .OR. IFAIL.EQ.7 .OR. IFAIL.EQ.9) THEN
*   WRITE (NOUT,99999) 'IP objective value = ', OBJMIP
*
*   Get information about the solution
*
  IFAIL = 0
*
  CALL H02BZF(N,M,BL,BU,CLAMDA,ISTATE,IWORK,LIWORK,RWORK,
+           LRWORK,IFAIL)
*
*   Print the solution
*
  CALL OUTSOL(N,M,A,LDA,BL,BU,X,ISTATE,CLAMDA,BIGBND,NAMES,
+           NOUT)
*
*   Increase the energy requirements and solve the modified IP
*   problem using the current IP solution as the starting point
*
  INIVAL = BL(N+1)
  READ (NIN,*) BL(N+1)
  WRITE (NOUT,99998) 'Increase the energy requirements from',
+           INIVAL, 'to', BL(N+1)
*
  IFAIL = -1
*
  CALL H02BBF(ITMAX,MSGGLVL,N,M,A,LDA,BL,BU,INTVAR,CVEC,MAXNOD,
+           INTFST,MAXDPT,TOLIV,TOLFES,BIGBND,X,OBJMIP,
+           IWORK,LIWORK,RWORK,LRWORK,IFAIL)
*
  IF (IFAIL.EQ.0 .OR. IFAIL.EQ.7 .OR. IFAIL.EQ.9) THEN
*   WRITE (NOUT,99999) 'IP objective value = ', OBJMIP
*
*   Get information about the solution
*
  IFAIL = 0
*

```

```

      CALL H02BZF(N,M,BL,BU,CLAMDA,ISTATE,IWORK,LIWORK,RWORK,
+         LRWORK,IFAIL)
*
*      Print the solution
*
      CALL OUTSOL(N,M,A,LDA,BL,BU,X,ISTATE,CLAMDA,BIGBND,NAMES,
+         NOUT)
*
      ELSE
+         WRITE (NOUT,99997) ' H02BBF terminated with IFAIL = ',
            IFAIL
      END IF
      ELSE
+         WRITE (NOUT,99997) ' H02BBF terminated with IFAIL = ', IFAIL
      END IF
      END IF
      STOP
*
99999 FORMAT (//1X,A,1P,G16.4)
99998 FORMAT (//1X,A,2X,1P,G10.4,2X,A,2X,1P,G10.4)
99997 FORMAT (1X,A,I3)
      END
      SUBROUTINE OUTSOL(N,M,A,LDA,BL,BU,X,ISTATE,CLAMDA,BIGBND,NAMES,
+         NOUT)
*      .. Scalar Arguments ..
      real          BIGBND
      INTEGER       LDA, M, N, NOUT
*      .. Array Arguments ..
      real          A(LDA,*), BL(N+M), BU(N+M), CLAMDA(N+M), X(N)
      INTEGER       ISTATE(N+M)
      CHARACTER*8   NAMES(N+M)
*      .. Local Scalars ..
      real          B1, B2, RES, RES2, V, WLAM
      INTEGER       IS, J, K
      CHARACTER*80  REC
*      .. Local Arrays ..
      CHARACTER*2   LSTATE(-2:4)
*      .. External Functions ..
      real          sdot
      EXTERNAL      sdot
*      .. Intrinsic Functions ..
      INTRINSIC    ABS
*      .. Data statements ..
      DATA         LSTATE(-2)/'--'/, LSTATE(-1)/'++'/,
+         LSTATE(0)/'FR'/, LSTATE(1)/'LL'/,
+         LSTATE(2)/'UL'/, LSTATE(3)/'EQ'/,
+         LSTATE(4)/'TF'/
*      .. Executable Statements ..
*
      WRITE (NOUT,99999)
      DO 20 J = 1, N + M
        B1 = BL(J)
        B2 = BU(J)
        WLAM = CLAMDA(J)
        IS = ISTATE(J)
        IF (J.LE.N) THEN
*          The variables x.
          K = J
          V = X(J)
        ELSE
*          The linear constraints A*x.
          IF (J.EQ.N+1) WRITE (NOUT,99998)
          K = J - N
          V = sdot(N,A(K,1),LDA,X,1)
        END IF
*
*      Print a line for the j-th variable or constraint.
*
        RES = V - B1
        RES2 = B2 - V

```

```

IF (ABS(RES).GT.ABS(RES2)) RES = RES2
WRITE (REC,99997) NAMES(J), LSTATE(IS), V, B1, B2, WLAM, RES
IF (B1.LE.-BIGBND) REC(29:42) = '      None      '
IF (B2.GE.BIGBND) REC(43:56) = '      None      '
WRITE (NOUT,'(A)') REC
20 CONTINUE
RETURN
*
99999 FORMAT (//1X,'Varbl',3X,'State',5X,'Value',5X,'Lower Bound',3X,
+ 'Upper Bound',4X,'Lagr Mult',3X,'Residual',/)
99998 FORMAT (//1X,'L Con',3X,'State',5X,'Value',5X,'Lower Bound',3X,
+ 'Upper Bound',4X,'Lagr Mult',3X,'Residual',/)
99997 FORMAT (1X,A8,2X,A2,1X,1P,3G14.4,1P,G12.4,1P,G12.4)
END
    
```

9.2. Program Data

H02BZF Example Program Data

```

6 3                                :Values of N and M
0 0                                :Values of ITMAX and MSGVLV
0                                  :Value of MAXNOD
0 9                                :Values of INTFST and MAXDPT
0.0 0.0                            :Values of TOLFES and TOLIV
3.0 24.0 13.0 9.0 20.0 19.0       :End of CVEC
'Oatmeal' 110.0 4.0 2.0
'Chicken' 205.0 32.0 12.0
'Eggs'    160.0 13.0 54.0
'Milk'    160.0 8.0 285.0
'Pie'     420.0 4.0 22.0
'Bacon'   260.0 14.0 80.0
1.0E+20                                       :End of matrix A
0.0 0.0 0.0 0.0 0.0 0.0                   :Value of BIGBND
'Energy' 2000.0 'Protein' 55.0 'Calcium' 800.0 :End of BL
4.0 3.0 2.0 8.0 2.0 2.0 1.0E+20 1.0E+20 1.0E+20 :End of BU
1 1 1 1 1 1                                :End of INTVAR
0.0 0.0 0.0 0.0 0.0 0.0                   :End of X
2200.0                                       :Change 'Energy' in RHS
    
```

9.3. Program Results

H02BZF Example Program Results

IP objective value = 97.00

Varbl	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
Oatmeal	EQ	4.000	4.000	4.000	3.000	0.0000
Chicken	LL	0.0000	0.0000	3.000	24.00	0.0000
Eggs	LL	0.0000	0.0000	2.000	13.00	0.0000
Milk	LL	5.000	5.000	8.000	9.000	0.0000
Pie	EQ	2.000	2.000	2.000	20.00	0.0000
Bacon	LL	0.0000	0.0000	2.000	19.00	0.0000

L Con	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
Energy	FR	2080.	2000.	None	0.0000	80.00
Protein	FR	64.00	55.00	None	0.0000	9.000
Calcium	FR	1477.	800.0	None	0.0000	677.0

Increase the energy requirements from 2000. to 2200.

IP objective value = 106.0

Varbl	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
Oatmeal	EQ	4.000	4.000	4.000	3.000	0.0000
Chicken	LL	0.0000	0.0000	3.000	24.00	0.0000
Eggs	LL	0.0000	0.0000	2.000	13.00	0.0000
Milk	LL	6.000	6.000	8.000	9.000	0.0000
Pie	EQ	2.000	2.000	2.000	20.00	0.0000
Bacon	LL	0.0000	0.0000	2.000	19.00	0.0000

L Con	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
Energy	FR	2240.	2200.	None	0.0000	40.00
Protein	FR	72.00	55.00	None	0.0000	17.00
Calcium	FR	1762.	800.0	None	0.0000	962.0

---





## H03ABF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

H03ABF solves the classical Transportation ('Hitchcock') problem.

### 2. Specification

```

SUBROUTINE H03ABF (KOST, MMM, MA, MB, M, K15, MAXIT, K7, K9, NUMIT,
1                K6, K8, K11, K12, Z, IFAIL)
      INTEGER      KOST(MMM,MB), MMM, MA, MB, M, K15(M), MAXIT,
1                K7(M), K9(M), NUMIT, K6(M), K8(M), K11(M), K12(M),
2                IFAIL
      real        Z

```

### 3. Description

H03ABF solves the Transportation problem by minimizing

$$z = \sum_i \sum_j c_{ij} x_{ij}.$$

subject to the constraints

$$\sum_j x_{ij} = A_i \quad (\text{Availabilities})$$

$$\sum_i x_{ij} = B_j \quad (\text{Requirements})$$

where the  $x_{ij}$  can be interpreted as quantities of goods sent from source  $i$  to destination  $j$ , for  $i = 1, 2, \dots, m_a$ ;  $j = 1, 2, \dots, m_b$ , at a cost of  $c_{ij}$  per unit, and it is assumed that  $\sum_i A_i = \sum_j B_j$  and  $x_{ij} \geq 0$ .

H03ABF uses the 'stepping stone' method, modified to accept degenerate cases.

### 4. References

- [1] HADLEY, G.  
 Linear Programming.  
 Addison-Wesley, New York, 1962.

### 5. Parameters

- 1: KOST(MMM,MB) – INTEGER array. *Input*  
*On entry:* the coefficients  $c_{ij}$ , for  $i = 1, 2, \dots, m_a$ ;  $j = 1, 2, \dots, m_b$ .
- 2: MMM – INTEGER. *Input*  
*On entry:* the first dimension of the array KOST as declared in the (sub)program from which H03ABF is called.  
*Constraint:* MMM  $\geq$  MA.
- 3: MA – INTEGER. *Input*  
*On entry:* the number of sources,  $m_a$ .  
*Constraint:* MA  $\geq$  1.
- 4: MB – INTEGER. *Input*  
*On entry:* the number of destinations,  $m_b$ .  
*Constraint:* MB  $\geq$  1.

- 5: M – INTEGER. Input  
*On entry:* the value of  $m_a + m_b$ .
- 6: K15(M) – INTEGER array. Input/Output  
*On entry:* K15( $i$ ) must be set to the availabilities  $A_i$ , for  $i = 1, 2, \dots, m_a$ ; and K15( $m_a + j$ ) must be set to the requirements  $B_j$ , for  $j = 1, 2, \dots, m_b$ .  
*On exit:* the contents of K15 are undefined.
- 7: MAXIT – INTEGER. Input  
*On entry:* the maximum number of iterations allowed.  
*Constraint:* MAXIT  $\geq 1$ .
- 8: K7(M) – INTEGER array. Workspace
- 9: K9(M) – INTEGER array. Workspace
- 10: NUMIT – INTEGER. Output  
*On exit:* the number of iterations performed.
- 11: K6(M) – INTEGER array. Output  
*On exit:* K6( $k$ ), for  $k = 1, 2, \dots, m_a + m_b - 1$ , contains the source indices of the optimal solution (see K11 below).
- 12: K8(M) – INTEGER array. Output  
*On exit:* K8( $k$ ), for  $k = 1, 2, \dots, m_a + m_b - 1$ , contains the destination indices of the optimal solution (see K11 below).
- 13: K11(M) – INTEGER array. Output  
*On exit:* K11( $k$ ), for  $k = 1, 2, \dots, m_a + m_b - 1$ , contains the optimal quantities  $x_{ij}$  which, sent from source  $i = K6(k)$  to destination  $j = K8(k)$ , minimize  $z$ .
- 14: K12(M) – INTEGER array. Output  
*On exit:* K12( $k$ ), for  $k = 1, 2, \dots, m_a + m_b - 1$ , contains the unit cost  $c_{ij}$  associated with the route from source  $i = K6(k)$  to destination  $j = K8(k)$ .
- 15: Z – *real*. Output  
*On exit:* the value of the minimized total cost.
- 16: IFAIL – INTEGER. Input/Output  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry the sum of the availabilities does not equal the sum of the requirements.

IFAIL = 2

During computation MAXIT has been exceeded.

IFAIL = 3

On entry, MAXIT < 1.

IFAIL = 4

On entry, MA < 1,

or MB < 1,

or  $M \neq MA + MB$ ,

or MA > MMM.

## 7. Accuracy

All operations are performed in integer arithmetic so that there are no rounding errors.

## 8. Further Comments

An accurate estimate of the run time for a particular problem is difficult to achieve.

## 9. Example

A company has three warehouses and three stores. The warehouses have a surplus of 12 units of a given commodity divided among them as follows:

Warehouse	Surplus
1	1
2	5
3	6

The stores altogether need 12 units of commodity, with the following requirements:

Store	Requirement
1	4
2	4
3	4

Costs of shipping one unit of the commodity from warehouse  $i$  to store  $j$  are displayed in the following matrix:

		Store		
		1	2	3
Warehouse	1	8	8	11
	2	5	8	14
	3	4	3	10

It is required to find the units of commodity to be moved from the warehouses to the stores, such that the transportation costs are minimized. The maximum number of iterations allowed is 200.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      H03ABF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER      MAMAX, MBMAX, M, MMM
PARAMETER   (MAMAX=5, MBMAX=5, M=MAMAX+MBMAX, MMM=MAMAX)
INTEGER      NIN, NOUT
PARAMETER   (NIN=5, NOUT=6)
*      .. Local Scalars ..
real       Z
INTEGER      I, IFAIL, J, L, MA, MB
```

```

*      .. Local Arrays ..
      INTEGER          K11(M), K12(M), K15(M), K6(M), K7(M), K8(M),
+          K9(M), KOST(MMM,MBMAX)
*      .. External Subroutines ..
      EXTERNAL        H03ABF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'H03ABF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) MA, MB
      IF (MA.GT.0 .AND. MA.LE.MAMAX .AND. MB.GT.0 .AND. MB.LE.MBMAX)
+      THEN
          READ (NIN,*) (K15(I),I=1,MA+MB)
          DO 20 I = 1, MA
              READ (NIN,*) (KOST(I,J),J=1,MB)
20      CONTINUE
          IFAIL = 0
*
          CALL H03ABF(KOST,MMM,MA,MB,MA+MB,K15,200,K7,K9,L,K6,K8,K11,K12,
+              Z,IFAIL)
*
          WRITE (NOUT,*)
          WRITE (NOUT,99999) 'Total cost = ', Z
          WRITE (NOUT,*)
          WRITE (NOUT,*) 'Goods from to'
          WRITE (NOUT,*)
          WRITE (NOUT,99998) (K11(I),K6(I),K8(I),I=1,MA+MB-1)
          END IF
          STOP
*
99999 FORMAT (1X,A,F5.1)
99998 FORMAT (1X,I3,I6,I5)
      END

```

## 9.2. Program Data

```

H03ABF Example Program Data
  3      3
  1      5      6      4      4      4
  8      8      11
  5      8      14
  4      3      10

```

## 9.3. Program Results

H03ABF Example Program Results

Total cost = 77.0

Goods from to

```

  4      3      2
  2      3      3
  1      2      3
  1      1      3
  4      2      1

```

---

## H03ADF – NAG Fortran Library Routine Document

**Note.** Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

### 1 Purpose

H03ADF finds the shortest path through a directed or undirected acyclic network using Dijkstra's algorithm.

### 2 Specification

```

SUBROUTINE H03ADF(N, NS, NE, DIRECT, NNZ, D, IROW, ICOL, SPLN,
1          PATH, IWORK, WORK, IFAIL)
  INTEGER   N, NS, NE, NNZ, IROW(NNZ), ICOL(NNZ),
1          PATH(N), IWORK(3*N+1), IFAIL
  real      D(NNZ), SPLN, WORK(2*N)
  LOGICAL   DIRECT

```

### 3 Description

This routine attempts to find the shortest path through a **directed** or **undirected acyclic** network, which consists of a set of points called **vertices** and a set of curves called **arcs** that connect certain pairs of distinct vertices. An acyclic network is one in which there are no paths connecting a vertex to itself. An arc whose origin vertex is  $i$  and whose destination vertex is  $j$  can be written as  $i \rightarrow j$ . In an undirected network the arcs  $i \rightarrow j$  and  $j \rightarrow i$  are equivalent (i.e.,  $i \leftrightarrow j$ ), whereas in a directed network they are different. Note that the shortest path may not be unique and in some cases may not even exist (e.g. if the network is disconnected).

The network is assumed to consist of  $n$  vertices which are labelled by the integers  $1, 2, \dots, n$ . The lengths of the arcs between the vertices are defined by the  $n$  by  $n$  **distance matrix**  $D$ , in which the element  $d_{ij}$  gives the length of the arc  $i \rightarrow j$ ;  $d_{ij} = 0$  if there is no arc connecting vertices  $i$  and  $j$  (as is the case for an acyclic network when  $i = j$ ). Thus the matrix  $D$  is usually **sparse**. For example, if  $n = 4$  and the network is directed, then

$$D = \begin{pmatrix} 0 & d_{12} & d_{13} & d_{14} \\ d_{21} & 0 & d_{23} & d_{24} \\ d_{31} & d_{32} & 0 & d_{34} \\ d_{41} & d_{42} & d_{43} & 0 \end{pmatrix}.$$

If the network is undirected,  $D$  is **symmetric** since  $d_{ij} = d_{ji}$  (i.e., the length of the arc  $i \rightarrow j \equiv$  the length of the arc  $j \rightarrow i$ ).

The method used by H03ADF is described in detail in Section 8.

### 4 References

- [1] Dijkstra E W (1959) A note on two problems in connection with graphs *Numer. Math.* 1 269–271

### 5 Parameters

- 1: N — INTEGER *Input*  
*On entry:*  $n$ , the number of vertices.  
*Constraint:*  $N \geq 2$ .

- 2: NS — INTEGER *Input*
- 3: NE — INTEGER *Input*
- On entry:*  $n_s$  and  $n_e$ , the labels of the first and last vertices, respectively, between which the shortest path is sought.
- Constraints:*
- $$1 \leq NS \leq N,$$
- $$1 \leq NE \leq N,$$
- $$NS \neq NE.$$
- 4: DIRECT — LOGICAL *Input*
- On entry:* indicates whether the network is directed or undirected as follows:
- if DIRECT = .TRUE., the network is directed;  
if DIRECT = .FALSE., the network is undirected.
- 5: NNZ — INTEGER *Input*
- On entry:* the number of non-zero elements in the distance matrix  $D$ .
- Constraints:*
- if DIRECT = .TRUE.,  $1 \leq NNZ \leq N \times (N-1)$ ;  
if DIRECT = .FALSE.,  $1 \leq NNZ \leq N \times (N-1)/2$ .
- 6: D(NNZ) — *real* array *Input*
- On entry:* the non-zero elements of the distance matrix  $D$ , ordered by increasing row index and increasing column index within each row. More precisely,  $D(k)$  must contain the value of the non-zero element with indices (IROW( $k$ ),ICOL( $k$ )); this is the length of the arc from the vertex with label IROW( $k$ ) to the vertex with label ICOL( $k$ ). Elements with the same row and column indices are not allowed. If DIRECT = .FALSE., then only those non-zero elements in the strict upper triangle of  $D$  need be supplied since  $d_{ij} = d_{ji}$ . (F11ZAF may be used to sort the elements of an arbitrarily ordered matrix into the required form. This is illustrated in Section 9.)
- Constraint:*  $D(k) > 0.0$ , for  $k = 1, 2, \dots, NNZ$ .
- 7: IROW(NNZ) — INTEGER array *Input*
- 8: ICOL(NNZ) — INTEGER array *Input*
- On entry:* IROW( $k$ ) and ICOL( $k$ ) must contain the row and column indices, respectively, for the non-zero element stored in  $D(k)$ .
- Constraints:*
- IROW and ICOL must satisfy the following constraints (which may be imposed by a call to F11ZAF):  
IROW( $k-1$ ) < IROW( $k$ ), or  
IROW( $k-1$ ) = IROW( $k$ ) and ICOL( $k-1$ ) < ICOL( $k$ ), for  $k = 2, 3, \dots, NNZ$ .  
In addition, if DIRECT = .TRUE.,  $1 \leq IROW(k) \leq N$ ,  $1 \leq ICOL(k) \leq N$  and  $IROW(k) \neq ICOL(k)$ ;  
if DIRECT = .FALSE.,  $1 \leq IROW(k) < ICOL(k) \leq N$ .
- 9: SPLEN — *real* *Output*
- On exit:* the length of the shortest path between the specified vertices  $n_s$  and  $n_e$ .
- 10: PATH(N) — INTEGER array *Output*
- On exit:* contains details of the shortest path between the specified vertices  $n_s$  and  $n_e$ . More precisely,  $NS = \text{PATH}(1) \rightarrow \text{PATH}(2) \rightarrow \dots \rightarrow \text{PATH}(p) = NE$  for some  $p \leq n$ . The remaining  $(n-p)$  elements are set to zero.

- 11: IWORK(3\*N+1) — INTEGER array Workspace
- 12: WORK(2\*N) — *real* array Workspace
- 13: IFAIL — INTEGER Input/Output

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6 Error Indicators and Warnings

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors detected by the routine:

IFAIL = 1

On entry,  $N < 2$ ,  
 or  $NS < 1$ ,  
 or  $NS > N$ ,  
 or  $NE < 1$ ,  
 or  $NE > N$ ,  
 or  $NS = NE$ .

IFAIL = 2

On entry,  $NNZ > N \times (N-1)$  when DIRECT = .TRUE.,  
 or  $NNZ > N \times (N-1)/2$  when DIRECT = .FALSE.,  
 or  $NNZ < 1$ .

IFAIL = 3

On entry,  $IROW(k) < 1$  or  $IROW(k) > N$  or  $ICOL(k) < 1$  or  $ICOL(k) > N$  or  $IROW(k) = ICOL(k)$  for some  $k$  when DIRECT = .TRUE..

IFAIL = 4

On entry,  $IROW(k) < 1$  or  $IROW(k) \geq ICOL(k)$  or  $ICOL(k) > N$  for some  $k$  when DIRECT = .FALSE..

IFAIL = 5

On entry,  $D(k) \leq 0.0$  for some  $k$ .

IFAIL = 6

On entry,  $IROW(k-1) > IROW(k)$  or  $IROW(k-1) = IROW(k)$  and  $ICOL(k-1) > ICOL(k)$  for some  $k$ .

IFAIL = 7

On entry,  $IROW(k-1) = IROW(k)$  and  $ICOL(k-1) = ICOL(k)$  for some  $k$ .

IFAIL = 8

No connected network exists between vertices NS and NE.

## 7 Accuracy

The results are exact, except for the obvious rounding errors in summing the distances in the length of the shortest path.

## 8 Further Comments

This routine is based upon Dijkstra's algorithm (see [1]), which attempts to find a path  $n_s \rightarrow n_e$  between two specified vertices  $n_s$  and  $n_e$  of shortest length  $d(n_s, n_e)$ .

The algorithm proceeds by assigning labels to each vertex, which may be **temporary** or **permanent**. A temporary label can be changed, whereas a permanent one cannot. For example, if vertex  $p$  has a permanent label  $(q, r)$ , then  $r$  is the distance  $d(n_s, r)$  and  $q$  is the previous vertex on a shortest length  $n_s \rightarrow p$  path. If the label is temporary, then it has the same meaning but it refers only to the shortest  $n_s \rightarrow p$  path found so far. A shorter one may be found later, in which case the label may become permanent.

The algorithm consists of the following steps.

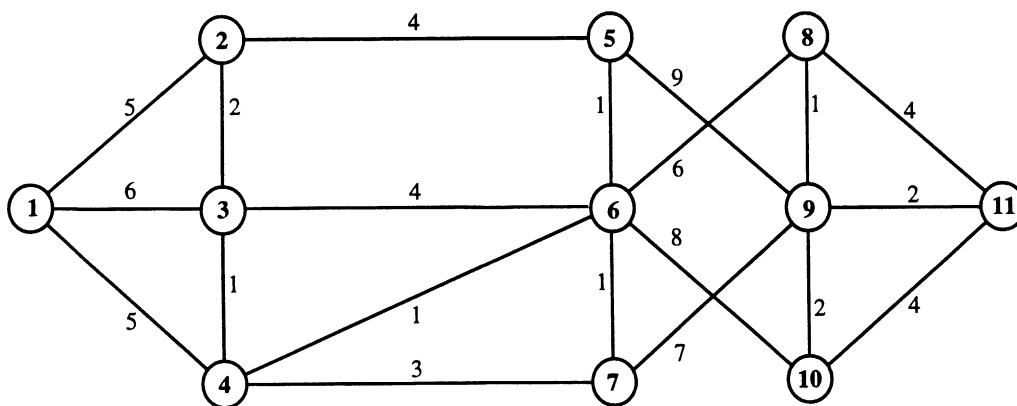
- (1) Assign the permanent label  $(-, 0)$  to vertex  $n_s$  and temporary labels  $(-, \infty)$  to every other vertex. Set  $k = n_s$  and go to (2).
- (2) Consider each vertex  $y$  adjacent to vertex  $k$  with a temporary label in turn. Let the label at  $k$  be  $(p, q)$  and at  $y$  be  $(r, s)$ . If  $q + d_{ky} < s$ , then a new temporary label  $(k, q + d_{ky})$  is assigned to vertex  $y$ ; otherwise no change is made in the label of  $y$ . When all vertices  $y$  with temporary labels adjacent to  $k$  have been considered, go to (3).
- (3) From the set of temporary labels, select the one with the smallest second component and declare that label to be permanent. The vertex it is attached to becomes the new vertex  $k$ . If  $k = n_e$  go to (4). Otherwise go to (2) unless no new vertex can be found (e.g. when the set of temporary labels is 'empty' but  $k \neq n_e$ , in which case no connected network exists between vertices  $n_s$  and  $n_e$ ).
- (4) To find the shortest path, let  $(y, z)$  denote the label of vertex  $n_e$ . The column label ( $z$ ) gives  $d(n_s, n_e)$  while the row label ( $y$ ) then links back to the previous vertex on a shortest length  $n_s \rightarrow n_e$  path. Go to vertex  $y$ . Suppose that the (permanent) label of vertex  $y$  is  $(w, x)$ , then the next previous vertex is  $w$  on a shortest length  $n_s \rightarrow y$  path. This process continues until vertex  $n_s$  is reached. Hence the shortest path is

$$n_s \rightarrow \dots \rightarrow w \rightarrow y \rightarrow n_e,$$

which has length  $d(n_s, n_e)$ .

## 9 Example

To find the shortest path between vertices 1 and 11 for the undirected network



### 9.1 Program Text

```
* H03ADF Example Program Text
* Mark 18 Release. NAG Copyright 1997.
* .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
```



```

INTEGER          NMAX, NNZMAX
PARAMETER        (NMAX=100,NNZMAX=1000)
CHARACTER        DUP, ZERO
PARAMETER        (DUP='Fail',ZERO='Remove')
*
.. Local Scalars ..
  real           SPLEN
INTEGER          IFAIL, J, LENC, N, NE, NNZ, NS
LOGICAL          DIRECT
*
.. Local Arrays ..
  real           D(NNZMAX), WORK(2*NMAX)
INTEGER          ICOL(NNZMAX), IROW(NNZMAX), IWORK(3*NMAX+1),
+               PATH(NMAX)
*
.. External Subroutines ..
EXTERNAL         F11ZAF, H03ADF
*
.. Executable Statements ..
WRITE (NOUT,*) 'H03ADF Example Program Results'
*
Skip heading in data file
READ (NIN,*)
READ (NIN,*) N, NS, NE, NNZ, DIRECT
IF (N.LE.NMAX .AND. NNZ.LE.NNZMAX) THEN
*
*   Read D, IROW and ICOL from data file.
*
  READ (NIN,*) (D(J),IROW(J),ICOL(J),J=1,NNZ)
*
*   Reorder the elements of D into the form required by H03ADF.
*
  IFAIL = 0
  CALL F11ZAF(N,NNZ,D,IROW,ICOL,DUP,ZERO,IWORK,IWORK(N+2),IFAIL)
*
*   Find the shortest path between vertices NS and NE.
*
  IFAIL = 0
  CALL H03ADF(N,NS,NE,DIRECT,NNZ,D,IROW,ICOL,SPLEN,PATH,IWORK,
+           WORK,IFAIL)
*
  IF (IFAIL.EQ.0) THEN
*
*   Print details of shortest path.
*
  DO 20 J = 0, N - 1
    IF (PATH(J+1).EQ.0) THEN
      LENC = J
      GO TO 40
    END IF
  20  CONTINUE
  LENC = N
  40  CONTINUE
  WRITE (NOUT,99999) 'Shortest path = ', (PATH(J),J=1,LENC)
  WRITE (NOUT,99998) 'Length of shortest path = ', SPLEN
  END IF
  END IF
  STOP
*
99999 FORMAT (/1X,A,10(I2,: ' to '))
99998 FORMAT (/1X,A,G16.6)
END

```

## 9.2 Program Data

### H03ADF Example Program Data

```
11 1 11 20 F :Values of N, NS, NE, NNZ and DIRECT
6.0 6 8
1.0 8 9
2.0 9 11
4.0 2 5
1.0 3 4
6.0 1 3
4.0 3 6
1.0 4 6
2.0 2 3
3.0 4 7
5.0 1 2
7.0 6 10
1.0 5 6
4.0 8 11
9.0 5 9
1.0 6 7
8.0 7 9
4.0 10 11
2.0 9 10
5.0 1 4 :End of D, IROW, ICOL
```

## 9.3 Program Results

### H03ADF Example Program Results

Shortest path = 1 to 4 to 6 to 8 to 9 to 11

Length of shortest path = 15.0000

---

## Chapter M01 – Sorting

**Note.** Please refer to the Users' Note for your implementation to check that a routine is available.

<b>Routine Name</b>	<b>Mark of Introduction</b>	<b>Purpose</b>
M01CAF	12	Sort a vector, real numbers
M01CBF	12	Sort a vector, integer numbers
M01CCF	12	Sort a vector, character data
M01DAF	12	Rank a vector, real numbers
M01DBF	12	Rank a vector, integer numbers
M01DCF	12	Rank a vector, character data
M01DEF	12	Rank rows of a matrix, real numbers
M01DFE	12	Rank rows of a matrix, integer numbers
M01DJF	12	Rank columns of a matrix, real numbers
M01DKF	12	Rank columns of a matrix, integer numbers
M01DZF	12	Rank arbitrary data
M01EAF	12	Rearrange a vector according to given ranks, real numbers
M01EBF	12	Rearrange a vector according to given ranks, integer numbers
M01ECF	12	Rearrange a vector according to given ranks, character data
M01ZAF	12	Invert a permutation
M01ZBF	12	Check validity of a permutation
M01ZCF	12	Decompose a permutation into cycles

---



## Chapter M01

### Sorting

#### Contents

<b>1</b>	<b>Scope of the Chapter</b>	<b>2</b>
<b>2</b>	<b>Background to the Problems</b>	<b>2</b>
<b>3</b>	<b>Recommendations on Choice and Use of Available Routines</b>	<b>2</b>
<b>4</b>	<b>Index</b>	<b>3</b>
<b>5</b>	<b>Routines Withdrawn or Scheduled for Withdrawal</b>	<b>4</b>
<b>6</b>	<b>References</b>	<b>4</b>

## 1 Scope of the Chapter

This chapter is concerned with sorting numeric or character data. It handles only the simplest types of data structure and it is concerned only with **internal** sorting – that is, sorting a set of data which can all be stored within the program.

Users with large files of data or complex data structures to be sorted should use a comprehensive sorting program or package.

## 2 Background to the Problems

The usefulness of sorting is obvious (perhaps a little too obvious, since sorting can be expensive and is sometimes done when not strictly necessary). Sorting may traditionally be associated with data processing and non-numerical programming, but it has many uses within the realm of numerical analysis. For example, within the NAG Fortran Library, sorting is used to arrange eigenvalues in ascending order of absolute value; in the manipulation of sparse matrices and in the ranking of observations for nonparametric statistics.

The general problem may be defined as follows. We are given  $N$  items of data

$$R_1, R_2, \dots, R_N.$$

Each item  $R_i$  contains a key  $K_i$  which can be ordered relative to any other key according to some specified criterion (for example, ascending numeric value). The problem is to determine a permutation

$$p(1), p(2), \dots, p(N)$$

which puts the keys in order:

$$K_{p(1)} \leq K_{p(2)} \leq \dots \leq K_{p(N)}$$

Sometimes we may wish actually to **rearrange** the items so that their keys are in order; for other purposes we may simply require a table of **indices** so that the items can be referred to in sorted order; or yet again we may require a table of **ranks**, that is, the positions of each item in the sorted order.

For example, given the single-character items, to be sorted into alphabetic order:

E B A D C

the indices of the items in sorted order are

3 2 5 4 1

and the ranks of the items are

5 2 1 4 3.

Indices may be converted to ranks, and vice versa, by simply computing the inverse permutation.

The items may consist solely of the key (each item may simply be a number). On the other hand, the items may contain additional information (for example, each item may be an eigenvalue of a matrix and its associated eigenvector, the eigenvalue being the key). In the latter case there may be many distinct items with equal keys, and it may be important to preserve the original order among them (if this is achieved, the sorting is called ‘**stable**’).

There are a number of ingenious algorithms for sorting. For a fascinating discussion of them, and of the whole subject, see Knuth [1].

## 3 Recommendations on Choice and Use of Available Routines

**Note.** Refer to the Users’ Note for your implementation to check that a routine is available.

Four categories of routines are provided:

- routines which rearrange the data into sorted order (M01C-);

- routines which determine the ranks of the data, leaving the data unchanged (M01D-);
- routines which rearrange the data according to pre-determined ranks (M01E-);
- service routines (M01Z-).

In the first three categories, routines are provided for *real* and integer numeric data, and for character data. Utilities for the manipulation of sparse matrices can be found in Chapter F11.

If the task is simply to rearrange a one-dimensional array of data into sorted order, then an M01C- routine should be used, since this requires no extra workspace and is faster than any other method. There are no M01C- routines for more complicated data structures, because the cost of rearranging the data is likely to outstrip the cost of comparisons. Instead, a combination of M01D- and M01E- routines, or some other approach, must be used as described below.

For many applications it is in fact preferable to separate the task of determining the sorted order (ranking) from the task of rearranging data into a pre-determined order; the latter task may not need to be performed at all. Frequently it may be sufficient to refer to the data in sorted order via an index vector, without rearranging it. Frequently also one set of data (e.g. a column of a matrix) is used for determining a set of ranks, which are then applied to other data (e.g. the remaining columns of the matrix).

To determine the ranks of a set of data, use an M01D- routine. Routines are provided for ranking one-dimensional arrays, and for ranking rows or columns of two-dimensional arrays. For ranking an arbitrary data structure, use M01DZF, which is, however, much less efficient than the other M01D- routines.

To create an index vector so that data can be referred to in sorted order, first call an M01D- routine to determine the ranks, and then call M01ZAF to convert the vector of ranks into an index vector.

To rearrange data according to pre-determined ranks: use an M01E- routine if the data is stored in a one-dimensional array; or if the data is stored in a more complicated structure

- either** use an index vector to generate a new copy of the data in the desired order
- or** rearrange the data without using extra storage by first calling M01ZCF and then using the simple code-framework given in the document for M01ZCF (assuming that the elements of data all occupy equal storage).

Examples of these operations can be found in the routine documents of the relevant routines.

## 4 Index

### Ranking:

arbitrary data	M01DZF
columns of a matrix, integer numbers	M01DKF
columns of a matrix, <i>real</i> numbers	M01DJF
rows of a matrix, integer numbers	M01DFF
rows of a matrix, <i>real</i> numbers	M01DEF
vector, character data	M01DCF
vector, integer numbers	M01DBF
vector, <i>real</i> numbers	M01DAF

### Rearranging (according to pre-determined ranks):

vector, character data	M01ECF
vector, integer numbers	M01EBF
vector, <i>real</i> numbers	M01EAF

### Service routines:

check validity of a permutation	M01ZBF
decompose a permutation into cycles	M01ZCF
invert a permutation (ranks to indices or vice versa)	M01ZAF

### Sorting (i.e., rearranging into sorted order):

vector, character data	M01CCF
vector, integer numbers	M01CBF
vector, <i>real</i> numbers	M01CAF

## 5 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have been withdrawn. Advice on replacing calls to these routines is given in the document ‘Advice on Replacement Calls for Withdrawn/Superseded Routines’.

M01AAF	M01ABF	M01ACF	M01ADF	M01AEF	M01AFF
M01AGF	M01AHF	M01AJF	M01AKF	M01ALF	M01AMF
M01ANF	M01APF	M01AQF	M01ARF	M01BAF	M01BBF
M01BCF	M01BDF				

## 6 References

- [1] Knuth D E (1973) *The Art of Computer Programming (Volume 3)* Addison-Wesley (2nd Edition)
-



## M01CAF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01CAF rearranges a vector of *real* numbers into ascending or descending order.

### 2. Specification

```

SUBROUTINE M01CAF (RV, M1, M2, ORDER, IFAIL)
  INTEGER          M1, M2, IFAIL
  real           RV(M2)
  CHARACTER*1     ORDER

```

### 3. Description

M01CAF is based on Singleton's implementation of the 'median-of-three' Quicksort algorithm [2], but with two additional modifications. First, small subfiles are sorted by an insertion sort on a separate final pass (Sedgewick [1]). Second, if a subfile is partitioned into two very unbalanced subfiles, the larger of them is flagged for special treatment: before it is partitioned, its end-points are swapped with two random points within it; this makes the worst case behaviour extremely unlikely.

### 4. References

- [1] SEDGEWICK, R.  
Implementing Quicksort programs.  
Comm. ACM 21, pp. 847-857, 1978.
- [2] SINGLETON, R.C.  
An efficient algorithm for sorting with minimal storage: Algorithm 347.  
Comm. ACM 12, pp. 185-187, 1969.

### 5. Parameters

- 1: RV(M2) – *real* array. *Input/Output*  
*On entry:* elements M1 to M2 of RV must contain *real* values to be sorted.  
*On exit:* these values are rearranged into sorted order.
- 2: M1 – INTEGER. *Input*  
*On entry:* the index of the first element of RV to be sorted.  
*Constraint:* M1 > 0.
- 3: M2 – INTEGER. *Input*  
*On entry:* the index of the last element of RV to be sorted.  
*Constraint:* M2 ≥ M1.
- 4: ORDER – CHARACTER\*1. *Input*  
*On entry:* if ORDER is 'A' or 'a', the values will be sorted into ascending (i.e. non-decreasing) order; if ORDER is 'D' or 'd', into descending order.  
*Constraint:* ORDER = 'A', 'a', 'D' or 'd'.

## 5: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M2 < 1,  
or M1 < 1,  
or M1 > M2.

IFAIL = 2

On entry, ORDER is not 'A', 'a', 'D' or 'd'.

## 7. Accuracy

Not applicable.

## 8. Further Comments

The average time taken by the routine is approximately proportional to  $n \times \log n$ , where  $n = M2 - M1 + 1$ . The worst case time is proportional to  $n^2$  but this is extremely unlikely to occur.

## 9. Example

The example program reads a list of *real* numbers and sorts them into ascending order.

## 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      M01CAF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NMAX
      PARAMETER       (NMAX=100)
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5, NOUT=6)
*      .. Local Scalars ..
      INTEGER          I, IFAIL, N
*      .. Local Arrays ..
      real            RV(NMAX)
*      .. External Subroutines ..
      EXTERNAL        M01CAF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'M01CAF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N
      IF (N.GE.1 .AND. N.LE.NMAX) THEN
          READ (NIN,*) (RV(I), I=1, N)
          IFAIL = 0
*

```

```

      CALL M01CAF(RV,1,N,'Ascending',IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Sorted numbers'
      WRITE (NOUT,*)
      WRITE (NOUT,99999) (RV(I),I=1,N)
      END IF
      STOP
*
99999 FORMAT (1X,10F7.1)
      END

```

## 9.2. Program Data

M01CAF Example Program Data  
 16  
 1.3 5.9 4.1 2.3 0.5 5.8 1.3 6.5  
 2.3 0.5 6.5 9.9 2.1 1.1 1.2 8.6

## 9.3. Program Results

M01CAF Example Program Results

Sorted numbers

0.5	0.5	1.1	1.2	1.3	1.3	2.1	2.3	2.3	4.1
5.8	5.9	6.5	6.5	8.6	9.9				

---



## M01CBF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01CBF rearranges a vector of integer numbers into ascending or descending order.

### 2. Specification

```
SUBROUTINE M01CBF (IV, M1, M2, ORDER, IFAIL)
  INTEGER          IV(M2), M1, M2, IFAIL
  CHARACTER*1     ORDER
```

### 3. Description

M01CBF is based on Singleton's implementation of the 'median-of-three' Quicksort algorithm [2], but with two additional modifications. First, small subfiles are sorted by an insertion sort on a separate final pass (Sedgewick [1]). Second, if a subfile is partitioned into two very unbalanced subfiles, the larger of them is flagged for special treatment: before it is partitioned, its end-points are swapped with two random points within it; this makes the worst case behaviour extremely unlikely.

### 4. References

- [1] SEDGEWICK, R.  
Implementing Quicksort programs.  
Comm. ACM 21, pp. 847-857, 1978.
- [2] SINGLETON, R.C.  
An efficient algorithm for sorting with minimal storage: Algorithm 347.  
Comm. ACM 12, pp. 185-187, 1969.

### 5. Parameters

- 1: IV(M2) – INTEGER array. *Input/Output*  
*On entry:* elements M1 to M2 of IV must contain integer values to be sorted.  
*On exit:* these values are rearranged into sorted order.
- 2: M1 – INTEGER. *Input*  
*On entry:* the index of the first element of IV to be sorted.  
*Constraint:* M1 > 0.
- 3: M2 – INTEGER. *Input*  
*On entry:* the index of the last element of IV to be sorted.  
*Constraint:* M2 ≥ M1.
- 4: ORDER – CHARACTER\*1. *Input*  
*On entry:* if ORDER is 'A' or 'a', the values will be sorted into ascending (i.e. non-decreasing) order; if ORDER is 'D' or 'd', into descending order.  
*Constraint:* ORDER = 'A', 'a', 'D' or 'd'.
- 5: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry  $IFAIL = 0$  or  $-1$ , explanatory error messages are output on the current error message unit (as defined by X04AAF).

$IFAIL = 1$

On entry,  $M2 < 1$ ,  
or  $M1 < 1$ ,  
or  $M1 > M2$ .

$IFAIL = 2$

On entry, ORDER is not 'A', 'a', 'D' or 'd'.

## 7. Accuracy

Not applicable.

## 8. Further Comments

The average time taken by the routine is approximately proportional to  $n \times \log n$ , where  $n = M2 - M1 + 1$ . The worst case time is proportional to  $n^2$  but this is extremely unlikely to occur.

## 9. Example

The example program reads a list of integers and sorts them into descending order.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      M01CBF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NMAX
PARAMETER       (NMAX=100)
INTEGER          NIN, NOUT
PARAMETER       (NIN=5, NOUT=6)
*      .. Local Scalars ..
INTEGER          I, IFAIL, N
*      .. Local Arrays ..
INTEGER          IV(NMAX)
*      .. External Subroutines ..
EXTERNAL        M01CBF
*      .. Executable Statements ..
WRITE (NOUT,*) 'M01CBF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
READ (NIN,*) N
IF (N.GE.1 .AND. N.LE.NMAX) THEN
  READ (NIN,*) (IV(I), I=1, N)
  IFAIL = 0
*
  CALL M01CBF(IV, 1, N, 'Descending', IFAIL)
*
  WRITE (NOUT,*)
  WRITE (NOUT,*) 'Sorted numbers'
  WRITE (NOUT,*)
  WRITE (NOUT,99999) (IV(I), I=1, N)
END IF
STOP
*
99999 FORMAT (1X,10I7)
END
```

**9.2. Program Data**

M01CBF Example Program Data  
16  
23 45 45 67 69 90 999 1  
78 112 24 69 96 99 45 78

**9.3. Program Results**

M01CBF Example Program Results  
Sorted numbers

999	112	99	96	90	78	78	69	69	67
45	45	45	24	23	1				

---





## M01CCF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01CCF rearranges a vector of character data so that a specified substring is in ASCII or reverse ASCII order.

### 2. Specification

```

SUBROUTINE M01CCF (CH, M1, M2, L1, L2, ORDER, IFAIL)
  INTEGER          M1, M2, L1, L2, IFAIL
  CHARACTER*1     ORDER
  CHARACTER*(*)   CH(M2)

```

### 3. Description

M01CCF is based on Singleton's implementation of the 'median-of-three' Quicksort algorithm [2], but with two additional modifications. First, small subfiles are sorted by an insertion sort on a separate final pass (Sedgewick [1]). Second, if a subfile is partitioned into two very unbalanced subfiles, the larger of them is flagged for special treatment: before it is partitioned, its end-points are swapped with two random points within it; this makes the worst case behaviour extremely unlikely.

Only the substring (L1:L2) of each element of the array CH is used to determine the sorted order, but the entire elements are rearranged into sorted order.

### 4. References

- [1] SEDGEWICK, R.  
Implementing Quicksort programs.  
Comm. ACM 21, pp. 847-857, 1978.
- [2] SINGLETON, R.C.  
An efficient algorithm for sorting with minimal storage: Algorithm 347.  
Comm. ACM 12, pp. 185-187, 1969.

### 5. Parameters

- 1: CH(M2) – CHARACTER\*(\*) array. *Input/Output*  
*On entry:* elements M1 to M2 of CH must contain character data to be sorted.  
*Constraint:* the length of each element of CH must not exceed 255.  
*On exit:* these values are rearranged into sorted order.
- 2: M1 – INTEGER. *Input*  
*On entry:* the index of the first element of CH to be sorted.  
*Constraint:* M1 > 0.
- 3: M2 – INTEGER. *Input*  
*On entry:* the index of the last element of CH to be sorted.  
*Constraint:* M2 ≥ M1.

- 4: L1 – INTEGER. *Input*  
 5: L2 – INTEGER. *Input*  
*On entry:* only the substring (L1:L2) of each element of CH is to be used in determining the sorted order.  
*Constraint:*  $0 < L1 \leq L2 \leq \text{LEN}(\text{CH}(1))$ .
- 6: ORDER – CHARACTER\*1. *Input*  
*On entry:* if ORDER is 'A' or 'a', the values will be sorted into ASCII order; if ORDER is 'R' or 'r', into reverse ASCII order.  
*Constraint:* ORDER = 'A', 'a', 'R' or 'r'.
- 7: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M2 < 1,  
 or M1 < 1,  
 or M1 > M2,  
 or L2 < 1,  
 or L1 < 1,  
 or L1 > L2,  
 or L2 > LEN(CH(1)).

IFAIL = 2

On entry, ORDER is not 'A', 'a', 'R' or 'r'.

IFAIL = 3

On entry, the length of each element of CH exceeds 255.

## 7. Accuracy

Not applicable.

## 8. Further Comments

The average time taken by the routine is approximately proportional to  $n \times \log n$ , where  $n = M2 - M1 + 1$ . The worst case time is proportional to  $n^2$ , but this is extremely unlikely to occur.

The routine relies on the Fortran 77 intrinsic functions LLT and LGT to order characters according to the ASCII collating sequence.

## 9. Example

The example program reads a file of 12-character records, and sorts them into reverse ASCII order on characters 7 to 12.

## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      M01CCF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          MMAX
      PARAMETER        (MMAX=100)
*      .. Local Scalars ..
      INTEGER          I, IFAIL, L1, L2, M
*      .. Local Arrays ..
      CHARACTER*12     CH(MMAX)
*      .. External Subroutines ..
      EXTERNAL         M01CCF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'M01CCF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      DO 20 M = 1, MMAX
         READ (NIN,'(A)',END=40) CH(M)
20    CONTINUE
40    M = M - 1
         L1 = 7
         L2 = 12
         IFAIL = 0
*
         CALL M01CCF(CH,1,M,L1,L2,'Reverse ASCII',IFAIL)
*
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Records sorted on columns ', L1, ' to ', L2
         WRITE (NOUT,*)
         WRITE (NOUT,99998) (CH(I),I=1,M)
         STOP
*
99999  FORMAT (1X,A,I2,A,I2)
99998  FORMAT (1X,A)
      END

```

## 9.2. Program Data

```

M01CCF Example Program Data
A02AAF  289
A02ABF  523
A02ACF  531
C02ADF  169
C02AEF  599
C05ADF  1351
C05AGF  240
C05AJF  136
C05AVF  211
C05AXF  183
C05AZF  2181

```

**9.3. Program Results**

M01CCF Example Program Results

Records sorted on columns 7 to 12

C05AZF	2181
C05ADF	1351
C02AEF	599
A02ACF	531
A02ABF	523
A02AAF	289
C05AGF	240
C05AVF	211
C05AXF	183
C02ADF	169
C05AJF	136

---

## M01DAF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01DAF ranks a vector of *real* numbers in ascending or descending order.

### 2. Specification

```

SUBROUTINE M01DAF (RV, M1, M2, ORDER, IRANK, IFAIL)
  INTEGER          M1, M2, IRANK(M2), IFAIL
  real           RV(M2)
  CHARACTER*1     ORDER

```

### 3. Description

M01DAF uses a variant of list-merging, as described by Knuth [1] pp. 165-166. The routine takes advantage of natural ordering in the data, and uses a simple list insertion in a preparatory pass to generate ordered lists of length at least 10. The ranking is stable: equal elements preserve their ordering in the input data.

### 4. References

- [1] KNUTH, D.E.  
The Art of Computer Programming, (Vol. 3, Sorting and Searching).  
Addison Wesley, 1973

### 5. Parameters

- 1: RV(M2) – *real* array. *Input*  
*On entry:* elements M1 to M2 of RV must contain *real* values to be ranked.
- 2: M1 – INTEGER. *Input*  
*On entry:* the index of the first element of RV to be ranked.  
*Constraint:* M1 > 0.
- 3: M2 – INTEGER. *Input*  
*On entry:* the index of the last element of RV to be ranked.  
*Constraint:* M2 ≥ M1.
- 4: ORDER – CHARACTER\*1. *Input*  
*On entry:* if ORDER is 'A' or 'a', the values will be ranked in ascending (i.e. non-decreasing) order; if ORDER is 'D' or 'd', into descending order.  
*Constraint:* ORDER = 'A', 'a', 'D' or 'd'.
- 5: IRANK(M2) – INTEGER array. *Output*  
*On exit:* elements M1 to M2 of IRANK contain the ranks of the corresponding elements of RV. Note that the ranks are in the range M1 to M2: thus, if RV(*i*) is the first element in the rank order, IRANK(*i*) is set to M1.
- 6: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry `IFAIL = 0` or `-1`, explanatory error messages are output on the current error message unit (as defined by `X04AAF`).

`IFAIL = 1`

On entry, `M2 < 1`,  
or `M1 < 1`,  
or `M1 > M2`.

`IFAIL = 2`

On entry, `ORDER` is not 'A', 'a', 'D' or 'd'.

## 7. Accuracy

Not applicable.

## 8. Further Comments

The average time taken by the routine is approximately proportional to  $n \times \log n$ , where  $n = M2 - M1 + 1$ .

## 9. Example

The example program reads a list of *real* numbers and ranks them in ascending order.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      M01DAF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NMAX
      PARAMETER       (NMAX=100)
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5, NOUT=6)
*      .. Local Scalars ..
      INTEGER          I, IFAIL, N
*      .. Local Arrays ..
      real            RV(NMAX)
      INTEGER          IRANK(NMAX)
*      .. External Subroutines ..
      EXTERNAL        M01DAF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'M01DAF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N
      IF (N.GE.1 .AND. N.LE.NMAX) THEN
          READ (NIN,*) (RV(I), I=1, N)
          IFAIL = 0
*

```

```

      CALL M01DAF(RV,1,N,'Ascending',IRANK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) '  Data  Ranks'
      WRITE (NOUT,*)
      DO 20 I = 1, N
        WRITE (NOUT,99999) RV(I), IRANK(I)
20    CONTINUE
      END IF
      STOP
*
99999 FORMAT (1X,F7.1,I7)
      END

```

## 9.2. Program Data

M01DAF Example Program Data

12

5.3 4.6 7.8 1.7 5.3 9.9 3.2 4.3 7.8 4.5 1.2 7.6

## 9.3. Program Results

M01DAF Example Program Results

Data	Ranks
5.3	7
4.6	6
7.8	10
1.7	2
5.3	8
9.9	12
3.2	3
4.3	4
7.8	11
4.5	5
1.2	1
7.6	9

---





## M01DBF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01DBF ranks a vector of integer numbers in ascending or descending order.

### 2. Specification

```

SUBROUTINE M01DBF (IV, M1, M2, ORDER, IRANK, IFAIL)
  INTEGER          IV(M2), M1, M2, IRANK(M2), IFAIL
  CHARACTER*1     ORDER

```

### 3. Description

M01DBF uses a variant of list-merging, as described by Knuth [1] pp. 165-166. The routine takes advantage of natural ordering in the data, and uses a simple list insertion in a preparatory pass to generate ordered lists of length at least 10. The ranking is stable: equal elements preserve their ordering in the input data.

### 4. References

- [1] KNUTH, D.E.  
The Art of Computer Programming, (Vol. 3, Sorting and Searching).  
Addison Wesley, 1973

### 5. Parameters

- 1: IV(M2) – INTEGER array. *Input*  
*On entry:* elements M1 to M2 of IV must contain integer values to be ranked.
- 2: M1 – INTEGER. *Input*  
*On entry:* the index of the first element of IV to be ranked.  
*Constraint:* M1 > 0.
- 3: M2 – INTEGER. *Input*  
*On entry:* M2 must specify the index of the last element of IV to be ranked.  
*Constraint:* M2 ≥ M1.
- 4: ORDER – CHARACTER\*1. *Input*  
*On entry:* if ORDER is 'A' or 'a', the values will be ranked in ascending (i.e. non-decreasing) order; if ORDER is 'D' or 'd', into descending order.  
*Constraint:* ORDER = 'A', 'a', 'D' or 'd'.
- 5: IRANK(M2) – INTEGER array. *Output*  
*On exit:* elements M1 to M2 of IRANK contain the ranks of the corresponding elements of IV. Note that the ranks are in the range M1 to M2: thus, if IV(*i*) is the first element in the rank order, IRANK(*i*) is set to M1.
- 6: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry `IFAIL = 0` or `-1`, explanatory error messages are output on the current error message unit (as defined by `X04AAF`).

`IFAIL = 1`

On entry, `M2 < 1`,  
or `M1 < 1`,  
or `M1 > M2`.

`IFAIL = 2`

On entry, `ORDER` is not 'A', 'a', 'D' or 'd'.

## 7. Accuracy

Not applicable.

## 8. Further Comments

The average time taken by the routine is approximately proportional to  $n \times \log n$ , where  $n = M2 - M1 + 1$ .

## 9. Example

The example program reads a list of integers and ranks them in descending order.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      M01DBF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
          INTEGER          NMAX
          PARAMETER        (NMAX=100)
          INTEGER          NIN, NOUT
          PARAMETER        (NIN=5, NOUT=6)
*      .. Local Scalars ..
          INTEGER          I, IFAIL, N
*      .. Local Arrays ..
          INTEGER          IRANK(NMAX), IV(NMAX)
*      .. External Subroutines ..
          EXTERNAL        M01DBF
*      .. Executable Statements ..
          WRITE (NOUT,*) 'M01DBF Example Program Results'
          Skip heading in data file
          READ (NIN,*)
          READ (NIN,*) N
          IF (N.GE.1 .AND. N.LE.NMAX) THEN
              READ (NIN,*) (IV(I), I=1, N)
              IFAIL = 0
          *

```

```

        CALL M01DBF(IV,1,N,'Descending',IRANK,IFAIL)
*
        WRITE (NOUT,*)
        WRITE (NOUT,*) '  Data  Ranks'
        WRITE (NOUT,*)
        DO 20 I = 1, N
            WRITE (NOUT,99999) IV(I), IRANK(I)
20      CONTINUE
        END IF
        STOP
*
99999 FORMAT (1X,2I7)
        END

```

## 9.2. Program Data

M01DBF Example Program Data

12

34 44 89 64 69 69 23 1 999 65 22 76

## 9.3. Program Results

M01DBF Example Program Results

Data	Ranks
34	9
44	8
89	2
64	7
69	4
69	5
23	10
1	12
999	1
65	6
22	11
76	3

---



## M01DCF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01DCF ranks a vector of character data in ASCII or reverse ASCII order of a specified substring.

### 2. Specification

```

SUBROUTINE M01DCF (CH, M1, M2, L1, L2, ORDER, IRANK, IFAIL)
  INTEGER          M1, M2, L1, L2, IRANK(M2), IFAIL
  CHARACTER*1     ORDER
  CHARACTER*(*)   CH(M2)

```

### 3. Description

M01DCF uses a variant of list-merging, as described by Knuth [1] pp. 165-166. The routine takes advantage of natural ordering in the data, and uses a simple list insertion in a preparatory pass to generate ordered lists of length at least 10. The ranking is stable: equal elements preserve their ordering in the input data.

Only the substring (L1:L2) of each element of the array CH is used to determine the rank order.

### 4. References

- [1] KNUTH, D.E.  
The Art of Computer Programming, (Vol. 3, Sorting and Searching).  
Addison Wesley, 1973

### 5. Parameters

- 1: CH(M2) – CHARACTER\*(\*) array. *Input*  
*On entry:* elements M1 to M2 of CH must contain character data to be ranked.  
*Constraint:* the length of each element of CH must not exceed 255.
- 2: M1 – INTEGER. *Input*  
*On entry:* the index of the first element of CH to be ranked.  
*Constraint:* M1 > 0.
- 3: M2 – INTEGER. *Input*  
*On entry:* the index of the last element of CH to be ranked.  
*Constraint:* M2 ≥ M1.
- 4: L1 – INTEGER. *Input*  
 5: L2 – INTEGER. *Input*  
*On entry:* only the substring (L1:L2) of each element of CH is to be used in determining the rank order.  
*Constraint:* 0 < L1 ≤ L2 ≤ LEN(CH(1))
- 6: ORDER – CHARACTER\*1. *Input*  
*On entry:* if ORDER is 'A' or 'a', the values will be ranked in ASCII order; if ORDER is 'R' or 'r', in reverse ASCII order.  
*Constraint:* ORDER = 'A', 'a', 'R' or 'r'.

- 7: IRANK(M2) – INTEGER array. *Output*

*On exit:* elements M1 to M2 of IRANK contain the ranks of the corresponding elements of CH. Note that the ranks are in the range M1 to M2: thus, if CH(*i*) is the first element in the rank order, IRANK(*i*) is set to M1.

- 8: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M2 < 1,  
or M1 < 1,  
or M1 > M2,  
or L2 < 1,  
or L1 < 1,  
or L1 > L2,  
or L2 > LEN(CH(1)).

IFAIL = 2

On entry, ORDER is not 'A', 'a', 'R' or 'r'.

IFAIL = 3

On entry, the length of each element of CH exceeds 255.

## 7. Accuracy

Not applicable.

## 8. Further Comments

The average time taken by the routine is approximately proportional to  $n \times \log n$ , where  $n = M2 - M1 + 1$ .

The routine relies on the Fortran 77 intrinsic functions LLT and LGT to order characters according to the ASCII collating sequence.

## 9. Example

The example program reads a file of 12-character records, and ranks them in reverse ASCII order on characters 7 to 12.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      M01DCF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5, NOUT=6)
      INTEGER          MMAX
      PARAMETER        (MMAX=100)
```

```

*      .. Local Scalars ..
      INTEGER          I, IFAIL, L1, L2, M
*      .. Local Arrays ..
      INTEGER          IRANK(MMAX)
      CHARACTER*12     CH(MMAX)
*      .. External Subroutines ..
      EXTERNAL         M01DCF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'M01DCF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      DO 20 M = 1, MMAX
        READ (NIN,'(A)',END=40) CH(M)
20    CONTINUE
40    M = M - 1
      L1 = 7
      L2 = 12
      IFAIL = 0
*
      CALL M01DCF(CH,1,M,L1,L2,'Reverse ASCII',IRANK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Records ranked on columns ', L1, ' to ', L2
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Data           Ranks'
      WRITE (NOUT,*)
      WRITE (NOUT,99998) (CH(I),IRANK(I),I=1,M)
      STOP
*
99999  FORMAT (1X,A,I2,A,I2)
99998  FORMAT (1X,A,I7)
      END

```

## 9.2. Program Data

```

M01DCF Example Program Data
A02AAF  289
A02ABF  523
A02ACF  531
C02ADF  169
C02AEF  599
C05ADF 1351
C05AGF  240
C05AJF  136
C05AVF  211
C05AXF  183
C05AZF 2181

```

## 9.3. Program Results

M01DCF Example Program Results

Records ranked on columns 7 to 12

Data		Ranks
A02AAF	289	6
A02ABF	523	5
A02ACF	531	4
C02ADF	169	10
C02AEF	599	3
C05ADF	1351	2
C05AGF	240	7
C05AJF	136	11
C05AVF	211	8
C05AXF	183	9
C05AZF	2181	1





## M01DEF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01DEF ranks the rows of a matrix of *real* numbers in ascending or descending order.

### 2. Specification

```

SUBROUTINE M01DEF (RM, LDM, M1, M2, N1, N2, ORDER, IRANK, IFAIL)
  INTEGER          LDM, M1, M2, N1, N2, IRANK(M2), IFAIL
  real           RM(LDM,N2)
  CHARACTER*1     ORDER

```

### 3. Description

M01DEF ranks rows M1 to M2 of a matrix, using the data in columns N1 to N2 of those rows. The ordering is determined by first ranking the data in column N1, then ranking any tied rows according to the data in column N1 + 1, and so on up to column N2.

M01DEF uses a variant of list-merging, as described by Knuth [1] pp. 165-166. The routine takes advantage of natural ordering in the data, and uses a simple list insertion in a preparatory pass to generate ordered lists of length at least 10. The ranking is stable: equal rows preserve their ordering in the input data.

### 4. References

- [1] KNUTH, D.E.  
The Art of Computer Programming, (Vol. 3, Sorting and Searching).  
Addison Wesley, 1973

### 5. Parameters

- 1: RM(LDM,N2) – *real* array. *Input*  
*On entry:* columns N1 to N2 of rows M1 to M2 of RM must contain *real* data to be ranked.
- 2: LDM – INTEGER. *Input*  
*On entry:* the first dimension of the array RM as declared in the (sub)program from which M01DEF is called.  
*Constraint:* LDM ≥ M2.
- 3: M1 – INTEGER. *Input*  
*On entry:* the index of the first row of RM to be ranked.  
*Constraint:* M1 > 0.
- 4: M2 – INTEGER. *Input*  
*On entry:* the index of the last row of RM to be ranked.  
*Constraint:* M2 ≥ M1.
- 5: N1 – INTEGER. *Input*  
*On entry:* the index of the first column of RM to be used.  
*Constraint:* N1 > 0.

- 6: N2 – INTEGER. *Input*  
*On entry:* the index of the last column of RM to be used.  
*Constraint:*  $N2 \geq N1$ .
- 7: ORDER – CHARACTER\*1. *Input*  
*On entry:* if ORDER is 'A' or 'a', the rows will be ranked in ascending (i.e. non-decreasing) order; if ORDER is 'D' or 'd', into descending order.  
*Constraint:* ORDER = 'A', 'a', 'D' or 'd'.
- 8: IRANK(M2) – INTEGER array. *Output*  
*On exit:* elements M1 to M2 of IRANK contain the ranks of the corresponding rows of RM. Note that the ranks are in the range M1 to M2: thus, if the  $i$ th row of RM is the first in the rank order, IRANK( $i$ ) is set to M1.
- 9: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry,  $M2 < 1$ ,  
 or  $N2 < 1$ ,  
 or  $M1 < 1$ ,  
 or  $M1 > M2$ ,  
 or  $N1 < 1$ ,  
 or  $N1 > N2$ ,  
 or  $LDM < M2$ .

IFAIL = 2

On entry, ORDER is not 'A', 'a', 'D' or 'd'.

## 7. Accuracy

Not applicable.

## 8. Further Comments

The average time taken by the routine is approximately proportional to  $n \times \log n$ , where  $n = M2 - M1 + 1$ .

## 9. Example

The example program reads a matrix of *real* numbers and ranks the rows in ascending order.

## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      M01DEF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=20,NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*      .. Local Scalars ..
      INTEGER          I, IFAIL, J, M, N
*      .. Local Arrays ..
      real            RM(MMAX,NMAX)
      INTEGER          IRANK(MMAX)
*      .. External Subroutines ..
      EXTERNAL         M01DEF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'M01DEF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) M, N
      IF (M.GE.1 .AND. M.LE.MMAX .AND. N.GE.1 .AND. N.LE.NMAX) THEN
        DO 20 I = 1, M
          READ (NIN,*) (RM(I,J),J=1,N)
20      CONTINUE
          IFAIL = 0
*
          CALL M01DEF(RM,MMAX,1,M,1,N,'Ascending',IRANK,IFAIL)
*
          WRITE (NOUT,*)
          WRITE (NOUT,*) 'Data                      Ranks'
          WRITE (NOUT,*)
          DO 40 I = 1, M
            WRITE (NOUT,99999) (RM(I,J),J=1,N), IRANK(I)
40      CONTINUE
          END IF
          STOP
*
99999  FORMAT (1X,3F7.1,I11)
      END

```

## 9.2. Program Data

```

M01DEF Example Program Data
12 3
6.0 5.0 4.0
5.0 2.0 1.0
2.0 4.0 9.0
4.0 9.0 6.0
4.0 9.0 5.0
4.0 1.0 2.0
3.0 4.0 1.0
2.0 4.0 6.0
1.0 6.0 4.0
9.0 3.0 2.0
6.0 2.0 5.0
4.0 9.0 6.0

```

**9.3. Program Results**

## M01DEF Example Program Results

Data			Ranks
6.0	5.0	4.0	11
5.0	2.0	1.0	9
2.0	4.0	9.0	3
4.0	9.0	6.0	7
4.0	9.0	5.0	6
4.0	1.0	2.0	5
3.0	4.0	1.0	4
2.0	4.0	6.0	2
1.0	6.0	4.0	1
9.0	3.0	2.0	12
6.0	2.0	5.0	10
4.0	9.0	6.0	8

---

## M01DFF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01DFF ranks the rows of a matrix of integer numbers in ascending or descending order.

### 2. Specification

```
SUBROUTINE M01DFF (IM, LDM, M1, M2, N1, N2, ORDER, IRANK, IFAIL)
  INTEGER          IM(LDM,N2), LDM, M1, M2, N1, N2, IRANK(M2), IFAIL
  CHARACTER*1      ORDER
```

### 3. Description

M01DFF ranks rows M1 to M2 of a matrix, using the data in columns N1 to N2 of those rows. The ordering is determined by first ranking the data in column N1, then ranking any tied rows according to the data in column N1 + 1, and so on up to column N2.

M01DFF uses a variant of list-merging, as described by Knuth [1] pp. 165-166. The routine takes advantage of natural ordering in the data, and uses a simple list insertion in a preparatory pass to generate ordered lists of length at least 10. The ranking is stable: equal rows preserve their ordering in the input data.

### 4. References

- [1] KNUTH, D.E.  
The Art of Computer Programming, (Vol. 3, Sorting and Searching).  
Addison Wesley, 1973

### 5. Parameters

- 1: IM(LDM,N2) – INTEGER array. *Input*  
*On entry:* columns N1 to N2 of rows M1 to M2 of IM must contain INTEGER data to be ranked.
- 2: LDM – INTEGER. *Input*  
*On entry:* the first dimension of the array IM as declared in the (sub)program from which M01DFF is called.  
*Constraint:* LDM ≥ M2.
- 3: M1 – INTEGER. *Input*  
*On entry:* the index of the first row of IM to be ranked.  
*Constraint:* M1 > 0.
- 4: M2 – INTEGER. *Input*  
*On entry:* the index of the last row of IM to be ranked.  
*Constraint:* M2 ≥ M1.
- 5: N1 – INTEGER. *Input*  
*On entry:* the index of the first column of IM to be used.  
*Constraint:* N1 > 0.

- 6: N2 – INTEGER. *Input*  
*On entry:* the index of the last column of IM to be used.  
*Constraint:*  $N2 \geq N1$ .
- 7: ORDER – CHARACTER\*1. *Input*  
*On entry:* if ORDER is 'A' or 'a', the rows will be ranked in ascending (i.e. non-decreasing) order; if ORDER is 'D' or 'd', into descending order.  
*Constraint:* ORDER = 'A', 'a', 'D' or 'd'.
- 8: IRANK(M2) – INTEGER array. *Output*  
*On exit:* elements M1 to M2 of IRANK contain the ranks of the corresponding rows of IM. Note that the ranks are in the range M1 to M2: thus, if the *i*th row of IM is the first in the rank order, IRANK(*i*) is set to M1.
- 9: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M2 < 1,  
 or N2 < 1,  
 or M1 < 1,  
 or M1 > M2,  
 or N1 < 1,  
 or N1 > N2,  
 or LDM < M2.

IFAIL = 2

On entry, ORDER is not 'A', 'a', 'D' or 'd'.

## 7. Accuracy

Not applicable.

## 8. Further Comments

The average time taken by the routine is approximately proportional to  $n \times \log n$ , where  $n = M2 - M1 + 1$ .

## 9. Example

The example program reads a matrix of integers and ranks the rows in descending order.

## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      M01DFF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          MMAX, NMAX
PARAMETER        (MMAX=20, NMAX=20)
INTEGER          NIN, NOUT
PARAMETER        (NIN=5, NOUT=6)
*      .. Local Scalars ..
INTEGER          I, IFAIL, J, M, N
*      .. Local Arrays ..
INTEGER          IM(MMAX, NMAX), IRANK(MMAX)
*      .. External Subroutines ..
EXTERNAL         M01DFF
*      .. Executable Statements ..
WRITE (NOUT,*) 'M01DFF Example Program Results'
Skip heading in data file
READ (NIN,*)
READ (NIN,*) M, N
IF (M.GE.1 .AND. M.LE.MMAX .AND. N.GE.1 .AND. N.LE.NMAX) THEN
  DO 20 I = 1, M
    READ (NIN,*) (IM(I, J), J=1, N)
20  CONTINUE
    IFAIL = 0
*
    CALL M01DFF(IM, MMAX, 1, M, 1, N, 'Descending', IRANK, IFAIL)
*
    WRITE (NOUT,*)
    WRITE (NOUT,*) 'Data                      Ranks'
    WRITE (NOUT,*)
    DO 40 I = 1, M
      WRITE (NOUT, 99999) (IM(I, J), J=1, N), IRANK(I)
, 40  CONTINUE
    END IF
    STOP
*
99999 FORMAT (1X, 3I7, I11)
END

```

## 9.2. Program Data

```

M01DFF Example Program Data
12 3
6 5 4
5 2 1
2 4 9
4 9 6
4 9 5
4 1 2
3 4 1
2 4 6
1 6 4
9 3 2
6 2 5
4 9 6

```

**9.3. Program Results**

## M01DFF Example Program Results

Data			Ranks
6	5	4	2
5	2	1	4
2	4	9	10
4	9	6	5
4	9	5	7
4	1	2	8
3	4	1	9
2	4	6	11
1	6	4	12
9	3	2	1
6	2	5	3
4	9	6	6

---



## M01DJF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01DJF ranks the columns of a matrix of *real* numbers in ascending or descending order.

### 2. Specification

```

SUBROUTINE M01DJF (RM, LDM, M1, M2, N1, N2, ORDER, IRANK, IFAIL)
  INTEGER          LDM, M1, M2, N1, N2, IRANK(N2), IFAIL
  real           RM(LDM,N2)
  CHARACTER*1     ORDER

```

### 3. Description

M01DJF ranks columns N1 to N2 of a matrix, using the data in rows M1 to M2 of those columns. The ordering is determined by first ranking the data in row M1, then ranking any tied columns according to the data in row M1 + 1, and so on up to row M2.

M01DJF uses a variant of list-merging, as described by Knuth [1] pp. 165-166. The routine takes advantage of natural ordering in the data, and uses a simple list insertion in a preparatory pass to generate ordered lists of length at least 10. The ranking is stable: equal columns preserve their ordering in the input data.

### 4. References

[1] KNUTH, D.E.

The Art of Computer Programming (Vol. 3, Sorting and Searching).  
Addison Wesley, 1973

### 5. Parameters

- 1: RM(LDM,N2) – *real* array. *Input*  
*On entry:* rows M1 to M2 of columns N1 to N2 of RM must contain *real* data to be ranked.
- 2: LDM – INTEGER. *Input*  
*On entry:* the first dimension of the array RM as declared in the (sub)program from which M01DJF is called.  
*Constraint:* LDM ≥ M2.
- 3: M1 – INTEGER. *Input*  
*On entry:* the index of the first row of RM to be used.  
*Constraint:* M1 > 0.
- 4: M2 – INTEGER. *Input*  
*On entry:* the index of the last row of RM to be used.  
*Constraint:* M2 ≥ M1.
- 5: N1 – INTEGER. *Input*  
*On entry:* the index of the first column of RM to be ranked.  
*Constraint:* N1 > 0.

- 6: N2 – INTEGER. *Input*  
*On entry:* the index of the last column of RM to be ranked.  
*Constraint:*  $N2 \geq N1$ .
- 7: ORDER – CHARACTER\*1. *Input*  
*On entry:* if ORDER is 'A' or 'a', the columns will be ranked in ascending (i.e. non-decreasing) order; if ORDER is 'D' or 'd', into descending order.  
*Constraint:* ORDER = 'A', 'a', 'D' or 'd'.
- 8: IRANK(N2) – INTEGER array. *Output*  
*On exit:* elements N1 to N2 of IRANK contain the ranks of the corresponding columns of RM. Note that the ranks are in the range N1 to N2: thus, if the *i*th column of RM is the first in the rank order, IRANK(*i*) is set to N1.
- 9: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M2 < 1,  
 or N2 < 1,  
 or M1 < 1,  
 or M1 > M2,  
 or N1 < 1,  
 or N1 > N2,  
 or LDM < M2.

IFAIL = 2

On entry, ORDER is not 'A', 'a', 'D' or 'd'.

## 7. Accuracy

Not applicable.

## 8. Further Comments

The average time taken by the routine is approximately proportional to  $n \times \log n$ , where  $n = N2 - N1 + 1$ .

## 9. Example

The example program reads a matrix of *real* numbers and ranks the columns in ascending order.

## 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      M01DJF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          MMAX, NMAX
PARAMETER        (MMAX=20,NMAX=20)
INTEGER          NIN, NOUT
PARAMETER        (NIN=5,NOUT=6)
*      .. Local Scalars ..
INTEGER          I, IFAIL, J, M, N
*      .. Local Arrays ..
real           RM(MMAX,NMAX)
INTEGER          IRANK(NMAX)
*      .. External Subroutines ..
EXTERNAL         M01DJF
*      .. Executable Statements ..
WRITE (NOUT,*) 'M01DJF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
READ (NIN,*) M, N
IF (M.GE.1 .AND. M.LE.MMAX .AND. N.GE.1 .AND. N.LE.NMAX) THEN
  DO 20 I = 1, M
    READ (NIN,*) (RM(I,J),J=1,N)
20  CONTINUE
    IFAIL = 0
*
*      CALL M01DJF(RM,MMAX,1,M,1,N,'Ascending',IRANK,IFAIL)
*
  WRITE (NOUT,*)
  WRITE (NOUT,*) 'Data'
  WRITE (NOUT,*)
  DO 40 I = 1, M
    WRITE (NOUT,99999) (RM(I,J),J=1,N)
40  CONTINUE
  WRITE (NOUT,*)
  WRITE (NOUT,*) 'Ranks'
  WRITE (NOUT,*)
  WRITE (NOUT,99998) (IRANK(I),I=1,N)
  END IF
  STOP
*
99999 FORMAT (1X,12F6.1)
99998 FORMAT (1X,12I6)
END

```

## 9.2. Program Data

```

M01DJF Example Program Data
3 12
5.0 4.0 3.0 2.0 2.0 1.0 9.0 4.0 4.0 2.0 2.0 1.0
3.0 8.0 2.0 5.0 5.0 6.0 9.0 8.0 9.0 5.0 4.0 1.0
9.0 1.0 6.0 1.0 2.0 4.0 8.0 1.0 2.0 2.0 6.0 2.0

```

9.3. Program Results

M01DJF Example Program Results

Data

5.0	4.0	3.0	2.0	2.0	1.0	9.0	4.0	4.0	2.0	2.0	1.0
3.0	8.0	2.0	5.0	5.0	6.0	9.0	8.0	9.0	5.0	4.0	1.0
9.0	1.0	6.0	1.0	2.0	4.0	8.0	1.0	2.0	2.0	6.0	2.0

Ranks

11	8	7	4	5	2	12	9	10	6	3	1
----	---	---	---	---	---	----	---	----	---	---	---

---

## M01DKF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01DKF ranks the columns of a matrix of integer numbers in ascending or descending order.

### 2. Specification

```
SUBROUTINE M01DKF (IM, LDM, M1, M2, N1, N2, ORDER, IRANK, IFAIL)
  INTEGER          IM(LDM,N2), LDM, M1, M2, N1, N2, IRANK(N2), IFAIL
  CHARACTER*1     ORDER
```

### 3. Description

M01DKF ranks columns N1 to N2 of a matrix, using the data in rows M1 to M2 of those columns. The ordering is determined by first ranking the data in row M1, then ranking any tied columns according to the data in row M1 + 1, and so on up to row M2.

M01DKF uses a variant of list-merging, as described by Knuth [1] pp. 165-166. The routine takes advantage of natural ordering in the data, and uses a simple list insertion in a preparatory pass to generate ordered lists of length at least 10. The ranking is stable: equal columns preserve their ordering in the input data.

### 4. References

- [1] KNUTH, D.E.  
The Art of Computer Programming, (Vol. 3, Sorting and Searching).  
Addison Wesley, 1973

### 5. Parameters

- 1: IM(LDM,N2) – INTEGER array. *Input*  
*On entry:* rows M1 to M2 of columns N1 to N2 of IM must contain integer data to be ranked.
- 2: LDM – INTEGER. *Input*  
*On entry:* the first dimension of the array IM as declared in the (sub)program from which M01DKF is called.  
*Constraint:* LDM ≥ M2.
- 3: M1 – INTEGER. *Input*  
*On entry:* the index of the first row of IM to be used.  
*Constraint:* M1 > 0.
- 4: M2 – INTEGER. *Input*  
*On entry:* the index of the last row of IM to be used.  
*Constraint:* M2 ≥ M1.
- 5: N1 – INTEGER. *Input*  
*On entry:* the index of the first column of IM to be ranked.  
*Constraint:* N1 > 0.

- 6: N2 – INTEGER. *Input*  
*On entry:* the index of the last column of IM to be ranked.  
*Constraint:*  $N2 \geq N1$ .
- 7: ORDER – CHARACTER\*1. *Input*  
*On entry:* if ORDER is 'A' or 'a', the columns will be ranked in ascending (i.e. non-decreasing) order; if ORDER is 'D' or 'd', into descending order.  
*Constraint:* ORDER = 'A', 'a', 'D' or 'd'.
- 8: IRANK(N2) – INTEGER array. *Output*  
*On exit:* elements N1 to N2 of IRANK contain the ranks of the corresponding columns of IM. Note that the ranks are in the range N1 to N2: thus, if the *i*th column of IM is the first in the rank order, IRANK(*i*) is set to N1.
- 9: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M2 < 1,  
 or N2 < 1,  
 or M1 < 1,  
 or M1 > M2,  
 or N1 < 1,  
 or N1 > N2,  
 or LDM < M2.

IFAIL = 2

On entry, ORDER is not 'A', 'a', 'D' or 'd'.

## 7. Accuracy

Not applicable.

## 8. Further Comments

The average time taken by the routine is approximately proportional to  $n \times \log n$ , where  $n = N2 - N1 + 1$ .

## 9. Example

The example program reads a matrix of integers and ranks the columns in descending order.

## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      M01DKF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=20,NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*      .. Local Scalars ..
      INTEGER          I, IFAIL, J, M, N
*      .. Local Arrays ..
      INTEGER          IM(MMAX,NMAX), IRANK(NMAX)
*      .. External Subroutines ..
      EXTERNAL         M01DKF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'M01DKF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) M, N
      IF (M.GE.1 .AND. M.LE.MMAX .AND. N.GE.1 .AND. N.LE.NMAX) THEN
        DO 20 I = 1, M
          READ (NIN,*) (IM(I,J),J=1,N)
20      CONTINUE
          IFAIL = 0
*
*      CALL M01DKF(IM,MMAX,1,M,1,N,'Descending',IRANK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Data'
      WRITE (NOUT,*)
      DO 40 I = 1, M
        WRITE (NOUT,99999) (IM(I,J),J=1,N)
40      CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Ranks'
      WRITE (NOUT,*)
      WRITE (NOUT,99999) (IRANK(I),I=1,N)
      END IF
      STOP
*
99999 FORMAT (1X,12I6)
      END

```

## 9.2. Program Data

```

M01DKF Example Program Data
3 12
5 4 3 2 2 1 9 4 4 2 2 1
3 8 2 5 5 6 9 8 9 5 4 1
9 1 6 1 2 4 8 1 2 2 6 2

```

## 9.3. Program Results

M01DKF Example Program Results

Data

5	4	3	2	2	1	9	4	4	2	2	1
3	8	2	5	5	6	9	8	9	5	4	1
9	1	6	1	2	4	8	1	2	2	6	2

Ranks

2	4	6	9	7	11	1	5	3	8	10	12
---	---	---	---	---	----	---	---	---	---	----	----

---





## M01DZF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01DZF ranks arbitrary data according to a user-supplied comparison routine.

### 2. Specification

```

SUBROUTINE M01DZF (COMPAR, M1, M2, IRANK, IFAIL)
  INTEGER          M1, M2, IRANK(M2), IFAIL
  LOGICAL          COMPARE
  EXTERNAL         COMPARE

```

### 3. Description

M01DZF is a general-purpose routine for ranking arbitrary data. M01DZF does not access the data directly; instead it calls a user-supplied routine COMPARE to determine the relative ordering of any two data items. The data items are identified simply by an integer in the range M1 to M2.

M01DZF uses a variant of list-merging, as described by Knuth [1] pp. 165-166. The routine takes advantage of natural ordering in the data, and uses a simple list insertion in a preparatory pass to generate ordered lists of length at least 10.

### 4. References

- [1] KNUTH, D.E.  
The Art of Computer Programming, (Vol. 3, Sorting and Searching).  
Addison Wesley, 1973.

### 5. Parameters

- 1: COMPARE – LOGICAL FUNCTION, supplied by the user. *External Procedure*

COMPARE must specify the relative ordering of any two data items; it must return `.TRUE.` if item I must come strictly *after* item J in the rank ordering.

Its specification is:

```

LOGICAL FUNCTION COMPARE(I, J)
INTEGER          I, J
1:  I – INTEGER.                               Input
2:  J – INTEGER.                               Input

```

*On entry:* I and J identify the data items to be compared.  
*Constraint:*  $M1 \leq I, J \leq M2$ .

COMPARE must be declared as EXTERNAL in the (sub)program from which M01DZF is called. Parameters denoted as *Input* must not be changed by this procedure.

- 2: M1 – INTEGER. *Input*  
3: M2 – INTEGER. *Input*

*On entry:* M1 and M2 must specify the range of data items to be ranked, and the range of ranks to be assigned. Specifically, M01DZF ranks the data items identified by integers in the range M1 to M2, and assigns ranks in the range M1 to M2 which are stored in elements M1 to M2 of IRANK.

*Constraint:*  $0 < M1 \leq M2$ .

## 4: IRANK(M2) – INTEGER array.

*Output*

*On exit:* elements M1 to M2 of IRANK contain the ranks of the data items M1 to M2. Note that the ranks are in the range M1 to M2: thus, if item  $i$  is first in the rank ordering, IRANK( $i$ ) contains M1.

## 5: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M2 < 1,  
or M1 < 1,  
or M1 > M2.

## 7. Accuracy

Not applicable.

## 8. Further Comments

The average time taken by the routine is approximately proportional to  $n \times \log n$ , where  $n = M2 - M1 + 1$ ; it will usually be dominated by the time taken in COMPAR.

## 9. Example

The example program reads records, each of which contains an integer key and a *real* number. The program ranks the records first of all in ascending order of the integer key; records with equal keys are ranked in descending order of the *real* number if the key is negative, in ascending order of the *real* number if the key is positive, and in their original order if the key is zero. After calling M01DZF, the program calls M01ZAF to convert the ranks to indices, and prints the records in rank order. Note the use of COMMON to communicate the data between the main program and the function COMPAR.

## 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      M01DZF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NMAX
PARAMETER       (NMAX=100)
INTEGER          NIN, NOUT
PARAMETER       (NIN=5, NOUT=6)
*      .. Arrays in Common ..
real           RV(NMAX)
INTEGER          IV(NMAX)
*      .. Local Scalars ..
INTEGER          I, IFAIL, N
*      .. Local Arrays ..
INTEGER          IRANK(NMAX)
*      .. External Functions ..
LOGICAL         COMPAR
EXTERNAL        COMPAR
```

```

*      .. External Subroutines ..
EXTERNAL          M01DZF, M01ZAF
*      .. Common blocks ..
COMMON           RV, IV
*      .. Executable Statements ..
WRITE (NOUT,*) 'M01DZF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
READ (NIN,*) N
IF (N.GE.1 .AND. N.LE.NMAX) THEN
  READ (NIN,*) (IV(I),RV(I),I=1,N)
  IFAIL = 0
*
  CALL M01DZF(COMP,1,N,IRANK,IFAIL)
  CALL M01ZAF(IRANK,1,N,IFAIL)
*
  WRITE (NOUT,*)
  WRITE (NOUT,*) '  Data in sorted order'
  WRITE (NOUT,*)
  DO 20 I = 1, N
    WRITE (NOUT,99999) IV(IRANK(I)), RV(IRANK(I))
  20  CONTINUE
  END IF
  STOP
*
99999 FORMAT (1X,I7,F7.1)
END
*
LOGICAL FUNCTION COMP(I,J)
*      .. Parameters ..
INTEGER          NMAX
PARAMETER       (NMAX=100)
*      .. Scalar Arguments ..
INTEGER          I, J
*      .. Arrays in Common ..
real            RV(NMAX)
INTEGER          IV(NMAX)
*      .. Common blocks ..
COMMON           RV, IV
*      .. Executable Statements ..
IF (IV(I).NE.IV(J)) THEN
  COMP = IV(I) .GT. IV(J)
ELSE
  IF (IV(I).LT.0) THEN
    COMP = RV(I) .LT. RV(J)
  ELSE IF (IV(I).GT.0) THEN
    COMP = RV(I) .GT. RV(J)
  ELSE
    COMP = I .LT. J
  END IF
END IF
RETURN
END

```

## 9.2. Program Data

```

M01DZF Example Program Data
12
 2  3.0
 1  4.0
-1  6.0
 0  5.0
 2  2.0
-2  7.0
 0  4.0
 1  3.0
 1  5.0
-1  2.0
 1  0.0
 2  1.0

```

**9.3. Program Results**

M01DZF Example Program Results

Data in sorted order

-2	7.0
-1	6.0
-1	2.0
0	4.0
0	5.0
1	0.0
1	3.0
1	4.0
1	5.0
2	1.0
2	2.0
2	3.0

---

## M01EAF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01EAF rearranges a vector of *real* numbers into the order specified by a vector of ranks.

### 2. Specification

```
SUBROUTINE M01EAF (RV, M1, M2, IRANK, IFAIL)
  INTEGER          M1, M2, IRANK(M2), IFAIL
  real            RV(M2)
```

### 3. Description

M01EAF is designed to be used typically in conjunction with the M01D- ranking routines. After one of the M01D- routines has been called to determine a vector of ranks, M01EAF can be called to rearrange a vector of *real* numbers into the rank order. If the vector of ranks has been generated in some other way, then M01ZBF should be called to check its validity before M01EAF is called.

### 4. References

None.

### 5. Parameters

1: RV(M2) – *real* array. *Input/Output*

*On entry:* elements M1 to M2 of RV must contain *real* values to be rearranged.

*On exit:* these values are rearranged into rank order. For example, if  $IRANK(i) = M1$ , then the initial value of  $RV(i)$  is moved to  $RV(M1)$ .

2: M1 – INTEGER. *Input*

3: M2 – INTEGER. *Input*

*On entry:* M1 and M2 must specify the range of the ranks supplied in IRANK and the elements of RV to be rearranged.

*Constraint:*  $0 < M1 \leq M2$ .

4: IRANK(M2) – INTEGER array. *Input*

*On entry:* elements M1 to M2 of IRANK must contain a permutation of the integers M1 to M2, which are interpreted as a vector of ranks.

5: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

### 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry,  $M2 < 1$ ,  
or  $M1 < 1$ ,  
or  $M1 > M2$ .

IFAIL = 2

Elements  $M1$  to  $M2$  of IRANK contain a value outside the range  $M1$  to  $M2$ .

IFAIL = 3

Elements  $M1$  to  $M2$  of IRANK contain a repeated value.

If IFAIL = 2 or 3, elements  $M1$  to  $M2$  of IRANK do not contain a permutation of the integers  $M1$  to  $M2$ . On exit, the contents of RV may be corrupted. To check the validity of IRANK without the risk of corrupting RV, use M01ZBF.

## 7. Accuracy

Not applicable.

## 8. Further Comments

The average time taken by the routine is approximately proportional to  $n$ , where  $n = M2 - M1 + 1$ .

## 9. Example

The example program reads a matrix of *real* numbers and rearranges its rows so that the elements of the  $k$ th column are in ascending order. To do this, the program first calls M01DAF to rank the elements of the  $k$ th column, and then calls M01EAF to rearrange each column into the order specified by the ranks. The value of  $k$  is read from the data-file.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      M01EAF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          MMAX, NMAX
PARAMETER       (MMAX=20, NMAX=20)
INTEGER          NIN, NOUT
PARAMETER       (NIN=5, NOUT=6)
*      .. Local Scalars ..
INTEGER          I, IFAIL, J, K, M, N
*      .. Local Arrays ..
real           RM(MMAX, NMAX)
INTEGER          IRANK(MMAX)
*      .. External Subroutines ..
EXTERNAL        M01DAF, M01EAF
*      .. Executable Statements ..
WRITE (NOUT,*) 'M01EAF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
READ (NIN,*) M, N, K
IF (M.GE.1 .AND. M.LE.MMAX .AND. N.GE.1 .AND. N.LE.NMAX .AND.
+   K.GE.1 .AND. K.LE.N) THEN
DO 20 I = 1, M
    READ (NIN,*) (RM(I, J), J=1, N)
20  CONTINUE
    IFAIL = 0
*
```

```

      CALL M01DAF(RM(1,K),1,M,'Ascending',IRANK,IFAIL)
*
      DO 40 J = 1, N
*
          CALL M01EAF(RM(1,J),1,M,IRANK,IFAIL)
*
40      CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Matrix sorted on column', K
      WRITE (NOUT,*)
      DO 60 I = 1, M
          WRITE (NOUT,99998) (RM(I,J),J=1,N)
60      CONTINUE
      END IF
      STOP
*
99999 FORMAT (1X,A,I3)
99998 FORMAT (1X,3F7.1)
      END

```

## 9.2. Program Data

M01EAF Example Program Data

```

12 3 1
6.0 5.0 4.0
5.0 2.0 1.0
2.0 4.0 9.0
4.0 9.0 6.0
4.0 9.0 5.0
4.0 1.0 2.0
3.0 4.0 1.0
2.0 4.0 6.0
1.0 6.0 4.0
9.0 3.0 2.0
6.0 2.0 5.0
4.0 9.0 6.0

```

## 9.3. Program Results

M01EAF Example Program Results

Matrix sorted on column 1

```

1.0    6.0    4.0
2.0    4.0    9.0
2.0    4.0    6.0
3.0    4.0    1.0
4.0    9.0    6.0
4.0    9.0    5.0
4.0    1.0    2.0
4.0    9.0    6.0
5.0    2.0    1.0
6.0    5.0    4.0
6.0    2.0    5.0
9.0    3.0    2.0

```

---





## M01EBF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01EBF rearranges a vector of integer numbers into the order specified by a vector of ranks.

### 2. Specification

```
SUBROUTINE M01EBF (IV, M1, M2, IRANK, IFAIL)
  INTEGER          IV(M2), M1, M2, IRANK(M2), IFAIL
```

### 3. Description

M01EBF is designed to be used typically in conjunction with the M01D- ranking routines. After one of the M01D- routines has been called to determine a vector of ranks, M01EBF can be called to rearrange a vector of integer numbers into the rank order. If the vector of ranks has been generated in some other way, then M01ZBF should be called to check its validity before M01EBF is called.

### 4. References

None.

### 5. Parameters

1: IV(M2) – INTEGER array. *Input/Output*

*On entry:* elements M1 to M2 of IV must contain integer values to be rearranged.

*On exit:* these values are rearranged into rank order. For example, if  $IRANK(i) = M1$ , then the initial value of  $IV(i)$  is moved to  $IV(M1)$ .

2: M1 – INTEGER. *Input*

3: M2 – INTEGER. *Input*

*On entry:* M1 and M2 specify the range of the ranks supplied in IRANK and the elements of IV to be rearranged.

*Constraint:*  $0 < M1 \leq M2$ .

4: IRANK(M2) – INTEGER array. *Input*

*On entry:* elements M1 to M2 of IRANK must contain a permutation of the integers M1 to M2, which are interpreted as a vector of ranks.

5: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

### 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M2 < 1,  
or M1 < 1,  
or M1 > M2.

IFAIL = 2

Elements M1 to M2 of IRANK contain a value outside the range M1 to M2.

IFAIL = 3

Elements M1 to M2 of IRANK contain a repeated value.

If IFAIL = 2 or 3, elements M1 to M2 of IRANK do not contain a permutation of the integers M1 to M2. On exit, the contents of IV may be corrupted. To check the validity of IRANK without the risk of corrupting IV, use M01ZBF.

## 7. Accuracy

Not applicable.

## 8. Further Comments

The average time taken by the routine is approximately proportional to  $n$ , where  $n = M2 - M1 + 1$ .

## 9. Example

The example program reads a matrix of integers and rearranges its rows so that the elements of the  $k$ th column are in ascending order. To do this, the program first calls M01DBF to rank the elements of the  $k$ th column, and then calls M01EBF to rearrange each column into the order specified by the ranks. The value of  $k$  is read from the data-file.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      M01EBF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          MMAX, NMAX
      PARAMETER        (MMAX=20,NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*      .. Local Scalars ..
      INTEGER          I, IFAIL, J, K, M, N
*      .. Local Arrays ..
      INTEGER          IM(MMAX,NMAX), IRANK(MMAX)
*      .. External Subroutines ..
      EXTERNAL         M01DBF, M01EBF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'M01EBF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) M, N, K
      IF (M.GE.1 .AND. M.LE.MMAX .AND. N.GE.1 .AND. N.LE.NMAX .AND.
+      K.GE.1 .AND. K.LE.N) THEN
      DO 20 I = 1, M
          READ (NIN,*) (IM(I,J),J=1,N)
20      CONTINUE
      IFAIL = 0
*
      CALL M01DBF(IM(1,K),1,M,'Ascending',IRANK,IFAIL)
*
      DO 40 J = 1, N
*

```

```

                CALL M01EBF(IM(1,J),1,M,IRANK,IFAIL)
*
40    CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Matrix sorted on column', K
      WRITE (NOUT,*)
      DO 60 I = 1, M
        WRITE (NOUT,99998) (IM(I,J),J=1,N)
60    CONTINUE
      END IF
      STOP
*
99999 FORMAT (1X,A,I3)
99998 FORMAT (1X,3I7)
      END

```

## 9.2. Program Data

```

M01EBF Example Program Data
12 3 1
6 5 4
5 2 1
2 4 9
4 9 6
4 9 5
4 1 2
3 4 1
2 4 6
1 6 4
9 3 2
6 2 5
4 9 6

```

## 9.3. Program Results

M01EBF Example Program Results

Matrix sorted on column 1

1	6	4
2	4	9
2	4	6
3	4	1
4	9	6
4	9	5
4	1	2
4	9	6
5	2	1
6	5	4
6	2	5
9	3	2

---



## M01ECF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01ECF rearranges a vector of character data into the order specified by a vector of ranks.

### 2. Specification

```
SUBROUTINE M01ECF (CH, M1, M2, IRANK, IFAIL)
  INTEGER          M1, M2, IRANK(M2), IFAIL
  CHARACTER*(*)   CH(M2)
```

### 3. Description

M01ECF is designed to be used typically in conjunction with the M01D- ranking routines. After one of the M01D- routines has been called to determine a vector of ranks, M01ECF can be called to rearrange a vector of character data into the rank order. If the vector of ranks has been generated in some other way, then M01ZBF should be called to check its validity before M01ECF is called.

### 4. References

None.

### 5. Parameters

1: CH(M2) – CHARACTER\*(\*) array. *Input/Output*

*On entry:* elements M1 to M2 of CH must contain character data to be rearranged.

*Constraint:* the length of each element of CH must not exceed 255.

*On exit:* these values are rearranged into rank order. For example, if  $IRANK(i) = M1$ , then the initial value of  $CH(i)$  is moved to  $CH(M1)$ .

2: M1 – INTEGER. *Input*

3: M2 – INTEGER. *Input*

*On entry:* the range of the ranks supplied in IRANK and the elements of CH to be rearranged.

*Constraint:*  $0 < M1 \leq M2$ .

4: IRANK(M2) – INTEGER array. *Input*

*On entry:* elements M1 to M2 of IRANK must contain a permutation of the integers M1 to M2, which are interpreted as a vector of ranks.

5: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

### 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M2 < 1,  
or M1 < 1,  
or M1 > M2.

IFAIL = 2

On entry, the length of each element of CH exceeds 255.

IFAIL = 3

Elements M1 to M2 of IRANK contain a value outside the range M1 to M2.

IFAIL = 4

Elements M1 to M2 of IRANK contain a repeated value.

If IFAIL = 3 or 4, elements M1 to M2 of IRANK do not contain a permutation of the integers M1 to M2. On exit, the contents of CH may be corrupted. To check the validity of IRANK without the risk of corrupting CH, use M01ZBF.

## 7. Accuracy

Not applicable.

## 8. Further Comments

The average time taken by the routine is approximately proportional to  $n$ , where  $n = M2 - M1 + 1$ .

## 9. Example

The example program reads a file of 12-character records, each of which contains in characters 1 to 6 a name of a NAG routine, and in characters 7 to 12 an integer frequency. The program first calls M01DBF to rank the integers in descending order, and then calls M01ECF to rearrange the names into the order specified by the ranks.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      M01ECF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
          INTEGER          NIN, NOUT
          PARAMETER       (NIN=5,NOUT=6)
          INTEGER          MMAX
          PARAMETER       (MMAX=100)
*      .. Local Scalars ..
          INTEGER          I, IFAIL, M
*      .. Local Arrays ..
          INTEGER          IFREQ(MMAX), IRANK(MMAX)
          CHARACTER*6      CH(MMAX)
*      .. External Subroutines ..
          EXTERNAL        M01DBF, M01ECF
*      .. Executable Statements ..
          WRITE (NOUT,*) 'M01ECF Example Program Results'
*      Skip heading in data file
          READ (NIN,*)
          DO 20 M = 1, MMAX
              READ (NIN,99999,END=40) CH(M), IFREQ(M)
20      CONTINUE
          40 M = M - 1
              IFAIL = 0
*
```

```
      CALL M01DBF(IFREQ,1,M,'Descending',IRANK,IFAIL)
      CALL M01ECF(CH,1,M,IRANK,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Names in order of frequency'
      WRITE (NOUT,*)
      WRITE (NOUT,99998) (CH(I),I=1,M)
      STOP
*
99999 FORMAT (A6,I6)
99998 FORMAT (1X,A)
      END
```

## 9.2. Program Data

```
M01ECF Example Program Data
A02AAF  289
A02ABF  523
A02ACF  531
C02ADF  169
C02AEF  599
C05ADF 1351
C05AGF  240
C05AJF  136
C05AVF  211
C05AXF  183
C05AZF 2181
```

## 9.3. Program Results

M01ECF Example Program Results

Names in order of frequency

```
C05AZF
C05ADF
C02AEF
A02ACF
A02ABF
A02AAF
C05AGF
C05AVF
C05AXF
C02ADF
C05AJF
```

---





## M01ZAF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01ZAF inverts a permutation, and hence converts a rank vector to an index vector, or vice versa.

### 2. Specification

```
SUBROUTINE M01ZAF (IPERM, M1, M2, IFAIL)
  INTEGER          IPERM(M2), M1, M2, IFAIL
```

### 3. Description

There are two common ways of describing a permutation using an integer vector IPERM. The first uses ranks: IPERM(*i*) holds the position to which the *i*th data element should be moved in order to sort the data; in other words its rank in the sorted order. The second uses indices: IPERM(*i*) holds the current position of the data element which would occur in *i*th position in sorted order. For example, given the values

3.5 5.9 2.9 0.5

to be sorted in ascending order, the ranks would be

3 4 2 1

and the indices would be

4 3 1 2

The M01D- routines generate ranks, and the M01E- routines require ranks to be supplied to specify the re-ordering. However if it is desired simply to refer to the data in sorted order without actually re-ordering them, indices are more convenient than ranks (see the example in Section 9).

M01ZAF can be used to convert ranks to indices, or indices to ranks, as the two permutations are inverses of one another.

### 4. References

None.

### 5. Parameters

- 1: IPERM(M2) – INTEGER array. *Input/Output*  
*On entry:* elements M1 to M2 of IPERM must contain a permutation of the integers M1 to M2.  
*On exit:* these elements contain the inverse permutation of the integers M1 to M2.
- 2: M1 – INTEGER. *Input*
- 3: M2 – INTEGER. *Input*  
*On entry:* M1 and M2 must specify the range of elements used in the array IPERM and the range of values in the permutation, as specified under IPERM.  
*Constraint:*  $0 < M1 \leq M2$ .
- 4: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry `IFAIL = 0` or `-1`, explanatory error messages are output on the current error message unit (as defined by `X04AAF`).

`IFAIL = 1`

On entry, `M2 < 1`,  
or `M1 < 1`,  
or `M1 > M2`.

`IFAIL = 2`

Elements `M1` to `M2` of `IPERM` contain a value outside the range `M1` to `M2`.

`IFAIL = 3`

Elements `M1` to `M2` of `IPERM` contain a repeated value.

If `IFAIL = 2` or `3`, elements `M1` to `M2` of `IPERM` do not contain a permutation of the integers `M1` to `M2`; on exit these elements are usually corrupted. To check the validity of a permutation without the risk of corrupting it, use `M01ZBF`.

## 7. Accuracy

Not applicable.

## 8. Further Comments

None.

## 9. Example

The example program reads a matrix of *real* numbers and prints its rows in ascending order as ranked by `M01DEF`. The program first calls `M01DEF` to rank the rows, and then calls `M01ZAF` to convert the rank vector to an index vector, which is used to refer to the rows in sorted order.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      M01ZAF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          MMAX, NMAX
PARAMETER       (MMAX=20,NMAX=20)
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
INTEGER          I, IFAIL, J, M, N
*      .. Local Arrays ..
real           RM(MMAX,NMAX)
INTEGER          IPERM(MMAX)
*      .. External Subroutines ..
EXTERNAL        M01DEF, M01ZAF
*      .. Executable Statements ..
WRITE (NOUT,*) 'M01ZAF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
READ (NIN,*) M, N
IF (M.GE.1 .AND. M.LE.MMAX .AND. N.GE.1 .AND. N.LE.NMAX) THEN
DO 20 I = 1, M
    READ (NIN,*) (RM(I,J),J=1,N)
20  CONTINUE
    IFAIL = 0
*
```

```

CALL M01DEF(RM,MMAX,1,M,1,N,'Ascending',IPERM,IFAIL)
CALL M01ZAF(IPERM,1,M,IFAIL)
*
WRITE (NOUT,*)
WRITE (NOUT,*) 'Matrix sorted by rows'
WRITE (NOUT,*)
DO 40 I = 1, M
    WRITE (NOUT,99999) (RM(IPERM(I),J),J=1,N)
40 CONTINUE
END IF
STOP
*
99999 FORMAT (1X,3F7.1)
END

```

## 9.2. Program Data

M01ZAF Example Program Data

```

12 3
6.0 5.0 4.0
5.0 2.0 1.0
2.0 4.0 9.0
4.0 9.0 6.0
4.0 9.0 5.0
4.0 1.0 2.0
3.0 4.0 1.0
2.0 4.0 6.0
1.0 6.0 4.0
9.0 3.0 2.0
6.0 2.0 5.0
4.0 9.0 6.0

```

## 9.3. Program Results

M01ZAF Example Program Results

Matrix sorted by rows

```

1.0    6.0    4.0
2.0    4.0    6.0
2.0    4.0    9.0
3.0    4.0    1.0
4.0    1.0    2.0
4.0    9.0    5.0
4.0    9.0    6.0
4.0    9.0    6.0
5.0    2.0    1.0
6.0    2.0    5.0
6.0    5.0    4.0
9.0    3.0    2.0

```

---



## M01ZBF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01ZBF checks the validity of a permutation.

### 2. Specification

```
SUBROUTINE M01ZBF (IPERM, M1, M2, IFAIL)
  INTEGER          IPERM(M2), M1, M2, IFAIL
```

### 3. Description

M01ZBF can be used to check the validity of user-supplied ranks or indices, without the ranks or indices being corrupted.

### 4. References

None.

### 5. Parameters

1: IPERM(M2) – INTEGER array. *Input*

*On entry:* elements M1 to M2 of IPERM must be set to values which are supposed to be a permutation of the integers M1 to M2. If they are a valid permutation, the routine exits with IFAIL = 0.

2: M1 – INTEGER. *Input*

3: M2 – INTEGER. *Input*

*On entry:* the range of elements used in the array IPERM and the range of values in the permutation, as specified under IPERM.

*Constraint:*  $0 < M1 \leq M2$ .

4: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

### 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M2 < 1,  
or M1 < 1,  
or M1 > M2.

IFAIL = 2

Elements M1 to M2 of IPERM contain a value outside the range M1 to M2.

IFAIL = 3

Elements M1 to M2 of IPERM contain a repeated value.

If IFAIL = 2 or 3, elements M1 to M2 of IPERM do not contain a permutation of the integers M1 to M2.

## 7. Accuracy

Not applicable.

## 8. Further Comments

None.

## 9. Example

The example program reads in a vector of *real* numbers, and a vector of ranks; it calls M01ZBF to check the validity of the ranks before calling M01EAF to rearrange the *real* numbers into the specified order.

## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      M01ZBF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NMAX
      PARAMETER       (NMAX=100)
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5, NOUT=6)
*      .. Local Scalars ..
      INTEGER          I, IFAIL, N
*      .. Local Arrays ..
      real            RV(NMAX)
      INTEGER          IRANK(NMAX)
*      .. External Subroutines ..
      EXTERNAL         M01EAF, M01ZBF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'M01ZBF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) N
      IF (N.GE.1 .AND. N.LE.NMAX) THEN
          READ (NIN,*) (RV(I),I=1,N)
          READ (NIN,*) (IRANK(I),I=1,N)
          IFAIL = 0
*
          CALL M01ZBF(IRANK,1,N,IFAIL)
          CALL M01EAF(RV,1,N,IRANK,IFAIL)
*
          WRITE (NOUT,*)
          WRITE (NOUT,*) 'Numbers in rank order'
          WRITE (NOUT,*)
          WRITE (NOUT,99999) (RV(I),I=1,N)
      END IF
      STOP
*
      99999 FORMAT (1X,10F7.1)
      END
```

## 9.2. Program Data

M01ZBF Example Program Data

```
12
5.3 4.6 7.8 1.7 5.3 9.9 3.2 4.3 7.8 4.5 1.2 7.6
 7  6 10  2  8 12  3  4 11  5  1  9
```

### 9.3. Program Results

M01ZBF Example Program Results

Numbers in rank order

1.2	1.7	3.2	4.3	4.5	4.6	5.3	5.3	7.6	7.8
7.8	9.9								

---





## M01ZCF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

M01ZCF decomposes a permutation into cycles, as an aid to re-ordering ranked data.

### 2. Specification

```
SUBROUTINE M01ZCF (IPERM, M1, M2, ICYCL, IFAIL)
  INTEGER          IPERM(M2), M1, M2, ICYCL(M2), IFAIL
```

### 3. Description

M01ZCF is provided as an aid to re-ordering arbitrary data structures without using additional storage. However users should consider carefully whether it is necessary to rearrange their data, or whether it would be simpler and more efficient to refer to the data in sorted order using an index vector, or to create a copy of the data in sorted order.

To rearrange data into a different order without using additional storage, the simplest method is to decompose the permutation which specifies the new order, into cycles; and then to do a cyclic permutation of the data items in each cycle. (This is the method used by the M01E- re-ordering routines.) Given a vector IRANK which specifies the ranks of the data (as generated by the M01D- routines), M01ZCF generates a new vector ICYCL, in which the permutation is represented in its component cycles, with the first element of each cycle negated. For example, the permutation

5 7 4 2 1 6 3

is composed of the cycles

(1 5) (2 7 3 4) (6)

and the vector ICYCL generated by M01ZCF contains

-1 5 -2 7 3 4 -6

In order to rearrange the data according to the specified ranks:

item 6 must be left in place;

items 1 and 5 must be interchanged;

items 4, 2, 7 and 3 must be moved one place round the cycle.

The complete rearrangement can be achieved by the following code:

```
DO 10 K = M1, M2
  I = ICYCL(K)
  IF (I.LT.0) THEN
    J = -I
  ELSE
    [swap items I and J]
  ENDIF
10 CONTINUE
```

### 4. References

None.

### 5. Parameters

1: IPERM(M2) – INTEGER array.

*Input*

*On entry:* elements M1 to M2 of IPERM must contain a permutation of the integers M1 to M2.

- 2: M1 – INTEGER. *Input*  
 3: M2 – INTEGER. *Input*  
*On entry:* M1 and M2 must specify the range of elements used in the array IPERM and the range of values in the permutation, as specified under IPERM.  
*Constraint:*  $0 < M1 \leq M2$ .
- 4: ICYCL(M2) – INTEGER array. *Output*  
*On exit:* elements M1 to M2 of ICYCL contain a representation of the permutation as a list of cycles, with the first integer in each cycle negated. (See Section 3.)
- 5: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M2 < 1,  
 or M1 < 1,  
 or M1 > M2.

IFAIL = 2

Elements M1 to M2 of IPERM contain a value outside the range M1 to M2.

IFAIL = 3

Elements M1 to M2 of IPERM contain a repeated value.

If IFAIL = 2 or 3, elements M1 to M2 of IPERM do not contain a permutation of the integers M1 to M2.

## 7. Accuracy

Not applicable.

## 8. Further Comments

None.

## 9. Example

The example program reads a matrix of *real* numbers and rearranges its columns so that the elements of the *l*th row are in ascending order. To do this, the program first calls M01DJF to rank the elements of the *l*th row, and then calls M01ZCF to decompose the rank vector into cycles. It then rearranges the columns using the framework of code suggested in Section 3. The value of *l* is read from the data-file.

## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      M01ZCF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          MMAX, NMAX
      PARAMETER       (MMAX=20,NMAX=20)
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            T
      INTEGER          I, IFAIL, II, J, K, L, M, N
*      .. Local Arrays ..
      real            RM(MMAX,NMAX)
      INTEGER          ICYCL(NMAX), IRANK(NMAX)
*      .. External Subroutines ..
      EXTERNAL        M01DJF, M01ZCF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'M01ZCF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) M, N, L
      IF (M.GE.1 .AND. M.LE.MMAX .AND. N.GE.1 .AND. N.LE.NMAX .AND.
+       L.GE.1 .AND. L.LE.M) THEN
          DO 20 I = 1, M
              READ (NIN,*) (RM(I,J),J=1,N)
          20  CONTINUE
              IFAIL = 0
*
              CALL M01DJF(RM,MMAX,L,L,1,N,'Ascending',IRANK,IFAIL)
              CALL M01ZCF(IRANK,1,N,ICYCL,IFAIL)
*
              DO 60 K = 1, N
                  I = ICYCL(K)
                  IF (I.LT.0) THEN
                      J = -I
                  ELSE
*                      Swap columns I and J
                      DO 40 II = 1, M
                          T = RM(II,J)
                          RM(II,J) = RM(II,I)
                          RM(II,I) = T
                      40  CONTINUE
                  END IF
          60  CONTINUE
              WRITE (NOUT,*)
              WRITE (NOUT,99999) 'Matrix sorted on row', L
              WRITE (NOUT,*)
              DO 80 I = 1, M
                  WRITE (NOUT,99998) (RM(I,J),J=1,N)
          80  CONTINUE
              END IF
              STOP
*
          99999 FORMAT (1X,A,I3)
          99998 FORMAT (1X,12F6.1)
              END

```

## 9.2. Program Data

```

M01ZCF Example Program Data
3 12 3
5.0 4.0 3.0 2.0 2.0 1.0 9.0 4.0 4.0 2.0 2.0 1.0
3.0 8.0 2.0 5.0 5.0 6.0 9.0 8.0 9.0 5.0 4.0 1.0
9.0 1.0 6.0 1.0 2.0 4.0 8.0 1.0 2.0 2.0 6.0 2.0

```

**9.3. Program Results**

M01ZCF Example Program Results

Matrix sorted on row 3

4.0	2.0	4.0	2.0	4.0	2.0	1.0	1.0	3.0	2.0	9.0	5.0
8.0	5.0	8.0	5.0	9.0	5.0	1.0	6.0	2.0	4.0	9.0	3.0
1.0	1.0	1.0	2.0	2.0	2.0	2.0	4.0	6.0	6.0	8.0	9.0

---

## Chapter P01 – Error Trapping

**Note.** Please refer to the Users' Note for your implementation to check that a routine is available.

<b>Routine Name</b>	<b>Mark of Introduction</b>	<b>Purpose</b>
P01ABF	12	Return value of error indicator/terminate with error message

---



# Chapter P01

## Error Trapping

### Contents

<b>1</b>	<b>Scope of the Chapter</b>	<b>2</b>
<b>2</b>	<b>Background to the Problems</b>	<b>2</b>
2.1	Errors, Failure and Warning Conditions . . . . .	2
2.2	The IFAIL Parameter . . . . .	2
2.3	Hard Fail Option . . . . .	2
2.4	Soft Fail Option . . . . .	3
2.5	Historical Note . . . . .	3
<b>3</b>	<b>Recommendations on Choice and Use of Available Routines</b>	<b>4</b>

## 1 Scope of the Chapter

This chapter is concerned with the trapping of error, failure or warning conditions by NAG Library routines. This introduction document describes the commonly occurring parameter IFAIL.

## 2 Background to the Problems

### 2.1 Errors, Failure and Warning Conditions

The error, failure or warning conditions considered here are those that can be detected by explicit coding in a Library routine. Such conditions must be anticipated by the author of the routine. They should not be confused with run-time errors detected by the compiling system, e.g. detection of overflow or failure to assign an initial value to a variable.

In the rest of this document we use the word ‘error’ to cover all types of error, failure or warning conditions detected by the routine. They fall roughly into three classes.

- (i) On entry to the routine the value of a parameter is out of range. This means that it is not useful, or perhaps even meaningful, to begin computation.
- (ii) During computation the routine decides that it cannot yield the desired results, and indicates a failure condition. For example, a matrix inversion routine will indicate a failure condition if it considers that the matrix is singular and so cannot be inverted.
- (iii) Although the routine completes the computation and returns results, it cannot guarantee that the results are completely reliable; it therefore returns a warning. For example, an optimization routine may return a warning if it cannot guarantee that it has found a local minimum.

**All three classes of errors are handled in the same way by the Library.**

Each error which can be detected by a Library routine is associated with a number. These numbers, with explanations of the errors, are listed in Section 6 (Error Indicators and Warnings) in the routine document. Unless the document specifically states to the contrary, the user should not assume that the routine necessarily tests for the occurrence of the errors in their order of error number, i.e., the detection of an error does not imply that other errors have or have not been detected.

### 2.2 The IFAIL Parameter

Most of the NAG Library routines which can be called directly by the user have a parameter called IFAIL. This parameter is concerned with the NAG Library error trapping mechanism (and, for some routines, with controlling the output of error messages and advisory messages).

IFAIL has **two** purposes:

- (i) to allow the user to specify what action the Library routine should take if an error is detected;
- (ii) to inform the user of the outcome of the call of the routine.

For purpose (i), the user **must** assign a value to IFAIL before the call to the Library routine. Since IFAIL is reset by the routine for purpose (ii), the parameter must be the name of a variable, **not** a literal or constant.

The value assigned to IFAIL before entry should be either 0 (**hard fail** option), or 1 or – 1 (**soft fail** option). If after completing its computation the routine has not detected an error, IFAIL is reset to 0 to indicate a **successful call**. Control returns to the calling program in the normal way. If the routine does detect an error, its action depends on whether the hard or soft fail option was chosen.

### 2.3 Hard Fail Option

If the user sets IFAIL to 0 before calling the Library routine, execution of the program will terminate if the routine detects an error. Before the program is stopped, this error message is output:

```
** ABNORMAL EXIT from NAG Library routine XXXXXX: IFAIL = n  
  
** NAG hard failure - execution terminated
```



where **XXXXXX** is the routine name, and **n** is the number associated with the detected error. An explanation of error number **n** is given in Section 6 of the routine document **XXXXXX**.

In addition, most routines output explanatory error messages immediately before the standard termination message shown above.

In some implementations of the NAG Library, when the hard fail option is invoked, the error message may be accompanied by dump or tracing information. The output channel used for the output of the error message is determined by X04AAF.

The hard fail option should be selected if the user is in any doubt about continuing the execution of the program after an unsuccessful call to a NAG Library routine.

## 2.4 Soft Fail Option

To select this option, the user must set IFAIL to 1 or -1 before calling the Library routine.

If the routine detects an error, IFAIL is reset to the associated error number; further computation within the routine is suspended and control returns to the calling program.

If the user sets IFAIL to 1, then no error message is output (**silent exit**).

If the user sets IFAIL to -1 (**noisy exit**), then before control is returned to the calling program, the following error message is output:

```
** ABNORMAL EXIT from NAG Library routine XXXXXX: IFAIL = n

** NAG soft failure - control returned
```

In addition, most routines output explanatory error messages immediately before the above standard message.

**It is most important to test the value of IFAIL on exit if the soft fail option is selected.** A non-zero exit value of IFAIL implies that the call was not successful so it is imperative that the user's program be coded to take appropriate action. That action may simply be to print IFAIL with an explanatory caption and then terminate the program. Many of the example programs in Section 9 of the routine documents have IFAIL-exit tests of this form. In the more ambitious case, where the user wishes his or her program to continue, it is essential that the program can branch to a point at which it is **sensible** to resume computation.

The soft fail option puts the onus on the user to handle any errors detected by the Library routine. With the proviso that the user is able to implement it **properly**, it is clearly more flexible than the hard fail option since it allows computation to continue in the case of errors. In particular there are at least two cases where its flexibility is useful:

- (i) where additional information about the error or the progress of computation is returned via some of the other parameters;
- (ii) exceptionally, certain routine documents may advise further calls with IFAIL left with its value on exit after the first call of the routine. In such cases the user should not reset the IFAIL-exit value between calls;
- (iii) in some routines, 'partial' success can be achieved, e.g. a probable solution found but not all conditions fully satisfied, so the routine returns a warning. On the basis of the advice in Section 6 and elsewhere in the routine document, the user may decide that this partially successful call is adequate for certain purposes.

## 2.5 Historical Note

The error handling mechanism described above was introduced into the NAG Library at Mark 12. It supersedes the earlier mechanism which for most routines allowed IFAIL to be set by the user to 0 or 1 only. The new mechanism is compatible with the old except that the details of the messages output on hard failure have changed. The new mechanism also allows the user to set IFAIL to -1 (soft failure, noisy exit).

A few routines (introduced mainly at Marks 7 and 8) use IFAIL in a different way to control the output of error messages, and also of advisory messages (see Chapter X04). In those routines IFAIL is regarded as a decimal integer whose least significant digits are denoted  $ba$  with the following significance:

$a = 0$ : hard failure       $a = 1$ : soft failure  
 $b = 0$ : silent exit         $b = 1$ : noisy exit

Details are given in the documents of the relevant routines; for those routines this alternative use of IFAIL remains valid.

### 3 Recommendations on Choice and Use of Available Routines

**Note.** Refer to the Users' Note for your implementation to check that a routine is available.

To implement the error mechanism described in Section 2, NAG Library routines call P01ABF.

This routine is therefore primarily of interest only to writers of NAG Fortran Library software. It is included in the general user manual for completeness. Users need not know how to call P01ABF directly though they may be aware of its existence.

---

## P01ABF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

P01ABF either returns the value of IERROR (soft failure), or terminates execution of the program (hard failure). Diagnostic messages may be output.

### 2. Specification

```
INTEGER FUNCTION P01ABF (IFAIL, IERROR, SRNAME, NREC, REC)
INTEGER          IFAIL, IERROR, NREC
CHARACTER*(*)   SRNAME, REC(*)
```

### 3. Description

P01ABF is intended for use by other NAG Fortran Library routines. If a routine does not terminate normally, P01ABF is called to provide uniform handling of error or warning conditions. Associated with abnormal termination are error or warning indicators, which are listed in Section 6 of the routine documents. P01ABF is called with the indicator value stored in IERROR and the name of the calling library routine in SRNAME. Messages relating to the reason for termination may optionally be supplied in the array REC. The action of P01ABF then depends on the entry values of IFAIL and IERROR.

If IERROR = 0 (successful termination), the value 0 is simply returned through the routine name. No message is output.

Non-zero values of IERROR indicate abnormal termination and the action taken depends on the value of IFAIL.

If IFAIL = -1 or 1, the value of IERROR is returned through the routine name (soft failure); if IFAIL = 0, then execution of the user's program is terminated without returning to the calling routine (hard failure).

If IFAIL = 1, then no output occurs from P01ABF (silent exit). If IFAIL = 0 or -1, then P01ABF writes messages to the unit number specified by a call to X04AAF (noisy exit). Any messages supplied by the calling routine in the array REC are output first, followed by a record of the form

```
** ABNORMAL EXIT from NAG Fortran Library routine XXXXXX: IFAIL =      n
```

where XXXXXX is the value of SRNAME and n is the value of IERROR. For soft failure this is followed by the record

```
** NAG soft failure: control returned
```

and for hard failure by

```
** NAG hard failure: execution terminated
```

See also Section 8 concerning compatibility with error-handling mechanisms in the NAG Fortran Library before Mark 12.

### 4. References

None.

### 5. Parameters

1: IFAIL – INTEGER. *Input*

*On entry:* the value of IFAIL determines the action of the routine as described in Section 3.

2: IERROR – INTEGER. *Input*

*On entry:* the value of the error or warning indicator. Unless IFAIL = 0, the value of IERROR is returned through the routine name.

3: SRNAME – CHARACTER\*(\*). *Input*

*On entry:* the name of the routine which has called P01ABF. If this is a NAG Fortran Library routine, the length of SRNAME is always 6.

4: NREC – INTEGER. *Input*

*On entry:* the number of elements (records) in the array REC.

*Constraint:* NREC ≥ 0.

5: REC(\*) – CHARACTER\*(\*) array. *Input*

**Note:** the dimension of REC must be greater than or equal to max(1,NREC).

*On entry:* an internal file. Unless IFAIL = 1, or NREC = 0, the first NREC elements of REC are written to the unit determined by X04AAF. If NREC = 0, then REC is not referenced.

## 6. Error Indicators and Warnings

None.

## 7. Accuracy

Not applicable.

## 8. Further Comments

P01ABF was introduced at Mark 12 of the NAG Fortran Library, to supersede the earlier error-handling P01AAF. P01ABF is compatible with P01AAF except that the details of the messages printed on hard failure have changed.

Compatibility includes even those routines (introduced mainly at Marks 7 and 8) which used IFAIL in a different way to control the output of messages. In those routines IFAIL is regarded as a decimal integer whose least significant two digits are denoted *ba* with the following meanings:

a = 0: hard failure	a = 1: soft failure
b = 0: silent exit	b = 1: noisy exit

However this aspect of P01ABF will not, and should not, be used in future.

## 9. Example

In the simple program below, the subroutine MYSQRT uses P01ABF in the way in which it is usually employed by a NAG Fortran Library routine. Within MYSQRT, error number 1 is associated with an attempt to find a real square root of a negative number. The first call illustrates soft failure with a silent exit, the second call soft failure with a noisy exit and the third call hard failure. Output from the main program is all in upper case characters; output from P01ABF uses lower case characters.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      P01ABF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER       (NOUT=6)
*      .. Local Scalars ..
      real            Y
      INTEGER          IFAIL
*      .. External Subroutines ..
      EXTERNAL        MYSQRT
```

```

*      .. Executable Statements ..
      WRITE (NOUT,*) 'P01ABF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*)
+ 'Soft failure, silent exit - message output from the main program'
      IFAIL = 1
      CALL MYSQRT(-1.0e0,Y,IFAIL)
      IF (IFAIL.EQ.0) THEN
        WRITE (NOUT,99999) Y
      ELSE
        WRITE (NOUT,*)
+      'Attempt to take the square root of a negative number'
      END IF
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Soft failure, noisy exit'
      IFAIL = -1
      CALL MYSQRT(-2.0e0,Y,IFAIL)
      IF (IFAIL.EQ.0) THEN
        WRITE (NOUT,99999) Y
      END IF
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Hard failure, noisy exit'
      IFAIL = 0
      CALL MYSQRT(-3.0e0,Y,IFAIL)
      WRITE (NOUT,99999) Y
      STOP
*
99999 FORMAT (1X,F10.4)
      END
*
      SUBROUTINE MYSQRT(X,Y,IFAIL)
*      Simple routine to compute square root
*      .. Parameters ..
      CHARACTER*6          SRNAME
      PARAMETER            (SRNAME='MYSQRT')
*      .. Scalar Arguments ..
      real                X, Y
      INTEGER              IFAIL
*      .. Local Arrays ..
      CHARACTER*51         REC(1)
*      .. External Functions ..
      INTEGER              P01ABF
      EXTERNAL             P01ABF
*      .. Intrinsic Functions ..
      INTRINSIC            SQRT
*      .. Executable Statements ..
      IF (X.GE.0.0e0) THEN
        Y = SQRT(X)
        IFAIL = 0
      ELSE
        WRITE (REC,99999) '** Attempt to take the square root of ', X
        IFAIL = P01ABF(IFAIL,1,SRNAME,1,REC)
      END IF
      RETURN
*
99999 FORMAT (1X,A,1P,e12.5)
      END

```

## 9.2. Program Data

None.

### 9.3. Program Results

#### P01ABF Example Program Results

Soft failure, silent exit - message output from the main program  
Attempt to take the square root of a negative number

Soft failure, noisy exit

```
** Attempt to take the square root of -2.00000E+00  
** ABNORMAL EXIT from NAG Library routine MYSQRT: IFAIL =      1  
** NAG soft failure - control returned
```

Hard failure, noisy exit

```
** Attempt to take the square root of -3.00000E+00  
** ABNORMAL EXIT from NAG Library routine MYSQRT: IFAIL =      1  
** NAG hard failure - execution terminated
```

---

## Chapter S – Approximations of Special Functions

**Note.** Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
S01BAF	14	$\ln(1 + x)$
S01EAF	14	Complex exponential, $e^z$
S07AAF	1	$\tan x$
S09AAF	1	$\arcsin x$
S09ABF	3	$\arccos x$
S10AAF	3	$\tanh x$
S10ABF	4	$\sinh x$
S10ACF	4	$\cosh x$
S11AAF	4	$\operatorname{arctanh} x$
S11ABF	4	$\operatorname{arcsinh} x$
S11ACF	4	$\operatorname{arccosh} x$
S13AAF	1	Exponential integral $E_1(x)$
S13ACF	2	Cosine integral $\operatorname{Ci}(x)$
S13ADF	5	Sine integral $\operatorname{Si}(x)$
S14AAF	1	Gamma function
S14ABF	8	Log Gamma function
S14ACF	14	$\psi(x) - \ln x$
S14ADF	14	Scaled derivatives of $\psi(x)$
S14BAF	14	Incomplete gamma functions $P(a, x)$ and $Q(a, x)$
S15ABF	3	Cumulative normal distribution function $P(x)$
S15ACF	4	Complement of cumulative normal distribution function $Q(x)$
S15ADF	4	Complement of error function $\operatorname{erfc}(x)$
S15AEF	4	Error function $\operatorname{erf}(x)$
S15AFF	7	Dawson's integral
S15DDF	14	Scaled complex complement of error function, $\exp(-z^2)\operatorname{erfc}(-iz)$
S17ACF	1	Bessel function $Y_0(x)$
S17ADF	1	Bessel function $Y_1(x)$
S17AEF	5	Bessel function $J_0(x)$
S17AFF	5	Bessel function $J_1(x)$
S17AGF	8	Airy function $\operatorname{Ai}(x)$
S17AHF	8	Airy function $\operatorname{Bi}(x)$
S17AJF	8	Airy function $\operatorname{Ai}'(x)$
S17AKF	8	Airy function $\operatorname{Bi}'(x)$
S17DCF	13	Bessel functions $Y_{\nu+a}(z)$ , real $a \geq 0$ , complex $z$ , $\nu = 0, 1, 2, \dots$
S17DEF	13	Bessel functions $J_{\nu+a}(z)$ , real $a \geq 0$ , complex $z$ , $\nu = 0, 1, 2, \dots$
S17DGF	13	Airy functions $\operatorname{Ai}(z)$ and $\operatorname{Ai}'(z)$ , complex $z$
S17DHF	13	Airy functions $\operatorname{Bi}(z)$ and $\operatorname{Bi}'(z)$ , complex $z$
S17DLF	13	Hankel functions $H_{\nu+a}^{(j)}(z)$ , $j = 1, 2$ , real $a \geq 0$ , complex $z$ , $\nu = 0, 1, 2, \dots$
S18ACF	1	Modified Bessel function $K_0(x)$
S18ADF	1	Modified Bessel function $K_1(x)$
S18AEF	5	Modified Bessel function $I_0(x)$
S18AFF	5	Modified Bessel function $I_1(x)$
S18CCF	10	Modified Bessel function $e^x K_0(x)$
S18CDF	10	Modified Bessel function $e^x K_1(x)$
S18CEF	10	Modified Bessel function $e^{- x } I_0(x)$
S18CFF	10	Modified Bessel function $e^{- x } I_1(x)$
S18DCF	13	Modified Bessel functions $K_{\nu+a}(z)$ , real $a \geq 0$ , complex $z$ , $\nu = 0, 1, 2, \dots$
S18DEF	13	Modified Bessel functions $I_{\nu+a}(z)$ , real $a \geq 0$ , complex $z$ , $\nu = 0, 1, 2, \dots$
S19AAF	11	Kelvin function $\operatorname{ber} x$

S19ABF	11	Kelvin function $\text{bei } x$
S19ACF	11	Kelvin function $\text{ker } x$
S19ADF	11	Kelvin function $\text{kei } x$
S20ACF	5	Fresnel integral $S(x)$
S20ADF	5	Fresnel integral $C(x)$
S21BAF	8	Degenerate symmetrised elliptic integral of 1st kind $R_C(x, y)$
S21BBF	8	Symmetrised elliptic integral of 1st kind $R_F(x, y, z)$
S21BCF	8	Symmetrised elliptic integral of 2nd kind $R_D(x, y, z)$
S21BDF	8	Symmetrised elliptic integral of 3rd kind $R_J(x, y, z, r)$
S21CAF	15	Jacobian elliptic functions $\text{sn}$ , $\text{cn}$ and $\text{dn}$

---



# Chapter S

## Approximations of Special Functions

### Contents

<b>1</b>	<b>Scope of the Chapter</b>	<b>2</b>
<b>2</b>	<b>Background to the Problems</b>	<b>2</b>
2.1	Functions of a Single Real Argument . . . . .	2
2.2	Approximations to Elliptic Integrals . . . . .	4
2.3	Bessel and Airy Functions of a Complex Argument . . . . .	5
<b>3</b>	<b>Recommendations on Choice and Use of Available Routines</b>	<b>5</b>
3.1	Elliptic Integrals . . . . .	5
3.2	Bessel and Airy Functions . . . . .	6
<b>4</b>	<b>Index</b>	<b>6</b>
<b>5</b>	<b>References</b>	<b>7</b>

## 1 Scope of the Chapter

This chapter is concerned with the provision of some commonly occurring physical and mathematical functions.

## 2 Background to the Problems

The majority of the routines in this chapter approximate real-valued functions of a single real argument, and the techniques involved are described in Section 2.1. In addition the chapter contains routines for elliptic integrals (see Section 2.2), Bessel and Airy functions of a complex argument (see Section 2.3), exponential of a complex argument, and complementary error function of a complex argument.

### 2.1 Functions of a Single Real Argument

Most of the routines for functions of a single real argument have been based on truncated Chebyshev expansions. This method of approximation was adopted as a compromise between the conflicting requirements of efficiency and ease of implementation on many different machine ranges. For details of the reasons behind this choice and the production and testing procedures followed in constructing this chapter see Schonfelder [7].

Basically, if the function to be approximated is  $f(x)$ , then for  $x \in [a, b]$  an approximation of the form

$$f(x) = g(x) \sum'_{r=0} C_r T_r(t)$$

is used ( $\sum'$  denotes, according to the usual convention, a summation in which the first term is halved), where  $g(x)$  is some suitable auxiliary function which extracts any singularities, asymptotes and, if possible, zeros of the function in the range in question and  $t = t(x)$  is a mapping of the general range  $[a, b]$  to the specific range  $[-1, +1]$  required by the Chebyshev polynomials,  $T_r(t)$ . For a detailed description of the properties of the Chebyshev polynomials see Clenshaw [5] and Fox and Parker [6].

The essential property of these polynomials for the purposes of function approximation is that  $T_n(t)$  oscillates between  $\pm 1$  and it takes its extreme values  $n + 1$  times in the interval  $[-1, +1]$ . Therefore, provided the coefficients  $C_r$  decrease in magnitude sufficiently rapidly the error made by truncating the Chebyshev expansion after  $n$  terms is approximately given by

$$E(t) \simeq C_n T_n(t).$$

That is, the error oscillates between  $\pm C_n$  and takes its extreme value  $n + 1$  times in the interval in question. Now this is just the condition that the approximation be a mini-max representation, one which minimizes the maximum error. By suitable choice of the interval,  $[a, b]$ , the auxiliary function,  $g(x)$ , and the mapping of the independent variable,  $t(x)$ , it is almost always possible to obtain a Chebyshev expansion with rapid convergence and hence truncations that provide near mini-max polynomial approximations to the required function. The difference between the true mini-max polynomial and the truncated Chebyshev expansion is seldom sufficiently great enough to be of significance.

The evaluation of the Chebyshev expansions follows one of two methods. The first and most efficient, and hence the most commonly used, works with the equivalent simple polynomial. The second method, which is used on the few occasions when the first method proves to be unstable, is based directly on the truncated Chebyshev series, and uses backward recursion to evaluate the sum. For the first method, a suitably truncated Chebyshev expansion (truncation is chosen so that the error is less than the *machine precision*) is converted to the equivalent simple polynomial. That is, we evaluate the set of coefficients  $b_r$  such that

$$y(t) = \sum_{r=0}^{n-1} b_r t^r = \sum'_{r=0}^{n-1} C_r T_r(t).$$

The polynomial can then be evaluated by the efficient Horner's method of nested multiplications,

$$y(t) = (b_0 + t(b_1 + t(b_2 + \dots t(b_{n-2} + tb_{n-1})))) \dots).$$

This method of evaluation results in efficient routines but for some expansions there is considerable loss of accuracy due to cancellation effects. In these cases the second method is used. It is well known that if

$$\begin{aligned} b_{n-1} &= C_{n-1} \\ b_{n-2} &= 2tb_{n-1} + C_{n-2} \\ b_j &= 2tb_{j+1} - b_{j+2} + C_j, \quad j = n-3, n-4, \dots, 0 \end{aligned}$$

then

$$\sum_{r=0}^n C_r T_r(t) = \frac{1}{2}(b_0 - b_2)$$

and this is always stable. This method is most efficiently implemented by using three variables cyclically and explicitly constructing the recursion.

That is,

$$\begin{aligned} \alpha &= C_{n-1} \\ \beta &= 2t\alpha + C_{n-2} \\ \gamma &= 2t\beta - \alpha + C_{n-3} \\ \alpha &= 2t\gamma - \beta + C_{n-4} \\ \beta &= 2t\alpha - \gamma + C_{n-5} \\ &\dots \\ &\dots \\ \text{say } \alpha &= 2t\gamma - \beta + C_2 \\ \beta &= 2t\alpha - \gamma + C_1 \\ y(t) &= t\beta - \alpha + \frac{1}{2}C_0 \end{aligned}$$

The auxiliary functions used are normally functions compounded of simple polynomial (usually linear) factors extracting zeros, and the primary compiler-provided functions, sin, cos, ln, exp, sqrt, which extract singularities and/or asymptotes or in some cases basic oscillatory behaviour, leaving a smooth well-behaved function to be approximated by the Chebyshev expansion which can therefore be rapidly convergent.

The mappings of  $[a, b]$  to  $[-1, +1]$  used range from simple linear mappings to the case when  $b$  is infinite, and considerable improvement in convergence can be obtained by use of a bilinear form of mapping. Another common form of mapping is used when the function is even; that is, it involves only even powers in its expansion. In this case an approximation over the whole interval  $[-a, a]$  can be provided using a mapping  $t = 2(x/a)^2 - 1$ . This embodies the evenness property but the expansion in  $t$  involves all powers and hence removes the necessity of working with an expansion with half its coefficients zero.

For many of the routines an analysis of the error in principle is given, namely, if  $E$  and  $\nabla$  are the absolute errors in function and argument and  $\epsilon$  and  $\delta$  are the corresponding relative errors, then

$$\begin{aligned} E &\simeq |f'(x)|\nabla \\ E &\simeq |xf'(x)|\delta \\ \epsilon &\simeq \left| \frac{xf'(x)}{f(x)} \right| \delta. \end{aligned}$$

If we ignore errors that arise in the argument of the function by propagation of data errors etc., and consider only those errors that result from the fact that a real number is being represented in the

computer in floating-point form with finite precision, then  $\delta$  is bounded and this bound is independent of the magnitude of  $x$ . For example, on an 11-digit machine

$$|\delta| \leq 10^{-11}.$$

(This of course implies that the absolute error  $\nabla = x\delta$  is also bounded but the bound is now dependent on  $x$ .) However, because of this the last two relations above are probably of more interest. If possible the relative error propagation is discussed; that is, the behaviour of the error amplification factor  $|xf'(x)/f(x)|$  is described, but in some cases, such as near zeros of the function which cannot be extracted explicitly, absolute error in the result is the quantity of significance and here the factor  $|xf'(x)|$  is described. In general, testing of the functions has shown that their error behaviour follows fairly well these theoretical error behaviours. In regions where the error amplification factors are less than or of the order of one, the errors are slightly larger than the above predictions. The errors are here limited largely by the finite precision of arithmetic in the machine, but  $\epsilon$  is normally no more than a few times greater than the bound on  $\delta$ . In regions where the amplification factors are large, of order ten or greater, the theoretical analysis gives a good measure of the accuracy obtainable.

It should be noted that the definitions and notations used for the functions in this chapter are all taken from Abramowitz and Stegun [1]. Users are strongly recommended to consult this book for details before using the routines in this chapter.

## 2.2 Approximations to Elliptic Integrals

The functions provided here are symmetrised variants of the classic elliptic integrals. These alternative definitions have been suggested by Carlson (see [2], [3] and [4]) and he also developed the basic algorithms used in this chapter.

The standard integral of the first kind is represented by

$$R_F(x, y, z) = \frac{1}{2} \int_0^\infty \frac{dt}{\sqrt{(t+x)(t+y)(t+z)}},$$

where  $x, y, z \geq 0$  and at most one may be equal to zero.

The normalisation factor,  $\frac{1}{2}$ , is chosen so as to make

$$R_F(x, x, x) = 1/\sqrt{x}.$$

If any two of the variables are equal,  $R_F$  degenerates into the second function

$$R_C(x, y) = R_F(x, y, y) = \frac{1}{2} \int_0^\infty \frac{dt}{\sqrt{t+x}(t+y)},$$

where the argument restrictions are now  $x \geq 0$  and  $y \neq 0$ .

This function is related to the logarithm or inverse hyperbolic functions if  $0 < y < x$ , and to the inverse circular functions if  $0 \leq x \leq y$ .

The integrals of the second kind are defined by

$$R_D(x, y, z) = \frac{3}{2} \int_0^\infty \frac{dt}{\sqrt{(t+x)(t+y)(t+z)^3}}$$

with  $z > 0$ ,  $x \geq 0$  and  $y \geq 0$ , but only one of  $x$  or  $y$  may be zero.

The function is a degenerate special case of the integral of the third kind

$$R_J(x, y, z, \rho) = \frac{3}{2} \int_0^\infty \frac{dt}{\sqrt{(t+x)(t+y)(t+z)(t+\rho)}}$$

with  $\rho \neq 0$  and  $x, y, z \geq 0$  with at most one equality holding. Thus  $R_D(x, y, z) = R_J(x, y, z, z)$ . The normalisation of both these functions is chosen so that

$$R_D(x, x, x) = R_J(x, x, x, x) = 1/(x\sqrt{x}).$$

The algorithms used for all these functions are based on duplication theorems. These allow a recursion system to be established which constructs a new set of arguments from the old using a combination of arithmetic and geometric means. The value of the function at the original arguments can then be simply related to the value at the new arguments. These recursive reductions are used until the arguments differ from the mean by an amount small enough for a Taylor series about the mean to give sufficient accuracy when retaining terms of order less than six. Each step of the recurrences reduces the difference from the mean by a factor of four, and as the truncation error is of order six, the truncation error goes like  $(4096)^{-n}$ , where  $n$  is the number of iterations.

The above forms can be related to the more traditional canonical forms (see Abramowitz and Stegun [1], 17.2).

If we write  $q = \cos^2 \phi$ ,  $r = 1 - m \cdot \sin^2 \phi$ ,  $s = 1 + n \cdot \sin^2 \phi$ , where  $0 < \phi \leq \frac{1}{2}\pi$ , we have:

the elliptic integral of the first kind:

$$F(\phi|m) = \int_0^{\sin \phi} (1-t^2)^{-1/2}(1-mt^2)^{-1/2} dt = \sin \phi \cdot R_F(q, r, 1);$$

the elliptic integral of the second kind:

$$\begin{aligned} E(\phi|m) &= \int_0^{\sin \phi} (1-t^2)^{-1/2}(1-mt^2)^{1/2} dt \\ &= \sin \phi \cdot R_F(q, r, 1) - \frac{1}{3}m \cdot \sin^3 \phi \cdot R_D(q, r, 1) \end{aligned}$$

the elliptic integral of the third kind:

$$\begin{aligned} \Pi(n; \phi|m) &= \int_0^{\sin \phi} (1-t^2)^{-1/2}(1-mt^2)^{-1/2}(1+nt^2)^{-1} dt \\ &= \sin \phi \cdot R_F(q, r, 1) - \frac{1}{3}n \cdot \sin^3 \phi \cdot R_J(q, r, 1, s). \end{aligned}$$

Also the complete elliptic integral of the first kind:

$$K(m) = \int_0^{\pi/2} (1-m \cdot \sin^2 \theta)^{-1/2} d\theta = R_F(0, 1-m, 1);$$

the complete elliptic integral of the second kind:

$$E(m) = \int_0^{\pi/2} (1-m \cdot \sin^2 \theta)^{1/2} d\theta = R_F(0, 1-m, 1) - \frac{1}{3}m \cdot R_D(0, 1-m, 1).$$

### 2.3 Bessel and Airy Functions of a Complex Argument

The routines for Bessel and Airy functions of a real argument are based on Chebyshev expansions, as described in Section 2.1. The routines for functions of a complex argument, however, use different methods. These routines relate all functions to the modified Bessel functions  $I_\nu(z)$  and  $K_\nu(z)$  computed in the right-half complex plane, including their analytic continuations.  $I_\nu$  and  $K_\nu$  are computed by different methods according to the values of  $z$  and  $\nu$ . The methods include power series, asymptotic expansions and Wronskian evaluations. The relations between functions are based on well known formulae (see Abramowitz and Stegun [1]).

## 3 Recommendations on Choice and Use of Available Routines

**Note.** Refer to the Users' Note for your implementation to check that a routine is available.

### 3.1 Elliptic Integrals

**IMPORTANT ADVICE:** users who encounter elliptic integrals in the course of their work are strongly recommended to look at transforming their analysis directly to one of the Carlson forms, rather than to

the traditional canonical Legendre forms. In general, the extra symmetry of the Carlson forms is likely to simplify the analysis, and these symmetric forms are much more stable to calculate.

The routine S21BAF for  $R_C$  is largely included as an auxiliary to the other routines for elliptic integrals. This integral essentially calculates elementary functions, e.g.

$$\begin{aligned}\ln x &= (x-1).R_C\left(\left(\frac{1+x}{2}\right)^2, x\right), \quad x > 0; \\ \arcsin x &= x.R_C(1-x^2, 1), \quad |x| \leq 1; \\ \operatorname{arcsinh} x &= x.R_C(1+x^2, 1), \quad \text{etc.}\end{aligned}$$

In general this method of calculating these elementary functions is not recommended as there are usually much more efficient specific routines available in the Library. However, S21BAF may be used, for example, to compute  $\ln x/(x-1)$  when  $x$  is close to 1, without the loss of significant figures that occurs when  $\ln x$  and  $x-1$  are computed separately.

### 3.2 Bessel and Airy Functions

For computing the Bessel functions  $J_\nu(x)$ ,  $Y_\nu(x)$ ,  $I_\nu(x)$  and  $K_\nu(x)$  where  $x$  is real and  $\nu = 0$  or  $1$ , special routines are provided, which are much faster than the more general routines that allow a complex argument and arbitrary real  $\nu \geq 0$ . Similarly, special routines are provided for computing the Airy functions and their derivatives  $\operatorname{Ai}(x)$ ,  $\operatorname{Bi}(x)$ ,  $\operatorname{Ai}'(x)$ ,  $\operatorname{Bi}'(x)$  for a real argument which are much faster than the routines for complex arguments.

## 4 Index

Airy function, Ai, real argument	S17AGF
Airy function, Ai', real argument	S17AJF
Airy function, Ai or Ai', complex argument, optionally scaled	S17DGF
Airy function, Bi, real argument	S17AHF
Airy function, Bi', real argument	S17AKF
Airy function, Bi or Bi', complex argument, optionally scaled	S17DHF
Arccos, inverse circular cosine	S09ABF
Arccosh, inverse hyperbolic cosine	S11ACF
Arcsin, inverse circular sine	S09AAF
Arcsinh, inverse hyperbolic sine	S11ABF
Arctanh, inverse hyperbolic tangent	S11AAF
Bessel function, $J_0$ , real argument	S17AEF
Bessel function, $J_1$ , real argument	S17AFF
Bessel function, $J_\nu$ , complex argument, optionally scaled	S17DEF
Bessel function, $Y_0$ , real argument	S17ACF
Bessel function, $Y_1$ , real argument	S17ADF
Bessel function, $Y_\nu$ , complex argument, optionally scaled	S17DCF
Complement of the Cumulative Normal distribution	S15ACF
Complement of the Error function, real argument	S15ADF
Complement of the Error function, scaled, complex argument	S15DDF
Cosine, hyperbolic	S10ACF
Cosine Integral	S13ACF
Cumulative Normal distribution function	S15ABF
Dawson's Integral	S15AFF
Digamma function, scaled	S14ADF
Elliptic functions, Jacobian, sn, cn, dn	S21CAF
Elliptic integral, symmetrised, degenerate of 1st kind, $R_C$	S21BAF
Elliptic integral, symmetrised, of 1st kind, $R_F$	S21BBF
Elliptic integral, symmetrised, of 2nd kind, $R_D$	S21BCF
Elliptic integral, symmetrised, of 3rd kind, $R_J$	S21BDF
Erf, real argument	S15AEF
Erfc, real argument	S15ADF
Erfc, scaled, complex argument	S15DDF

Error function, real argument	S15AEF
Exponential, complex	S01EAF
Exponential Integral	S13AAF
Fresnel Integral, $C$	S20ADF
Fresnel Integral, $S$	S20ACF
Gamma function	S14AAF
Gamma function, incomplete	S14BAF
Generalized Factorial function	S14AAF
Hankel function $H_\nu^{(1)}$ or $H_\nu^{(2)}$ , complex argument, optionally scaled	S17DLF
Incomplete Gamma function	S14BAF
Jacobian elliptic functions, sn, cn, dn	S21CAF
Kelvin function, $\text{bei } x$	S19ABF
Kelvin function, $\text{ber } x$	S19AAF
Kelvin function, $\text{kei } x$	S19ADF
Kelvin function, $\text{ker } x$	S19ACF
Logarithm of Gamma function	S14ABF
Logarithm of $1 + x$	S01BAF
Modified Bessel function, $I_0$ , real argument	S18AEF
Modified Bessel function, $I_1$ , real argument	S18AFF
Modified Bessel function, $I_\nu$ , complex argument, optionally scaled	S18DEF
Modified Bessel function, $K_0$ , real argument	S18ACF
Modified Bessel function, $K_1$ , real argument	S18ADF
Modified Bessel function, $K_\nu$ , complex argument, optionally scaled	S18DCF
Psi function	S14ACF
Psi function and derivatives, scaled	S14ADF
Scaled modified Bessel function, $e^{- x }I_0(x)$ , real argument	S18CEF
Scaled modified Bessel function, $e^{- x }I_1(x)$ , real argument	S18CFF
Scaled modified Bessel function, $e^x K_0(x)$ , real argument	S18CCF
Scaled modified Bessel function, $e^x K_1(x)$ , real argument	S18CDF
Sine, hyperbolic	S10ABF
Sine integral	S13ADF
Tangent, circular	S07AAF
Tangent, hyperbolic	S10AAF
Trigamma function, scaled	S14ADF

## 5 References

- [1] Abramowitz M and Stegun I A (1972) *Handbook of Mathematical Functions* Dover Publications (3rd Edition)
- [2] Carlson B C (1965) On computing elliptic integrals and functions *J. Math. Phys.* **44** 36–51
- [3] Carlson B C (1977) *Special Functions of Applied Mathematics* Academic Press
- [4] Carlson B C (1977) Elliptic integrals of the first kind *SIAM J. Math. Anal.* **8** 231–242
- [5] Clenshaw C W (1962) Mathematical tables *Chebyshev Series for Mathematical Functions* HMSO
- [6] Fox L and Parker I B (1968) *Chebyshev Polynomials in Numerical Analysis* Oxford University Press
- [7] Schonfelder J L (1976) The production of special function routines for a multi-machine library *Softw. Pract. Exper.* **6** (1)





## S01BAF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S01BAF returns a value of the shifted logarithmic function,  $\ln(1+x)$ , via the routine name.

## 2. Specification

```

real FUNCTION S01BAF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

## 3. Description

This routine computes values of  $\ln(1+x)$ , retaining full relative precision even when  $|x|$  is small. The routine is based on the Chebyshev expansion

$$\ln \frac{1 + p^2 + 2p\bar{x}}{1 + p^2 - 2p\bar{x}} = 4 \sum_{k=0}^{\infty} \frac{p^{2k+1}}{2k+1} T_{2k+1}(\bar{x}).$$

Setting  $\bar{x} = \frac{x(1+p^2)}{2p(x+2)}$ , and choosing  $p = \frac{q-1}{q+1}$ ,  $q = \sqrt[4]{2}$  the expansion is valid in the domain  $x \in \left[ \frac{1}{\sqrt{2}} - 1, \sqrt{2} - 1 \right]$ . Outside this domain,  $\ln(1+x)$  is computed by the Fortran intrinsic logarithmic function.

## 4. References

- [1] LYUSTERNIK, L.A., CHERVONENKIS, O.A. and YANPOLSKII, A.R.  
Handbook for Computing Elementary Functions, p. 57.  
Pergamon Press, 1965.

## 5. Parameters

- 1: **X** – *real*. *Input*  
*On entry:* the argument  $x$  of the function.  
*Constraint:*  $X > -1.0$ .
- 2: **IFAIL** – **INTEGER**. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry,  $X \leq -1.0$ .

The result is returned as zero.

## 7. Accuracy

The returned result should be accurate almost to *machine precision*, with a limit of about 20 significant figures due to the precision of internal constants. Note however that if  $x$  lies very close to  $-1.0$  and is not exact (for example if  $x$  is the result of some previous computation and has been rounded), then precision will be lost in the computation of  $1 + x$ , and hence  $\ln(1+x)$ , in S01BAF.

## 8. Further Comments

Empirical tests show that the time taken for a call of S01BAF usually lies between about 1.25 and 2.5 times the time for a call to the standard Fortran function LOG.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S01BAF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5, NOUT=6)
*      .. External Functions ..
      real             S01BAF
      EXTERNAL          S01BAF
*      .. Local Scalars ..
      real             X, Y
      INTEGER           IFAIL
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S01BAF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '          X          Y'
20  READ (NIN,*,END=40) X
      IFAIL = 0
*
      Y = S01BAF(X, IFAIL)
*
      WRITE (NOUT,99999) X, Y
      GO TO 20
40  STOP
*
99999 FORMAT (1X,1P,2e12.4)
      END
```

### 9.2. Program Data

```
S01BAF Example Program Data
 2.50E+0
 1.25E-1
-9.06E-1
 1.29E-3
-7.83E-6
 1.00E-9
```

### 9.3. Program Results

S01BAF Example Program Results

X	Y
2.5000E+00	1.2528E+00
1.2500E-01	1.1778E-01
-9.0600E-01	-2.3645E+00
1.2900E-03	1.2892E-03
-7.8300E-06	-7.8300E-06
1.0000E-09	1.0000E-09

---



## S01EAF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S01EAF evaluates the exponential function  $e^z$ , for *complex*  $z$ .

## 2. Specification

```
complex FUNCTION S01EAF (Z, IFAIL)
      INTEGER          IFAIL
      complex         Z
```

## 3. Description

This routine evaluates the exponential function  $e^z$ , taking care to avoid machine overflow, and giving a warning if the result cannot be computed to more than half precision. The function is evaluated as  $e^z = e^x (\cos y + i \sin y)$ , where  $x$  and  $y$  are the real and imaginary parts respectively of  $z$ .

Since  $\cos y$  and  $\sin y$  are less than or equal to 1 in magnitude, it is possible that  $e^x$  may overflow although  $e^x \cos y$  or  $e^x \sin y$  does not. In this case the alternative formula  $\text{sign}(\cos y)e^{x+\ln|\cos y|}$  is used for the real part of the result, and  $\text{sign}(\sin y)e^{x+\ln|\sin y|}$  for the imaginary part. If either part of the result still overflows, a warning is returned through parameter IFAIL.

If  $\text{Im } z$  is too large, precision may be lost in the evaluation of  $\sin y$  and  $\cos y$ . Again, a warning is returned through IFAIL.

## 4. References

None.

## 5. Parameters

- 1:  $Z$  – *complex*. *Input*  
*On entry*: the argument  $z$  of the function.
- 2: IFAIL – INTEGER. *Input/Output*  
*On entry*: IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit*: IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

The real part of the result overflows, and is set to the largest safe number with the correct sign. The imaginary part of the result is meaningful.

IFAIL = 2

The imaginary part of the result overflows, and is set to the largest safe number with the correct sign. The real part of the result is meaningful.

IFAIL = 3

Both real and imaginary parts of the result overflow, and are set to the largest safe number with the correct signs.

IFAIL = 4

The computed result is accurate to less than half precision, due to the size of  $\text{Im } z$ .

IFAIL = 5

The computed result has no precision, due to the size of  $\text{Im } z$ , and is set to zero.

## 7. Accuracy

Accuracy is limited in general only by the accuracy of the Fortran intrinsic functions in the computation of  $\sin y$ ,  $\cos y$  and  $e^x$ , where  $x = \text{Re } z$ ,  $y = \text{Im } z$ . As  $y$  gets larger, precision will probably be lost due to argument reduction in the evaluation of the sine and cosine functions, until the warning error IFAIL = 4 occurs when  $y$  gets larger than  $\sqrt{1/\varepsilon}$ , where  $\varepsilon$  is the *machine precision*. Note that on some machines, the intrinsic functions SIN and COS will not operate on arguments larger than about  $\sqrt{1/\varepsilon}$ , and so IFAIL can never return as 4.

In the comparatively rare event that the result is computed by the formulae  $\text{sign}(\cos y)e^{x+\ln|\cos y|}$  and  $\text{sign}(\sin y)e^{x+\ln|\sin y|}$ , a further small loss of accuracy may be expected due to rounding errors in the logarithmic function.

## 8. Further Comments

None.

## 9. Example

The following program reads values of the argument  $z$  from a file, evaluates the function at each value of  $z$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S01EAF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5, NOUT=6)
*      .. Local Scalars ..
      complex         W, Z
      INTEGER          IFAIL
*      .. External Functions ..
      complex         S01EAF
      EXTERNAL        S01EAF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S01EAF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*)
+ '          Z                      exp(Z)'
20 READ (NIN,*,END=40) Z
      IFAIL = 0
*
      W = S01EAF(Z, IFAIL)
*
      WRITE (NOUT,99999) Z, W
      GO TO 20
40 STOP
*
99999 FORMAT (1X, '(', F12.4, ', ', F12.4, ')', ' ', '(', F12.4, ', ', F12.4, ')')
      END
```

## 9.2. Program Data

```
S01EAF Example Program Data
( 1.0, 0.0)
(-0.5, 2.0)
( 0.0,-2.0)
(-2.5,-1.5)
```

## 9.3. Program Results

```
S01EAF Example Program Results
```

z		exp(z)	
( 1.0000,	0.0000)	( 2.7183,	0.0000)
( -0.5000,	2.0000)	( -0.2524,	0.5515)
( 0.0000,	-2.0000)	( -0.4161,	-0.9093)
( -2.5000,	-1.5000)	( 0.0058,	-0.0819)

---





## S07AAF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S07AAF returns the value of the circular tangent,  $\tan x$ , via the routine name.

## 2. Specification

```
real FUNCTION S07AAF (X, IFAIL)
  INTEGER      IFAIL
  real        X
```

## 3. Description

The routine calculates an approximate value for the circular tangent of its argument,  $\tan x$ . It is based on the Chebyshev expansion

$$\tan \theta = \theta y(t) = \theta \sum_{r=0}^{\infty} c_r T_r(t)$$

$$\text{where } -\frac{\pi}{4} < \theta < \frac{\pi}{4} \text{ and } -1 < t < +1, \quad t = 2\left(\frac{4\theta}{\pi}\right)^2 - 1.$$

The reduction to the standard range is accomplished by taking

$$x = N\pi/2 + \theta$$

where  $N$  is an integer and  $-\frac{\pi}{4} < \theta < \frac{\pi}{4}$ ,

$$\text{i.e. } \theta = x - \left(\frac{2x}{\pi}\right)\frac{\pi}{2} \text{ where } N = \left[\frac{2x}{\pi}\right] = \text{the nearest integer to } \frac{2x}{\pi}.$$

From the properties of  $\tan x$  it follows that

$$\tan x = \begin{cases} \tan \theta, & N \text{ even} \\ -1/\tan \theta, & N \text{ odd} \end{cases}$$

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 4.3, p. 71, 1968.

## 5. Parameters

- 1: **X** – *real*. *Input*  
*On entry:* the argument  $x$  of the function.
- 2: **IFAIL** – **INTEGER**. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

**IFAIL** = 1

The routine has been called with an argument that is too large; the default result returned is zero.

IFAIL = 2

The routine has been called with an argument that is too close to an odd multiple of  $\pi/2$ , at which the function is infinite; the routine returns a value with the correct sign but a more or less arbitrary but large magnitude (see Section 7).

## 7. Accuracy

If  $\delta$  and  $\varepsilon$  are the relative errors in the argument and result respectively, then in principle

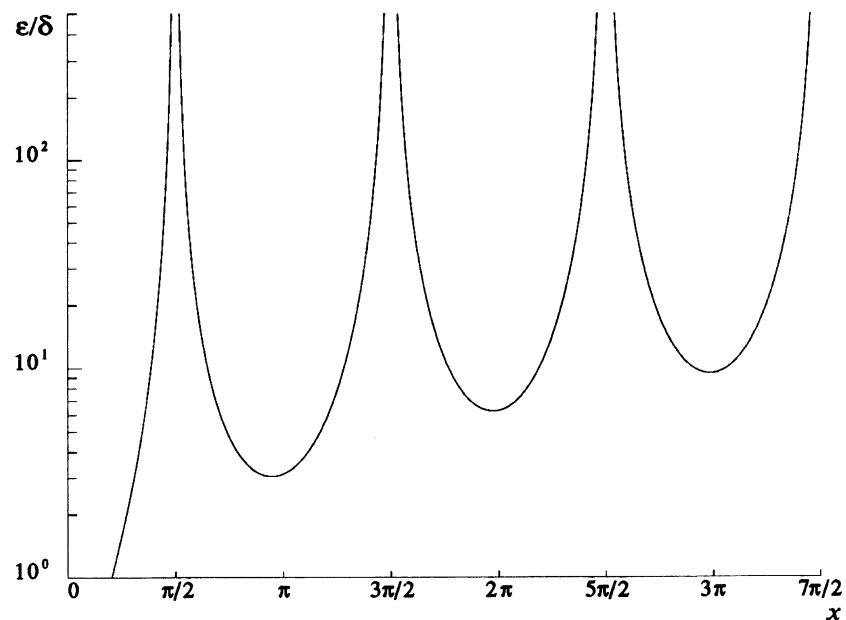
$$\varepsilon \geq \frac{2x}{\sin 2x} \delta.$$

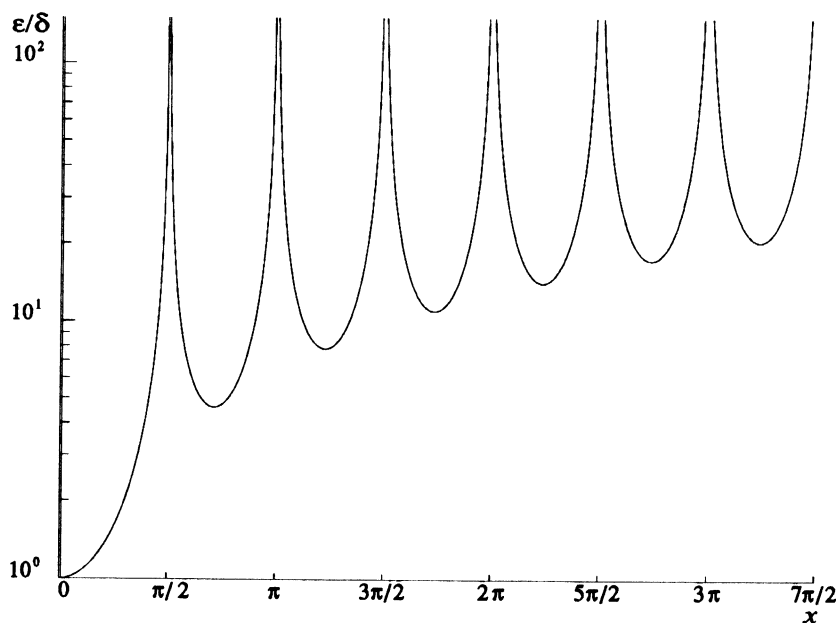
That is a relative error in the argument,  $x$ , is amplified by at least a factor  $2x/\sin 2x$  in the result. Similarly if  $E$  is the absolute error in the result this is given by

$$E \geq \frac{x}{\cos^2 x} \delta.$$

The equalities should hold if  $\delta$  is greater than the *machine precision* ( $\delta$  is a result of data errors etc.) but if  $\delta$  is simply the round off error in the machine it is possible that internal calculation rounding will lose an extra figure.

The graphs below show the behaviour of these amplification factors.





In the principal range it is possible to preserve relative accuracy even near the zero of  $\tan x$  at  $x = 0$  but at the other zeros only absolute accuracy is possible. Near the infinities of  $\tan x$  both the relative and absolute errors become infinite and the routine must fail (error 2).

If  $N$  is odd and  $|\theta| \leq xF_2$  the routine could not return better than two figures and in all probability would produce a result that was in error in its most significant figure. Therefore the routine fails and it returns the value

$$-\text{sign}\theta \left( \frac{1}{|xF_2|} \right) \approx -\text{sign}\theta \tan \left( \frac{\pi}{2} - |xF_2| \right)$$

which is the value of the tangent at the nearest argument for which a valid call could be made.

Accuracy is also unavoidably lost if the routine is called with a large argument. If  $|x| > F_1$  the routine fails (error 1) and returns zero. (See the Users' Note for your implementation for specific values of  $F_1$  and  $F_2$ ).

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S07AAF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER        (NIN=5, NOUT=6)
*      .. Local Scalars ..
real           X, Y
INTEGER          IFAIL
```

```

*      .. External Functions ..
      real                S07AAF
      EXTERNAL           S07AAF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S07AAF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S07AAF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END

```

## 9.2. Program Data

```

S07AAF Example Program Data
      -2.0
      -0.5
      1.0
      3.0
      1.5708

```

## 9.3. Program Results

```

S07AAF Example Program Results

```

X	Y	IFAIL
-2.000E+00	2.185E+00	0
-5.000E-01	-5.463E-01	0
1.000E+00	1.557E+00	0
3.000E+00	-1.425E-01	0
1.571E+00	-2.722E+05	0

---

## S09AAF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S09AAF returns the value of the inverse circular sine,  $\arcsin x$ , via the routine name. The value is in the principal range  $(-\pi/2, \pi/2)$

## 2. Specification

```

real FUNCTION S09AAF (X, IFAIL)
  INTEGER          IFAIL
  real            X

```

## 3. Description

The routine calculates an approximate value for the inverse circular sine,  $\arcsin x$ . It is based on the Chebyshev expansion

$$\arcsin x = x \times y(x) = x \sum_{r=0}^{\infty} a_r T_r(t)$$

where  $-\frac{1}{\sqrt{2}} \leq x \leq \frac{1}{\sqrt{2}}$  and  $t = 4x^2 - 1$ .

For  $x^2 \leq \frac{1}{2}$ ,  $\arcsin x = x \times y(x)$ .

For  $\frac{1}{2} < x^2 \leq 1$ ,  $\arcsin x = \text{sign } x \left\{ \frac{\pi}{2} - \arcsin \sqrt{1-x^2} \right\}$ .

For  $x^2 > 1$ ,  $\arcsin x$  is undefined and the routine fails.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 4.4, p. 79, 1968.

## 5. Parameters

1: X – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

*Constraint:*  $|X| \leq 1.0$ .

2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

The routine has been called with an argument greater than 1.0 in absolute value;  $\arcsin x$  is undefined and the routine returns zero.

## 7. Accuracy

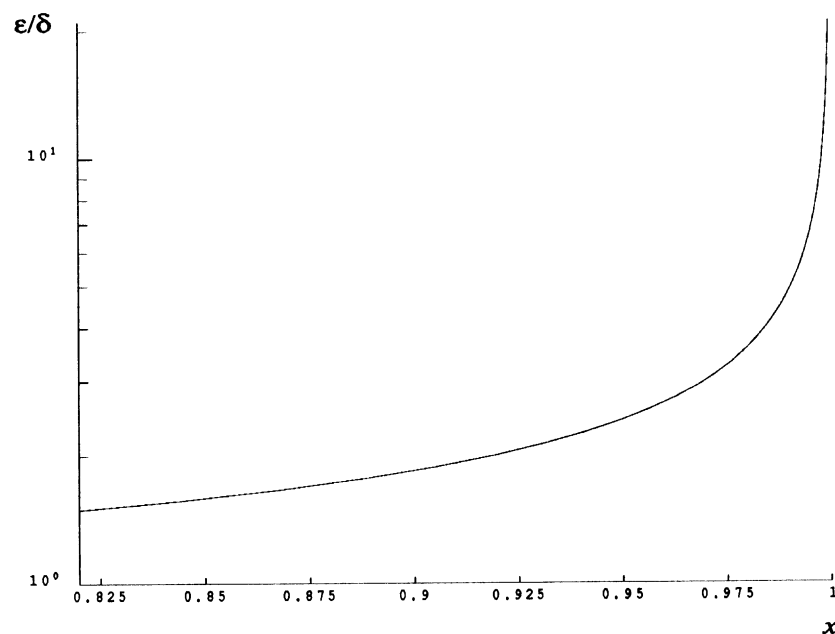
If  $\delta$  and  $\varepsilon$  are the relative errors in the argument and result, respectively, then in principle

$$|\varepsilon| \approx \left| \frac{x}{\arcsin x \sqrt{1-x^2}} \times \delta \right|.$$

That is, a relative error in the argument  $x$ , is amplified by at least a factor  $\frac{x}{\arcsin x \sqrt{1-x^2}}$  in the result.

The equality should hold if  $\delta$  is greater than the *machine precision* ( $\delta$  is a result of data errors etc.) but if  $\delta$  is produced simply by round off error in the machine it is possible that rounding in internal calculations may lose an extra figure in the result.

This factor stays close to one except near  $|x| = 1$  where its behaviour is shown in the following graph.



For  $|x|$  close to unity,  $1 - |x| \sim \delta$ , the above analysis is no longer applicable owing to the fact that both argument and result are subject to finite bounds, ( $|x| \leq 1$  and  $|\arcsin x| \leq \frac{1}{2}\pi$ ). In this region  $\varepsilon \sim \sqrt{\delta}$ ; that is the result will have approximately half as many correct significant figures as the argument.

For  $|x| = 1$  the result will be correct to full *machine precision*.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      S09AAF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
real           X, Y
INTEGER         IFAIL
*      .. External Functions ..
real           S09AAF
EXTERNAL        S09AAF
*      .. Executable Statements ..
WRITE (NOUT,*) 'S09AAF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
WRITE (NOUT,*)
WRITE (NOUT,*) '      X          Y          IFAIL'
WRITE (NOUT,*)
20 READ (NIN,*,END=40) X
   IFAIL = 1
*
   Y = S09AAF(X, IFAIL)
*
   WRITE (NOUT,99999) X, Y, IFAIL
   GO TO 20
40 STOP
*
99999 FORMAT (1X,1P,2e12.3,I7)
      END

```

### 9.2. Program Data

```

S09AAF Example Program Data
      -0.5
       0.1
       0.9
       2.0
      -1.5

```

### 9.3. Program Results

S09AAF Example Program Results

X	Y	IFAIL
-5.000E-01	-5.236E-01	0
1.000E-01	1.002E-01	0
9.000E-01	1.120E+00	0
2.000E+00	0.000E+00	1
-1.500E+00	0.000E+00	1

---





## S09ABF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S09ABF returns the value of the inverse circular cosine,  $\arccos x$ , via the routine name; the result is in the principal range  $(0, \pi)$ .

## 2. Specification

```
real FUNCTION S09ABF (X, IFAIL)
      INTEGER          IFAIL
      real             X
```

## 3. Description

The routine calculates an approximate value for the inverse circular cosine,  $\arccos x$ . It is based on the Chebyshev expansion

$$\arcsin x = x y(t) = x \sum_{r=0}^{\infty} a_r T_r(t)$$

where  $-\frac{1}{\sqrt{2}} \leq x \leq \frac{1}{\sqrt{2}}$ , and  $t = 4x^2 - 1$ .

For  $x^2 \leq \frac{1}{2}$ ,  $\arccos x = \frac{\pi}{2} - \arcsin x$ .

For  $-1 \leq x < -\frac{1}{\sqrt{2}}$ ,  $\arccos x = \pi - \arcsin \sqrt{1-x^2}$ .

For  $\frac{1}{\sqrt{2}} < x \leq 1$ ,  $\arccos x = \arcsin \sqrt{1-x^2}$ .

For  $|x| > 1$ ,  $\arccos x$  is undefined and the routine fails.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 4.4., p. 79, 1968.

## 5. Parameters

1: X – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

*Constraint:*  $|X| \leq 1.0$ .

2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

S09ABF has been called with  $|X| > 1.0$ , for which  $\arccos$  is undefined. A zero result is returned.

## 7. Accuracy

If  $\delta$  and  $\varepsilon$  are the relative errors in the argument and the result, respectively, then in principle

$$|\varepsilon| \approx \left| \frac{x}{\arccos x \sqrt{1-x^2}} \times \delta \right|.$$

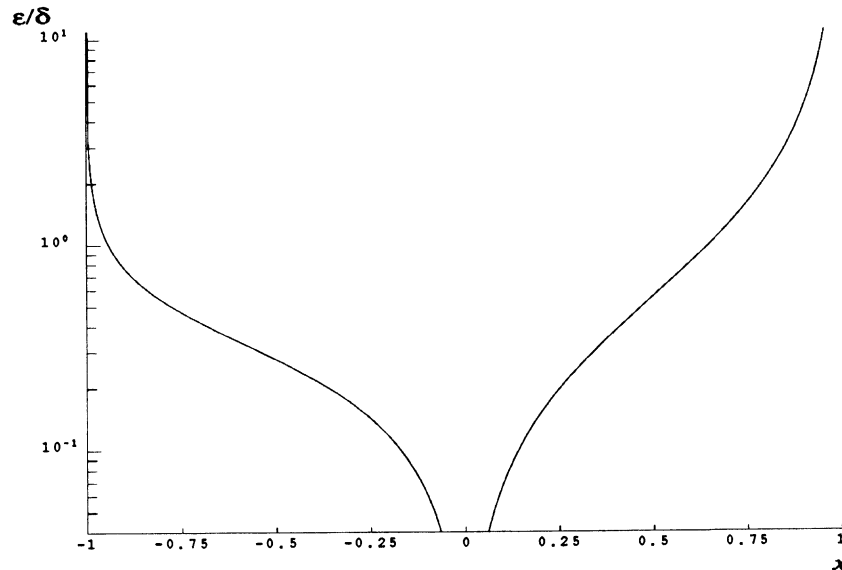
The equality should hold if  $\delta$  is greater than the *machine precision* ( $\delta$  is due to data errors etc.), but if  $\delta$  is due simply to round-off in the machine it is possible that rounding etc. in internal calculations may lose one extra figure.

The behaviour of the amplification factor  $\frac{x}{\arccos x \sqrt{1-x^2}}$  is shown in the graph below.

In the region of  $x = 0$  this factor tends to zero and the accuracy will be limited by the *machine precision*. For  $|x|$  close to one,  $1 - |x| \sim \delta$ , the above analysis is not applicable owing to the fact that both the argument and the result are bounded  $|x| \leq 1$ ,  $0 \leq \arccos x \leq \pi$ .

In the region of  $x \sim -1$  we have  $\varepsilon \sim \sqrt{\delta}$ , that is the result will have approximately half as many correct significant figures as the argument.

In the region  $x \sim +1$ , we have that the absolute error in the result,  $E$ , is given by  $E \sim \sqrt{\delta}$ , that is the result will have approximately half as many decimal places correct as there are correct figures in the argument.



## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      S09ABF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real             X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real             S09ABF
      EXTERNAL         S09ABF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S09ABF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1

*
      Y = S09ABF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END

```

## 9.2. Program Data

```

S09ABF Example Program Data
      -0.5
       0.1
       0.9
       2.0
      -1.5

```

## 9.3. Program Results

S09ABF Example Program Results

X	Y	IFAIL
-5.000E-01	2.094E+00	0
1.000E-01	1.471E+00	0
9.000E-01	4.510E-01	0
2.000E+00	0.000E+00	1
-1.500E+00	0.000E+00	1

---



## S10AAF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S10AAF returns a value for the hyperbolic tangent,  $\tanh x$ , via the routine name.

## 2. Specification

```

real FUNCTION S10AAF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

## 3. Description

The routine calculates an approximate value for the hyperbolic tangent of its argument,  $\tanh x$ .

For  $|x| \leq 1$  it is based on the Chebyshev expansion

$$\tanh x = x \sum_{r=0}^{\infty} a_r T_r(t)$$

where  $-1 \leq x \leq 1$ ,  $-1 \leq t \leq 1$ , and  $t = 2x^2 - 1$ .

For  $1 < |x| < E_1$  (See the Users' Note for your implementation for value of  $E_1$ )

$$\tanh x = \frac{e^{2x} - 1}{e^{2x} + 1}$$

For  $|x| \geq E_1$ ,  $\tanh x = \text{sign } x$  to within the representation accuracy of the machine and so this approximation is used.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 4.5, p. 83, 1968.

## 5. Parameters

1: X – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

There are no error exits from this routine. The parameter IFAIL is included for consistency with the other routines in this chapter.

## 7. Accuracy

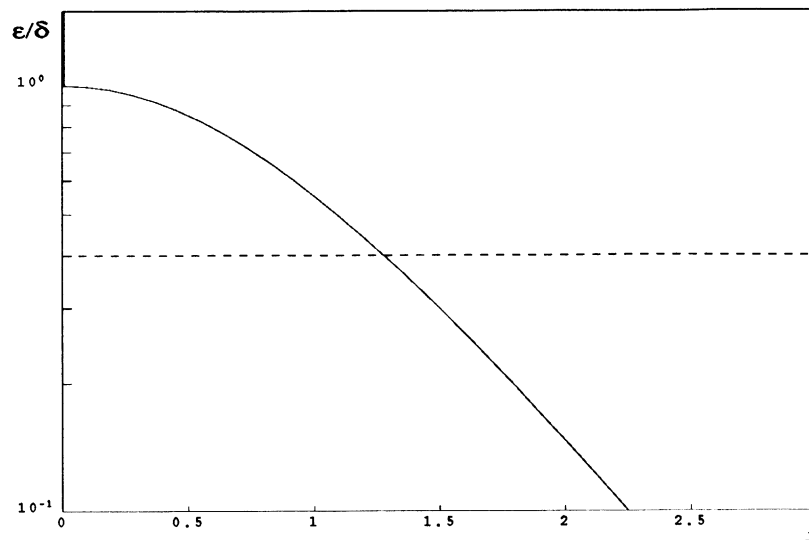
If  $\delta$  and  $\varepsilon$  are the relative errors in the argument and the result respectively, then in principle,

$$|\varepsilon| \approx \left| \frac{2x}{\sinh 2x} \delta \right|.$$

That is, a relative error in the argument,  $x$ , is amplified by a factor approximately  $\frac{2x}{\sinh 2x}$ , in the result.

The equality should hold if  $\delta$  is greater than the *machine precision* ( $\delta$  due to data errors etc.) but if  $\delta$  is due simply to the round-off in the machine representation it is possible that an extra figure may be lost in internal calculation round-off.

The behaviour of the amplification factor is shown in the following graph:



It should be noted that this factor is always less than or equal to 1.0 and away from  $x = 0$  the accuracy will eventually be limited entirely by the precision of machine representation.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S10AAF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S10AAF
      EXTERNAL         S10AAF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S10AAF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20  READ (NIN,*,END=40) X
      IFAIL = 1
*
*      Y = S10AAF(X,IFAIL)
*
```

```
        WRITE (NOUT,99999) X, Y, IFAIL
        GO TO 20
    40 STOP
*
99999 FORMAT (1X,1P,2E12.3,I7)
        END
```

## 9.2. Program Data

```
S10AAF Example Program Data
      -20.0
      -5.0
      0.5
      5.0
```

## 9.3. Program Results

S10AAF Example Program Results

X	Y	IFAIL
-2.000E+01	-1.000E+00	0
-5.000E+00	-9.999E-01	0
5.000E-01	4.621E-01	0
5.000E+00	9.999E-01	0

---





## S10ABF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S10ABF returns the value of the hyperbolic sine,  $\sinh x$ , via the routine name.

## 2. Specification

```

real FUNCTION S10ABF (X, IFAIL)
      INTEGER      IFAIL
      real         X

```

## 3. Description

The routine calculates an approximate value for the hyperbolic sine of its argument,  $\sinh x$ .

For  $|x| \leq 1$  it uses the Chebyshev expansion

$$\sinh x = x y(t) = x \sum_{r=0}^{\infty} a_r T_r(t)$$

where  $t = 2x^2 - 1$ .

For  $1 < |x| \leq E_1$ ,  $\sinh x = \frac{1}{2}(e^x - e^{-x})$

where  $E_1$  is a machine-dependent constant, details of which are given in the Users' Note for your implementation.

For  $|x| > E_1$ , the routine fails owing to the danger of setting overflow in calculating  $e^x$ . The result returned for such calls is  $\sinh(\text{sign } x E_1)$ , i.e. it returns the result for the nearest valid argument.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 4.5, p. 83, 1968.

## 5. Parameters

1: **X** – *real*. *Input*  
*On entry:* the argument  $x$  of the function.

2: **IFAIL** – **INTEGER**. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

**IFAIL** = 1

The routine has been called with an argument too large in absolute magnitude. There is a danger of setting overflow. The result is the value of  $\sinh$  at the closest argument for which a valid call could be made. (See Section 3 and the Users' Note for your implementation).

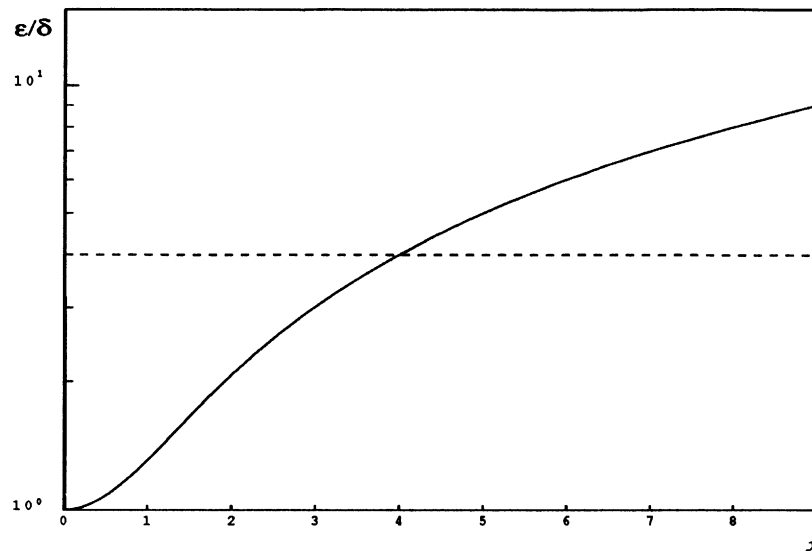
## 7. Accuracy

If  $\delta$  and  $\varepsilon$  are the relative errors in the argument and result, respectively, then in principle

$$|\varepsilon| \approx |x \coth x \times \delta|.$$

That is the relative error in the argument,  $x$ , is amplified by a factor, approximately  $x \coth x$ . The equality should hold if  $\delta$  is greater than the *machine precision* ( $\delta$  is a result of data errors etc.) but, if  $\delta$  is simply a result of round-off in the machine representation of  $x$ , then it is possible that an extra figure may be lost in internal calculation round-off.

The behaviour of the error amplification factor can be seen in the following graph:



It should be noted that for  $|x| \geq 2$

$$\varepsilon \sim x\delta = \Delta$$

where  $\Delta$  is the absolute error in the argument.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S10ABF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5, NOUT=6)
*      .. Local Scalars ..
      real             X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real             S10ABF
      EXTERNAL         S10ABF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S10ABF Example Program Results'
```

```

*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S10ABF(X, IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END

```

## 9.2. Program Data

```

S10ABF Example Program Data
      -10.0
      -0.5
      0.0
      0.5
      25.0

```

## 9.3. Program Results

S10ABF Example Program Results

X	Y	IFAIL
-1.000E+01	-1.101E+04	0
-5.000E-01	-5.211E-01	0
0.000E+00	0.000E+00	0
5.000E-01	5.211E-01	0
2.500E+01	3.600E+10	0

---



## S10ACF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

S10ACF returns the value of the hyperbolic cosine,  $\cosh x$ , via the routine name.

### 2. Specification

```

real FUNCTION S10ACF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

### 3. Description

The routine calculates an approximate value for the hyperbolic cosine,  $\cosh x$ .

For  $|x| \leq E_1$ ,  $\cosh x = \frac{1}{2}(e^x + e^{-x})$ .

For  $|x| > E_1$ , the routine fails owing to danger of setting overflow in calculating  $e^x$ . The result returned for such calls is  $\cosh E_1$ , i.e. it returns the result for the nearest valid argument. The value of machine-dependent constant  $E_1$  may be given in the Users' Note for your implementation.

### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 4.5, p. 83, 1968.

### 5. Parameters

- 1: X – *real*. *Input*  
*On entry:* the argument  $x$  of the function.
- 2: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

### 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

The routine has been called with an argument too large in absolute magnitude. There is a danger of overflow. The result returned is the value of  $\cosh x$  at the nearest valid argument.

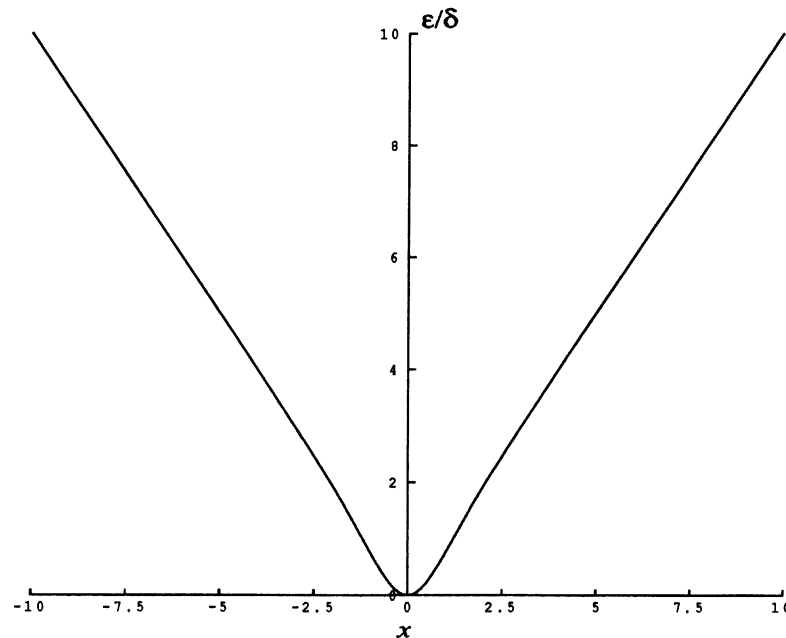
### 7. Accuracy

If  $\delta$  and  $\varepsilon$  are the relative errors in the argument and result, respectively, then in principle

$$\varepsilon \simeq x \tanh x \times \delta.$$

That is, the relative error in the argument,  $x$ , is amplified by a factor, at least  $x \tanh x$ . The equality should hold if  $\delta$  is greater than the *machine precision* ( $\delta$  is due to data errors etc.) but if  $\delta$  is simply a result of round-off in the machine representation of  $x$  then it is possible that an extra figure may be lost in internal calculation round-off.

The behaviour of the error amplification factor is shown by the following graph:



It should be noted that near  $x = 0$  where this amplification factor tends to zero the accuracy will be limited eventually by the machine precision. Also for  $|x| \geq 2$

$$\varepsilon \sim x\delta = \Delta$$

where  $\Delta$  is the absolute error in the argument  $x$ .

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S10ACF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
real           X, Y
INTEGER          IFAIL
*      .. External Functions ..
real          S10ACF
EXTERNAL        S10ACF
*      .. Executable Statements ..
WRITE (NOUT,*) 'S10ACF Example Program Results'
```

```

*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S10ACF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END

```

## 9.2. Program Data

```

S10ACF Example Program Data
      -10.0
      -0.5
      0.0
      0.5
      25.0

```

## 9.3. Program Results

S10ACF Example Program Results

X	Y	IFAIL
-1.000E+01	1.101E+04	0
-5.000E-01	1.128E+00	0
0.000E+00	1.000E+00	0
5.000E-01	1.128E+00	0
2.500E+01	3.600E+10	0

---





## S11AAF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S11AAF returns the value of the inverse hyperbolic tangent,  $\operatorname{arctanh} x$ , via the routine name.

## 2. Specification

```
real FUNCTION S11AAF (X, IFAIL)
  INTEGER      IFAIL
  real        X
```

## 3. Description

The routine calculates an approximate value for the inverse hyperbolic tangent of its argument,  $\operatorname{arctanh} x$ .

For  $x^2 \leq \frac{1}{2}$  it is based on the Chebyshev expansion

$$\operatorname{arctanh} x = x \sum_{r=0}^{\infty} a_r T_r(t)$$

where  $-\frac{1}{\sqrt{2}} \leq x \leq \frac{1}{\sqrt{2}}$ ,  $-1 \leq t \leq 1$ , and  $t = 4x^2 - 1$ .

For  $\frac{1}{2} < x^2 < 1$ , it uses

$$\operatorname{arctanh} x = \frac{1}{2} \ln \left( \frac{1+x}{1-x} \right).$$

For  $|x| \geq 1$ , the routine fails as  $\operatorname{arctanh} x$  is undefined.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 4.6, p. 86, 1968.

## 5. Parameters

1: X – *real*. *Input*

*On entry:* the argument  $x$  of the function.

*Constraint:*  $|X| < 1.0$ .

2: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

The routine has been called with an argument greater than or equal to 1.0 in magnitude, for which  $\operatorname{arctanh}$  is not defined. The result is returned as zero.

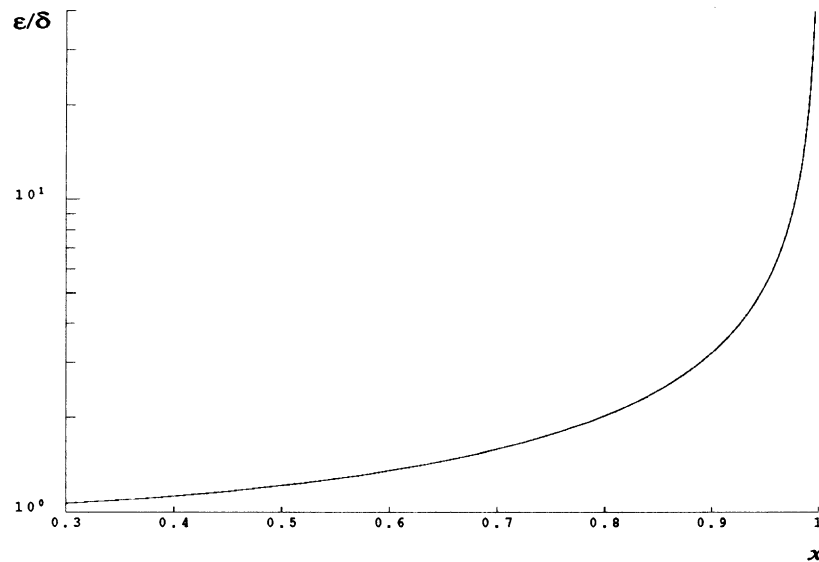
## 7. Accuracy

If  $\delta$  and  $\varepsilon$  are the relative errors in the argument and result, respectively, then in principle

$$|\varepsilon| \approx \left| \frac{x}{(1-x^2)\operatorname{arctanh} x} \times \delta \right|.$$

That is, the relative error in the argument,  $x$ , is amplified by at least a factor  $\frac{x}{(1-x^2)\operatorname{arctanh} x}$  in the result. The equality should hold if  $\delta$  is greater than the *machine precision* ( $\delta$  due to data errors etc.) but if  $\delta$  is simply due to round-off in the machine representation then it is possible that an extra figure may be lost in internal calculation round-off.

The behaviour of the amplification factor is shown in the following graph:



The factor is not significantly greater than one except for arguments close to  $|x| = 1$ . However in the region where  $|x|$  is close to one,  $1 - |x| \sim \delta$ , the above analysis is inapplicable since  $x$  is bounded by definition,  $|x| < 1$ . In this region where  $\operatorname{arctanh}$  is tending to infinity we have

$$\varepsilon \sim 1/\ln \delta$$

which implies an obvious, unavoidable serious loss of accuracy near  $|x| \sim 1$ , e.g. if  $x$  and 1 agree to 6 significant figures, the result for  $\operatorname{arctanh} x$  would be correct to at most about one figure.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      S11AAF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S11AAF
      EXTERNAL        S11AAF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S11AAF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S11AAF(X, IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END

```

### 9.2. Program Data

```

S11AAF Example Program Data
      -0.5
      0.0
      0.5
      -0.9999
      3.0

```

### 9.3. Program Results

S11AAF Example Program Results

X	Y	IFAIL
-5.000E-01	-5.493E-01	0
0.000E+00	0.000E+00	0
5.000E-01	5.493E-01	0
-9.999E-01	-4.952E+00	0
3.000E+00	0.000E+00	1

---



## S11ABF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

S11ABF returns the value of the inverse hyperbolic sine,  $\operatorname{arcsinh} x$ , via the routine name.

### 2. Specification

```

real FUNCTION S11ABF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

### 3. Description

The routine calculates an approximate value for the inverse hyperbolic sine of its argument,  $\operatorname{arcsinh} x$ .

For  $|x| \leq 1$  it is based on the Chebyshev expansion

$$\operatorname{arcsinh} x = x \times y(t) = x \sum_{r=0}^{\infty} c_r T_r(t), \quad \text{where } t = 2x^2 - 1.$$

For  $|x| > 1$  it uses the fact that

$$\operatorname{arcsinh} x = \operatorname{sign} x \times \ln(|x| + \sqrt{x^2 + 1}).$$

This form is used directly for  $1 < |x| < 10^k$ , where  $k = n/2 + 1$ , and the machine uses approximately  $n$  decimal place arithmetic.

For  $|x| \geq 10^k$ ,  $\sqrt{x^2 + 1}$  is equal to  $|x|$  to within the accuracy of the machine and hence we can guard against premature overflow and, without loss of accuracy, calculate

$$\operatorname{arcsinh} x = \operatorname{sign} x \times (\ln 2 + \ln |x|).$$

### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 4.6, p. 86, 1968.

### 5. Parameters

1: X – *real*. *Input*  
*On entry:* the argument  $x$  of the function.

2: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

### 6. Error Indicators and Warnings

Errors detected by the routine:

There are no error exits from this routine. The parameter IFAIL is included for consistency with the other routines in this chapter.

## 7. Accuracy

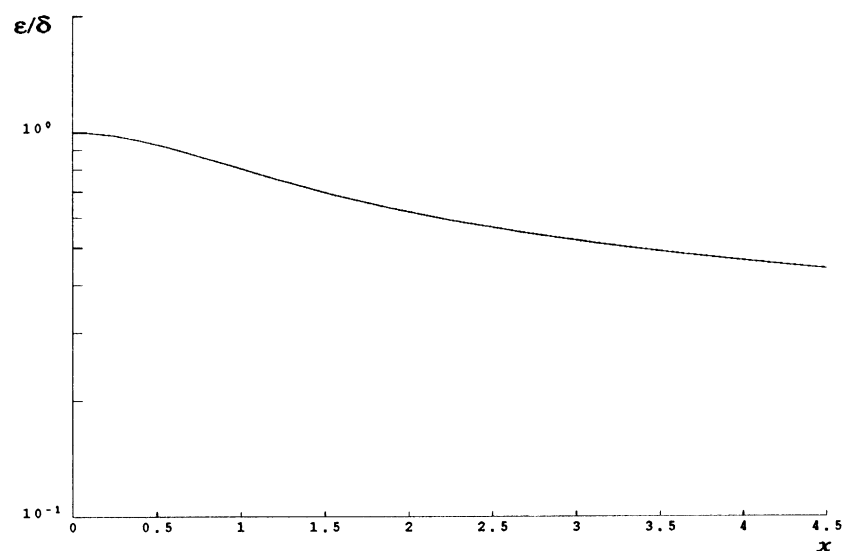
If  $\delta$  and  $\varepsilon$  are the relative errors in the argument and the result, respectively, then in principle

$$|\varepsilon| \approx \left| \frac{x}{\sqrt{1+x^2} \operatorname{arcsinh} x} \delta \right|.$$

That is, the relative error in the argument,  $x$ , is amplified by a factor at least  $\frac{x}{\sqrt{1+x^2} \operatorname{arcsinh} x}$ , in the result.

The equality should hold if  $\delta$  is greater than the *machine precision* ( $\delta$  due to data errors etc.) but if  $\delta$  is simply due to round-off in the machine representation it is possible that an extra figure may be lost in internal calculation round-off.

The behaviour of the amplification factor is shown in the following graph:



It should be noted that this factor is always less than or equal to one. For large  $x$  we have the absolute error in the result,  $E$ , in principle, given by

$$E \sim \delta.$$

This means that eventually accuracy is limited by *machine precision*.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S11ABF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5, NOUT=6)
*      .. Local Scalars ..
      real             X, Y
      INTEGER           IFAIL
```

```

*      .. External Functions ..
      real                S11ABF
      EXTERNAL           S11ABF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S11ABF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X            Y            IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S11ABF(X, IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END

```

## 9.2. Program Data

```

S11ABF Example Program Data
      -2.0
      -0.5
      1.0
      6.0

```

## 9.3. Program Results

S11ABF Example Program Results

X	Y	IFAIL
-2.000E+00	-1.444E+00	0
-5.000E-01	-4.812E-01	0
1.000E+00	8.814E-01	0
6.000E+00	2.492E+00	0

---





## S11ACF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S11ACF returns the value of the inverse hyperbolic cosine,  $\operatorname{arccosh} x$ , via the routine name. The result is in the principal positive branch.

## 2. Specification

```
real FUNCTION S11ACF (X, IFAIL)
      INTEGER          IFAIL
      real             X
```

## 3. Description

The routine calculates an approximate value for the inverse hyperbolic cosine,  $\operatorname{arccosh} x$ . It is based on the relation

$$\operatorname{arccosh} x = \ln(x + \sqrt{x^2 - 1}).$$

This form is used directly for  $1 < x < 10^k$ , where  $k = n/2 + 1$ , and the machine uses approximately  $n$  decimal place arithmetic.

For  $x \geq 10^k$ ,  $\sqrt{x^2 - 1}$  is equal to  $\sqrt{x}$  to within the accuracy of the machine and hence we can guard against premature overflow and, without loss of accuracy, calculate

$$\operatorname{arccosh} x = \ln 2 + \ln x.$$

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 4.6, p. 86, 1968.

## 5. Parameters

1: X – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

*Constraint:*  $X \geq 1.0$ .

2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

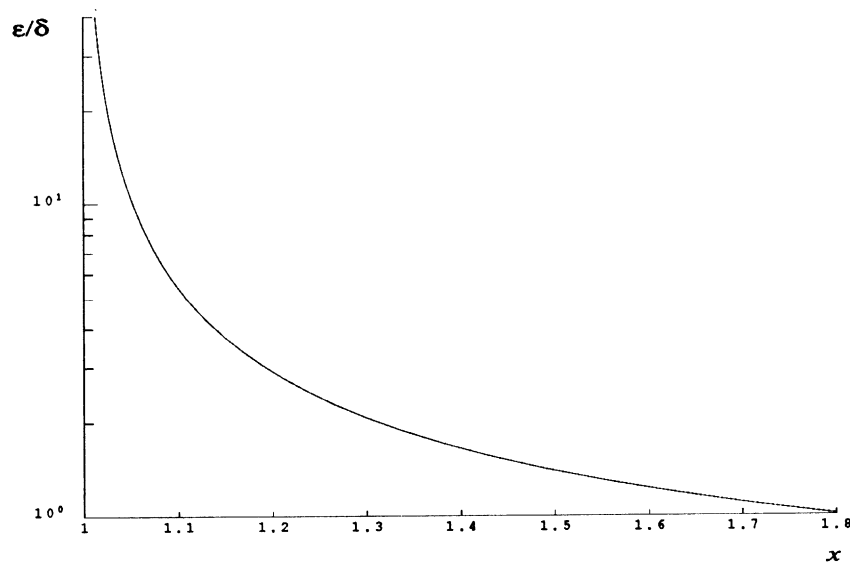
The routine has been called with an argument less than 1.0, for which  $\operatorname{arccosh} x$  is not defined. The result returned is zero.

## 7. Accuracy

If  $\delta$  and  $\varepsilon$  are the relative errors in the argument and result respectively, then in principle

$$|\varepsilon| \approx \left| \frac{x}{\sqrt{x^2-1} \operatorname{arccosh} x} \times \delta \right|.$$

That is the relative error in the argument is amplified by a factor at least  $\frac{x}{\sqrt{x^2-1} \operatorname{arccosh} x}$  in the result. The equality should apply if  $\delta$  is greater than the *machine precision* ( $\delta$  due to data errors etc.) but if  $\delta$  is simply a result of round-off in the machine representation it is possible that an extra figure may be lost in internal calculation and round-off. The behaviour of the amplification factor is shown in the following graph:



It should be noted that for  $x > 2$  the factor is always less than 1.0. For large  $x$  we have the absolute error  $E$  in the result, in principle, given by

$$E \sim \delta.$$

This means that eventually accuracy is limited by *machine precision*. More significantly for  $x$  close to 1,  $x - 1 \sim \delta$ , the above analysis becomes inapplicable due to the fact that both function and argument are bounded,  $x \geq 1$ ,  $\operatorname{arccosh} x \geq 0$ . In this region we have

$$E \sim \sqrt{\delta}.$$

That is, there will be approximately half as many decimal places correct in the result as there were correct figures in the argument.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      S11ACF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S11ACF
      EXTERNAL         S11ACF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S11ACF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S11ACF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2e12.3,I7)
      END

```

### 9.2. Program Data

```

S11ACF Example Program Data
      1.00
      2.0
      5.0
      10.0
      -0.5

```

### 9.3. Program Results

S11ACF Example Program Results

X	Y	IFAIL
1.000E+00	0.000E+00	0
2.000E+00	1.317E+00	0
5.000E+00	2.292E+00	0
1.000E+01	2.993E+00	0
-5.000E-01	0.000E+00	1

---



## S13AAF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised terms* and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S13AAF returns the value of the exponential integral  $E_1(x)$ , via the routine name.

## 2. Specification

```

real FUNCTION S13AAF (X, IFAIL)
      INTEGER          IFAIL
      real            X

```

## 3. Description

The routine calculates an approximate value for

$$E_1(x) = \int_x^{\infty} \frac{e^{-u}}{u} du, \quad x > 0.$$

For  $0 < x \leq 4$ , the approximation is based on the Chebyshev expansion

$$E_1(x) = y(t) - \ln x = \sum_r' a_r T_r(t) - \ln x,$$

where  $t = \frac{1}{2}x - 1$ .

For  $x > 4$ ,

$$E_1(x) = \frac{e^{-x}}{x} y(t) = \frac{e^{-x}}{x} \sum_r' a_r T_r(t),$$

$$\text{where } t = -1.0 + 14.5/(x+3.25) = \frac{11.25-x}{3.25+x}.$$

In both cases,  $-1 \leq t \leq +1$ .

To guard against producing underflows, if  $x > x_{hi}$  the result is set directly to zero. For the value of  $x_{hi}$  see the Users' Note for your implementation.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 5.1, p, 228, 1968.

## 5. Parameters

1: X – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

*Constraint:*  $X > 0.0$ .

2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

The routine has been called with an argument less than or equal to zero for which the function is not defined. The result returned is zero.

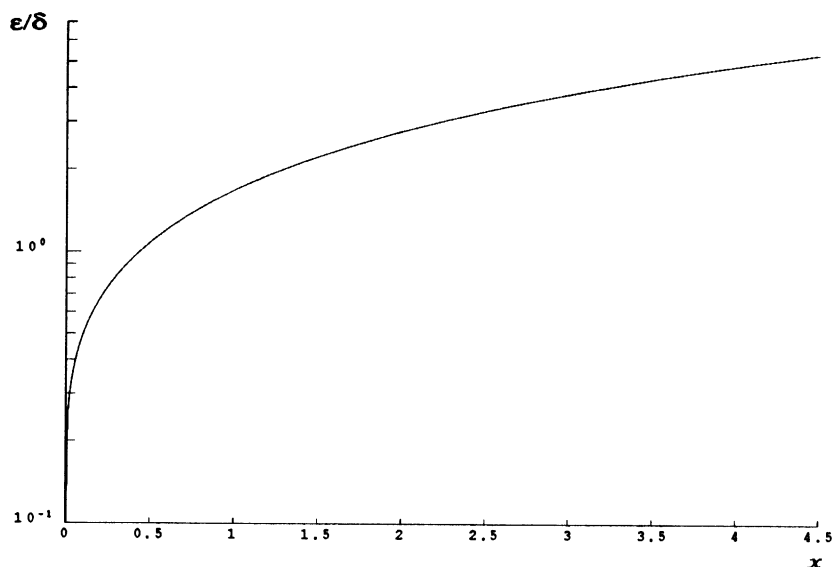
## 7. Accuracy

If  $\delta$  and  $\varepsilon$  are the relative errors in argument and result respectively, then in principle,

$$|\varepsilon| \approx \left| \frac{e^{-x}}{E_1(x)} \times \delta \right|$$

so the relative error in the argument is amplified in the result by at least a factor  $e^{-x}/E_1(x)$ . The equality should hold if  $\delta$  is greater than the *machine precision* ( $\delta$  due to data errors etc.) but if  $\delta$  is simply a result of round-off in the machine representation, it is possible that an extra figure may be lost in internal calculation and round-off.

The behaviour of this amplification factor is shown in the following graph:



It should be noted that, for small  $x$ , the amplification factor tends to zero and eventually the error in the result will be limited by *machine precision*.

For large  $x$ ,

$$\varepsilon \sim x\delta = \Delta,$$

the absolute error in the argument.

## 8. Further Comments

None.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      S13AAF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S13AAF
      EXTERNAL         S13AAF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S13AAF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X      Y      IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S13AAF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END

```

### 9.2. Program Data

```

S13AAF Example Program Data
      2.0
     -1.0

```

### 9.3. Program Results

S13AAF Example Program Results

X	Y	IFAIL
2.000E+00	4.890E-02	0
-1.000E+00	0.000E+00	1

---





## S13ACF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S13ACF returns the value of the cosine integral

$$\text{Ci}(x) = \gamma + \ln x + \int_0^x \frac{\cos u - 1}{u} du, \quad x > 0$$

via the routine name, where  $\gamma$  denotes Euler's constant.

## 2. Specification

```

real FUNCTION S13ACF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

## 3. Description

The routine calculates an approximate value for  $\text{Ci}(x)$ .

For  $0 < x \leq 16$  it is based on the Chebyshev expansion

$$\text{Ci}(x) = \ln x + \sum_{r=0}^{\infty} a_r T_r(t), \quad t = 2\left(\frac{x}{16}\right)^2 - 1.$$

For  $16 < x < x_{hi}$  where the value of  $x_{hi}$  is given in the Users' Note for your implementation,

$$\text{Ci}(x) = \frac{f(x) \sin x}{x} - \frac{g(x) \cos x}{x^2}$$

where  $f(x) = \sum_{r=0}^{\infty} f_r T_r(t)$  and  $g(x) = \sum_{r=0}^{\infty} g_r T_r(t)$ ,  $t = 2\left(\frac{16}{x}\right)^2 - 1$ .

For  $x \geq x_{hi}$ ,  $\text{Ci}(x) = 0$  to within the accuracy possible (see Section 7).

## 4. References

- [1] ABRAMOWITZ, .M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 5.2, p. 231, 1968.

## 5. Parameters

- 1: **X** – *real*. *Input*  
*On entry:* the argument  $x$  of the function.  
*Constraint:*  $X > 0.0$ .
- 2: **IFAIL** – **INTEGER**. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

**IFAIL** = 1

The routine has been called with an argument less than or equal to zero for which the function is not defined. The result returned is zero.

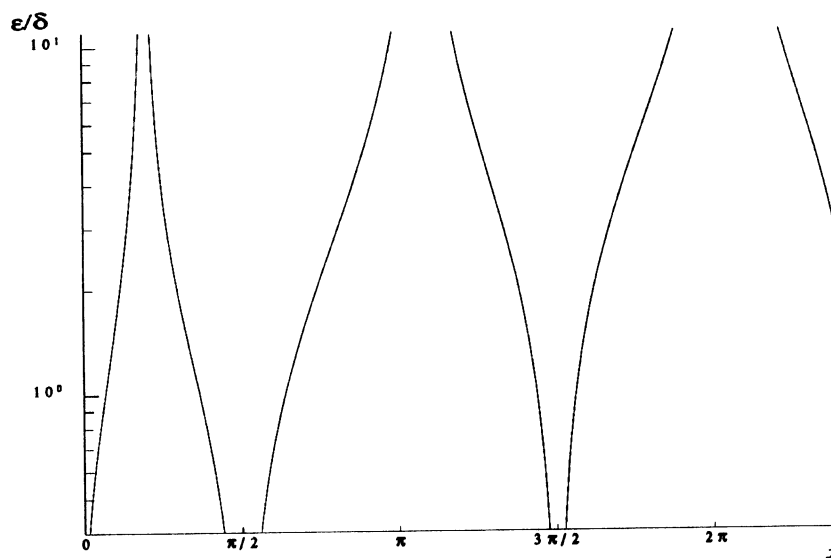
## 7. Accuracy

If  $E$  and  $\varepsilon$  are the absolute and relative errors in the result and  $\delta$  is the relative error in the argument then in principle these are related by

$$|E| \approx |\delta \cos x| \text{ and } |\varepsilon| \approx \left| \frac{\delta \cos x}{\text{Ci}(x)} \right|.$$

That is accuracy will be limited by *machine precision* near the origin and near the zeros of  $\cos x$ , but near the zeros of  $\text{Ci}(x)$  only absolute accuracy can be maintained.

The behaviour of this amplification is shown in the following graph:



For large values of  $x$ ,  $\text{Ci}(x) \sim \frac{\sin x}{x}$  therefore  $\varepsilon \sim \delta x \cot x$  and since  $\delta$  is limited by the finite precision of the machine it becomes impossible to return results which have any relative accuracy. That is, when  $x \geq 1/\delta$  we have that  $|\text{Ci}(x)| \leq 1/x \sim E$  and hence is not significantly different from zero.

Hence  $x_{hi}$  is chosen such that for values of  $x \geq x_{hi}$ ,  $\text{Ci}(x)$  in principle would have values less than the *machine precision* and so is essentially zero.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S13ACF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5, NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER         IFAIL
```

```

*      .. External Functions ..
      real          S13ACF
      EXTERNAL      S13ACF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S13ACF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S13ACF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END

```

## 9.2. Program Data

```

S13ACF Example Program Data
      0.2
      0.4
      0.6
      0.8
      1.0

```

## 9.3. Program Results

S13ACF Example Program Results

X	Y	IFAIL
2.000E-01	-1.042E+00	0
4.000E-01	-3.788E-01	0
6.000E-01	-2.227E-02	0
8.000E-01	1.983E-01	0
1.000E+00	3.374E-01	0

---



## S13ADF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S13ADF returns the value of the sine integral

$$\text{Si}(x) = \int_0^x \frac{\sin u}{u} du,$$

via the routine name.

## 2. Specification

```

real FUNCTION S13ADF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

## 3. Description

The routine calculates an approximate value for  $\text{Si}(x)$ .

For  $|x| \leq 16.0$  it is based on the Chebyshev expansion

$$\text{Si}(x) = x \sum_{r=0}^{\infty} a_r T_r(t), \quad t = 2 \left( \frac{x}{16} \right)^2 - 1.$$

For  $16 < |x| < x_{hi}$ , where  $x_{hi}$  is an implementation dependent number,

$$\text{Si}(x) = \text{sign} \left\{ \frac{\pi}{2} \frac{f(x) \cos x}{x} - \frac{g(x) \sin x}{x^2} \right\}$$

where  $f(x) = \sum_{r=0}^{\infty} f_r T_r(t)$  and  $g(x) = \sum_{r=0}^{\infty} g_r T_r(t)$ ,  $t = 2 \left( \frac{16}{x} \right)^2 - 1$ .

For  $|x| \geq x_{hi}$ ,  $\text{Si}(x) = \frac{1}{2} \pi \text{sign } x$  to within *machine precision*.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 5.2, p. 231, 1968.

## 5. Parameters

1: X – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

There are no failure exits from this routine. The parameter IFAIL has been included for consistency with other routines in this chapter.

## 7. Accuracy

If  $\delta$  and  $\varepsilon$  are the relative errors in the argument and result, respectively, then in principle

$$|\varepsilon| \approx \left| \frac{\delta \sin x}{\text{Si}(x)} \right|.$$

The equality may hold if  $\delta$  is greater than the *machine precision* ( $\delta$  due to data errors etc.) but if  $\delta$  is simply due to round-off in the machine representation, then since the factor relating  $\delta$  to  $\varepsilon$  is always less than one, the accuracy will be limited by *machine precision*.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S13ADF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5, NOUT=6)
*      .. Local Scalars ..
real           X, Y
INTEGER         IFAIL
*      .. External Functions ..
real          S13ADF
EXTERNAL        S13ADF
*      .. Executable Statements ..
WRITE (NOUT,*) 'S13ADF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
WRITE (NOUT,*)
WRITE (NOUT,*) '      X          Y          IFAIL'
WRITE (NOUT,*)
20 READ (NIN,*,END=40) X
   IFAIL = 1
*
   Y = S13ADF(X, IFAIL)
*
   WRITE (NOUT,99999) X, Y, IFAIL
   GO TO 20
40 STOP
*
99999 FORMAT (1X,1P,2e12.3,I7)
END
```

### 9.2. Program Data

```
S13ADF Example Program Data
0.0
0.2
0.4
0.6
0.8
1.0
```

### 9.3. Program Results

S13ADF Example Program Results

X	Y	IFAIL
0.000E+00	0.000E+00	0
2.000E-01	1.996E-01	0
4.000E-01	3.965E-01	0
6.000E-01	5.881E-01	0
8.000E-01	7.721E-01	0
1.000E+00	9.461E-01	0

---





## S14AAF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

S14AAF returns the value of the Gamma function  $\Gamma(x)$ , via the routine name.

### 2. Specification

```

real FUNCTION S14AAF (X, IFAIL)
      INTEGER          IFAIL
      real            X

```

### 3. Description

This routine evaluates an approximation to the Gamma function  $\Gamma(x)$ . The routine is based on the Chebyshev expansion:

$$\Gamma(1+u) = \sum_{r=0}^{\infty} a_r T_r(t), \quad \text{where } 0 \leq u < 1, \quad t = 2u - 1,$$

and uses the property  $\Gamma(1+x) = x\Gamma(x)$ . If  $x = N + 1 + u$  where  $N$  is integral and  $0 \leq u < 1$  then it follows that:

$$\text{for } N > 0 \quad \Gamma(x) = (x-1)(x-2) \dots (x-N)\Gamma(1+u),$$

$$\text{for } N = 0 \quad \Gamma(x) = \Gamma(1+u),$$

$$\text{for } N < 0 \quad \Gamma(x) = \frac{\Gamma(1+u)}{x(x+1)(x+2) \dots (x-N-1)}.$$

There are four possible failures for this routine:

- (i) if  $x$  is too large, there is a danger of overflow since  $\Gamma(x)$  could become too large to be represented in the machine;
- (ii) if  $x$  is too large and negative, there is a danger of underflow;
- (iii) if  $x$  is equal to a negative integer,  $\Gamma(x)$  would overflow since it has poles at such points;
- (iv) if  $x$  is too near zero, there is again the danger of overflow on some machines. For small  $x$ ,  $\Gamma(x) \simeq \frac{1}{x}$ , and on some machines there exists a range of non-zero but small values of  $x$  for which  $1/x$  is larger than the greatest representable value.

### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 6, p. 255, 1968.

### 5. Parameters

- 1: **X** – *real*. *Input*  
*On entry:* the argument  $x$  of the function.  
*Constraint:* X must not be zero or a negative integer.
- 2: **IFAIL** – **INTEGER**. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

**9.3. Program Results**

S14AAF Example Program Results

X	Y	IFAIL
1.000E+00	1.000E+00	0
1.250E+00	9.064E-01	0
1.500E+00	8.862E-01	0
1.750E+00	9.191E-01	0
2.000E+00	1.000E+00	0
5.000E+00	2.400E+01	0
1.000E+01	3.629E+05	0
-1.500E+00	2.363E+00	0

---

## S14ABF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S14ABF returns a value for the logarithm of the Gamma function,  $\ln \Gamma(x)$ , via the routine name.

## 2. Specification

```

real FUNCTION S14ABF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

## 3. Description

This routine evaluates an approximation to  $\ln \Gamma(x)$ . It is based on two Chebyshev expansions.

For  $0 < x \leq x_{small}$ ,  $\ln \Gamma(x) = -\ln x$  to within machine accuracy.

For  $x_{small} < x \leq 15.0$ , the recursive relation  $\Gamma(1+x) = x\Gamma(x)$  is used to reduce the calculation to one involving  $\Gamma(1+u)$ ,  $0 \leq u < 1$  which is evaluated as:

$$\Gamma(1+u) = \sum_{r=0}^{\infty} a_r T_r(t), \quad t = 2u - 1.$$

Once  $\Gamma(x)$  has been calculated, the required result is produced by taking the logarithm.

For  $15.0 < x \leq x_{big}$ ,

$$\ln \Gamma(x) = (x - \frac{1}{2}) \ln x - x + \frac{1}{2} \ln 2\pi + y(x)/x$$

$$\text{where } y(x) = \sum_{r=0}^{\infty} b_r T_r(t), \quad t = 2\left(\frac{15}{x}\right)^2 - 1.$$

For  $x_{big} < x \leq x_{vbig}$  the term  $y(x)/x$  is negligible and so its calculation is omitted.

For  $x > x_{vbig}$  there is a danger of setting overflow so the routine must fail.

For  $x \leq 0.0$  the function is not defined and the routine fails.

Note:  $x_{small}$  is calculated so that if  $x < x_{small}$ ,  $\Gamma(x) = 1/x$  to within machine accuracy.

$x_{big}$  is calculated so that if  $x > x_{big}$ ,

$$\ln \Gamma(x) = (x - \frac{1}{2}) \ln x - x + \frac{1}{2} \ln 2\pi$$

to within machine accuracy.

$x_{vbig}$  is calculated so that  $\ln \Gamma(x_{vbig})$  is close to the value returned by X02ALF.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 6, pp. 255, 1968.

## 5. Parameters

- 1: X – *real*. *Input*  
On entry: the argument  $x$  of the function.  
Constraint: X > 0.0.
- 2: IFAIL – INTEGER. *Input/Output*  
On entry: IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

$X \leq 0.0$ , the function is undefined. On soft failure, the routine returns zero.

IFAIL = 2

$X$  is too large, the function would overflow. On soft failure, the routine returns the value of the function at the largest permissible argument.

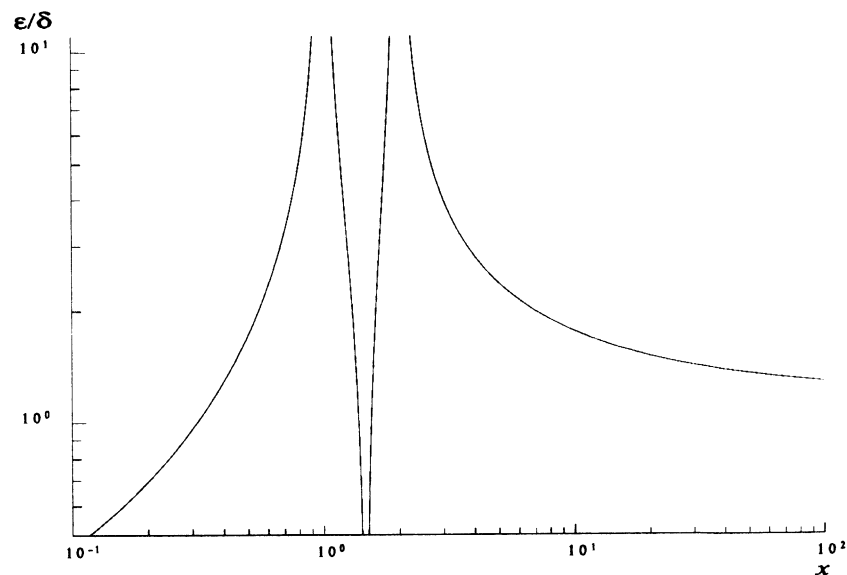
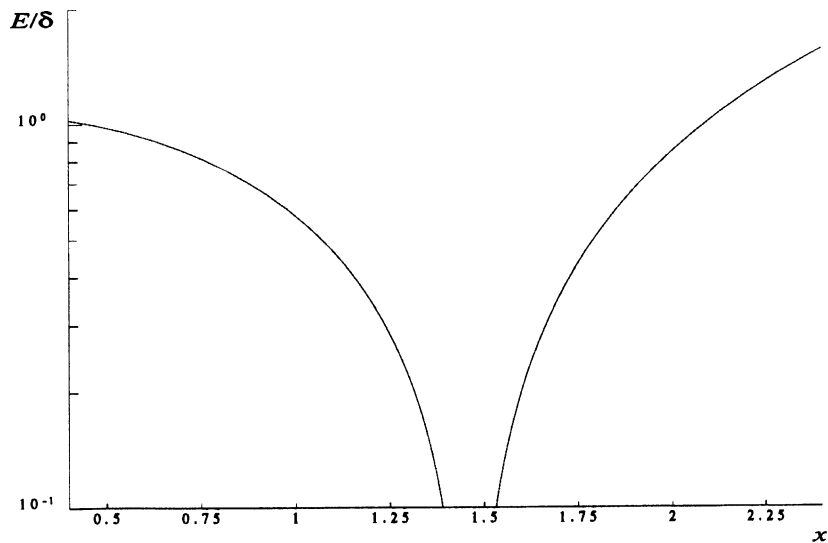
7. Accuracy

Let  $\delta$  and  $\varepsilon$  be the relative errors in the argument and result respectively, and  $E$  be the absolute error in the result.

If  $\delta$  is somewhat larger than the relative *machine precision*, then

$$E \approx |x \Psi(x)| \delta \text{ and } \varepsilon \approx \left| \frac{x \Psi(x)}{\ln \Gamma(x)} \right| \delta$$

where  $\Psi(x)$  is the digamma function  $\frac{\Gamma'(x)}{\Gamma(x)}$ . The following graphs show the behaviour of these error amplification factors:



These show that relative error can be controlled, since except near  $x = 1$  or  $2$  relative error is attenuated by the function or at least is not greatly amplified.

For large  $x$ ,  $\varepsilon \simeq \left(1 + \frac{1}{\ln x}\right)\delta$  and for small  $x$ ,  $\varepsilon \simeq \frac{1}{\ln x}\delta$ .

The function  $\ln \Gamma(x)$  has zeros at  $x = 1$  and  $2$  and hence relative accuracy is not maintainable near those points. However absolute accuracy can still be provided near those zeros as is shown above.

If however,  $\delta$  is of the order of the *machine precision*, then rounding errors in the routine's internal arithmetic may result in errors which are slightly larger than those predicted by the equalities. It should be noted that even in areas where strong attenuation of errors is predicted the relative precision is bounded by the effective *machine precision*.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S14ABF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
real           X, Y
INTEGER          IFAIL
*      .. External Functions ..
real           S14ABF
EXTERNAL        S14ABF
*      .. Executable Statements ..
WRITE (NOUT,*) 'S14ABF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
WRITE (NOUT,*)
WRITE (NOUT,*) '      X          Y          IFAIL'
WRITE (NOUT,*)
20 READ (NIN,*,END=40) X
   IFAIL = 1
*
   Y = S14ABF(X,IFAIL)
*
   WRITE (NOUT,99999) X, Y, IFAIL
   GO TO 20
40 STOP
*
99999 FORMAT (1X,1P,2e12.3,I7)
END
```

**9.2. Program Data**

S14ABF Example Program Data

```
1.0
1.25
1.5
1.75
2.0
5.0
10.0
20.0
1000.0
0.0
-5.0
```

**9.3. Program Results**

S14ABF Example Program Results

X	Y	IFAIL
1.000E+00	0.000E+00	0
1.250E+00	-9.827E-02	0
1.500E+00	-1.208E-01	0
1.750E+00	-8.440E-02	0
2.000E+00	0.000E+00	0
5.000E+00	3.178E+00	0
1.000E+01	1.280E+01	0
2.000E+01	3.934E+01	0
1.000E+03	5.905E+03	0
0.000E+00	0.000E+00	1
-5.000E+00	0.000E+00	1

---

## S14ACF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S14ACF returns a value of the function  $\psi(x) - \ln x$ , where  $\psi$  is the psi function

$$\psi(x) = \frac{d}{dx} \ln \Gamma(x) = \frac{\Gamma'(x)}{\Gamma(x)}.$$

## 2. Specification

```
real FUNCTION S14ACF (X, IFAIL)
      INTEGER      IFAIL
      real         X
```

## 3. Description

This routine returns a value of the function  $\psi(x) - \ln x$ . The psi function is computed without the logarithmic term so that when  $x$  is large, sums or differences of psi functions may be computed without unnecessary loss of precision, by analytically combining the logarithmic terms. For example, the difference  $d = \Psi(x+\frac{1}{2}) - \psi(x)$  has an asymptotic behaviour for large  $x$  given by  $d \sim \ln(x+\frac{1}{2}) - \ln x + O\left(\frac{1}{x^2}\right) \sim \ln\left(1+\frac{1}{2x}\right) \sim \frac{1}{2x}$ .

Computing  $d$  directly would amount to subtracting two large numbers which are close to  $\ln(x+\frac{1}{2})$  and  $\ln x$  to produce a small number close to  $\frac{1}{2x}$ , resulting in a loss of significant digits. However, using this routine to compute  $f(x) = \psi(x) - \ln x$ , we can compute  $d = f(x+\frac{1}{2}) - f(x) + \ln\left(1+\frac{1}{2x}\right)$ , and the dominant logarithmic term may be computed accurately from its power series when  $x$  is large. Thus we avoid the unnecessary loss of precision.

The routine is derived from the routine PSIFN in Amos [1].

## 4. References

- [1] AMOS, D.E.  
Algorithm 610: A Portable FORTRAN Subroutine for Derivatives of the Psi Function.  
ACM Trans. Math. Software 9, pp. 494-502, 1983.
- [2] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions, Ch. 6, pp. 258-260.  
Dover Publications, 1968.

## 5. Parameters

- 1: X – *real*. *Input*  
*On entry:* the argument  $x$  of the function.  
*Constraint:* X > 0.0.
- 2: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry  $IFAIL = 0$  or  $-1$ , explanatory error messages are output on the current error message unit (as defined by X04AAF).

$IFAIL = 1$

On entry,  $X \leq 0.0$ . S14ACF returns the value zero.

$IFAIL = 2$

No result is computed because underflow is likely. The value of  $X$  is too large. S14ACF returns the value zero.

$IFAIL = 3$

No result is computed because overflow is likely. The value of  $X$  is too small. S14ACF returns the value zero.

## 7. Accuracy

All constants in subroutine S14ACF are given to approximately 18 digits of precision. Calling the number of digits of precision in the floating-point arithmetic being used  $t$ , then clearly the maximum number of correct digits in the results obtained is limited by  $p = \min(t, 18)$ .

With the above proviso, results returned by this routine should be accurate almost to full precision, except at points close to the zero of  $\psi(x)$ ,  $x \approx 1.461632$ , where only absolute rather than relative accuracy can be obtained.

## 8. Further Comments

None.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S14ACF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5, NOUT=6)
*      .. Local Scalars ..
      real            F, X
      INTEGER          IFAIL
*      .. External Functions ..
      real            S14ACF
      EXTERNAL         S14ACF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S14ACF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '          X          psi(X)-log(X)'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 0
*
      F = S14ACF(X, IFAIL)
*
```



```
        WRITE (NOUT,99999) X, F
        GO TO 20
    40 STOP
*
99999 FORMAT (1X,F12.4,F15.4)
END
```

### 9.2. Program Data

S14ACF Example Program Data  
0.1  
0.5  
3.6  
8.0

### 9.3. Program Results

S14ACF Example Program Results

X	psi(X)-log(X)
0.1000	-8.1212
0.5000	-1.2704
3.6000	-0.1453
8.0000	-0.0638

---



## S14ADF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S14ADF returns a sequence of values of scaled derivatives of the psi function  $\psi(x)$ .

## 2. Specification

```
SUBROUTINE S14ADF (X, N, M, ANS, IFAIL)
  INTEGER          N, M, IFAIL
  real            X, ANS(M)
```

## 3. Description

This routine computes  $m$  values of the function

$$w(k,x) = \frac{(-1)^{k+1} \psi^{(k)}(x)}{k!},$$

for  $x > 0$ ,  $k = n, n+1, \dots, n+m-1$ , where  $\psi$  is the psi function

$$\psi(x) = \frac{d}{dx} \ln \Gamma(x) = \frac{\Gamma'(x)}{\Gamma(x)},$$

and  $\psi^{(k)}$  denotes the  $k$ th derivative of  $\psi$ .

The routine is derived from the routine PSIFN in Amos [1]. The basic method of evaluation of  $w(k,x)$  is the asymptotic series

$$w(k,x) \sim \varepsilon(k,x) + \frac{1}{2x^{k+1}} + \frac{1}{x^k} \sum_{j=1}^{\infty} B_{2j} \frac{(2j+k-1)!}{(2j)! k! x^{2j}}$$

for large  $x$  greater than a machine dependent value  $x_{\min}$ , followed by backward recurrence using

$$w(k,x) = w(k,x+1) + x^{-k-1}$$

for smaller values of  $x$ , where  $\varepsilon(k,x) = -\ln x$  when  $k = 0$ ,  $\varepsilon(k,x) = \frac{1}{kx^k}$  when  $k > 0$ , and  $B_{2j}$ ,  $j = 1, 2, \dots$ , are the Bernoulli numbers.

When  $k$  is large, the above procedure may be inefficient, and the expansion

$$w(k,x) = \sum_{j=1}^{\infty} \frac{1}{(x+j)^{k+1}},$$

which converges rapidly for large  $k$ , is used instead.

## 4. References

- [1] AMOS, D.E.  
Algorithm 610: A Portable FORTRAN Subroutine for Derivatives of the Psi Function.  
ACM Trans. Math. Software 9, pp. 494-502, 1983.
- [2] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions, Ch. 6, pp. 258-260.  
Dover Publications, 1968.

## 5. Parameters

1:  $X$  – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

*Constrain:*  $X > 0.0$ .

- 2: N – INTEGER. *Input*  
*On entry:* the first member  $n$  of the sequence of functions.  
*Constraint:*  $N \geq 0$ .
- 3: M – INTEGER. *Input*  
*On entry:* the number of members  $m$  required in the sequence  $w(k,x)$ , for  $k = n, n+1, \dots, n+m-1$ .  
*Constraint:*  $M \geq 1$ .
- 4: ANS(M) – *real* array. *Output*  
*On exit:* the first  $m$  elements of ANS contain the required values  $w(k,x)$ , for  $k = n, n+1, \dots, n+m-1$ .
- 5: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry,  $X \leq 0.0$ .

IFAIL = 2

On entry,  $N < 0$ .

IFAIL = 3

On entry,  $M < 1$ .

IFAIL = 4

No results are returned because underflow is likely. Either  $X$  or  $N + M - 1$  is too large. If possible, reduce the value of  $M$  and call S14ADF again.

IFAIL = 5

No results are returned because overflow is likely. Either  $X$  is too small, or  $N + M - 1$  is too large. If possible, reduce the value of  $M$  and call S14ADF again.

IFAIL = 6

No results are returned because there is not enough internal workspace to continue computation.  $N + M - 1$  may be too large. If possible, reduce the value of  $M$  and call S14ADF again.

## 7. Accuracy

All constants in subroutine S14ADF are given to approximately 18 digits of precision. Calling the number of digits of precision in the floating-point arithmetic being used  $t$ , then clearly the maximum number of correct digits in the results obtained is limited by  $p = \min(t, 18)$ . Empirical tests of S14ADF, taking values of  $x$  in the range  $0.0 < x < 50.0$ , and  $n$  in the range  $1 \leq n \leq 50$ , have shown that the maximum relative error is a loss of approximately two decimal places of precision. Tests with  $n = 0$ , i.e. testing the function  $-\psi(x)$ , have shown somewhat better accuracy, except at points close to the zero of  $\psi(x)$ ,  $x \approx 1.461632$ , where only absolute accuracy can be obtained.

## 8. Further Comments

The time taken for a call of S14ADF is approximately proportional to  $m$ , plus a constant. In general, it is much cheaper to call S14ADF with  $m$  greater than 1 to evaluate the function  $w(k,x)$ , for  $k = n, n+1, \dots, n+m-1$ , rather than to make  $m$  separate calls of S14ADF.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S14ADF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5, NOUT=6)
      INTEGER          MMAX
      PARAMETER       (MMAX=4)
*      .. Local Scalars ..
      real            X
      INTEGER          I, IFAIL, M, N
*      .. Local Arrays ..
      real            W(MMAX)
*      .. External Subroutines ..
      EXTERNAL        S14ADF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S14ADF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      + '      X              W(1)              W(2)              W(3)              W(4)'
      WRITE (NOUT,*)
      N = 0
      M = 4
20    READ (NIN,*,END=40) X
*
      CALL S14ADF(X,N,M,W,IFAIL)
*
      WRITE (NOUT,99999) X, (W(I),I=1,M)
      GO TO 20
40    STOP
*
99999  FORMAT (1X,1P,5(e12.4,2X))
      END
```

### 9.2. Program Data

```
S14ADF Example Program Data
0.1
0.5
3.6
8.0
```

**9.3. Program Results**

## S14ADF Example Program Results

X	W(1)	W(2)	W(3)	W(4)
1.0000E-01	1.0424E+01	1.0143E+02	1.0009E+03	1.0001E+04
5.0000E-01	1.9635E+00	4.9348E+00	8.4144E+00	1.6235E+01
3.6000E+00	-1.1357E+00	3.1988E-01	5.0750E-02	1.0653E-02
8.0000E+00	-2.0156E+00	1.3314E-01	8.8498E-03	7.8321E-04

---

## S14BAF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised terms* and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S14BAF computes values for the incomplete gamma functions  $P(a,x)$  and  $Q(a,x)$ .

## 2. Specification

```

SUBROUTINE S14BAF (A, X, TOL, P, Q, IFAIL)
  INTEGER          IFAIL
  real             A, X, TOL, P, Q

```

## 3. Description

This subroutine evaluates the incomplete gamma functions in the normalised form

$$P(a,x) = \frac{1}{\Gamma(a)} \int_0^x t^{a-1} e^{-t} dt,$$

$$Q(a,x) = \frac{1}{\Gamma(a)} \int_x^\infty t^{a-1} e^{-t} dt,$$

with  $x \geq 0$  and  $a > 0$ , to a user-specified accuracy. With this normalisation,  $P(a,x) + Q(a,x) = 1$ .

Several methods are used to evaluate the functions depending on the arguments  $a$  and  $x$ , the methods including Taylor expansion for  $P(a,x)$ , Legendre's continued fraction for  $Q(a,x)$ , and power series for  $Q(a,x)$ . When both  $a$  and  $x$  are large, and  $a \simeq x$ , the uniform asymptotic expansion of Temme [3] is employed for greater efficiency – specifically, this expansion is used when  $a \geq 20$  and  $0.7a \leq x \leq 1.4a$ .

Once either of  $P$  or  $Q$  is computed, the other is obtained by subtraction from 1. In order to avoid loss of relative precision in this subtraction, the smaller of  $P$  and  $Q$  is computed first.

This routine is derived from subroutine GAM in Gautschi [2].

## 4. References

- [1] GAUTSCHI, W.  
A Computational Procedure for Incomplete Gamma Functions.  
ACM Trans. Math. Software 5, pp. 466-481, 1979.
- [2] GAUTSCHI, W.  
Algorithm 542: Incomplete Gamma Functions.  
ACM Trans. Math. Software 5, pp. 482-489, 1979.
- [3] TEMME, N.M.  
On the computation of the incomplete gamma functions for large values of the parameters.  
In: 'Algorithms for Approximation', J.C. Mason and M.G. Cox. (ed).  
Oxford University Press, 1987.

## 5. Parameters

1: A – *real*.

*Input*

*On entry:* the argument  $a$  of the functions.

*Constraint:* A > 0.0.

- 2: **X – real.** *Input*  
*On entry:* the argument  $x$  of the functions.  
*Constraint:*  $X \geq 0.0$ .
- 3: **TOL – real.** *Input*  
*On entry:* the relative accuracy required by the user in the results. If S14BAF is entered with TOL greater than 1.0 or less than *machine precision*, then the value of *machine precision* is used instead.
- 4: **P – real.** *Output*  
 5: **Q – real.** *Output*  
*On exit:* the values of the functions  $P(a,x)$  and  $Q(a,x)$  respectively.
- 6: **IFAIL – INTEGER.** *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

If, on exit, IFAIL  $\neq$  0 then S14BAF returns with a value of 0.0 for  $P$  and  $Q$ .

IFAIL = 1

On entry,  $A \leq 0.0$ .

IFAIL = 2

On entry,  $X < 0.0$ .

IFAIL = 3

Convergence of the Taylor series or Legendre continued fraction fails within 600 iterations. This error is extremely unlikely to occur; if it does, contact NAG.

## 7. Accuracy

There are rare occasions when the relative accuracy attained is somewhat less than that specified by parameter TOL. However, the error should never exceed more than one or two decimal places. Note also that there is a limit of 18 decimal places on the achievable accuracy, because constants in the routine are given to this precision.

## 8. Further Comments

The time taken for a call of S14BAF depends on the precision requested through TOL, and also varies slightly with the input arguments  $a$  and  $x$ .

## 9. Example

The following program reads values of the argument  $a$  and  $x$  from a file, evaluates the function and prints the results.



### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      S14BAF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            A, P, Q, TOL, X
      INTEGER          IFAIL
*      .. External Functions ..
      real            X02AJF
      EXTERNAL         X02AJF
*      .. External Subroutines ..
      EXTERNAL         S14BAF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S14BAF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      TOL = X02AJF()
      WRITE (NOUT,*)
      WRITE (NOUT,*) '          A          X          P          Q'
20    READ (NIN,*,END=40) A, X
      IFAIL = 0
*
      CALL S14BAF(A,X,TOL,P,Q,IFAIL)
*
      WRITE (NOUT,99999) A, X, P, Q
      GO TO 20
40    STOP
*
99999 FORMAT (1X,4F12.4)
      END

```

### 9.2. Program Data

```

S14BAF Example Program Data
2.0  3.0
7.0  1.0
0.5  99.0
20.0 21.0
21.0 20.0

```

### 9.3. Program Results

S14BAF Example Program Results

A	X	P	Q
2.0000	3.0000	0.8009	0.1991
7.0000	1.0000	0.0001	0.9999
0.5000	99.0000	1.0000	0.0000
20.0000	21.0000	0.6157	0.3843
21.0000	20.0000	0.4409	0.5591

---



## S15ABF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

S15ABF returns the value of the cumulative Normal distribution function,  $P(x)$ , via the routine name.

### 2. Specification

```

real FUNCTION S15ABF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

### 3. Description

The routine evaluates an approximate value for the cumulative Normal distribution function

$$P(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-u^2/2} du.$$

The routine is based on the fact that

$$P(x) = \frac{1}{2} \operatorname{erfc}\left(\frac{-x}{\sqrt{2}}\right)$$

and it calls S15ADF to obtain a value of  $\operatorname{erfc}$  for the appropriate argument.

### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 7.1, p. 297 and Ch. 26.2, p. 931, 1968.

### 5. Parameters

- 1: **X** – *real*. *Input*  
*On entry:* the argument  $x$  of the function.
- 2: **IFAIL** – **INTEGER**. *Input/Output*  
*On entry:* **IFAIL** must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* **IFAIL** = 0 unless the routine detects an error (see Section 6).

### 6. Error Indicators and Warnings

Errors detected by the routine:

There are no failure exits from this routine. The parameter **IFAIL** is included for consistency with other routines in this chapter.

### 7. Accuracy

Because of its close relationship with  $\operatorname{erfc}$ , the accuracy of this routine is very similar to that in S15ADF. If  $\varepsilon$  and  $\delta$  are the relative errors in result and argument, respectively, they are in principle related by

$$|\varepsilon| \simeq \left| \frac{x e^{-x^2}}{\sqrt{2\pi} P(x)} \delta \right|$$

so that the relative error in the argument,  $x$ , is amplified by a factor,  $\frac{xe^{-x^2}}{\sqrt{2\pi} P(x)}$ , in the result.

For  $x$  small and for  $x$  positive this factor is always less than one and accuracy is mainly limited by *machine precision*.

For large negative  $x$  the factor behaves like  $\sim x^2$  and hence to a certain extent relative accuracy is unavoidably lost.

However the absolute error in the result,  $E$ , is given by

$$|E| \simeq \left| \frac{xe^{-x^2}}{\sqrt{2\pi}} \delta \right|$$

so absolute accuracy can be guaranteed for all  $x$ .

## 8. Further Comments

None.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S15ABF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
real           X, Y
INTEGER         IFAIL
*      .. External Functions ..
real          S15ABF
EXTERNAL        S15ABF
*      .. Executable Statements ..
WRITE (NOUT,*) 'S15ABF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
WRITE (NOUT,*)
WRITE (NOUT,*) '      X          Y          IFAIL'
WRITE (NOUT,*)
20 READ (NIN,*,END=40) X
   IFAIL = 1
*
*      Y = S15ABF(X,IFAIL)
*
WRITE (NOUT,99999) X, Y, IFAIL
GO TO 20
40 STOP
*
99999 FORMAT (1X,1P,2E12.3,I7)
END
```

### 9.2. Program Data

```
S15ABF Example Program Data
-20.0
-1.0
0.0
1.0
2.0
20.0
```

9.3. Program Results

S15ABF Example Program Results

X	Y	IFAIL
-2.000E+01	0.000E+00	0
-1.000E+00	1.587E-01	0
0.000E+00	5.000E-01	0
1.000E+00	8.413E-01	0
2.000E+00	9.772E-01	0
2.000E+01	1.000E+00	0

---



## S15ACF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S15ACF returns the value of the complement of the cumulative normal distribution function,  $Q(x)$ , via the routine name.

## 2. Specification

```

real FUNCTION S15ACF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

## 3. Description

The routine evaluates an approximate value for the complement of the cumulative normal distribution function

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-u^2/2} du.$$

The routine is based on the fact that

$$Q(x) = \frac{1}{2} \operatorname{erfc} \left( \frac{x}{\sqrt{2}} \right)$$

and it calls S15ADF to obtain the necessary value of  $\operatorname{erfc}$ , the complementary error function.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 7.1, p. 297, Ch. 26.2, p. 931, 1968.

## 5. Parameters

- 1: **X** – *real*. *Input*  
*On entry:* the argument  $x$  of the function.
- 2: **IFAIL** – **INTEGER**. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

There are no failure exits from this routine. The parameter IFAIL is included for consistency with other routines in this chapter.

## 7. Accuracy

Because of its close relationship with  $\operatorname{erfc}$  the accuracy of this routine is very similar to that in S15ADF. If  $\varepsilon$  and  $\delta$  are the relative errors in result and argument, respectively, then in principle they are related by

$$|\varepsilon| \approx \left| \frac{x e^{-x^2/2}}{\sqrt{2\pi} Q(x)} \delta \right|.$$

For  $x$  negative or small positive this factor is always less than one and accuracy is mainly limited by *machine precision*. For large positive  $x$  we find  $\varepsilon \sim x^2 \delta$  and hence to a certain extent relative accuracy is unavoidably lost. However the absolute error in the result,  $E$ , is given by

$$|E| \approx \left| \frac{x e^{-x^2/2}}{\sqrt{2\pi}} \delta \right|$$

and since this factor is always less than one absolute accuracy can be guaranteed for all  $x$ .

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S15ACF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
real           X, Y
INTEGER         IFAIL
*      .. External Functions ..
real          S15ACF
EXTERNAL        S15ACF
*      .. Executable Statements ..
WRITE (NOUT,*) 'S15ACF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
WRITE (NOUT,*)
WRITE (NOUT,*) '      X          Y          IFAIL'
WRITE (NOUT,*)
20 READ (NIN,*,END=40) X
   IFAIL = 1
*
*      Y = S15ACF(X,IFAIL)
*
WRITE (NOUT,99999) X, Y, IFAIL
GO TO 20
40 STOP
*
99999 FORMAT (1X,1P,2E12.3,I7)
END
```

### 9.2. Program Data

```
S15ACF Example Program Data
-20.0
-1.0
 0.0
 1.0
 2.0
20.0
```



9.3. Program Results

S15ACF Example Program Results

X	Y	IFAIL
-2.000E+01	1.000E+00	0
-1.000E+00	8.413E-01	0
0.000E+00	5.000E-01	0
1.000E+00	1.587E-01	0
2.000E+00	2.275E-02	0
2.000E+01	0.000E+00	0

---



## S15ADF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S15ADF returns the value of the complementary error function,  $\text{erfc } x$ , via the routine name.

## 2. Specification

```

real FUNCTION S15ADF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

## 3. Description

The routine calculates an approximate value for the complement of the error function

$$\text{erfc } x = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-u^2} du = 1 - \text{erf } x.$$

For  $x \geq 0$ , it is based on the Chebyshev expansion

$$\text{erfc } x = e^{-x^2} y(x),$$

where  $y(x) = \sum_{r=0}^{\infty} a_r T_r(t)$  and  $t = (x-3.75)/(x+3.75)$ ,  $-1 \leq t \leq +1$ .

For  $x \geq x_{hi}$ , where there is a danger of setting underflow, the result is returned as zero.

For  $x < 0$ ,  $\text{erfc } x = 2 - e^{-x^2} y(|x|)$ .

For  $x < x_{low} < 0$ , the result is returned as 2.0 which is correct to within *machine precision*. The values of  $x_{hi}$  and  $x_{low}$  are given in the Users' Note for your implementation.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 7, p. 297, 1968.

## 5. Parameters

- 1: X – *real*. *Input*  
*On entry:* the argument  $x$  of the function.
- 2: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

There are no error exits from this routine. The parameter IFAIL is included for consistency with other routines in this chapter.

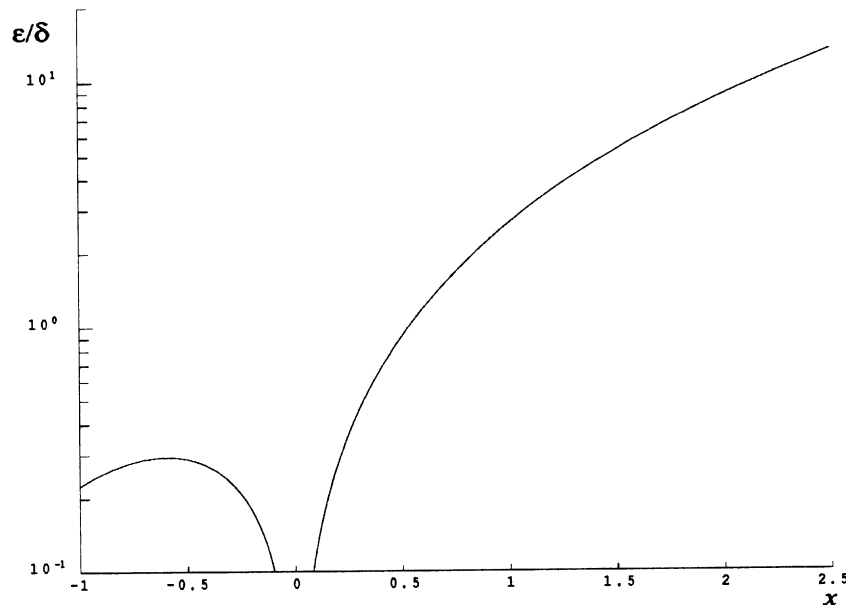
## 7. Accuracy

If  $\delta$  and  $\varepsilon$  are relative errors in the argument and result, respectively, then in principle

$$|\varepsilon| \simeq \left| \frac{2xe^{-x^2}}{\sqrt{\pi} \operatorname{erfc} x} \delta \right|.$$

That is, the relative error in the argument,  $x$ , is amplified by a factor  $\frac{2xe^{-x^2}}{\sqrt{\pi} \operatorname{erfc} x}$  in the result.

The behaviour of this factor is shown in the following graph:



It should be noted that near  $x = 0$  this factor behaves as  $\frac{2x}{\sqrt{\pi}}$  and hence the accuracy is largely

determined by the *machine precision*. Also for large negative  $x$ , where the factor is  $\sim \frac{xe^{-x^2}}{\sqrt{\pi}}$ , accuracy is mainly limited by *machine precision*. However, for large positive  $x$ , the factor becomes  $\sim 2x^2$  and to an extent relative accuracy is necessarily lost. The absolute accuracy  $E$  is given by

$$E \simeq \frac{2xe^{-x^2}}{\sqrt{\pi}} \delta$$

so absolute accuracy is guaranteed for all  $x$ .

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      S15ADF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S15ADF
      EXTERNAL         S15ADF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S15ADF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S15ADF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END

```

### 9.2. Program Data

```

S15ADF Example Program Data
      -10.0
      -1.0
      0.0
      1.0
      15.0

```

### 9.3. Program Results

S15ADF Example Program Results

X	Y	IFAIL
-1.000E+01	2.000E+00	0
-1.000E+00	1.843E+00	0
0.000E+00	1.000E+00	0
1.000E+00	1.573E-01	0
1.500E+01	0.000E+00	0

---



## S15AEF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S15AEF returns the value of the error function  $\operatorname{erf} x$ , via the routine name.

## 2. Specification

```

real FUNCTION S15AEF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

## 3. Description

Evaluates the error function,

$$\operatorname{erf} x = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

For  $|x| \leq 2$ ,

$$\operatorname{erf} x = x \sum_{r=0}' a_r T_r(t), \quad \text{where } t = \frac{1}{2}x^2 - 1.$$

For  $2 < |x| < x_{hi}$ ,

$$\operatorname{erf} x = \operatorname{sign} x \left\{ 1 - \frac{e^{-x^2}}{|x|\sqrt{\pi}} \sum_{r=0}' b_r T_r(t) \right\}, \quad \text{where } t = \frac{x-7}{x+3}.$$

For  $|x| \geq x_{hi}$ ,

$$\operatorname{erf} x = \operatorname{sign} x.$$

$x_{hi}$  is the value above which  $\operatorname{erf} x = \pm 1$  within *machine precision*. Its value is given in the Users' Note for your implementation.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 7.1, p. 297, 1968.

## 5. Parameters

1: X – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

There are no error exits from this routine. The parameter IFAIL is included for consistency with other routines in this chapter.

## 7. Accuracy

On a machine with approximately 11 significant figures the routine agrees with available tables to 10 figures and consistency checking with S15ADF of the relation

$$\operatorname{erf} x + \operatorname{erfc} x = 1.0$$

shows errors in at worst the 11th figure.

## 8. Further Comments

None.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S15AEF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S15AEF
      EXTERNAL         S15AEF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S15AEF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S15AEF(X, IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2e12.3,I7)
      END
```

### 9.2. Program Data

```
S15AEF Example Program Data
      -6.0
      -4.5
      -1.0
      1.0
      4.5
      6.0
```



### 9.3. Program Results

S15AEF Example Program Results

X	Y	IFAIL
-6.000E+00	-1.000E+00	0
-4.500E+00	-1.000E+00	0
-1.000E+00	-8.427E-01	0
1.000E+00	8.427E-01	0
4.500E+00	1.000E+00	0
6.000E+00	1.000E+00	0

---



## S17AFF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S17AFF returns the value of the Bessel function  $J_1(x)$ , via the routine name.

## 2. Specification

```
real FUNCTION S17AFF (X, IFAIL)
      INTEGER          IFAIL
      real             X
```

## 3. Description

This routine evaluates an approximation to the Bessel function of the first kind  $J_1(x)$ .

**Note:**  $J_1(-x) = -J_1(x)$ , so the approximation need only consider  $x \geq 0$ .

The routine is based on three Chebyshev expansions:

For  $0 < x \leq 8$ ,

$$J_1(x) = \frac{x}{8} \sum_{r=0}^{\infty} a_r T_r(t), \quad \text{with } t = 2\left(\frac{x}{8}\right)^2 - 1.$$

For  $x > 8$ ,

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

$$\text{where } P_1(x) = \sum_{r=0}^{\infty} b_r T_r(t),$$

$$\text{and } Q_1(x) = \frac{8}{x} \sum_{r=0}^{\infty} c_r T_r(t), \quad \text{with } t = 2\left(\frac{8}{x}\right)^2 - 1.$$

For  $x$  near zero,  $J_1(x) \simeq \frac{x}{2}$ . This approximation is used when  $x$  is sufficiently small for the result to be correct to *machine precision*.

For very large  $x$ , it becomes impossible to provide results with any reasonable accuracy (see Section 7), hence the routine fails. Such arguments contain insufficient information to determine the phase of oscillation of  $J_1(x)$ ; only the amplitude,  $\sqrt{\frac{2}{\pi|x|}}$ , can be determined and this is returned on soft failure. The range for which this occurs is roughly related to the *machine precision*; the routine will fail if  $|x| \geq 1/\text{machine precision}$  (see the Users' Note for your implementation for details).

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 358, 1968.
- [2] CLENSHAW, C.W.  
Mathematical Tables.  
Chebyshev Series for Mathematical Functions.  
National Physical Laboratory, HMSO, 5, 1962.

## 5. Parameters

1: X – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

## 2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

X is too large. On soft failure the routine returns the amplitude of the  $J_1$  oscillation,  $\sqrt{\frac{2}{\pi|x|}}$ .

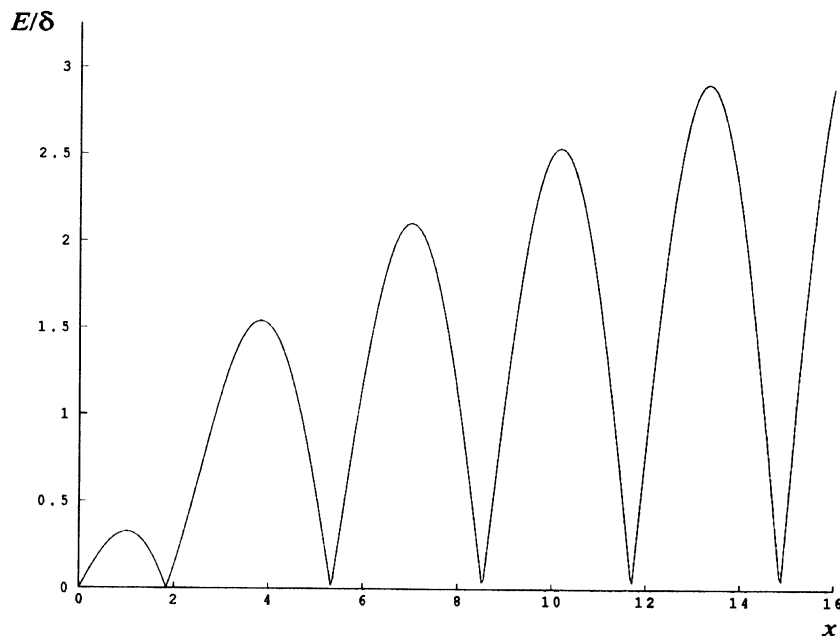
## 7. Accuracy

Let  $\delta$  be the relative error in the argument and  $E$  be the absolute error in the result. (Since  $J_1(x)$  oscillates about zero, absolute error and not relative error is significant.)

If  $\delta$  is somewhat larger than *machine precision* (e.g. if  $\delta$  is due to data errors etc.), then  $E$  and  $\delta$  are approximately related by:

$$E \approx |xJ_0(x) - J_1(x)| \delta$$

(provided  $E$  is also within machine bounds). The following graph displays the behaviour of the amplification factor  $|xJ_0(x) - J_1(x)|$ :



However, if  $\delta$  is of the same order as *machine precision*, then rounding errors could make  $E$  slightly larger than the above relation predicts.

For very large  $x$ , the above relation ceases to apply. In this region,  $J_1(x) \approx \sqrt{\frac{2}{\pi|x|}} \cos\left(x - \frac{3\pi}{4}\right)$ .

The amplitude  $\sqrt{\frac{2}{\pi|x|}}$  can be calculated with reasonable accuracy for all  $x$ , but  $\cos\left(x - \frac{3\pi}{4}\right)$  cannot. If  $x - \frac{3\pi}{4}$  is written as  $2N\pi + \theta$  where  $N$  is an integer and  $0 \leq \theta < 2\pi$ , then  $\cos\left(x - \frac{3\pi}{4}\right)$  is determined by  $\theta$  only. If  $x \geq \delta^{-1}$ ,  $\theta$  cannot be determined with any accuracy at all. Thus if  $x$  is greater than, or of the order of the reciprocal of *machine precision*, it is impossible to calculate the phase of  $J_1(x)$  and the routine must fail.

```

*      .. Executable Statements ..
      WRITE (NOUT,*) 'S15AFF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S15AFF(X, IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END

```

## 9.2. Program Data

```

S15AFF Example Program Data
      -2.0
      -0.5
      1.0
      1.5
      2.0
      5.0
      10.0

```

## 9.3. Program Results

S15AFF Example Program Results

X	Y	IFAIL
-2.000E+00	-3.013E-01	0
-5.000E-01	-4.244E-01	0
1.000E+00	5.381E-01	0
1.500E+00	4.282E-01	0
2.000E+00	3.013E-01	0
5.000E+00	1.021E-01	0
1.000E+01	5.025E-02	0

---



## S15DDF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S15DDF computes values of the function  $w(z) = e^{-z^2} \operatorname{erfc}(-iz)$ , for *complex*  $z$ .

## 2. Specification

*complex* FUNCTION S15DDF (Z, IFAIL)  
 INTEGER IFAIL  
*complex* Z

## 3. Description

This routine computes values of the function  $w(z) = e^{-z^2} \operatorname{erfc}(-iz)$ , where  $\operatorname{erfc} z$  is the complementary error function

$$\operatorname{erfc} z = \frac{2}{\sqrt{\pi}} \int_z^{\infty} e^{-t^2} dt,$$

for complex  $z$ . The method used is that in Gautschi [1] for  $z$  in the first quadrant of the complex plane, and is extended for  $z$  in other quadrants via the relations  $w(-z) = 2e^{-z^2} - w(z)$  and  $w(\bar{z}) = \overline{w(-z)}$ . Following advice in Gautschi [1], and Van Der Laan and Temme [3], the code in Gautschi [2] has been adapted to work in various precisions up to 18 decimal places. The real part of  $w(z)$  is sometimes known as the Voigt function.

## 4. References

- [1] GAUTSCHI, W.  
Efficient Computation of the Complex Error Function.  
SIAM J. Numer. Anal. 7, pp. 187-198, 1970.
- [2] GAUTSCHI, W.  
Algorithm 363: Complex Error Function.  
Comm. ACM 12, p. 635, 1969.
- [3] VAN DER LAAN, C.G. and TEMME, N.M.  
Calculation of special functions: the gamma function, the exponential integrals and error-like functions.  
CWI Tract 10, Ch. 5, Centre for Mathematics and Computer Science, Amsterdam, 1984.

## 5. Parameters

1: Z – *complex*. *Input*

*On entry*: the argument  $z$  of the function.

2: IFAIL – INTEGER. *Input/Output*

*On entry*: IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit*: IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

The real part of the result overflows, and is set to the largest safe number with the correct sign. The imaginary part of the result is meaningful.

IFAIL = 2

The imaginary part of the result overflows, and is set to the largest safe number with the correct sign. The real part of the result is meaningful.

IFAIL = 3

Both real and imaginary parts of the result overflow, and are set to the largest safe number with the correct signs.

IFAIL = 4

The result returned is accurate to less than half precision, due to the size of an intermediate result.

IFAIL = 5

The result returned has no precision, due to the size of an intermediate result, and is set to zero.

## 7. Accuracy

The accuracy of the returned result depends on the argument  $z$ . If  $z$  lies in the first or second quadrant of the complex plane (i.e.  $\text{Im } z$  is greater than or equal to zero), the result should be accurate almost to *machine precision*, except that there is a limit of about 18 decimal places on the achievable accuracy because constants in the routine are given to this precision. With such arguments, IFAIL can only return as zero.

If however  $\text{Im } z$  is less than zero, accuracy may be lost in two ways; firstly, in the evaluation of  $e^{-z^2}$ , if  $\text{Im}(-z^2)$  is large, in which case a warning will be issued through IFAIL = 4 or 5; and secondly, near the zeros of the required function, where precision is lost due to cancellation, in which case no warning is given – the result has absolute accuracy rather than relative accuracy. Note also that in this half-plane, one or both parts of the result may overflow – this is signalled through IFAIL = 1, 2 or 3.

## 8. Further Comments

The time taken for a call of S15DDF depends on the argument  $z$ , the time increasing as  $|z| \rightarrow 0.0$ .

This routine may be used to compute values of  $\text{erfc } z$  and  $\text{erf } z$  for *complex*  $z$  by the relations  $\text{erfc } z = e^{-z^2} w(iz)$ ,  $\text{erf } z = 1 - \text{erfc } z$ . (For *real* arguments, S15ADF and S15AEF should be used).

## 9. Example

The following program reads values of the argument  $z$  from a file, evaluates the function at each value of  $z$  and prints the results.



### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S15DDF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*      .. Local Scalars ..
      complex          W, Z
      INTEGER          IFAIL
*      .. External Functions ..
      complex          S15DDF
      EXTERNAL          S15DDF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S15DDF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '          Z          W'
      20 READ (NIN,*,END=40) Z
         IFAIL = 0
*
         W = S15DDF(Z,IFAIL)
*
      WRITE (NOUT,99999) Z, W
      GO TO 20
      40 STOP
*
      99999 FORMAT (1X,'(',F12.4,',',F12.4,')',
                  ('(',F12.4,',',F12.4,')'))
      END
```

### 9.2. Program Data

```
S15DDF Example Program Data
( 1.00E0,  0.00E0)    - Values for Z.
(-3.01E0,  0.75E0)
( 2.75E0, -1.52E0)
(-1.33E0, -0.54E0)
```

### 9.3. Program Results

S15DDF Example Program Results

Z		W	
( 1.0000,	0.0000)	( 0.3679,	0.6072)
( -3.0100,	0.7500)	( 0.0522,	-0.1838)
( 2.7500,	-1.5200)	( -0.1015,	0.1654)
( -1.3300,	-0.5400)	( -0.1839,	-0.7891)

---



## S17ACF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S17ACF returns the value of the Bessel function  $Y_0(x)$ , via the routine name.

## 2. Specification

```
real FUNCTION S17ACF (X, IFAIL)
INTEGER          IFAIL
real             X
```

## 3. Description

This routine evaluates an approximation to the Bessel function of the second kind  $Y_0(x)$ .

**Note:**  $Y_0(x)$  is undefined for  $x \leq 0$  and the routine will fail for such arguments.

The routine is based on four Chebyshev expansions:

For  $0 < x \leq 8$ ,

$$Y_0(x) = \frac{2}{\pi} \ln x \sum_{r=0}^{\infty} a_r T_r(t) + \sum_{r=0}^{\infty} b_r T_r(t), \quad \text{with } t = 2\left(\frac{x}{8}\right)^2 - 1$$

For  $x > 8$ ,

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

where  $P_0(x) = \sum_{r=0}^{\infty} c_r T_r(t)$ ,

and  $Q_0(x) = \frac{8}{x} \sum_{r=0}^{\infty} d_r T_r(t)$ , with  $t = 2\left(\frac{8}{x}\right)^2 - 1$ .

For  $x$  near zero,  $Y_0(x) \simeq \frac{2}{\pi} \left( \ln\left(\frac{x}{2}\right) + \gamma \right)$ , where  $\gamma$  denotes Euler's constant. This approximation is used when  $x$  is sufficiently small for the result to be correct to *machine precision*.

For very large  $x$ , it becomes impossible to provide results with any reasonable accuracy (see Section 7), hence the routine fails. Such arguments contain insufficient information to determine the phase of oscillation of  $Y_0(x)$ ; only the amplitude,  $\sqrt{\frac{2}{\pi x}}$ , can be determined and this is returned on soft failure. The range for which this occurs is roughly related to the *machine precision*: the routine will fail if  $x \gtrsim 1/\text{machine precision}$  (see the Users' Note for your implementation for details).

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 358, 1968.
- [2] CLENSHAW, C.W.  
Mathematical Tables.  
Chebyshev Series for Mathematical Functions.  
National Physical Laboratory, HMSO, 5, 1962.

## 5. Parameters

1:  $X$  – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

*Constraint:*  $X > 0.0$ .

2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

$X$  is too large. On soft failure the routine returns the amplitude of the  $Y_0$  oscillation,  $\sqrt{\frac{2}{\pi x}}$ .

IFAIL = 2

$X \leq 0.0$ ,  $Y_0$  is undefined. On soft failure the routine returns zero.

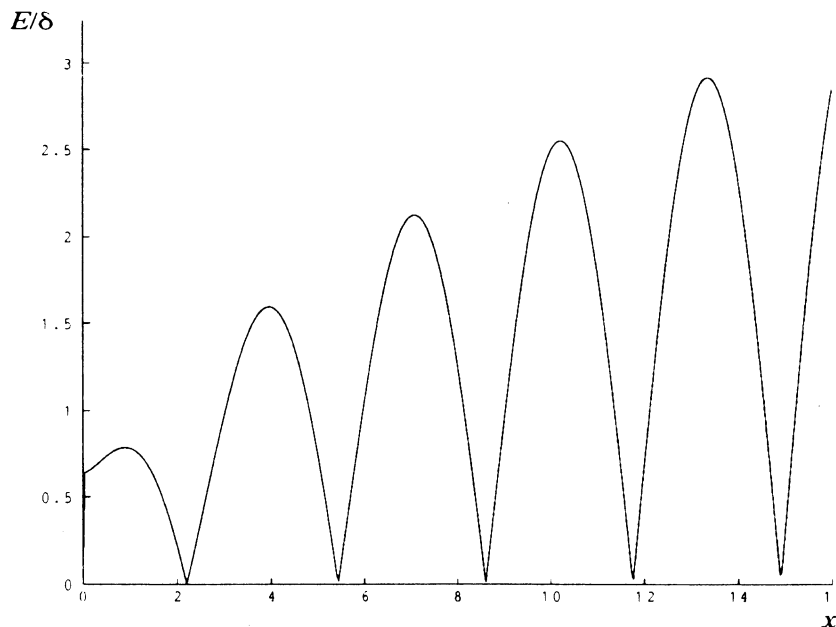
## 7. Accuracy

Let  $\delta$  be the relative error in the argument and  $E$  be the absolute error in the result. (Since  $Y_0(x)$  oscillates about zero, absolute error and not relative error is significant, except for very small  $x$ .)

If  $\delta$  is somewhat larger than the machine representation error (e.g. if  $\delta$  is due to data errors etc.), then  $E$  and  $\delta$  are approximately related by

$$E \simeq |xY_1(x)| \delta$$

(provided  $E$  is also within machine bounds). The following graph displays the behaviour of the amplification factor  $|xY_1(x)|$ :



However, if  $\delta$  is of the same order as the machine representation errors, then rounding errors could make  $E$  slightly larger than the above relation predicts.

For very small  $x$ , the errors are essentially independent of  $\delta$  and the routine should provide relative accuracy bounded by the *machine precision*.

For very large  $x$ , the above relation ceases to apply. In this region,  $Y_0(x) \approx \sqrt{\frac{2}{\pi x}} \sin\left(x - \frac{\pi}{4}\right)$ . The amplitude  $\sqrt{\frac{2}{\pi x}}$  can be calculated with reasonable accuracy for all  $x$ , but  $\sin\left(x - \frac{\pi}{4}\right)$  cannot. If  $x - \frac{\pi}{4}$  is written as  $2N\pi + \theta$  where  $N$  is an integer and  $0 \leq \theta < 2\pi$ , then  $\sin\left(x - \frac{\pi}{4}\right)$  is determined by  $\theta$  only. If  $x \geq \delta^{-1}$ ,  $\theta$  cannot be determined with any accuracy at all. Thus if  $x$  is greater than, or of the order of the inverse of *machine precision*, it is impossible to calculate the phase of  $Y_0(x)$  and the routine must fail.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S17ACF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
          INTEGER          NIN, NOUT
          PARAMETER        (NIN=5,NOUT=6)
*      .. Local Scalars ..
          real             X, Y
          INTEGER          IFAIL
*      .. External Functions ..
          real             S17ACF
          EXTERNAL         S17ACF
*      .. Executable Statements ..
          WRITE (NOUT,*) 'S17ACF Example Program Results'
*      Skip heading in data file
          READ (NIN,*)
          WRITE (NOUT,*)
          WRITE (NOUT,*) '      X              Y              IFAIL'
          WRITE (NOUT,*)
20      READ (NIN,*,END=40) X
          IFAIL = 1
*
          Y = S17ACF(X, IFAIL)
*
          WRITE (NOUT,99999) X, Y, IFAIL
          GO TO 20
40      STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
          END
```

### 9.2. Program Data

```
S17ACF Example Program Data
0.0
0.5
1.0
3.0
6.0
8.0
10.0
-1.0
1000.0
```

**9.3. Program Results**

S17ACF Example Program Results

X	Y	IFAIL
0.000E+00	0.000E+00	2
5.000E-01	-4.445E-01	0
1.000E+00	8.826E-02	0
3.000E+00	3.769E-01	0
6.000E+00	-2.882E-01	0
8.000E+00	2.235E-01	0
1.000E+01	5.567E-02	0
-1.000E+00	0.000E+00	2
1.000E+03	4.716E-03	0

---

## S17ADF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S17ADF returns the value of the Bessel Function  $Y_1(x)$ , via the routine name.

## 2. Specification

```

real FUNCTION S17ADF (X, IFAIL)
      INTEGER      IFAIL
      real         X

```

## 3. Description

This routine evaluates an approximation to the Bessel Function of the second kind  $Y_1(x)$ .

**Note:**  $Y_1(x)$  is undefined for  $x \leq 0$  and the routine will fail for such arguments.

The routine is based on four Chebyshev expansions:

For  $0 < x \leq 8$ ,

$$Y_1(x) = \frac{2}{\pi} \ln x \frac{x}{8} \sum_{r=0}' a_r T_r(t) - \frac{2}{\pi x} + \frac{x}{8} \sum_{r=0}' b_r T_r(t), \quad \text{with } t = 2\left(\frac{x}{8}\right)^2 - 1;$$

For  $x > 8$ ,

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

$$\text{where } P_1(x) = \sum_{r=0}' c_r T_r(t),$$

$$\text{and } Q_1(x) = \frac{8}{x} \sum_{r=0}' d_r T_r(t), \quad \text{with } t = 2\left(\frac{8}{x}\right)^2 - 1.$$

For  $x$  near zero,  $Y_1(x) \simeq -\frac{2}{\pi x}$ . This approximation is used when  $x$  is sufficiently small for the result to be correct to *machine precision*. For extremely small  $x$ , there is a danger of overflow in calculating  $-\frac{2}{\pi x}$  and for such arguments the routine will fail.

For very large  $x$ , it becomes impossible to provide results with any reasonable accuracy (see Section 7), hence the routine fails. Such arguments contain *insufficient information* to determine the phase of oscillation of  $Y_1(x)$ , only the amplitude,  $\sqrt{\frac{2}{\pi x}}$ , can be determined and this is returned on soft failure. The range for which this occurs is roughly related to *machine precision*; the routine will fail if  $x \geq 1/\text{machine precision}$  (see the Users' Note for your implementation for details).

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 358, 1968.
- [2] CLENSHAW, C.W.  
Mathematical Tables.  
Chebyshev series for mathematical functions.  
National Physical Laboratory, H.M.S.O., 5, 1962.

## 5. Parameters

1:  $X$  – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

*Constraint:*  $X > 0.0$ .

2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

$X$  is too large. On soft failure the routine returns the amplitude of the  $Y_1$  oscillation,  $\sqrt{\frac{2}{\pi x}}$ .

IFAIL = 2

$X \leq 0.0$ ,  $Y_1$  is undefined. On soft failure the routine returns zero.

IFAIL = 3

$X$  is too close to zero, there is a danger of overflow. On soft failure, the routine returns the value of  $Y_1(x)$  at the smallest valid argument.

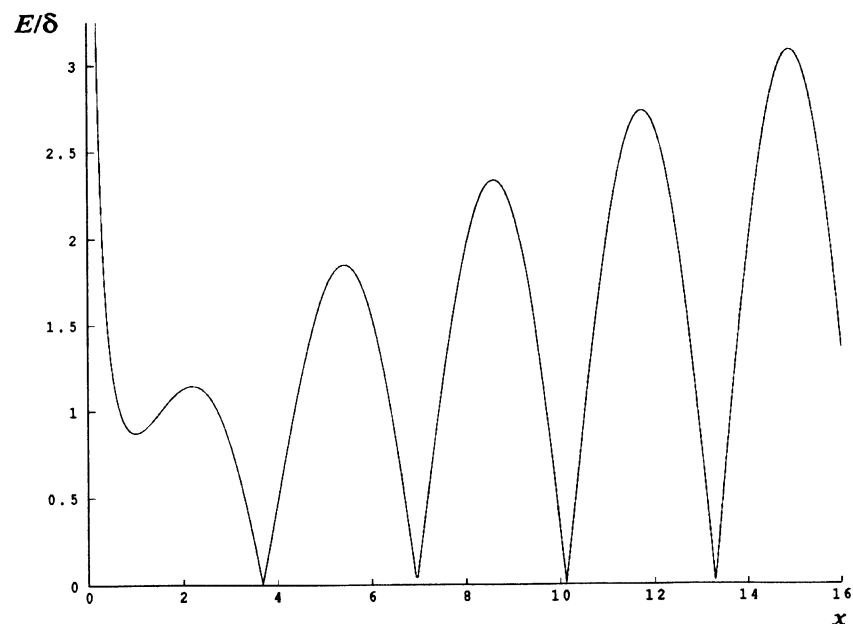
## 7. Accuracy

Let  $\delta$  be the relative error in the argument and  $E$  be the absolute error in the result. (Since  $Y_1(x)$  oscillates about zero, absolute error and not relative error is significant, except for very small  $x$ .)

If  $\delta$  is somewhat larger than the *machine precision* (e.g. if  $\delta$  is due to data errors etc.), then  $E$  and  $\delta$  are approximately related by:

$$E \simeq |xY_0(x) - Y_1(x)| \delta$$

(provided  $E$  is also within machine bounds). The following graph displays the behaviour of the amplification factor  $|xY_0(x) - Y_1(x)|$ :





However, if  $\delta$  is of the same order as *machine precision*, then rounding errors could make  $E$  slightly larger than the above relation predicts.

For very small  $x$ , absolute error becomes large, but the relative error in the result is of the same order as  $\delta$ .

For very large  $x$ , the above relation ceases to apply. In this region,  $Y_1(x) \approx \frac{2}{\pi x} \sin\left(x - \frac{3\pi}{4}\right)$ . The amplitude  $\frac{2}{\pi x}$  can be calculated with reasonable accuracy for all  $x$ , but  $\sin\left(x - \frac{3\pi}{4}\right)$  cannot. If  $x - \frac{3\pi}{4}$  is written as  $2N\pi + \theta$  where  $N$  is an integer and  $0 \leq \theta < 2\pi$ , then  $\sin\left(x - \frac{3\pi}{4}\right)$  is determined by  $\theta$  only. If  $x > \delta^{-1}$ ,  $\theta$  cannot be determined with any accuracy at all. Thus if  $x$  is greater than, or of the order of, the inverse of the *machine precision*, it is impossible to calculate the phase of  $Y_1(x)$  and the routine must fail.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S17ADF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
real           X, Y
INTEGER         IFAIL
*      .. External Functions ..
real          S17ADF
EXTERNAL        S17ADF
*      .. Executable Statements ..
WRITE (NOUT,*) 'S17ADF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
WRITE (NOUT,*)
WRITE (NOUT,*) '      X          Y          IFAIL'
WRITE (NOUT,*)
20 READ (NIN,*,END=40) X
   IFAIL = 1
*
*      Y = S17ADF(X,IFAIL)
*
WRITE (NOUT,99999) X, Y, IFAIL
GO TO 20
40 STOP
*
99999 FORMAT (1X,1P,2E12.3,17)
END
```

## 9.2. Program Data

```
S17ADF Example Program Data
      0.0
      0.5
      1.0
      3.0
      6.0
      8.0
     10.0
     -1.0
    1000.0
```

## 9.3. Program Results

```
S17ADF Example Program Results
```

X	Y	IFAIL
0.000E+00	0.000E+00	2
5.000E-01	-1.471E+00	0
1.000E+00	-7.812E-01	0
3.000E+00	3.247E-01	0
6.000E+00	-1.750E-01	0
8.000E+00	-1.581E-01	0
1.000E+01	2.490E-01	0
-1.000E+00	0.000E+00	2
1.000E+03	-2.478E-02	0

---

## S17AEF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S17AEF returns the value of the Bessel Function  $J_0(x)$ , via the routine name.

## 2. Specification

```

real FUNCTION S17AEF (X, IFAIL)
      INTEGER          IFAIL
      real!           X

```

## 3. Description

This routine evaluates an approximation to the Bessel Function of the first kind  $J_0(x)$ .

**Note:**  $J_0(-x) = J_0(x)$ , so the approximation need only consider  $x \geq 0$ .

The routine is based on three Chebyshev expansions:

For  $0 < x \leq 8$ ,

$$J_0(x) = \sum_{r=0}' a_r T_r(t), \quad \text{with } t = 2\left(\frac{x}{8}\right)^2 - 1.$$

For  $x > 8$ ,

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

$$\text{where } P_0(x) = \sum_{r=0}' b_r T_r(t),$$

$$\text{and } Q_0(x) = \frac{8}{x} \sum_{r=0}' c_r T_r(t), \quad \text{with } t = 2\left(\frac{8}{x}\right)^2 - 1.$$

For  $x$  near zero,  $J_0(x) \simeq 1$ . This approximation is used when  $x$  is sufficiently small for the result to be correct to *machine precision*.

For very large  $x$ , it becomes impossible to provide results with any reasonable accuracy (see Section 7), hence the routine fails. Such arguments contain insufficient information to determine the phase of oscillation of  $J_0(x)$ ; only the amplitude,  $\sqrt{\frac{2}{\pi|x|}}$ , can be determined and this is returned on soft failure. The range for which this occurs is roughly related to the *machine precision*; the routine will fail if  $|x| \geq 1/\text{machine precision}$  (see the Users' Note for your implementation).

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 358, 1968.
- [2] CLENSHAW, C.W.  
Mathematical Tables.  
Chebyshev Series for Mathematical Functions.  
National Physical Laboratory, HMSO, 5, 1962.

## 5. Parameters

1:  $X$  – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

$X$  is too large. On soft failure the routine returns the amplitude of the  $J_0$  oscillation,  $\sqrt{\frac{2}{\pi|x|}}$ .

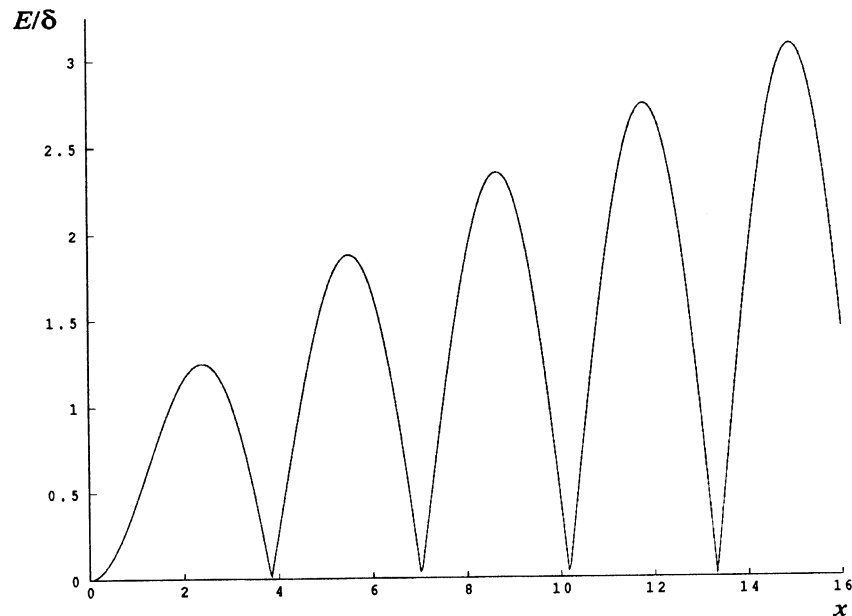
## 7. Accuracy

Let  $\delta$  be the relative error in the argument and  $E$  be the absolute error in the result. (Since  $J_0(x)$  oscillates about zero, absolute error and not relative error is significant.)

If  $\delta$  is somewhat larger than the *machine precision* (e.g. if  $\delta$  is due to data errors etc.), then  $E$  and  $\delta$  are approximately related by:

$$E \approx |xJ_1(x)| \delta$$

(provided  $E$  is also within machine bounds). The following graph displays the behaviour of the amplification factor  $|xJ_1(x)|$ :



However, if  $\delta$  is of the same order as *machine precision*, then rounding errors could make  $E$  slightly larger than the above relation predicts.

For very large  $x$ , the above relation ceases to apply. In this region,  $J_0(x) \approx \sqrt{\frac{2}{\pi|x|}} \cos\left(x - \frac{\pi}{4}\right)$ .

The amplitude  $\sqrt{\frac{2}{\pi|x|}}$  can be calculated with reasonable accuracy for all  $x$ , but  $\cos\left(x - \frac{\pi}{4}\right)$  cannot. If  $x - \frac{\pi}{4}$  is written as  $2N\pi + \theta$  where  $N$  is an integer and  $0 \leq \theta < 2\pi$ , then  $\cos\left(x - \frac{\pi}{4}\right)$  is

determined by  $\theta$  only. If  $x \geq \delta^{-1}$ ,  $\theta$  cannot be determined with any accuracy at all. Thus if  $x$  is greater than, or of the order of, the inverse of the *machine precision*, it is impossible to calculate the phase of  $J_0(x)$  and the routine must fail.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S17AEF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S17AEF
      EXTERNAL         S17AEF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S17AEF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S17AEF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END
```

### 9.2. Program Data

```
S17AEF Example Program Data
      0.0
      0.5
      1.0
      3.0
      6.0
      8.0
      10.0
      -1.0
      1000.0
```

**9.3. Program Results**

S17AEF Example Program Results

X	Y	IFAIL
0.000E+00	1.000E+00	0
5.000E-01	9.385E-01	0
1.000E+00	7.652E-01	0
3.000E+00	-2.601E-01	0
6.000E+00	1.506E-01	0
8.000E+00	1.717E-01	0
1.000E+01	-2.459E-01	0
-1.000E+00	7.652E-01	0
1.000E+03	2.479E-02	0

---

## S17AFF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S17AFF returns the value of the Bessel Function  $J_1(x)$ , via the routine name.

## 2. Specification

```
real FUNCTION S17AFF (X, IFAIL)
  INTEGER          IFAIL
  real            X
```

## 3. Description

This routine evaluates an approximation to the Bessel Function of the first kind  $J_1(x)$ .

Note:  $J_1(-x) = -J_1(x)$ , so the approximation need only consider  $x \geq 0$ .

The routine is based on three Chebyshev expansions:

For  $0 < x \leq 8$ ,

$$J_1(x) = \frac{x}{8} \sum_{r=0}^{\infty} a_r T_r(t), \quad \text{with } t = 2\left(\frac{x}{8}\right)^2 - 1.$$

For  $x > 8$ ,

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

$$\text{where } P_1(x) = \sum_{r=0}^{\infty} b_r T_r(t),$$

$$\text{and } Q_1(x) = \frac{8}{x} \sum_{r=0}^{\infty} c_r T_r(t), \quad \text{with } t = 2\left(\frac{8}{x}\right)^2 - 1.$$

For  $x$  near zero,  $J_1(x) \approx \frac{x}{2}$ . This approximation is used when  $x$  is sufficiently small for the result to be correct to *machine precision*.

For very large  $x$ , it becomes impossible to provide results with any reasonable accuracy (see Section 7), hence the routine fails. Such arguments contain *insufficient information* to determine the phase of oscillation of  $J_1(x)$ ; only the amplitude,  $\sqrt{\frac{2}{\pi|x|}}$ , can be determined and this is returned on soft failure. The range for which this occurs is roughly related to the *machine precision*; the routine will fail if  $|x| \geq 1/\textit{machine precision}$  (see the Users' Note for your implementation for details).

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 358, 1968.
- [2] CLENSHAW, C.W.  
Mathematical Tables.  
Chebyshev Series for Mathematical Functions.  
National Physical Laboratory, HMSO, 5, 1962.

## 5. Parameters

1: X – *real*.

*Input*

*On entry*: the argument  $x$  of the function.

## 2: IFAIL – INTEGER.

Input/Output

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

X is too large. On soft failure the routine returns the amplitude of the  $J_1$  oscillation,  $\sqrt{\frac{2}{\pi|x|}}$ .

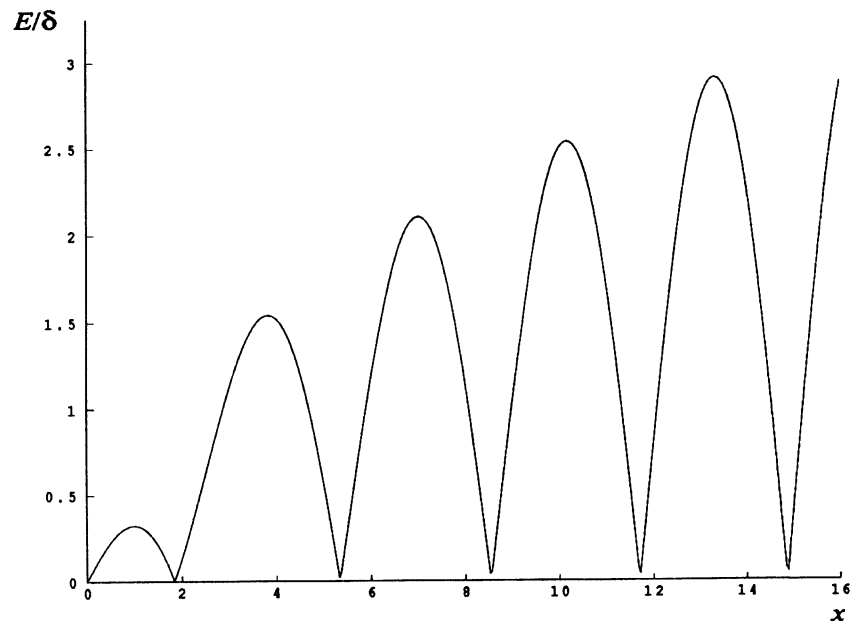
## 7. Accuracy

Let  $\delta$  be the relative error in the argument and  $E$  be the absolute error in the result. (Since  $J_1(x)$  oscillates about zero, absolute error and not relative error is significant.)

If  $\delta$  is somewhat larger than *machine precision* (e.g. if  $\delta$  is due to data errors etc.), then  $E$  and  $\delta$  are approximately related by:

$$E \simeq |xJ_0(x) - J_1(x)| \delta$$

(provided  $E$  is also within machine bounds). The following graph displays the behaviour of the amplification factor  $|xJ_0(x) - J_1(x)|$ :



However, if  $\delta$  is of the same order as *machine precision*, then rounding errors could make  $E$  slightly larger than the above relation predicts.

For very large  $x$ , the above relation ceases to apply. In this region,  $J_1(x) \simeq \sqrt{\frac{2}{\pi|x|}} \cos\left(x - \frac{3\pi}{4}\right)$ .

The amplitude  $\sqrt{\frac{2}{\pi|x|}}$  can be calculated with reasonable accuracy for all  $x$ , but  $\cos\left(x - \frac{3\pi}{4}\right)$  cannot. If  $x - \frac{3\pi}{4}$  is written as  $2N\pi + \theta$  where  $N$  is an integer and  $0 \leq \theta < 2\pi$ , then

$\cos\left(x - \frac{3\pi}{4}\right)$  is determined by  $\theta$  only. If  $x \geq \delta^{-1}$ ,  $\theta$  cannot be determined with any accuracy at all. Thus if  $x$  is greater than, or of the order of, *machine precision*, it is impossible to calculate the phase of  $J_1(x)$  and the routine must fail.



## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S17AFF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
real           X, Y
INTEGER         IFAIL
*      .. External Functions ..
real          S17AFF
EXTERNAL        S17AFF
*      .. Executable Statements ..
WRITE (NOUT,*) 'S17AFF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
WRITE (NOUT,*)
WRITE (NOUT,*) '      X          Y          IFAIL'
WRITE (NOUT,*)
20 READ (NIN,*,END=40) X
   IFAIL = 1
*
   Y = S17AFF(X,IFAIL)
*
   WRITE (NOUT,99999) X, Y, IFAIL
   GO TO 20
40 STOP
*
99999 FORMAT (1X,1P,2E12.3,I7)
END
```

### 9.2. Program Data

```
S17AFF Example Program Data
0.0
0.5
1.0
3.0
6.0
8.0
10.0
-1.0
1000.0
```

**9.3. Program Results**

## S17AFF Example Program Results

X	Y	IFAIL
0.000E+00	0.000E+00	0
5.000E-01	2.423E-01	0
1.000E+00	4.401E-01	0
3.000E+00	3.391E-01	0
6.000E+00	-2.767E-01	0
8.000E+00	2.346E-01	0
1.000E+01	4.347E-02	0
-1.000E+00	-4.401E-01	0
1.000E+03	4.728E-03	0

---

## S17AGF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S17AGF returns a value for the Airy function,  $\text{Ai}(x)$ , via the routine name.

## 2. Specification

```

real FUNCTION S17AGF (X, IFAIL)
      INTEGER      IFAIL
      real         X

```

## 3. Description

This routine evaluates an approximation to the Airy function,  $\text{Ai}(x)$ . It is based on a number of Chebyshev expansions:

For  $x < -5$ ,

$$\text{Ai}(x) = \frac{a(t) \sin z - b(t) \cos z}{(-x)^{\frac{1}{2}}}$$

where  $z = \frac{\pi}{4} + \frac{2}{3}\sqrt{-x^3}$ , and  $a(t)$  and  $b(t)$  are expansions in the variable  $t = -2\left(\frac{5}{x}\right)^3 - 1$ .

For  $-5 \leq x \leq 0$ ,

$$\text{Ai}(x) = f(t) - x g(t),$$

where  $f$  and  $g$  are expansions in  $t = -2\left(\frac{x}{5}\right)^3 - 1$ .

For  $0 < x < 4.5$ ,

$$\text{Ai}(x) = e^{-3x/2} y(t),$$

where  $y$  is an expansion in  $t = 4x/9 - 1$ .

For  $4.5 \leq x < 9$ ,

$$\text{Ai}(x) = e^{-5x/2} u(t),$$

where  $u$  is an expansion in  $t = 4x/9 - 3$ .

For  $x \geq 9$ ,

$$\text{Ai}(x) = \frac{e^{-z} v(t)}{x^{\frac{1}{2}}},$$

where  $z = \frac{2}{3}\sqrt{x^3}$  and  $v$  is an expansion in  $t = 2\left(\frac{18}{z}\right) - 1$ .

For  $|x| < \text{the machine precision}$ , the result is set directly to  $\text{Ai}(0)$ . This both saves time and guards against underflow in intermediate calculations.

For large negative arguments, it becomes impossible to calculate the phase of the oscillatory function with any precision and so the routine must fail. This occurs if  $x < -\left(\frac{3}{2\varepsilon}\right)^{\frac{3}{2}}$ , where  $\varepsilon$  is the *machine precision*.

For large positive arguments, where  $\text{Ai}$  decays in an essentially exponential manner, there is a danger of underflow so the routine must fail.

#### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 10.4, p. 446., 1968.

#### 5. Parameters

1:  $X$  – *real*. *Input*  
*On entry:* the argument  $x$  of the function.

2: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

#### 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

$X$  is too large and positive. On soft failure, the routine returns zero.

IFAIL = 2

$X$  is too large and negative. On soft failure, the routine returns zero.

#### 7. Accuracy

For negative arguments the function is oscillatory and hence absolute error is the appropriate measure. In the positive region the function is essentially exponential-like and here relative error is appropriate. The absolute error,  $E$ , and the relative error,  $\varepsilon$ , are related in principle to the relative error in the argument,  $\delta$ , by

$$E \approx |x \operatorname{Ai}'(x)| \delta, \quad \varepsilon \approx \left| \frac{x \operatorname{Ai}'(x)}{\operatorname{Ai}(x)} \right| \delta.$$

In practice, approximate equality is the best that can be expected. When  $\delta$ ,  $\varepsilon$  or  $E$  is of the order of the *machine precision*, the errors in the result will be somewhat larger.

For small  $x$ , errors are strongly damped by the function and hence will be bounded by the *machine precision*.

For moderate negative  $x$ , the error behaviour is oscillatory but the amplitude of the error grows like

$$\text{amplitude} \left( \frac{E}{\delta} \right) \sim \frac{|x|^{\frac{1}{2}}}{\sqrt{\pi}}.$$

However the phase error will be growing roughly like  $\frac{2}{3}\sqrt{|x|^3}$  and hence all accuracy will be lost for large negative arguments due to the impossibility of calculating sin and cos to any accuracy if  $\frac{2}{3}\sqrt{|x|^3} > \frac{1}{\delta}$ .

For large positive arguments, the relative error amplification is considerable:

$$\frac{\varepsilon}{\delta} \sim \sqrt{x^3}.$$

This means a loss of roughly two decimal places accuracy for arguments in the region of 20. However very large arguments are not possible due to the danger of setting underflow and so the errors are limited in practice.

## 8. Further Comments

None.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S17AGF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S17AGF
      EXTERNAL         S17AGF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S17AGF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*) '          X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S17AGF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2e12.3,I7)
      END
```

### 9.2. Program Data

```
S17AGF Example Program Data
      -10.0
      -1.0
      0.0
      1.0
      5.0
      10.0
      20.0
```

### 9.3. Program Results

S17AGF Example Program Results

X	Y	IFAIL
-1.000E+01	4.024E-02	0
-1.000E+00	5.356E-01	0
0.000E+00	3.550E-01	0
1.000E+00	1.353E-01	0
5.000E+00	1.083E-04	0
1.000E+01	1.105E-10	0
2.000E+01	1.692E-27	0



## S17AHF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S17AHF returns a value of the Airy function,  $\text{Bi}(x)$ , via the routine name.

## 2. Specification

```

real FUNCTION S17AHF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

## 3. Description

This routine evaluates an approximation to the Airy function  $\text{Bi}(x)$ . It is based on a number of Chebyshev expansions.

For  $x < -5$ ,

$$\text{Bi}(x) = \frac{a(t) \cos z + b(t) \sin z}{(-x)^{\frac{1}{3}}},$$

where  $z = \frac{\pi}{4} + \frac{2}{3}\sqrt{-x^3}$  and  $a(t)$  and  $b(t)$  are expansions in the variable  $t = -2\left(\frac{5}{x}\right)^3 - 1$ .

For  $-5 \leq x \leq 0$ ,

$$\text{Bi}(x) = \sqrt{3} (f(t) + xg(t)),$$

where  $f$  and  $g$  are expansions in  $t = -2\left(\frac{x}{5}\right)^3 - 1$ .

For  $0 < x < 4.5$ ,

$$\text{Bi}(x) = e^{11x/8}y(t),$$

where  $y$  is an expansion in  $t = 4x/9 - 1$ .

For  $4.5 \leq x \leq 9$ ,

$$\text{Bi}(x) = e^{5x/2}v(t),$$

where  $v$  is an expansion in  $t = 4x/9 - 3$ .

For  $x \geq 9$ ,

$$\text{Bi}(x) = \frac{e^z u(t)}{x^{\frac{1}{3}}},$$

where  $z = \frac{2}{3}\sqrt{x^3}$  and  $u$  is an expansion in  $t = 2\left(\frac{18}{z}\right) - 1$ .

For  $|x| <$  the *machine precision*, the result is set directly to  $\text{Bi}(0)$ . This both saves time and avoids possible intermediate underflows.

For large negative arguments, it becomes impossible to calculate the phase of the oscillating function with any accuracy so the routine must fail. This occurs if  $x < -\left(\frac{3}{2\varepsilon}\right)^{\frac{3}{2}}$ , where  $\varepsilon$  is the *machine precision*.

For large positive arguments, there is a danger of causing overflow since  $\text{Bi}$  grows in an essentially exponential manner, so the routine must fail.

#### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 10.4, p. 446, 1968.

#### 5. Parameters

1: X – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

#### 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

X is too large and positive. On soft failure, the routine returns zero.

IFAIL = 2

X is too large and negative. On soft failure, the routine returns zero.

#### 7. Accuracy

For negative arguments the function is oscillatory and hence absolute error is the appropriate measure. In the positive region the function is essentially exponential-like and here relative error is appropriate. The absolute error,  $E$ , and the relative error,  $\varepsilon$ , are related in principle to the relative error in the argument,  $\delta$ , by

$$E \simeq |xBi'(x)|\delta, \quad \varepsilon \simeq \left| \frac{xBi'(x)}{Bi(x)} \right| \delta.$$

In practice, approximate equality is the best that can be expected. When  $\delta$ ,  $\varepsilon$  or  $E$  is of the order of the *machine precision*, the errors in the result will be somewhat larger.

For small  $x$ , errors are strongly damped and hence will be bounded essentially by the *machine precision*.

For moderate to large negative  $x$ , the error behaviour is clearly oscillatory but the amplitude of the error grows like

$$\text{amplitude} \left( \frac{E}{\delta} \right) \sim \frac{|x|^{\frac{1}{2}}}{\sqrt{\pi}}.$$

However the phase error will be growing roughly as  $\frac{2}{3}\sqrt{|x|^3}$  and hence all accuracy will be lost for large negative arguments. This is due to the impossibility of calculating sin and cos to any accuracy if  $\frac{2}{3}\sqrt{|x|^3} > \frac{1}{\delta}$ .

For large positive arguments, the relative error amplification is considerable:

$$\frac{\varepsilon}{\delta} \sim \sqrt{x^3}.$$

This means a loss of roughly two decimal places accuracy for arguments in the region of 20. However very large arguments are not possible due to the danger of causing overflow and errors are therefore limited in practice.



## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S17AHF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S17AHF
      EXTERNAL         S17AHF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S17AHF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S17AHF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2e12.3,I7)
      END
```

### 9.2. Program Data

```
S17AHF Example Program Data
-10.0
-1.0
0.0
1.0
5.0
10.0
20.0
```

**9.3. Program Results****S17AHF Example Program Results**

X	Y	IFAIL
-1.000E+01	-3.147E-01	0
-1.000E+00	1.040E-01	0
0.000E+00	6.149E-01	0
1.000E+00	1.207E+00	0
5.000E+00	6.578E+02	0
1.000E+01	4.556E+08	0
2.000E+01	2.104E+25	0

---

## S17AJF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S17AJF returns a value of the derivative of the Airy function  $\text{Ai}(x)$ , via the routine name.

## 2. Specification

```

real FUNCTION S17AJF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

## 3. Description

This routine evaluates an approximation to the derivative of the Airy function  $\text{Ai}(x)$ . It is based on a number of Chebyshev expansions.

For  $x < -5$ ,

$$\text{Ai}'(x) = \sqrt[4]{-x} \left[ a(t) \cos z + \frac{b(t)}{\zeta} \sin z \right],$$

where  $z = \frac{\pi}{4} + \zeta$ ,  $\zeta = \frac{2}{3}\sqrt{-x^3}$  and  $a(t)$  and  $b(t)$  are expansions in variable  $t = -2\left(\frac{5}{x}\right)^3 - 1$ .

For  $-5 \leq x \leq 0$ ,

$$\text{Ai}'(x) = x^2 f(t) - g(t),$$

where  $f$  and  $g$  are expansions in  $t = -2\left(\frac{x}{5}\right)^3 - 1$ .

For  $0 < x < 4.5$ ,

$$\text{Ai}'(x) = e^{-11x/8} y(t),$$

where  $y(t)$  is an expansion in  $t = 4\left(\frac{x}{9}\right) - 1$ .

For  $4.5 \leq x < 9$ ,

$$\text{Ai}'(x) = e^{-5x/2} v(t),$$

where  $v(t)$  is an expansion in  $t = 4\left(\frac{x}{9}\right) - 3$ .

For  $x \geq 9$ ,

$$\text{Ai}'(x) = \sqrt[4]{-x} e^{-z} u(t),$$

where  $z = \frac{2}{3}\sqrt{x^3}$  and  $u(t)$  is an expansion in  $t = 2\left(\frac{18}{z}\right) - 1$ .

For  $|x| <$  the square of the *machine precision*, the result is set directly to  $\text{Ai}'(0)$ . This both saves time and avoids possible intermediate underflows.

For large negative arguments, it becomes impossible to calculate a result for the oscillating function with any accuracy and so the routine must fail. This occurs for  $x < -\left(\frac{\sqrt{\pi}}{\varepsilon}\right)^{4/7}$ , where  $\varepsilon$  is the *machine precision*.

For large positive arguments, where  $\text{Ai}'$  decays in an essentially exponential manner, there is a danger of underflow so the routine must fail.

#### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 10.4, p. 446, 1968.

#### 5. Parameters

- 1: X – *real*. *Input*  
*On entry:* the argument  $x$  of the function.
- 2: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

#### 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

X is too large and positive. On soft failure, the routine returns zero.

IFAIL = 2

X is too large and negative. On soft failure, the routine returns zero.

#### 7. Accuracy

For negative arguments the function is oscillatory and hence absolute error is the appropriate measure. In the positive region the function is essentially exponential in character and here relative error is needed. The absolute error,  $E$ , and the relative error,  $\varepsilon$ , are related in principle to the relative error in the argument,  $\delta$ , by

$$E \simeq |x^2 \text{Ai}(x)| \delta \quad \varepsilon \simeq \left| \frac{x^2 \text{Ai}(x)}{\text{Ai}'(x)} \right| \delta.$$

In practice, approximate equality is the best that can be expected. When  $\delta$ ,  $\varepsilon$  or  $E$  is of the order of the *machine precision*, the errors in the result will be somewhat larger.

For small  $x$ , positive or negative, errors are strongly attenuated by the function and hence will be roughly bounded by the *machine precision*.

For moderate to large negative  $x$ , the error, like the function, is oscillatory; however the amplitude of the error grows like

$$\frac{|x|^{7/4}}{\sqrt{\pi}}.$$

Therefore it becomes impossible to calculate the function with any accuracy if  $|x|^{7/4} > \frac{\sqrt{\pi}}{\delta}$ .

For large positive  $x$ , the relative error amplification is considerable:

$$\frac{\varepsilon}{\delta} \simeq \sqrt{x^3}.$$

However, very large arguments are not possible due to the danger of underflow. Thus in practice error amplification is limited.

#### 8. Further Comments

None.

#### 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      S17AJF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real             X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real             S17AJF
      EXTERNAL         S17AJF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S17AJF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S17AJF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END

```

## 9.2. Program Data

```

S17AJF Example Program Data
      -10.0
      -1.0
      0.0
      1.0
      5.0
      10.0
      20.0

```

## 9.3. Program Results

S17AJF Example Program Results

X	Y	IFAIL
-1.000E+01	9.963E-01	0
-1.000E+00	-1.016E-02	0
0.000E+00	-2.588E-01	0
1.000E+00	-1.591E-01	0
5.000E+00	-2.474E-04	0
1.000E+01	-3.521E-10	0
2.000E+01	-7.586E-27	0

---



## S17AKF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

S17AKF returns a value for the derivative of the Airy function  $\text{Bi}(x)$ , via the routine name.

### 2. Specification

```

real FUNCTION S17AKF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

### 3. Description

This routine calculates an approximate value for the derivative of the Airy function  $\text{Bi}(x)$ . It is based on a number of Chebyshev expansions.

For  $x < -5$ ,

$$\text{Bi}'(x) = \sqrt[4]{-x} \left[ -a(t) \sin z + \frac{b(t)}{\zeta} \cos z \right],$$

where  $z = \frac{\pi}{4} + \zeta$ ,  $\zeta = \frac{2}{3}\sqrt{-x^3}$  and  $a(t)$  and  $b(t)$  are expansions in the variable  $t = -2\left(\frac{5}{x}\right)^3 - 1$ .

For  $-5 \leq x \leq 0$ ,

$$\text{Bi}'(x) = \sqrt{3}(x^2 f(t) + g(t)),$$

where  $f$  and  $g$  are expansions in  $t = -2\left(\frac{x}{5}\right)^3 - 1$ .

For  $0 < x < 4.5$ ,

$$\text{Bi}'(x) = e^{3x/2} y(t),$$

where  $y(t)$  is an expansion in  $t = 4x/9 - 1$ .

For  $4.5 \leq x < 9$ ,

$$\text{Bi}'(x) = e^{21x/8} u(t),$$

where  $u(t)$  is an expansion in  $t = 4x/9 - 3$ .

For  $x \geq 9$ ,

$$\text{Bi}'(x) = \sqrt[4]{x} e^z v(t),$$

where  $z = \frac{2}{3}\sqrt{x^3}$  and  $v(t)$  is an expansion in  $t = 2\left(\frac{18}{z}\right) - 1$ .

For  $|x| <$  the square of the *machine precision*, the result is set directly to  $\text{Bi}'(0)$ . This saves time and avoids possible underflows in calculation.

For large negative arguments, it becomes impossible to calculate a result for the oscillating function with any accuracy so the routine must fail. This occurs for  $x < -\left(\frac{\sqrt{\pi}}{\varepsilon}\right)^\dagger$ , where  $\varepsilon$  is the *machine precision*.

For large positive arguments, where  $\text{Bi}'$  grows in an essentially exponential manner, there is a danger of overflow so the routine must fail.

#### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 10.4, p. 446, 1968.

#### 5. Parameters

- 1: X – *real*. *Input*  
*On entry:* the argument  $x$  of the function.
- 2: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

#### 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

X is too large and positive. On soft failure the routine returns zero.

IFAIL = 2

X is too large and negative. On soft failure the routine returns zero.

#### 7. Accuracy

For negative arguments the function is oscillatory and hence absolute error is appropriate. In the positive region the function has essentially exponential behaviour and hence relative error is needed. The absolute error,  $E$ , and the relative error  $\varepsilon$ , are related in principle to the relative error in the argument  $\delta$ , by

$$E \simeq |x^2 \text{Bi}(x)| \delta \quad \varepsilon \simeq \left| \frac{x^2 \text{Bi}(x)}{\text{Bi}'(x)} \right| \delta.$$

In practice, approximate equality is the best that can be expected. When  $\delta$ ,  $\varepsilon$  or  $E$  is of the order of the *machine precision*, the errors in the result will be somewhat larger.

For small  $x$ , positive or negative, errors are strongly attenuated by the function and hence will effectively be bounded by the *machine precision*.

For moderate to large negative  $x$ , the error is, like the function, oscillatory. However, the amplitude of the absolute error grows like  $\frac{|x|^{\frac{1}{2}}}{\sqrt{\pi}}$ . Therefore it becomes impossible to calculate the function with any accuracy if  $|x|^{\frac{1}{2}} > \frac{\sqrt{\pi}}{\delta}$ .

For large positive  $x$ , the relative error amplification is considerable:  $\frac{\varepsilon}{\delta} \sim \sqrt{x^3}$ . However, very large arguments are not possible due to the danger of overflow. Thus in practice the actual amplification that occurs is limited.

#### 8. Further Comments

None.

#### 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.



## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      S17AKF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S17AKF
      EXTERNAL         S17AKF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S17AKF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S17AKF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END

```

## 9.2. Program Data

```

S17AKF Example Program Data
      -10.0
      -1.0
      0.0
      1.0
      5.0
      10.0
      20.0

```

## 9.3. Program Results

S17AKF Example Program Results

X	Y	IFAIL
-1.000E+01	1.194E-01	0
-1.000E+00	5.924E-01	0
0.000E+00	4.483E-01	0
1.000E+00	9.324E-01	0
5.000E+00	1.436E+03	0
1.000E+01	1.429E+09	0
2.000E+01	9.382E+25	0

---



## S17DCF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised terms* and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S17DCF returns a sequence of values for the Bessel functions  $Y_{\nu+n}(z)$  for complex  $z$ , non-negative  $\nu$  and  $n = 0, 1, \dots, N-1$ , with an option for exponential scaling.

## 2. Specification

```

SUBROUTINE S17DCF (FNU, Z, N, SCALE, CY, NZ, CWRK, IFAIL)
  INTEGER          N, NZ, IFAIL
  real            FNU
  complex        Z, CY(N), CWRK(N)
  CHARACTER*1     SCALE

```

## 3. Description

This subroutine evaluates a sequence of values for the Bessel function  $Y_\nu(z)$ , where  $z$  is complex,  $-\pi < \arg z \leq \pi$ , and  $\nu$  is the real, non-negative order. The  $N$ -member sequence is generated for orders  $\nu, \nu+1, \dots, \nu+N-1$ . Optionally, the sequence is scaled by the factor  $e^{-|\operatorname{Im} z|}$ .

**Note:** although the routine may not be called with  $\nu$  less than zero, for negative orders the formula  $Y_{-\nu}(z) = Y_\nu(z)\cos(\pi\nu) + J_\nu(z)\sin(\pi\nu)$  may be used (for the Bessel function  $J_\nu(z)$ , see S17DEF).

The routine is derived from the routine CBESY in Amos [2]. It is based on the relation

$Y_\nu(z) = \frac{H_\nu^{(1)}(z) - H_\nu^{(2)}(z)}{2i}$ , where  $H_\nu^{(1)}(z)$  and  $H_\nu^{(2)}(z)$  are the Hankel functions of the first and second kinds respectively (see S17DLF).

When  $N$  is greater than 1, extra values of  $Y_\nu(z)$  are computed using recurrence relations.

For very large  $|z|$  or  $(\nu+N-1)$ , argument reduction will cause total loss of accuracy, and so no computation is performed. For slightly smaller  $|z|$  or  $(\nu+N-1)$ , the computation is performed but results are accurate to less than half of *machine precision*. If  $|z|$  is very small, near the machine underflow threshold, or  $(\nu+N-1)$  is too large, there is a risk of overflow and so no computation is performed. In all the above cases, a warning is given by the routine.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions, Ch. 9, p. 358.  
Dover Publications, 1968.
- [2] AMOS, D.E.  
Algorithm 644: A Portable Package for Bessel Functions of a Complex Argument and Nonnegative Order.  
ACM Trans. Math. Software, 12, pp. 265-273, 1986.

## 5. Parameters

- 1: FNU – *real*. *Input*  
*On entry:* the order,  $\nu$ , of the first member of the sequence of functions.  
*Constraint:* FNU  $\geq$  0.0.
- 2: Z – *complex*. *Input*  
*On entry:* the argument,  $z$ , of the functions.  
*Constraint:* Z  $\neq$  (0.0, 0.0).

- 3: N – INTEGER. *Input*  
*On entry:* the number,  $N$ , of members required in the sequence  $Y_v(z), Y_{v+1}(z), \dots, Y_{v+N-1}(z)$ .  
*Constraint:*  $N \geq 1$ .
- 4: SCALE – CHARACTER\*1. *Input*  
*On entry:* the scaling option.  
 If SCALE = 'U' or 'u', the results are returned unscaled.  
 If SCALE = 'S' or 's', the results are returned scaled by the factor  $e^{-|\text{Im}z|}$ .  
*Constraint:* SCALE = 'U', 'u', 'S' or 's'.
- 5: CY(N) – *complex* array. *Output*  
*On exit:* the  $N$  required function values: CY( $i$ ) contains  $Y_{v+i-1}(z)$ , for  $i = 1, 2, \dots, N$ .
- 6: NZ – INTEGER. *Output*  
*On exit:* the number of components of CY that are set to zero due to underflow. The positions of such components in the array CY are arbitrary.
- 7: CWRK(N) – *complex* array. *Workspace*
- 8: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, FNU < 0.0,  
 or Z = (0.0, 0.0),  
 or N < 1,  
 or SCALE  $\neq$  'U', 'u', 'S' or 's'.

IFAIL = 2

No computation has been performed due to the likelihood of overflow, because ABS(Z) is less than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 3

No computation has been performed due to the likelihood of overflow, because FNU + N - 1 is too large – how large depends on Z as well as the overflow threshold of the machine.

IFAIL = 4

The computation has been performed, but the errors due to argument reduction in elementary functions make it likely that the results returned by S17DCF are accurate to less than half of *machine precision*. This error exit may occur if either ABS(Z) or FNU + N - 1 is greater than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 5

No computation has been performed because the errors due to argument reduction in elementary functions mean that all precision in results returned by S17DCF would be lost. This error exit may occur if either  $\text{ABS}(Z)$  or  $\text{FNU} + N - 1$  is greater than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 6

No results are returned because the algorithm termination condition has not been met. This may occur because the parameters supplied to S17DCF would have caused overflow or underflow.

## 7. Accuracy

All constants in subroutine S17DCF are given to approximately 18 digits of precision. Calling the number of digits of precision in the floating-point arithmetic being used  $t$ , then clearly the maximum number of correct digits in the results obtained is limited by  $p = \min(t, 18)$ . Because of errors in argument reduction when computing elementary functions inside S17DCF, the actual number of correct digits is limited, in general, by  $p - s$ , where  $s \approx \max(1, |\log_{10} |z||, |\log_{10} \nu|)$  represents the number of digits lost due to the argument reduction. Thus the larger the values of  $|z|$  and  $\nu$ , the less the precision in the result. If S17DCF is called with  $N > 1$ , then computation of function values via recurrence may lead to some further small loss of accuracy.

If function values which should nominally be identical are computed by calls to S17DCF with different base values of  $\nu$  and different  $N$ , the computed values may not agree exactly. Empirical tests with modest values of  $\nu$  and  $z$  have shown that the discrepancy is limited to the least significant 3-4 digits of precision.

## 8. Further Comments

The time taken by the routine for a call of S17DCF is approximately proportional to the value of  $N$ , plus a constant. In general it is much cheaper to call S17DCF with  $N$  greater than 1, rather than to make  $N$  separate calls to S17DCF.

Paradoxically, for some values of  $z$  and  $\nu$ , it is cheaper to call S17DCF with a larger value of  $N$  than is required, and then discard the extra function values returned. However, it is not possible to state the precise circumstances in which this is likely to occur. It is due to the fact that the base value used to start recurrence may be calculated in different regions for different  $N$ , and the costs in each region may differ greatly.

Note that if the function required is  $Y_0(x)$  or  $Y_1(x)$ , i.e.  $\nu = 0.0$  or  $1.0$ , where  $x$  is real and positive, and only a single unscaled function value is required, then it may be much cheaper to call S17ACF or S17ADF respectively.

## 9. Example

The following example program prints a caption and then proceeds to read sets of data from the input data stream. The first datum is a value for the order FNU, the second is a complex value for the argument, Z, and the third is a value for the parameter SCALE. The program calls the routine with  $N = 2$  to evaluate the function for orders FNU and  $\text{FNU} + 1$ , and it prints the results. The process is repeated until the end of the input data stream is encountered.

## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      S17DCF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
      INTEGER          N
      PARAMETER       (N=2)
*      .. Local Scalars ..
      complex         Z
      real            FNU
      INTEGER          IFAIL, NZ
      CHARACTER*1     SCALE
*      .. Local Arrays ..
      complex         CWRK(N), CY(N)
*      .. External Subroutines ..
      EXTERNAL        S17DCF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S17DCF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Calling with N =', N
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      + '  FNU          Z          SCALE          CY(1)          CY(2)
      +  NZ IFAIL'
      WRITE (NOUT,*)
20  READ (NIN,*,END=40) FNU, Z, SCALE
      IFAIL = 0
*
      CALL S17DCF(FNU, Z, N, SCALE, CY, NZ, CWRK, IFAIL)
*
      WRITE (NOUT,99998) FNU, Z, SCALE, CY(1), CY(2), NZ, IFAIL
      GO TO 20
40  STOP
*
99999 FORMAT (1X,A,I2)
99998 FORMAT (1X,F7.4,' ('',F7.3,'','',F7.3,'') ',A,
+           2(' ('',F7.3,'','',F7.3,'')'),I4,I4)
      END

```

## 9.2. Program Data

```

S17DCF Example Program Data
0.00 ( 0.3, 0.4) 'U'
2.30 ( 2.0, 0.0) 'U'
2.12 (-1.0, 0.0) 'U'
1.58 (-2.3, 5.6) 'U'
1.58 (-2.3, 5.6) 'S'

```

## 9.3. Program Results

S17DCF Example Program Results

Calling with N = 2

FNU	Z	SCALE	CY(1)	CY(2)	NZ	IFAIL
0.0000	( 0.300, 0.400)	U	( -0.498, 0.670)	( -1.015, 0.949)	0	0
2.3000	( 2.000, 0.000)	U	( -0.740, 0.000)	( -1.412, 0.000)	0	0
2.1200	( -1.000, 0.000)	U	( -1.728, 0.860)	( 6.533, -2.615)	0	0
1.5800	( -2.300, 5.600)	U	( 36.476, -1.552)	( -2.679, 25.911)	0	0
1.5800	( -2.300, 5.600)	S	( 0.135, -0.006)	( -0.010, 0.096)	0	0

## S17DEF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S17DEF returns a sequence of values for the Bessel functions  $J_{\nu+n}(z)$  for complex  $z$ , non-negative  $\nu$  and  $n = 0, 1, \dots, N-1$ , with an option for exponential scaling.

## 2. Specification

```
SUBROUTINE S17DEF (FNU, Z, N, SCALE, CY, NZ, IFAIL)
  INTEGER          N, NZ, IFAIL
  real            FNU
  complex        Z, CY(N)
  CHARACTER*1     SCALE
```

## 3. Description

This subroutine evaluates a sequence of values for the Bessel function  $J_\nu(z)$ , where  $z$  is complex,  $-\pi < \arg z \leq \pi$ , and  $\nu$  is the real, non-negative order. The  $N$ -member sequence is generated for orders  $\nu, \nu+1, \dots, \nu+N-1$ . Optionally, the sequence is scaled by the factor  $e^{-|\operatorname{Im} z|}$ .

Note: although the routine may not be called with  $\nu$  less than zero, for negative orders the formula  $J_{-\nu}(z) = J_\nu(z)\cos(\pi\nu) - Y_\nu(z)\sin(\pi\nu)$  may be used (for the Bessel function  $Y_\nu(z)$ , see S17DCF).

The routine is derived from the routine CBESJ in Amos [2]. It is based on the relations  $J_\nu(z) = e^{\nu\pi i/2} I_\nu(-iz)$ ,  $\operatorname{Im} z \geq 0.0$ , and  $J_\nu(z) = e^{-\nu\pi i/2} I_\nu(iz)$ ,  $\operatorname{Im} z < 0.0$ .

The Bessel function  $I_\nu(z)$  is computed using a variety of techniques depending on the region under consideration.

When  $N$  is greater than 1, extra values of  $J_\nu(z)$  are computed using recurrence relations.

For very large  $|z|$  or  $(\nu+N-1)$ , argument reduction will cause total loss of accuracy, and so no computation is performed. For slightly smaller  $|z|$  or  $(\nu+N-1)$ , the computation is performed but results are accurate to less than half of *machine precision*. If  $\operatorname{Im} z$  is large, there is a risk of overflow and so no computation is performed. In all the above cases, a warning is given by the routine.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 358, 1968.
- [2] AMOS, D.E.  
Algorithm 644: A Portable Package for Bessel Functions of a Complex Argument and Nonnegative Order.  
ACM Trans. Math. Software, 12, pp. 265-273, 1986.

## 5. Parameters

- 1: FNU – *real*. *Input*  
*On entry:* the order,  $\nu$ , of the first member of the sequence of functions.  
*Constraint:* FNU  $\geq 0.0$ .
- 2: Z – *complex*. *Input*  
*On entry:* the argument  $z$  of the functions.

- 3: N – INTEGER. Input  
*On entry:* the number,  $N$ , of members required in the sequence  $J_\nu(z), J_{\nu+1}(z), \dots, J_{\nu+N-1}(z)$ .  
*Constraint:*  $N \geq 1$ .
- 4: SCALE – CHARACTER\*1. Input  
*On entry:* the scaling option.  
 If SCALE = 'U' or 'u', the results are returned unscaled.  
 If SCALE = 'S' or 's', the results are returned scaled by the factor  $e^{-|\text{Im}z|}$ .  
*Constraint:* SCALE = 'U', 'u', 'S' or 's'.
- 5: CY(N) – *complex* array. Output  
*On exit:* the  $N$  required function values: CY( $i$ ) contains  $J_{\nu+i-1}(z)$ , for  $i = 1, 2, \dots, N$ .
- 6: NZ – INTEGER. Output  
*On exit:* the number of components of CY that are set to zero due to underflow. If NZ > 0, then elements CY(N–NZ+1), CY(N–NZ+2), ..., CY(N) are set to zero.
- 7: IFAIL – INTEGER. Input/Output  
*On entry:* IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, FNU < 0.0,  
 or N < 1,  
 or SCALE  $\neq$  'U', 'u', 'S' or 's'.

IFAIL = 2

No computation has been performed due to the likelihood of overflow, because  $\text{Im} Z$  is larger than a machine-dependent threshold value (given in the Users' Note for your implementation). This error exit can only occur when SCALE = 'U' or 'u'.

IFAIL = 3

The computation has been performed, but the errors due to argument reduction in elementary functions make it likely that the results returned by S17DEF are accurate to less than half of *machine precision*. This error exit may occur if either  $\text{ABS}(Z)$  or  $\text{FNU} + N - 1$  is greater than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 4

No computation has been performed because the errors due to argument reduction in elementary functions mean that all precision in results returned by S17DEF would be lost. This error exit may occur when either  $\text{ABS}(Z)$  or  $\text{FNU} + N - 1$  is greater than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 5

No results are returned because the algorithm termination condition has not been met. This may occur because the parameters supplied to S17DEF would have caused overflow or underflow.



## 7. Accuracy

All constants in subroutine S17DEF are given to approximately 18 digits of precision. Calling the number of digits of precision in the floating-point arithmetic being used  $t$ , then clearly the maximum number of correct digits in the results obtained is limited by  $p = \min(t, 18)$ . Because of errors in argument reduction when computing elementary functions inside S17DEF, the actual number of correct digits is limited, in general, by  $p - s$ , where  $s \approx \max(1, |\log_{10} |z||, |\log_{10} v|)$  represents the number of digits lost due to the argument reduction. Thus the larger the values of  $|z|$  and  $v$ , the less the precision in the result. If S17DEF is called with  $N > 1$ , then computation of function values via recurrence may lead to some further small loss of accuracy.

If function values which should nominally be identical are computed by calls to S17DEF with different base values of  $v$  and different  $N$ , the computed values may not agree exactly. Empirical tests with modest values of  $v$  and  $z$  have shown that the discrepancy is limited to the least significant 3-4 digits of precision.

## 8. Further Comments

The time taken by the routine for a call of S17DEF is approximately proportional to the value of  $N$ , plus a constant. In general it is much cheaper to call S17DEF with  $N$  greater than 1, rather than to make  $N$  separate calls to S17DEF.

Paradoxically, for some values of  $z$  and  $v$ , it is cheaper to call S17DEF with a larger value of  $N$  than is required, and then discard the extra function values returned. However, it is not possible to state the precise circumstances in which this is likely to occur. It is due to the fact that the base value used to start recurrence may be calculated in different regions for different  $N$ , and the costs in each region may differ greatly.

Note that if the function required is  $J_0(x)$  or  $J_1(x)$ , i.e.  $v = 0.0$  or  $1.0$ , where  $x$  is real and positive, and only a single unscaled function value is required, then it may be much cheaper to call S17AEF or S17AFF respectively.

## 9. Example

The following example program prints a caption and then proceeds to read sets of data from the input data stream. The first datum is a value for the order FNU, the second is a complex value for the argument, Z, and the third is a value for the parameter SCALE. The program calls the routine with  $N = 2$  to evaluate the function for orders FNU and FNU + 1, and it prints the results. The process is repeated until the end of the input data stream is encountered.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S17DEF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5, NOUT=6)
      INTEGER          N
      PARAMETER        (N=2)
*      .. Local Scalars ..
      complex
      Z
      real
      FNU
      INTEGER          IFAIL, NZ
      CHARACTER*1      SCALE
*      .. Local Arrays ..
      complex
      CY(N)
*      .. External Subroutines ..
      EXTERNAL         S17DEF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S17DEF Example Program Results'
```

```

*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Calling with N =', N
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      +   FNU           Z           SCALE           CY(1)           CY(2)
      +   NZ IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) FNU, Z, SCALE
      IFAIL = 0
*
      CALL S17DEF(FNU,Z,N,SCALE,CY,NZ,IFAIL)
*
      WRITE (NOUT,99998) FNU, Z, SCALE, CY(1), CY(2), NZ, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,A,I2)
99998  FORMAT (1X,F7.4,' ('',F7.3,'','',F7.3,'') ',A,
      +       2(' ('',F7.3,'','',F7.3,'')'),I4,I4)
      END

```

## 9.2. Program Data

```

S17DEF Example Program Data
0.00   ( 0.3, 0.4)   'U'
2.30   ( 2.0, 0.0)   'U'
2.12   (-1.0, 0.0)   'U'
1.58   (-2.3, 5.6)   'U'
1.58   (-2.3, 5.6)   'S'

```

## 9.3. Program Results

S17DEF Example Program Results

Calling with N = 2

FNU	Z	SCALE	CY(1)	CY(2)	NZ	IFAIL
0.0000	( 0.300, 0.400)	U	( 1.017, -0.061)	( 0.157, 0.197)	0	0
2.3000	( 2.000, 0.000)	U	( 0.272, 0.000)	( 0.089, 0.000)	0	0
2.1200	( -1.000, 0.000)	U	( 0.088, 0.035)	( -0.014, -0.006)	0	0
1.5800	( -2.300, 5.600)	U	( -1.551, -36.476)	( 25.910, 2.677)	0	0
1.5800	( -2.300, 5.600)	S	( -0.006, -0.135)	( 0.096, 0.010)	0	0

## S17DGF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S17DGF returns the value of the Airy function  $Ai(z)$  or its derivative  $Ai'(z)$  for complex  $z$ , with an option for exponential scaling.

## 2. Specification

```
SUBROUTINE S17DGF (DERIV, Z, SCALE, AI, NZ, IFAIL)
  INTEGER          NZ, IFAIL
  complex        Z, AI
  CHARACTER*1     DERIV, SCALE
```

## 3. Description

This subroutine returns a value for the Airy function  $Ai(z)$  or its derivative  $Ai'(z)$ , where  $z$  is complex,  $-\pi < \arg z \leq \pi$ . Optionally, the value is scaled by the factor  $e^{2z\sqrt{z}/3}$ .

The routine is derived from the routine CAIRY in Amos [2]. It is based on the relations

$$Ai(z) = \frac{\sqrt{z}K_{1/3}(w)}{\pi\sqrt{3}}, \text{ and } Ai'(z) = \frac{-zK_{2/3}(w)}{\pi\sqrt{3}}, \text{ where } K_\nu \text{ is the modified Bessel function and } w = 2z\sqrt{z}/3.$$

For very large  $|z|$ , argument reduction will cause total loss of accuracy, and so no computation is performed. For slightly smaller  $|z|$ , the computation is performed but results are accurate to less than half of *machine precision*. If  $\text{Re } w$  is too large, and the unscaled function is required, there is a risk of overflow and so no computation is performed. In all the above cases, a warning is given by the routine.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 358, 1968.
- [2] AMOS, D.E.  
Algorithm 644: A Portable Package for Bessel Functions of a Complex Argument and Nonnegative Order.  
ACM Trans. Math. Software 12, pp. 265-273, 1986.

## 5. Parameters

- 1: DERIV – CHARACTER\*1. *Input*  
*On entry:* specifies whether the function or its derivative is required.  
 If DERIV = 'F' or 'f',  $Ai(z)$  is returned.  
 If DERIV = 'D' or 'd',  $Ai'(z)$  is returned.  
*Constraint:* DERIV = 'F', 'f', 'D' or 'd'.
- 2: Z – **complex**. *Input*  
*On entry:* the argument  $z$  of the function.

- 3: SCALE – CHARACTER\*1. *Input*  
*On entry:* the scaling option.  
 If SCALE = 'U' or 'u', the result is returned unscaled.  
 If SCALE = 'S' or 's', the result is returned scaled by the factor  $e^{2z\sqrt{z}/3}$ .  
*Constraint:* SCALE = 'U', 'u', 'S' or 's'.
- 4: AI – *complex*. *Output*  
*On exit:* the required function or derivative value.
- 5: NZ – INTEGER. *Output*  
*On exit:* NZ indicates whether or not AI is set to zero due to underflow. This can only occur when SCALE = 'U' or 'u'.  
 If NZ = 0, AI is not set to zero.  
 If NZ = 1, AI is set to zero.
- 6: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, DERIV  $\neq$  'F', 'f', 'D' or 'd',  
 or SCALE  $\neq$  'U', 'u', 'S' or 's'.

IFAIL = 2

No computation has been performed due to the likelihood of overflow, because  $\text{Re } w$  is too large, where  $w = 2Z\sqrt{Z}/3$  – how large depends on  $Z$  and the overflow threshold of the machine. This error exit can only occur when SCALE = 'U' or 'u'.

IFAIL = 3

The computation has been performed, but the errors due to argument reduction in elementary functions make it likely that the result returned by S17DGF is accurate to less than half of *machine precision*. This error exit may occur if  $\text{ABS}(Z)$  is greater than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 4

No computation has been performed because the errors due to argument reduction in elementary functions mean that all precision in the result returned by S17DGF would be lost. This error exit may occur if  $\text{ABS}(Z)$  is greater than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 5

No result is returned because the algorithm termination condition has not been met. This may occur because the parameters supplied to S17DGF would have caused overflow or underflow.

## 7. Accuracy

All constants in subroutine S17DGF are given to approximately 18 digits of precision. Calling the number of digits of precision in the floating-point arithmetic being used  $t$ , then clearly the maximum number of correct digits in the results obtained is limited by  $p = \min(t, 18)$ . Because of errors in argument reduction when computing elementary functions inside S17DGF, the actual number of correct digits is limited, in general, by  $p - s$ , where  $s \approx \max(1, |\log_{10} |z||)$  represents the number of digits lost due to the argument reduction. Thus the larger the value of  $|z|$ , the less the precision in the result.

Empirical tests with modest values of  $z$ , checking relations between Airy functions  $Ai(z)$ ,  $Ai'(z)$ ,  $Bi(z)$  and  $Bi'(z)$ , have shown errors limited to the least significant 3-4 digits of precision.

## 8. Further Comments

Note that if the function is required to operate on a real argument only, then it may be much cheaper to call S17AGF or S17AJF.

## 9. Example

The following example program prints a caption and then proceeds to read sets of data from the input data stream. The first datum is a value for the parameter DERIV, the second is a complex value for the argument, Z, and the third is a value for the parameter SCALE. The program calls the routine and prints the results. The process is repeated until the end of the input data stream is encountered.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S17DGF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      complex        AI, Z
      INTEGER          IFAIL, NZ
      CHARACTER*1     DERIV, SCALE
*      .. External Subroutines ..
      EXTERNAL        S17DGF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S17DGF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*)
+      'DERIV          Z          SCALE          AI          NZ  IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) DERIV, Z, SCALE
      IFAIL = 0
*
      CALL S17DGF(DERIV,Z,SCALE,AI,NZ,IFAIL)
*
      WRITE (NOUT,99999) DERIV, Z, SCALE, AI, NZ, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (3X,A,' (' ,F8.4,' ',' ,F8.4,' ) ',A,' (' ,F8.4,' ',' ,F8.4,
+          ' )',I4,I5)
      END
```

## 9.2. Program Data

S17DGF Example Program Data

'F'	( 0.3, 0.4)	'U'
'F'	( 0.2, 0.0)	'U'
'F'	( 1.1, -6.6)	'U'
'F'	( 1.1, -6.6)	'S'
'D'	(-1.0, 0.0)	'U'

## 9.3. Program Results

S17DGF Example Program Results

DERIV	Z	SCALE	AI	NZ	IFAIL
F	( 0.3000, 0.4000)	U	( 0.2716, -0.1002)	0	0
F	( 0.2000, 0.0000)	U	( 0.3037, 0.0000)	0	0
F	( 1.1000, -6.6000)	U	(-43.6632, -47.9030)	0	0
F	( 1.1000, -6.6000)	S	( 0.1655, 0.0597)	0	0
D	( -1.0000, 0.0000)	U	( -0.0102, 0.0000)	0	0

---

## S17DHF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S17DHF returns the value of the Airy function  $\text{Bi}(z)$  or its derivative  $\text{Bi}'(z)$  for complex  $z$ , with an option for exponential scaling.

## 2. Specification

```
SUBROUTINE S17DHF (DERIV, Z, SCALE, BI, IFAIL)
  INTEGER          IFAIL
  complex        Z, BI
  CHARACTER*1     DERIV, SCALE
```

## 3. Description

This subroutine returns a value for the Airy function  $\text{Bi}(z)$  or its derivative  $\text{Bi}'(z)$ , where  $z$  is complex,  $-\pi < \arg z \leq \pi$ . Optionally, the value is scaled by the factor  $e^{|\text{Re}(2z\sqrt{z/3})|}$ .

The routine is derived from the routine CBIRY in Amos [2]. It is based on the relations  $\text{Bi}(z) = \frac{\sqrt{z}}{\sqrt{3}}(I_{-1/3}(w) + I_{1/3}(w))$ , and  $\text{Bi}'(z) = \frac{z}{\sqrt{3}}(I_{-2/3}(w) + I_{2/3}(w))$ , where  $I_\nu$  is the modified Bessel function and  $w = 2z\sqrt{z/3}$ .

For very large  $|z|$ , argument reduction will cause total loss of accuracy, and so no computation is performed. For slightly smaller  $|z|$ , the computation is performed but results are accurate to less than half of *machine precision*. If  $\text{Re } z$  is too large, and the unscaled function is required, there is a risk of overflow and so no computation is performed. In all the above cases, a warning is given by the routine.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 358, 1968.
- [2] AMOS, D.E.  
Algorithm 644: A Portable Package for Bessel Functions of a Complex Argument and Nonnegative Order.  
ACM Trans. Math. Software, 12, pp. 265-273, 1986.

## 5. Parameters

1: DERIV – CHARACTER\*1.

*Input*

*On entry:* specifies whether the function or its derivative is required.

If DERIV = 'F' or 'f',  $\text{Bi}(z)$  is returned.

If DERIV = 'D' or 'd',  $\text{Bi}'(z)$  is returned.

*Constraint:* DERIV = 'F', 'f', 'D' or 'd'.

2: Z – **complex**.

*Input*

*On entry:* the argument  $z$  of the function.

## 3: SCALE – CHARACTER\*1.

Input

*On entry:* the scaling option.

If SCALE = 'U' or 'u', the result is returned unscaled.

If SCALE = 'S' or 's', the result is returned scaled by the factor  $e^{|\operatorname{Re}(2z\sqrt{z}/3)|}$ .

*Constraint:* SCALE = 'U', 'u', 'S' or 's'.

## 4: BI – complex.

Output

*On exit:* the required function or derivative value.

## 5: IFAIL – INTEGER.

Input/Output

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, DERIV  $\neq$  'F', 'f', 'D' or 'd',  
or SCALE  $\neq$  'U', 'u', 'S' or 's'.

IFAIL = 2

No computation has been performed due to the likelihood of overflow, because  $\operatorname{real}(Z)$  is too large – how large depends on the overflow threshold of the machine. This error exit can only occur when SCALE = 'U' or 'u'.

IFAIL = 3

The computation has been performed, but the errors due to argument reduction in elementary functions make it likely that the result returned by S17DHF is accurate to less than half of *machine precision*. This error exit may occur if  $\operatorname{ABS}(Z)$  is greater than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 4

No computation has been performed because the errors due to argument reduction in elementary functions mean that all precision in the result returned by S17DHF would be lost. This error exit may occur if  $\operatorname{ABS}(Z)$  is greater than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 5

No result is returned because the algorithm termination condition has not been met. This may occur because the parameters supplied to S17DHF would have caused overflow or underflow.

## 7. Accuracy

All constants in subroutine S17DHF are given to approximately 18 digits of precision. Calling the number of digits of precision in the floating-point arithmetic being used  $t$ , then clearly the maximum number of correct digits in the results obtained is limited by  $p = \min(t, 18)$ . Because of errors in argument reduction when computing elementary functions inside S17DHF, the actual number of correct digits is limited, in general, by  $p - s$ , where  $s \approx \max(1, |\log_{10} |z||)$  represents the number of digits lost due to the argument reduction. Thus the larger the value of  $|z|$ , the less the precision in the result.



Empirical tests with modest values of  $z$ , checking relations between Airy functions  $Ai(z)$ ,  $Ai'(z)$ ,  $Bi(z)$  and  $Bi'(z)$ , have shown errors limited to the least significant 3-4 digits of precision.

## 8. Further Comments

Note that if the function is required to operate on a real argument only, then it may be much cheaper to call S17AHF or S17AKF.

## 9. Example

The following example program prints a caption and then proceeds to read sets of data from the input data stream. The first datum is a value for the parameter DERIV, the second is a complex value for the argument, Z, and the third is a value for the parameter SCALE. The program calls the routine and prints the results. The process is repeated until the end of the input data stream is encountered.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S17DHF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
          INTEGER          NIN, NOUT
          PARAMETER        (NIN=5,NOUT=6)
*      .. Local Scalars ..
          complex          BI, Z
          INTEGER          IFAIL
          CHARACTER*1      DERIV, SCALE
*      .. External Subroutines ..
          EXTERNAL        S17DHF
*      .. Executable Statements ..
          WRITE (NOUT,*) 'S17DHF Example Program Results'
          Skip heading in data file
          READ (NIN,*)
          WRITE (NOUT,*)
          WRITE (NOUT,*)
          + 'DERIV          Z          SCALE          BI          IFAIL'
          WRITE (NOUT,*)
20      READ (NIN,*,END=40) DERIV, Z, SCALE
          IFAIL = 0
*
          CALL S17DHF(DERIV,Z,SCALE,BI,IFAIL)
*
          WRITE (NOUT,99999) DERIV, Z, SCALE, BI, IFAIL
          GO TO 20
40      STOP
*
99999  FORMAT (3X,A,'      (',F8.4,',',F8.4,')      ',A,'      (',F8.4,',',F8.4,
+          ',I5)
          END
```

### 9.2. Program Data

```
S17DHF Example Program Data
'F' ( 0.3, 0.4) 'U'
'F' ( 0.2, 0.0) 'U'
'F' ( 1.1, -6.6) 'U'
'F' ( 1.1, -6.6) 'S'
'D' (-1.0, 0.0) 'U'
```

## 9.3. Program Results

## S17DHF Example Program Results

DERIV	Z	SCALE	BI	IFAIL
F	( 0.3000, 0.4000)	U	( 0.7355, 0.1825)	0
F	( 0.2000, 0.0000)	U	( 0.7055, 0.0000)	0
F	( 1.1000, -6.6000)	U	(-47.9039, 43.6634)	0
F	( 1.1000, -6.6000)	S	( -0.1300, 0.1185)	0
D	( -1.0000, 0.0000)	U	( 0.5924, 0.0000)	0

---

## S17DLF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S17DLF returns a sequence of values for the Hankel functions  $H_{\nu+n}^{(1)}(z)$  or  $H_{\nu+n}^{(2)}(z)$  for complex  $z$ , non-negative  $\nu$  and  $n = 0, 1, \dots, N-1$ , with an option for exponential scaling.

## 2. Specification

```
SUBROUTINE S17DLF (M, FNU, Z, N, SCALE, CY, NZ, IFAIL)
  INTEGER          M, N, NZ, IFAIL
  real            FNU
  complex        Z, CY(N)
  CHARACTER*1     SCALE
```

## 3. Description

This subroutine evaluates a sequence of values for the Hankel function  $H_{\nu}^{(1)}(z)$  or  $H_{\nu}^{(2)}(z)$ , where  $z$  is complex,  $-\pi < \arg z \leq \pi$ , and  $\nu$  is the real, non-negative order. The  $N$ -member sequence is generated for orders  $\nu, \nu+1, \dots, \nu+N-1$ . Optionally, the sequence is scaled by the factor  $e^{-iz}$  if the function is  $H_{\nu}^{(1)}(z)$  or by the factor  $e^{iz}$  if the function is  $H_{\nu}^{(2)}(z)$ .

Note: although the routine may not be called with  $\nu$  less than zero, for negative orders the formulae  $H_{-\nu}^{(1)}(z) = e^{i\pi\nu} H_{\nu}^{(1)}(z)$ , and  $H_{-\nu}^{(2)}(z) = e^{-i\pi\nu} H_{\nu}^{(2)}(z)$  may be used.

The routine is derived from the routine CBESH in Amos [2]. It is based on the relation

$$H_{\nu}^{(m)}(z) = \frac{1}{p} e^{-p\nu} K_{\nu}(ze^{-p}),$$

where  $p = \frac{i\pi}{2}$  if  $m = 1$  and  $p = \frac{-i\pi}{2}$  if  $m = 2$ , and the Bessel function  $K_{\nu}(z)$  is computed in the right half-plane only. Continuation of  $K_{\nu}(z)$  to the left half-plane is computed in terms of the Bessel function  $I_{\nu}(z)$ . These functions are evaluated using a variety of different techniques, depending on the region under consideration.

When  $N$  is greater than 1, extra values of  $H_{\nu}^{(m)}(z)$  are computed using recurrence relations.

For very large  $|z|$  or  $(\nu+N-1)$ , argument reduction will cause total loss of accuracy, and so no computation is performed. For slightly smaller  $|z|$  or  $(\nu+N-1)$ , the computation is performed but results are accurate to less than half of *machine precision*. If  $|z|$  is very small, near the machine underflow threshold, or  $(\nu+N-1)$  is too large, there is a risk of overflow and so no computation is performed. In all the above cases, a warning is given by the routine.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 358, 1968.
- [2] AMOS, D.E.  
Algorithm 644: A Portable Package for Bessel Functions of a Complex Argument and Nonnegative Order.  
ACM Trans. Math. Software, 12, pp. 265-273, 1986.

## 5. Parameters

- 1: M – INTEGER. *Input*  
*On entry:* the kind of functions required.  
 If M = 1, the functions are  $H_\nu^{(1)}(z)$ .  
 If M = 2, the functions are  $H_\nu^{(2)}(z)$ .  
*Constraint:* M = 1 or 2.
- 2: FNU – *real*. *Input*  
*On entry:* the order,  $\nu$ , of the first member of the sequence of functions.  
*Constraint:* FNU  $\geq$  0.0.
- 3: Z – *complex*. *Input*  
*On entry:* the argument  $z$  of the functions.  
*Constraint:* Z  $\neq$  (0.0, 0.0).
- 4: N – INTEGER. *Input*  
*On entry:* the number,  $N$ , of members required in the sequence  $H_\nu^{(M)}(z)$ ,  $H_{\nu+1}^{(M)}(z)$ , ...,  $H_{\nu+N-1}^{(M)}(z)$ .  
*Constraint:* N  $\geq$  1.
- 5: SCALE – CHARACTER\*1. *Input*  
*On entry:* the scaling option.  
 If SCALE = 'U' or 'u', the results are returned unscaled.  
 If SCALE = 'S' or 's', the results are returned scaled by the factor  $e^{-iz}$  when M = 1, or by the factor  $e^{iz}$  when M = 2.  
*Constraint:* SCALE = 'U', 'u', 'S' or 's'.
- 6: CY(N) – *complex* array. *Output*  
*On exit:* the  $N$  required function values: CY( $i$ ) contains  $H_{\nu+i-1}^{(M)}(z)$ , for  $i = 1, 2, \dots, N$ .
- 7: NZ – INTEGER. *Output*  
*On exit:* the number of components of CY that are set to zero due to underflow. If NZ > 0, then if  $\text{Im } z > 0.0$  and M = 1, or  $\text{Im } z < 0.0$  and M = 2, elements CY(1), CY(2), ..., CY(NZ) are set to zero. In the complementary half-planes, NZ simply states the number of underflows, and not which elements they are.
- 8: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, M  $\neq$  1 and M  $\neq$  2,  
 or FNU < 0.0,  
 or Z = (0.0, 0.0),

or  $N < 1$ ,  
 or  $SCALE \neq 'U', 'u', 'S' \text{ or } 's'$ .

IFAIL = 2

No computation has been performed due to the likelihood of overflow, because  $ABS(Z)$  is less than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 3

No computation has been performed due to the likelihood of overflow, because  $FNU + N - 1$  is too large – how large depends on  $Z$  and the overflow threshold of the machine.

IFAIL = 4

The computation has been performed, but the errors due to argument reduction in elementary functions make it likely that the results returned by S17DLF are accurate to less than half of *machine precision*. This error exit may occur if either  $ABS(Z)$  or  $FNU + N - 1$  is greater than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 5

No computation has been performed because the errors due to argument reduction in elementary functions mean that all precision in results returned by S17DLF would be lost. This error exit may occur when either of  $ABS(Z)$  or  $FNU + N - 1$  is greater than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 6

No results are returned because the algorithm termination condition has not been met. This may occur because the parameters supplied to S17DLF would have caused overflow or underflow.

## 7. Accuracy

All constants in subroutine S17DLF are given to approximately 18 digits of precision. Calling the number of digits of precision in the floating-point arithmetic being used  $t$ , then clearly the maximum number of correct digits in the results obtained is limited by  $p = \min(t, 18)$ . Because of errors in argument reduction when computing elementary functions inside S17DLF, the actual number of correct digits is limited, in general, by  $p - s$ , where  $s \approx \max(1, |\log_{10} |z||, |\log_{10} v|)$  represents the number of digits lost due to the argument reduction. Thus the larger the values of  $|z|$  and  $v$ , the less the precision in the result. If S17DLF is called with  $N > 1$ , then computation of function values via recurrence may lead to some further small loss of accuracy.

If function values which should nominally be identical are computed by calls to S17DLF with different base values of  $v$  and different  $N$ , the computed values may not agree exactly. Empirical tests with modest values of  $v$  and  $z$  have shown that the discrepancy is limited to the least significant 3-4 digits of precision.

## 8. Further Comments

The time taken by the routine for a call of S17DLF is approximately proportional to the value of  $N$ , plus a constant. In general it is much cheaper to call S17DLF with  $N$  greater than 1, rather than to make  $N$  separate calls to S17DLF.

Paradoxically, for some values of  $z$  and  $v$ , it is cheaper to call S17DLF with a larger value of  $N$  than is required, and then discard the extra function values returned. However, it is not possible to state the precise circumstances in which this is likely to occur. It is due to the fact that the base value used to start recurrence may be calculated in different regions for different  $N$ , and the costs in each region may differ greatly.

## 9. Example

The following example program prints a caption and then proceeds to read sets of data from the input data stream. The first datum is a value for the kind of function, M, the second is a value for the order FNU, the third is a complex value for the argument, Z, and the fourth is a value for the parameter SCALE. The program calls the routine with N = 2 to evaluate the function for orders FNU and FNU + 1, and it prints the results. The process is repeated until the end of the input data stream is encountered.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S17DLF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
      INTEGER          N
      PARAMETER       (N=2)
*      .. Local Scalars ..
      complex         Z
      real            FNU
      INTEGER          IFAIL, M, NZ
      CHARACTER*1     SCALE
*      .. Local Arrays ..
      complex         CY(N)
*      .. External Subroutines ..
      EXTERNAL        S17DLF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S17DLF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Calling with N =', N
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      + 'M      FNU          Z          SCALE          CY(1)          CY(2)
      +      NZ IFAIL'
      WRITE (NOUT,*)
      20 READ (NIN,*,END=40) M, FNU, Z, SCALE
      IFAIL = 0
*
      CALL S17DLF(M,FNU,Z,N,SCALE,CY,NZ,IFAIL)
*
      WRITE (NOUT,99998) M, FNU, Z, SCALE, CY(1), CY(2), NZ, IFAIL
      GO TO 20
      40 STOP
*
      99999 FORMAT (1X,A,I2)
      99998 FORMAT (1X,I1,1X,F7.4,' (' ,F7.3,',',',F7.3,') ' ,A,
      +          2(' (' ,F7.3,',',',F7.3,')'),I4,I4)
      END
```

### 9.2. Program Data

```
S17DLF Example Program Data
1      0.00      ( 0.3,  0.4)      'U'
1      2.30      ( 2.0,  0.0)      'U'
1      2.12      (-1.0,  0.0)      'U'
2      6.00      ( 3.1, -1.6)      'U'
2      6.00      ( 3.1, -1.6)      'S'
```

### 9.3. Program Results

S17DLF Example Program Results

Calling with N = 2

M	FNU	Z	SCALE	CY(1)	CY(2)	NZ	IFAIL
1	0.0000	( 0.300, 0.400)	U	( 0.347, -0.559)	( -0.791, -0.818)	0	0
1	2.3000	( 2.000, 0.000)	U	( 0.272, -0.740)	( 0.089, -1.412)	0	0
1	2.1200	( -1.000, 0.000)	U	( -0.772, -1.693)	( 2.601, 6.527)	0	0
2	6.0000	( 3.100, -1.600)	U	( -1.371, -1.280)	( -1.491, -5.993)	0	0
2	6.0000	( 3.100, -1.600)	S	( 7.050, 6.052)	( 8.614, 29.352)	0	0

---





## S18ACF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S18ACF returns the value of the modified Bessel Function  $K_0(x)$ , via the routine name.

## 2. Specification

```
real FUNCTION S18ACF (X, IFAIL)
      INTEGER      IFAIL
      real         X
```

## 3. Description

This routine evaluates an approximation to the modified Bessel Function of the second kind  $K_0(x)$ .

**Note:**  $K_0(x)$  is undefined for  $x \leq 0$  and the routine will fail for such arguments.

The routine is based on five Chebyshev expansions:

For  $0 < x \leq 1$ ,

$$K_0(x) = -\ln x \sum_{r=0}' a_r T_r(t) + \sum_{r=0}' b_r T_r(t), \quad \text{where } t = 2x^2 - 1;$$

For  $1 < x \leq 2$ ,

$$K_0(x) = e^{-x} \sum_{r=0}' c_r T_r(t), \quad \text{where } t = 2x - 3;$$

For  $2 < x \leq 4$ ,

$$K_0(x) = e^{-x} \sum_{r=0}' d_r T_r(t), \quad \text{where } t = x - 3;$$

For  $x > 4$ ,

$$K_0(x) = \frac{e^{-x}}{\sqrt{x}} \sum_{r=0}' e_r T_r(t), \quad \text{where } t = \frac{9 - x}{1 + x}.$$

For  $x$  near zero,  $K_0(x) \simeq -\gamma - \ln\left(\frac{x}{2}\right)$ , where  $\gamma$  denotes Euler's constant. This approximation is used when  $x$  is sufficiently small for the result to be correct to *machine precision*.

For large  $x$ , where there is a danger of underflow due to the smallness of  $K_0$ , the result is set exactly to zero.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 374, 1968.

## 5. Parameters

1: **X** – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

*Constraint:*  $X > 0.0$ .

2: **IFAIL** – **INTEGER**.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

$X \leq 0.0$ ,  $K_0$  is undefined. On soft failure the routine returns zero.

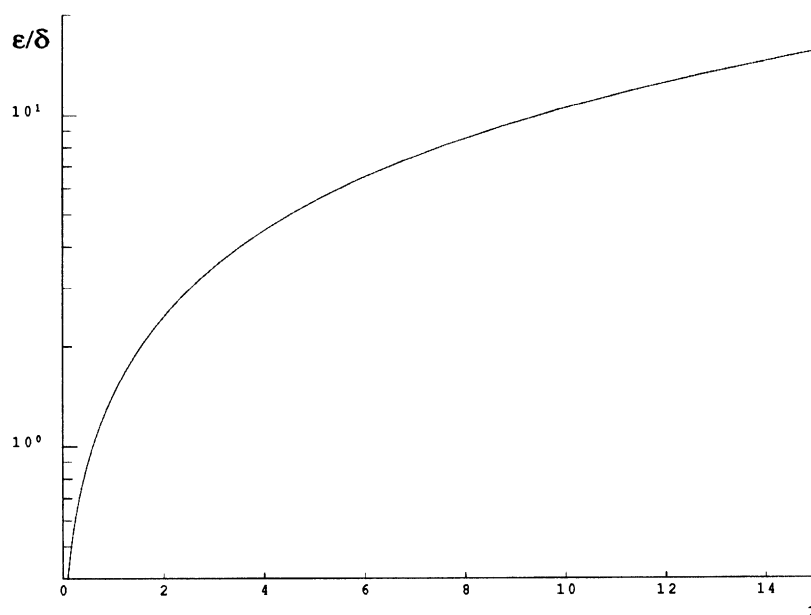
## 7. Accuracy

Let  $\delta$  and  $\varepsilon$  be the relative errors in the argument and result respectively.

If  $\delta$  is somewhat larger than the *machine precision* (i.e. if  $\delta$  is due to data errors etc.), then  $\varepsilon$  and  $\delta$  are approximately related by:

$$\varepsilon \simeq \left| \frac{xK_1(x)}{K_0(x)} \right| \delta.$$

The following graph shows the behaviour of the error amplification factor  $\left| \frac{xK_1(x)}{K_0(x)} \right|$ :



However, if  $\delta$  is of the same order as *machine precision*, then rounding errors could make  $\varepsilon$  slightly larger than the above relation predicts.

For small  $x$ , the amplification factor is approximately  $\left| \frac{1}{\ln x} \right|$ , which implies strong attenuation of the error, but in general  $\varepsilon$  can never be less than the *machine precision*.

For large  $x$ ,  $\varepsilon \simeq x\delta$  and we have strong amplification of the relative error. Eventually  $K_0$ , which is asymptotically given by  $\frac{e^{-x}}{\sqrt{x}}$ , becomes so small that it cannot be calculated without underflow and hence the routine will return zero. Note that for large  $x$  the errors will be dominated by those of the Fortran intrinsic function EXP.

## 8. Further Comments

For details of the time taken by the routine see the appropriate the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S18ACF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S18ACF
      EXTERNAL        S18ACF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S18ACF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S18ACF(X, IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END
```

## 9.2. Program Data

```
S18ACF Example Program Data
      0.0
      0.4
      0.6
      1.4
      1.6
      2.5
      3.5
      6.0
      8.0
      10.0
      -1.0
      1000.0
```

**9.3. Program Results**

S18ACF Example Program Results

X	Y	IFAIL
0.000E+00	0.000E+00	1
4.000E-01	1.115E+00	0
6.000E-01	7.775E-01	0
1.400E+00	2.437E-01	0
1.600E+00	1.880E-01	0
2.500E+00	6.235E-02	0
3.500E+00	1.960E-02	0
6.000E+00	1.244E-03	0
8.000E+00	1.465E-04	0
1.000E+01	1.778E-05	0
-1.000E+00	0.000E+00	1
1.000E+03	0.000E+00	0

---

## S18ADF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S18ADF returns the value of the modified Bessel Function  $K_1(x)$ , via the routine name.

## 2. Specification

```

real FUNCTION S18ADF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

## 3. Description

This routine evaluates an approximation to the modified Bessel Function of the second kind  $K_1(x)$ .

Note:  $K_1(x)$  is undefined for  $x \leq 0$  and the routine will fail for such arguments.

The routine is based on five Chebyshev expansions:

For  $0 < x \leq 1$ ,

$$K_1(x) = \frac{1}{x} + x \ln x \sum_{r=0}^{\prime} a_r T_r(t) - x \sum_{r=0}^{\prime} b_r T_r(t), \quad \text{where } t = 2x^2 - 1;$$

For  $1 < x \leq 2$ ,

$$K_1(x) = e^{-x} \sum_{r=0}^{\prime} c_r T_r(t), \quad \text{where } t = 2x - 3;$$

For  $2 < x \leq 4$ ,

$$K_1(x) = e^{-x} \sum_{r=0}^{\prime} d_r T_r(t), \quad \text{where } t = x - 3;$$

For  $x > 4$ ,

$$K_1(x) = \frac{e^{-x}}{\sqrt{x}} \sum_{r=0}^{\prime} e_r T_r(t), \quad \text{where } t = \frac{9 - x}{1 + x}.$$

For  $x$  near zero,  $K_1(x) \simeq \frac{1}{x}$ . This approximation is used when  $x$  is sufficiently small for the result to be correct to *machine precision*. For very small  $x$  on some machines, it is impossible to calculate  $\frac{1}{x}$  without overflow and the routine must fail.

For large  $x$ , where there is a danger of underflow due to the smallness of  $K_1$ , the result is set exactly to zero.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 374, 1968.

## 5. Parameters

1: X – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

*Constraint:* X > 0.0.

## 2: IFAIL – INTEGER.

Input/Output

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

$X \leq 0.0$ ,  $K_1$  is undefined. On soft failure the routine returns zero.

IFAIL = 2

$X$  is too small, there is a danger of overflow. On soft failure the routine returns approximately the largest representable value.

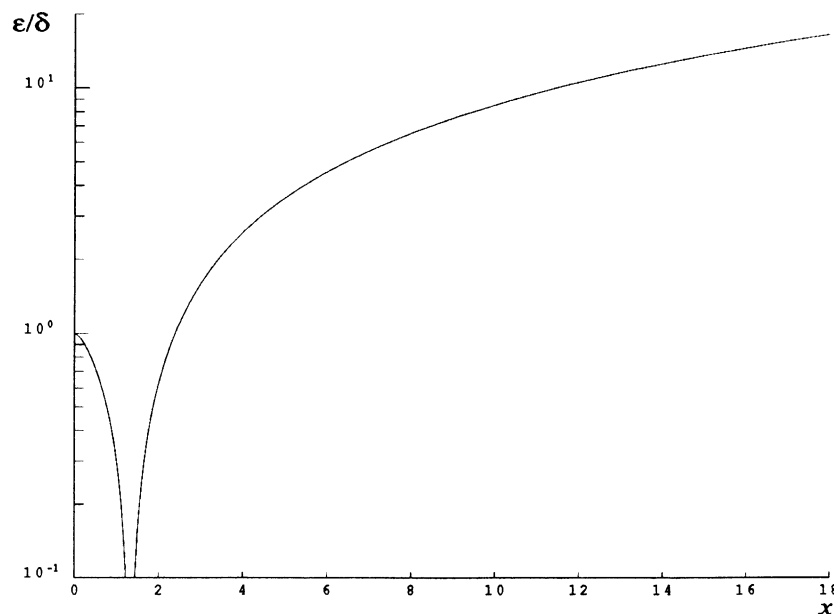
## 7. Accuracy

Let  $\delta$  and  $\varepsilon$  be the relative errors in the argument and result respectively.

If  $\delta$  is somewhat larger than the *machine precision* (i.e. if  $\delta$  is due to data errors etc.), then  $\varepsilon$  and  $\delta$  are approximately related by:

$$\varepsilon \simeq \left| \frac{xK_0(x) - K_1(x)}{K_1(x)} \right| \delta.$$

The following graph shows the behaviour of the error amplification factor  $\left| \frac{xK_0(x) - K_1(x)}{K_1(x)} \right|$ :



However if  $\delta$  is of the same order as the *machine precision*, then rounding errors could make  $\varepsilon$  slightly larger than the above relation predicts.

For small  $x$ ,  $\varepsilon \simeq \delta$  and there is no amplification of errors.

For large  $x$ ,  $\varepsilon \simeq x\delta$  and we have strong amplification of the relative error. Eventually  $K_1$ , which is asymptotically given by  $\frac{e^{-x}}{\sqrt{x}}$ , becomes so small that it cannot be calculated without underflow and hence the routine will return zero. Note that for large  $x$  the errors will be dominated by those of the Fortran intrinsic function EXP.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S18ADF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5, NOUT=6)
*      .. Local Scalars ..
real           X, Y
INTEGER          IFAIL
*      .. External Functions ..
real          S18ADF
EXTERNAL        S18ADF
*      .. Executable Statements ..
WRITE (NOUT,*) 'S18ADF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
WRITE (NOUT,*)
WRITE (NOUT,*) '      X          Y          IFAIL'
WRITE (NOUT,*)
20 READ (NIN,*,END=40) X
   IFAIL = 1
*
   Y = S18ADF(X,IFAIL)
*
   WRITE (NOUT,99999) X, Y, IFAIL
   GO TO 20
40 STOP
*
99999 FORMAT (1X,1P,2e12.3,I7)
END
```

### 9.2. Program Data

```
S18ADF Example Program Data
0.0
0.4
0.6
1.4
1.6
2.5
3.5
6.0
8.0
10.0
-1.0
1000.0
```

**9.3. Program Results**

S18ADF Example Program Results

X	Y	IFAIL
0.000E+00	0.000E+00	1
4.000E-01	2.184E+00	0
6.000E-01	1.303E+00	0
1.400E+00	3.208E-01	0
1.600E+00	2.406E-01	0
2.500E+00	7.389E-02	0
3.500E+00	2.224E-02	0
6.000E+00	1.344E-03	0
8.000E+00	1.554E-04	0
1.000E+01	1.865E-05	0
-1.000E+00	0.000E+00	1
1.000E+03	0.000E+00	0

---



## S18AEF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised terms* and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S18AEF returns the value of the modified Bessel Function  $I_0(x)$ , via the routine name.

## 2. Specification

```

real FUNCTION S18AEF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

## 3. Description

This routine evaluates an approximation to the modified Bessel Function of the first kind  $I_0(x)$ .

**Note:**  $I_0(-x) = I_0(x)$ , so the approximation need only consider  $x \geq 0$ .

The routine is based on three Chebyshev expansions:

For  $0 < x \leq 4$ ,

$$I_0(x) = e^x \sum_{r=0}^{\infty} a_r T_r(t), \quad \text{where } t = 2\left(\frac{x}{4}\right) - 1;$$

For  $4 < x \leq 12$ ,

$$I_0(x) = e^x \sum_{r=0}^{\infty} b_r T_r(t), \quad \text{where } t = \frac{x - 8}{4};$$

For  $x > 12$ ,

$$I_0(x) = \frac{e^x}{\sqrt{x}} \sum_{r=0}^{\infty} c_r T_r(t), \quad \text{where } t = 2\left(\frac{12}{x}\right) - 1.$$

For small  $x$ ,  $I_0(x) \simeq 1$ . This approximation is used when  $x$  is sufficiently small for the result to be correct to *machine precision*.

For large  $x$ , the routine must fail because of the danger of overflow in calculating  $e^x$ .

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 374, 1968.

## 5. Parameters

- 1: X – *real*. *Input*  
*On entry:* the argument  $x$  of the function.
- 2: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

X is too large. On soft failure the routine returns the approximate value of  $I_0(x)$  at the nearest valid argument.

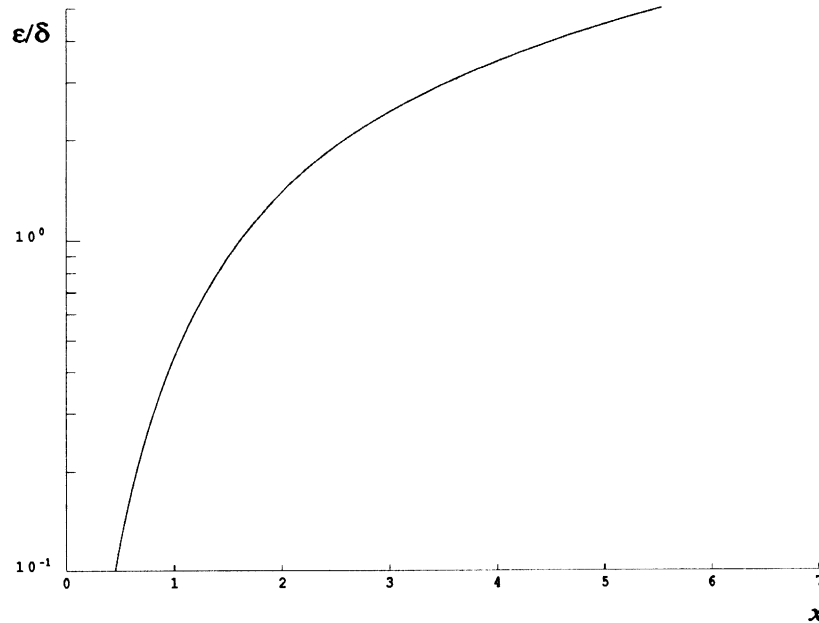
## 7. Accuracy

Let  $\delta$  and  $\varepsilon$  be the relative errors in the argument and result respectively.

If  $\delta$  is somewhat larger than the *machine precision* (i.e. if  $\delta$  is due to data errors etc.), then  $\varepsilon$  and  $\delta$  are approximately related by:

$$\varepsilon \approx \left| \frac{xI_1(x)}{I_0(x)} \right| \delta.$$

The following graph shows the behaviour of the error amplification factor  $\left| \frac{xI_1(x)}{I_0(x)} \right|$ :



However if  $\delta$  is of the same order as *machine precision*, then rounding errors could make  $\varepsilon$  slightly larger than the above relation predicts.

For small  $x$  the amplification factor is approximately  $\frac{x^2}{2}$ , which implies strong attenuation of the error, but in general  $\varepsilon$  can never be less than the *machine precision*.

For large  $x$ ,  $\varepsilon \approx x\delta$  and we have strong amplification of errors. However the routine must fail for quite moderate values of  $x$ , because  $I_0(x)$  would overflow; hence in practice the loss of accuracy for large  $x$  is not excessive. Note that for large  $x$  the errors will be dominated by those of the Fortran intrinsic function EXP.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      S18AEF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S18AEF
      EXTERNAL         S18AEF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S18AEF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S18AEF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2e12.3,I7)
      END

```

### 9.2. Program Data

```

S18AEF Example Program Data
      0.0
      0.5
      1.0
      3.0
      6.0
      8.0
      10.0
      15.0
      20.0
      -1.0

```

### 9.3. Program Results

S18AEF Example Program Results

X	Y	IFAIL
0.000E+00	1.000E+00	0
5.000E-01	1.063E+00	0
1.000E+00	1.266E+00	0
3.000E+00	4.881E+00	0
6.000E+00	6.723E+01	0
8.000E+00	4.276E+02	0
1.000E+01	2.816E+03	0
1.500E+01	3.396E+05	0
2.000E+01	4.356E+07	0
-1.000E+00	1.266E+00	0

---



## S18AFF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S18AFF returns a value for the modified Bessel Function  $I_1(X)$ , via the routine name.

## 2. Specification

```
real FUNCTION S18AFF (X, IFAIL)
      INTEGER      IFAIL
      real         X
```

## 3. Description

This routine evaluates an approximation to the modified Bessel Function of the first kind  $I_1(x)$ .

**Note:**  $I_1(-x) = -I_1(x)$ , so the approximation need only consider  $x \geq 0$ .

The routine is based on three Chebyshev expansions:

For  $0 < x \leq 4$ ,

$$I_1(x) = x \sum_{r=0}^{\infty} a_r T_r(t), \quad \text{where } t = 2\left(\frac{x}{4}\right)^2 - 1;$$

For  $4 < x \leq 12$ ,

$$I_1(x) = e^x \sum_{r=0}^{\infty} b_r T_r(t), \quad \text{where } t = \frac{x-8}{4};$$

For  $x > 12$ ,

$$I_1(x) = \frac{e^x}{\sqrt{x}} \sum_{r=0}^{\infty} c_r T_r(t), \quad \text{where } t = 2\left(\frac{12}{x}\right) - 1.$$

For small  $x$ ,  $I_1(x) \simeq x$ . This approximation is used when  $x$  is sufficiently small for the result to be correct to *machine precision*.

For large  $x$ , the routine must fail because  $I_1(x)$  cannot be represented without overflow.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 374, 1968.

## 5. Parameters

- 1: X – *real*. *Input*  
*On entry:* the argument  $x$  of the function.
- 2: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

X is too large. On soft failure the routine returns the approximate value of  $I_1(x)$  at the nearest valid argument.

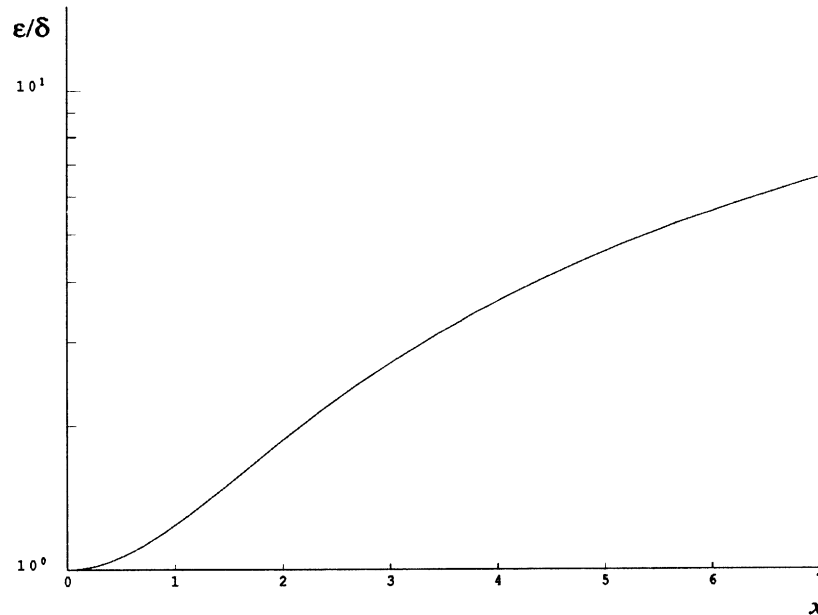
## 7. Accuracy

Let  $\delta$  and  $\varepsilon$  be the relative errors in the argument and result respectively.

If  $\delta$  is somewhat larger than the *machine precision* (i.e. if  $\delta$  is due to data errors etc.), then  $\varepsilon$  and  $\delta$  are approximately related by:

$$\varepsilon \approx \left| \frac{xI_0(x) - I_1(x)}{I_1(x)} \right| \delta.$$

The following graph shows the behaviour of the error amplification factor  $\left| \frac{xI_0(x) - I_1(x)}{I_1(x)} \right|$ :



However if  $\delta$  is of the same order as *machine precision*, then rounding errors could make  $\varepsilon$  slightly larger than the above relation predicts.

For small  $x$ ,  $\varepsilon \approx \delta$  and there is no amplification of errors.

For large  $x$ ,  $\varepsilon \approx x\delta$  and we have strong amplification of errors. However the routine must fail for quite moderate values of  $x$  because  $I_1(x)$  would overflow; hence in practice the loss of accuracy for large  $x$  is not excessive. Note that for large  $x$ , the errors will be dominated by those of the Fortran intrinsic function EXP.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      S18AFF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
real          X, Y
INTEGER          IFAIL
*      .. External Functions ..
real          S18AFF
EXTERNAL        S18AFF
*      .. Executable Statements ..
WRITE (NOUT,*) 'S18AFF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
WRITE (NOUT,*)
WRITE (NOUT,*) '      X          Y          IFAIL'
WRITE (NOUT,*)
20 READ (NIN,*,END=40) X
   IFAIL = 1
*
   Y = S18AFF(X,IFAIL)
*
   WRITE (NOUT,99999) X, Y, IFAIL
   GO TO 20
40 STOP
*
99999 FORMAT (1X,1P,2e12.3,I7)
END

```

### 9.2. Program Data

```

S18AFF Example Program Data
      0.0
      0.5
      1.0
      3.0
      6.0
      8.0
     10.0
     15.0
     20.0
     -1.0

```

### 9.3. Program Results

S18AFF Example Program Results

X	Y	IFAIL
0.000E+00	0.000E+00	0
5.000E-01	2.579E-01	0
1.000E+00	5.652E-01	0
3.000E+00	3.953E+00	0
6.000E+00	6.134E+01	0
8.000E+00	3.999E+02	0
1.000E+01	2.671E+03	0
1.500E+01	3.281E+05	0
2.000E+01	4.245E+07	0
-1.000E+00	-5.652E-01	0

---





## S18CCF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

S18CCF returns a value of the scaled modified Bessel function  $e^x K_0(x)$  via the routine name.

### 2. Specification

```

real FUNCTION S18CCF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

### 3. Description

This routine evaluates an approximation to  $e^x K_0(x)$ , where  $K_0$  is a modified Bessel function of the second kind. The scaling factor  $e^x$  removes most of the variation in  $K_0(x)$ .

The routine uses the same Chebyshev expansions as S18ACF, which returns the unscaled value of  $K_0(x)$ .

### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 374, 1968.

### 5. Parameters

1: X – *real*. *Input*

*On entry:* the argument  $x$  of the function.

*Constraint:*  $X > 0.0$ .

2: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

### 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry  $X \leq 0.0$ ,  $K_0$  is undefined.

On soft failure, S18CCF returns zero.

### 7. Accuracy

Relative errors in the argument are attenuated when propagated into the function value. When the accuracy of the argument is essentially limited by the *machine precision*, the accuracy of the function value will be similarly limited by at most a small multiple of the *machine precision*.

### 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

### 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S18CCF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
real           X, Y
INTEGER          IFAIL
*      .. External Functions ..
real           S18CCF
EXTERNAL        S18CCF
*      .. Executable Statements ..
WRITE (NOUT,*) 'S18CCF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
WRITE (NOUT,*)
WRITE (NOUT,*) '      X          Y          IFAIL'
WRITE (NOUT,*)
20 READ (NIN,*,END=40) X
   IFAIL = 1
*
   Y = S18CCF(X, IFAIL)
*
   WRITE (NOUT,99999) X, Y, IFAIL
   GO TO 20
40 STOP
*
99999 FORMAT (1X,1P,2e12.3,I7)
END
```

### 9.2. Program Data

```
S18CCF Example Program Data
0.0
0.4
0.6
1.4
2.5
10.0
1000.0
-1.0
```

### 9.3. Program Results

S18CCF Example Program Results

X	Y	IFAIL
0.000E+00	0.000E+00	1
4.000E-01	1.663E+00	0
6.000E-01	1.417E+00	0
1.400E+00	9.881E-01	0
2.500E+00	7.595E-01	0
1.000E+01	3.916E-01	0
1.000E+03	3.963E-02	0
-1.000E+00	0.000E+00	1

---

## S18CDF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised terms* and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S18CDF returns a value of the scaled modified Bessel function  $e^x K_1(x)$  via the routine name.

## 2. Specification

```
real FUNCTION S18CDF (X, IFAIL)
      INTEGER          IFAIL
      real             X
```

## 3. Description

This routine evaluates an approximation to  $e^x K_1(x)$ , where  $K_1$  is a modified Bessel function of the second kind. The scaling factor  $e^x$  removes most of the variation in  $K_1(x)$ .

The routine uses the same Chebyshev expansions as S18ADF, which returns the unscaled value of  $K_1(x)$ .

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 374, 1968.

## 5. Parameters

1: X – *real*. *Input*

*On entry:* the argument  $x$  of the function.

*Constraint:*  $X > 0.0$ .

2: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry  $X \leq 0.0$ ,  $K_1$  is undefined. On soft failure S18CDF returns zero.

IFAIL = 2

On entry X is too close to zero, there is a danger of causing overflow. On soft failure, S18CDF returns the value of the function at the smallest permitted value of the argument.

## 7. Accuracy

Relative errors in the argument are attenuated when propagated into the function value. When the accuracy of the argument is essentially limited by the *machine precision*, the accuracy of the function value will be similarly limited by at most a small multiple of the *machine precision*.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S18CDF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
real           X, Y
INTEGER         IFAIL
*      .. External Functions ..
real          S18CDF
EXTERNAL        S18CDF
*      .. Executable Statements ..
WRITE (NOUT,*) 'S18CDF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
WRITE (NOUT,*)
WRITE (NOUT,*) '      X          Y          IFAIL'
WRITE (NOUT,*)
20 READ (NIN,*,END=40) X
   IFAIL = 1
*
   Y = S18CDF(X,IFAIL)
*
   WRITE (NOUT,99999) X, Y, IFAIL
   GO TO 20
40 STOP
*
99999 FORMAT (1X,1P,2e12.3,I7)
      END
```

### 9.2. Program Data

```
S18CDF Example Program Data
      0.0
      0.4
      0.6
      1.4
      2.5
     10.0
    1000.0
     -1.0
```

### 9.3. Program Results

S18CDF Example Program Results

X	Y	IFAIL
0.000E+00	0.000E+00	1
4.000E-01	3.259E+00	0
6.000E-01	2.374E+00	0
1.400E+00	1.301E+00	0
2.500E+00	9.002E-01	0
1.000E+01	4.108E-01	0
1.000E+03	3.965E-02	0
-1.000E+00	0.000E+00	1

## S18CEF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised terms* and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

S18CEF returns a value of the scaled modified Bessel function  $e^{-|x|}I_0(x)$  via the routine name.

### 2. Specification

```

real FUNCTION S18CEF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

### 3. Description

This routine evaluates an approximation to  $e^{-|x|}I_0(x)$ , where  $I_0$  is a modified Bessel function of the first kind. The scaling factor  $e^{-|x|}$  removes most of the variation in  $I_0(x)$ .

The routine uses the same Chebyshev expansions as S18AEF, which returns the unscaled value of  $I_0(x)$ .

### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 374, 1968.

### 5. Parameters

- 1: X – *real*. *Input*  
*On entry:* the argument  $x$  of the function.
- 2: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

### 6. Error Indicators and Warnings

Errors detected by the routine:

There are no actual failure exits from this routine. IFAIL is always set to zero. This parameter is included for compatibility with other routines in this chapter.

### 7. Accuracy

Relative errors in the argument are attenuated when propagated into the function value. When the accuracy of the argument is essentially limited by the *machine precision*, the accuracy of the function value will be similarly limited by at most a small multiple of the *machine precision*.

### 8. Further Comments

For details of the time taken by the routine see appropriate the Users' Note for your implementation.

### 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      S18CEF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S18CEF
      EXTERNAL        S18CEF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S18CEF Example Program Results'
      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S18CEF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2e12.3,I7)
      END

```

## 9.2. Program Data

```

S18CEF Example Program Data
      0.0
      0.5
      1.0
      3.0
      6.0
      10.0
      1000.0
      -1.0

```

## 9.3. Program Results

S18CEF Example Program Results

X	Y	IFAIL
0.000E+00	1.000E+00	0
5.000E-01	6.450E-01	0
1.000E+00	4.658E-01	0
3.000E+00	2.430E-01	0
6.000E+00	1.667E-01	0
1.000E+01	1.278E-01	0
1.000E+03	1.262E-02	0
-1.000E+00	4.658E-01	0

---

## S18CFF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S18CFF returns a value of the scaled modified Bessel function  $e^{-|x|}I_1(x)$  via the routine name.

## 2. Specification

```

real FUNCTION S18CFF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

## 3. Description

This routine evaluates an approximation to  $e^{-|x|}I_1(x)$ , where  $I_1$  is a modified Bessel function of the first kind. The scaling factor  $e^{-|x|}$  removes most of the variation in  $I_1(x)$ .

The routine uses the same Chebyshev expansions as S18AFF, which returns the unscaled value of  $I_1(x)$ .

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 374, 1968.

## 5. Parameters

1: X – *real*. *Input*  
*On entry:* the argument  $x$  of the function.

2: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

There are no actual failure exits from this routine. IFAIL is always set to zero. This parameter is included for compatibility with other routines in this chapter.

## 7. Accuracy

Relative errors in the argument are attenuated when propagated into the function value. When the accuracy of the argument is essentially limited by the *machine precision*, the accuracy of the function value will be similarly limited by at most a small multiple of the *machine precision*.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S18CFF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S18CFF
      EXTERNAL         S18CFF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S18CFF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
        IFAIL = 1
*
        Y = S18CFF(X,IFAIL)
*
        WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2e12.3,I7)
      END
```

### 9.2. Program Data

```
S18CFF Example Program Data
      0.0
      0.5
      1.0
      3.0
      6.0
      10.0
      1000.0
      -1.0
```

### 9.3. Program Results

S18CFF Example Program Results

X	Y	IFAIL
0.000E+00	0.000E+00	0
5.000E-01	1.564E-01	0
1.000E+00	2.079E-01	0
3.000E+00	1.968E-01	0
6.000E+00	1.521E-01	0
1.000E+01	1.213E-01	0
1.000E+03	1.261E-02	0
-1.000E+00	-2.079E-01	0

---



## S18DCF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S18DCF returns a sequence of values for the modified Bessel functions  $K_{\nu+n}(z)$  for complex  $z$ , non-negative  $\nu$  and  $n = 0, 1, \dots, N-1$ , with an option for exponential scaling.

## 2. Specification

```

SUBROUTINE S18DCF (FNU, Z, N, SCALE, CY, NZ, IFAIL)
  INTEGER          N, NZ, IFAIL
  real            FNU
  complex        Z, CY(N)
  CHARACTER*1     SCALE

```

## 3. Description

This subroutine evaluates a sequence of values for the modified Bessel function  $K_{\nu}(z)$ , where  $z$  is complex,  $-\pi < \arg z \leq \pi$ , and  $\nu$  is the real, non-negative order. The  $N$ -member sequence is generated for orders  $\nu, \nu+1, \dots, \nu+N-1$ . Optionally, the sequence is scaled by the factor  $e^z$ .

The routine is derived from the routine CBESK in Amos [2].

Note: although the routine may not be called with  $\nu$  less than zero, for negative orders the formula  $K_{-\nu}(z) = K_{\nu}(z)$  may be used.

When  $N$  is greater than 1, extra values of  $K_{\nu}(z)$  are computed using recurrence relations.

For very large  $|z|$  or  $(\nu+N-1)$ , argument reduction will cause total loss of accuracy, and so no computation is performed. For slightly smaller  $|z|$  or  $(\nu+N-1)$ , the computation is performed but results are accurate to less than half of *machine precision*. If  $|z|$  is very small, near the machine underflow threshold, or  $(\nu+N-1)$  is too large, there is a risk of overflow and so no computation is performed. In all the above cases, a warning is given by the routine.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 358, 1968.
- [2] AMOS, D.E.  
Algorithm 644: A Portable Package for Bessel Functions of a Complex Argument and Nonnegative Order.  
ACM Trans. Math. Software 12, pp. 265-273, 1986.

## 5. Parameters

- 1: FNU – *real*. *Input*  
On entry: the order,  $\nu$ , of the first member of the sequence of functions.  
Constraint: FNU  $\geq$  0.0.
- 2: Z – *complex*. *Input*  
On entry: the argument  $z$  of the functions.  
Constraint: Z  $\neq$  (0.0, 0.0).

- 3: N – INTEGER. *Input*  
*On entry:* the number,  $N$ , of members required in the sequence  $K_\nu(z)$ ,  $K_{\nu+1}(z)$ , ...,  $K_{\nu+N-1}(z)$ .  
*Constraint:*  $N \geq 1$ .
- 4: SCALE – CHARACTER\*1. *Input*  
*On entry:* the scaling option.  
 If SCALE = 'U' or 'u', the results are returned unscaled.  
 If SCALE = 'S' or 's', the results are returned scaled by the factor  $e^z$ .  
*Constraint:* SCALE = 'U', 'u', 'S' or 's'.
- 5: CY(N) – *complex* array. *Output*  
*On exit:* the  $N$  required function values: CY( $i$ ) contains  $K_{\nu+i-1}(z)$ , for  $i = 1, 2, \dots, N$ .
- 6: NZ – INTEGER. *Output*  
*On exit:* the number of components of CY that are set to zero due to underflow. If  $NZ > 0$  and  $\text{Re } z \geq 0.0$ , elements CY(1),CY(2),...,CY(NZ) are set to zero. If  $\text{Re } z < 0.0$ , NZ simply states the number of underflows, and not which elements they are.
- 7: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, FNU < 0.0,  
 or  $Z = (0.0, 0.0)$ ,  
 or  $N < 1$ ,  
 or SCALE  $\neq$  'U', 'u', 'S' or 's'.

IFAIL = 2

No computation has been performed due to the likelihood of overflow, because  $\text{ABS}(Z)$  is less than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 3

No computation has been performed due to the likelihood of overflow, because  $\text{FNU} + N - 1$  is too large – how large depends on  $Z$  and the overflow threshold of the machine.

IFAIL = 4

The computation has been performed, but the errors due to argument reduction in elementary functions make it likely that the results returned by S18DCF are accurate to less than half of *machine precision*. This error exit may occur if either  $\text{ABS}(Z)$  or  $\text{FNU} + N - 1$  is greater than a machine-dependent threshold value (given in the Users' Note for your implementation).

**IFAIL = 5**

No computation has been performed because the errors due to argument reduction in elementary functions mean that all precision in results returned by S18DCF would be lost. This error exit may occur when either  $\text{ABS}(Z)$  or  $\text{FNU} + N - 1$  is greater than a machine-dependent threshold value (given in the Users' Note for your implementation).

**IFAIL = 6**

No results are returned because the algorithm termination condition has not been met. This may occur because the parameters supplied to S18DCF would have caused overflow or underflow.

**7. Accuracy**

All constants in subroutine S18DCF are given to approximately 18 digits of precision. Calling the number of digits of precision in the floating-point arithmetic being used  $t$ , then clearly the maximum number of correct digits in the results obtained is limited by  $p = \min(t, 18)$ . Because of errors in argument reduction when computing elementary functions inside S18DCF, the actual number of correct digits is limited, in general, by  $p - s$ , where  $s \approx \max(1, |\log_{10} |z||, |\log_{10} v|)$  represents the number of digits lost due to the argument reduction. Thus the larger the values of  $|z|$  and  $v$ , the less the precision in the result. If S18DCF is called with  $N > 1$ , then computation of function values via recurrence may lead to some further small loss of accuracy.

If function values which should nominally be identical are computed by calls to S18DCF with different base values of  $v$  and different  $N$ , the computed values may not agree exactly. Empirical tests with modest values of  $v$  and  $z$  have shown that the discrepancy is limited to the least significant 3-4 digits of precision.

**8. Further Comments**

The time taken by the routine for a call of S18DCF is approximately proportional to the value of  $N$ , plus a constant. In general it is much cheaper to call S18DCF with  $N$  greater than 1, rather than to make  $N$  separate calls to S18DCF.

Paradoxically, for some values of  $z$  and  $v$ , it is cheaper to call S18DCF with a larger value of  $N$  than is required, and then discard the extra function values returned. However, it is not possible to state the precise circumstances in which this is likely to occur. It is due to the fact that the base value used to start recurrence may be calculated in different regions for different  $N$ , and the costs in each region may differ greatly.

Note that if the function required is  $K_0(x)$  or  $K_1(x)$ , i.e.  $v = 0.0$  or  $1.0$ , where  $x$  is real and positive, and only a single function value is required, then it may be much cheaper to call S18ACF, S18ADF, S18CCF or S18CDF, depending on whether a scaled result is required or not.

**9. Example**

The following example program prints a caption and then proceeds to read sets of data from the input data stream. The first datum is a value for the order FNU, the second is a complex value for the argument, Z, and the third is a value for the parameter SCALE. The program calls the routine with  $N = 2$  to evaluate the function for orders FNU and FNU + 1, and it prints the results. The process is repeated until the end of the input data stream is encountered.

**9.1. Program Text**

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S18DCF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5, NOUT=6)
      INTEGER          N
      PARAMETER       (N=2)
```

```

*   .. Local Scalars ..
    complex          Z
    real            FNU
    INTEGER          IFAIL, NZ
    CHARACTER*1     SCALE
*   .. Local Arrays ..
    complex          CY(N)
*   .. External Subroutines ..
    EXTERNAL        S18DCF
*   .. Executable Statements ..
    WRITE (NOUT,*) 'S18DCF Example Program Results'
*   Skip heading in data file
    READ (NIN,*)
    WRITE (NOUT,*)
    WRITE (NOUT,99999) 'Calling with N =', N
    WRITE (NOUT,*)
    WRITE (NOUT,*)
+   ' FNU          Z          SCALE          CY(1)          CY(2)
+   NZ IFAIL'
    WRITE (NOUT,*)
20  READ (NIN,*,END=40) FNU, Z, SCALE
    IFAIL = 0
*
    CALL S18DCF(FNU,Z,N,SCALE,CY,NZ,IFAIL)
*
    WRITE (NOUT,99998) FNU, Z, SCALE, CY(1), CY(2), NZ, IFAIL
    GO TO 20
40  STOP
*
99999 FORMAT (1X,A,I2)
99998 FORMAT (1X,F7.4,' ('',F7.3,'','',F7.3,'') ',A,
+           2(' ('',F7.3,'','',F7.3,'')'),I4,I4)
    END

```

## 9.2. Program Data

```

S18DCF Example Program Data
0.00 ( 0.3, 0.4) 'U'
2.30 ( 2.0, 0.0) 'U'
2.12 (-1.0, 0.0) 'U'
5.10 ( 3.0, 2.0) 'U'
5.10 ( 3.0, 2.0) 'S'

```

## 9.3. Program Results

S18DCF Example Program Results

Calling with N = 2

FNU	Z	SCALE	CY(1)	CY(2)	NZ	IFAIL
0.0000	( 0.300, 0.400)	U	( 0.831, -0.803)	( 0.831, -1.735)	0	0
2.3000	( 2.000, 0.000)	U	( 0.325, 0.000)	( 0.909, 0.000)	0	0
2.1200	( -1.000, 0.000)	U	( 1.763, -1.047)	( -8.087, 3.147)	0	0
5.1000	( 3.000, 2.000)	U	( -0.426, 0.243)	( -0.810, 1.255)	0	0
5.1000	( 3.000, 2.000)	S	( -0.880, -9.803)	( -16.150, -25.293)	0	0

## S18DEF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S18DEF returns a sequence of values for the modified Bessel functions  $I_{\nu+n}(z)$  for complex  $z$ , non-negative  $\nu$  and  $n = 0, 1, \dots, N-1$ , with an option for exponential scaling.

## 2. Specification

```

SUBROUTINE S18DEF (FNU, Z, N, SCALE, CY, NZ, IFAIL)
  INTEGER          N, NZ, IFAIL
  real            FNU
  complex        Z, CY(N)
  CHARACTER*1     SCALE

```

## 3. Description

This subroutine evaluates a sequence of values for the modified Bessel function  $I_\nu(z)$ , where  $z$  is complex,  $-\pi < \arg z \leq \pi$ , and  $\nu$  is the real, non-negative order. The  $N$ -member sequence is generated for orders  $\nu, \nu+1, \dots, \nu+N-1$ . Optionally, the sequence is scaled by the factor  $e^{-|\operatorname{Re}z|}$ . The routine is derived from the routine CBESI in Amos [2].

**Note:** although the routine may not be called with  $\nu$  less than zero, for negative orders the formula  $I_{-\nu}(z) = I_\nu(z) + \frac{2}{\pi} \sin(\pi\nu) K_\nu(z)$  may be used (for the Bessel function  $K_\nu(z)$ , see S18DCF).

When  $N$  is greater than 1, extra values of  $I_\nu(z)$  are computed using recurrence relations.

For very large  $|z|$  or  $(\nu+N-1)$ , argument reduction will cause total loss of accuracy, and so no computation is performed. For slightly smaller  $|z|$  or  $(\nu+N-1)$ , the computation is performed but results are accurate to less than half of *machine precision*. If  $\operatorname{real}(z)$  is too large and the unscaled function is required, there is a risk of overflow and so no computation is performed. In all the above cases, a warning is given by the routine.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9, p. 358, 1968.
- [2] AMOS, D.E.  
Algorithm 644: A Portable Package for Bessel Functions of a Complex Argument and Nonnegative Order.  
ACM Trans. Math. Software, 12, pp. 265-273, 1986.

## 5. Parameters

- 1: FNU – *real*. *Input*  
*On entry:* the order,  $\nu$ , of the first member of the sequence of functions.  
*Constraint:* FNU  $\geq$  0.0.
- 2: Z – *complex*. *Input*  
*On entry:* the argument  $z$  of the functions.

- 3: N – INTEGER. *Input*  
*On entry:* the number,  $N$ , of members required in the sequence  $I_\nu(z), I_{\nu+1}(z), \dots, I_{\nu+N-1}(z)$ .  
*Constraint:*  $N \geq 1$ .
- 4: SCALE – CHARACTER\*1. *Input*  
*On entry:* the scaling option.  
 If SCALE = 'U' or 'u', the results are returned unscaled.  
 If SCALE = 'S' or 's', the results are returned scaled by the factor  $e^{-|\operatorname{Re} z|}$ .  
*Constraint:* SCALE = 'U', 'u', 'S' or 's'.
- 5: CY(N) – *complex* array. *Output*  
*On exit:* the  $N$  required function values: CY( $i$ ) contains  $I_{\nu+i-1}(z)$ , for  $i = 1, 2, \dots, N$ .
- 6: NZ – INTEGER. *Output*  
*On exit:* the number of components of CY that are set to zero due to underflow.  
 If  $NZ > 0$ , then elements CY(N–NZ+1), CY(N–NZ+2), ..., CY(N) are set to zero.
- 7: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, FNU < 0.0,  
 or N < 1,  
 or SCALE  $\neq$  'U', 'u', 'S' or 's'.

IFAIL = 2

No computation has been performed due to the likelihood of overflow, because  $\operatorname{real}(Z)$  is greater than a machine-dependent threshold value (given in the Users' Note for your implementation). This error exit can only occur when SCALE = 'U' or 'u'.

IFAIL = 3

The computation has been performed, but the errors due to argument reduction in elementary functions make it likely that the results returned by S18DEF are accurate to less than half of *machine precision*. This error exit may occur when either  $\operatorname{ABS}(Z)$  or  $\operatorname{FNU} + N - 1$  is greater than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 4

No computation has been performed because the errors due to argument reduction in elementary functions mean that all precision in results returned by S18DEF would be lost. This error exit may occur when either  $\operatorname{ABS}(Z)$  or  $\operatorname{FNU} + N - 1$  is greater than a machine-dependent threshold value (given in the Users' Note for your implementation).

IFAIL = 5

No results are returned because the algorithm termination condition has not been met. This may occur because the parameters supplied to S18DEF would have caused overflow or underflow.

## 7. Accuracy

All constants in subroutine S18DEF are given to approximately 18 digits of precision. Calling the number of digits of precision in the floating-point arithmetic being used  $t$ , then clearly the maximum number of correct digits in the results obtained is limited by  $p = \min(t, 18)$ . Because of errors in argument reduction when computing elementary functions inside S18DEF, the actual number of correct digits is limited, in general, by  $p - s$ , where  $s \approx \max(1, |\log_{10} |z||, |\log_{10} v|)$  represents the number of digits lost due to the argument reduction. Thus the larger the values of  $|z|$  and  $v$ , the less the precision in the result. If S18DEF is called with  $N > 1$ , then computation of function values via recurrence may lead to some further small loss of accuracy.

If function values which should nominally be identical are computed by calls to S18DEF with different base values of  $v$  and different  $N$ , the computed values may not agree exactly. Empirical tests with modest values of  $v$  and  $z$  have shown that the discrepancy is limited to the least significant 3-4 digits of precision.

## 8. Further Comments

The time taken by the routine for a call of S18DEF is approximately proportional to the value of  $N$ , plus a constant. In general it is much cheaper to call S18DEF with  $N$  greater than 1, rather than to make  $N$  separate calls to S18DEF.

Paradoxically, for some values of  $z$  and  $v$ , it is cheaper to call S18DEF with a larger value of  $N$  than is required, and then discard the extra function values returned. However, it is not possible to state the precise circumstances in which this is likely to occur. It is due to the fact that the base value used to start recurrence may be calculated in different regions for different  $N$ , and the costs in each region may differ greatly.

Note that if the function required is  $I_0(x)$  or  $I_1(x)$ , i.e.  $v = 0.0$  or  $1.0$ , where  $x$  is real and positive, and only a single function value is required, then it may be much cheaper to call S18AEF, S18AFF, S18CEF or S18CFF, depending on whether a scaled result is required or not.

## 9. Example

The following example program prints a caption and then proceeds to read sets of data from the input data stream. The first datum is a value for the order FNU, the second is a complex value for the argument, Z, and the third is a value for the parameter SCALE. The program calls the routine with  $N = 2$  to evaluate the function for orders FNU and FNU + 1, and it prints the results. The process is repeated until the end of the input data stream is encountered.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised terms* to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S18DEF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5, NOUT=6)
      INTEGER          N
      PARAMETER        (N=2)
*      .. Local Scalars ..
      complex         Z
      real            FNU
      INTEGER          IFAIL, NZ
      CHARACTER*1     SCALE
```

```

*      .. Local Arrays ..
      complex          CY(N)
*      .. External Subroutines ..
      EXTERNAL          S18DEF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S18DEF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Calling with N =', N
      WRITE (NOUT,*)
      WRITE (NOUT,*)
+      FNU              Z              SCALE              CY(1)              CY(2)
+      NZ IFAIL'
      WRITE (NOUT,*)
20  READ (NIN,*,END=40) FNU, Z, SCALE
      IFAIL = 0
*
      CALL S18DEF(FNU,Z,N,SCALE,CY,NZ,IFAIL)
*
      WRITE (NOUT,99998) FNU, Z, SCALE, CY(1), CY(2), NZ, IFAIL
      GO TO 20
40  STOP
*
99999 FORMAT (1X,A,I2)
99998 FORMAT (1X,F7.4,' (' ,F7.3,',',F7.3,') ' ,A,
+           2(' (' ,F7.3,',',F7.3,')'),I4,I4)
      END

```

## 9.2. Program Data

```

S18DEF Example Program Data
0.00 ( 0.3, -0.4) 'U'
2.30 ( 2.0, 0.0) 'U'
2.12 (-1.0, 0.0) 'U'
5.50 (-6.1, 9.8) 'U'
5.50 (-6.1, 9.8) 'S'

```

## 9.3. Program Results

S18DEF Example Program Results

Calling with N = 2

FNU	Z	SCALE	CY(1)	CY(2)	NZ	IFAIL
0.0000	( 0.300, -0.400)	U	( 0.982, -0.059)	( 0.143, -0.203)	0	0
2.3000	( 2.000, 0.000)	U	( 0.500, 0.000)	( 0.142, 0.000)	0	0
2.1200	( -1.000, 0.000)	U	( 0.103, 0.041)	( -0.016, -0.006)	0	0
5.5000	( -6.100, 9.800)	U	( 22.534, 13.710)	( -19.635, -1.660)	0	0
5.5000	( -6.100, 9.800)	S	( 0.051, 0.031)	( -0.044, -0.004)	0	0



## S19AAF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S19AAF returns a value for the Kelvin function  $\text{ber } x$  via the routine name.

## 2. Specification

```
real FUNCTION S19AAF (X, IFAIL)
      INTEGER      IFAIL
      real         X
```

## 3. Description

This routine evaluates an approximation to the Kelvin function  $\text{ber } x$ .

**Note:**  $\text{ber}(-x) = \text{ber } x$ , so the approximation need only consider  $x \geq 0.0$ .

The routine is based on several Chebyshev expansions:

For  $0 \leq x \leq 5$ ,

$$\text{ber } x = \sum_{r=0}^{\infty} a_r T_r(t) \text{ with } t = 2\left(\frac{x}{5}\right)^4 - 1;$$

for  $x > 5$ ,

$$\begin{aligned} \text{ber } x = & \frac{e^{\frac{x}{\sqrt{2}}}}{\sqrt{2\pi x}} \left[ \left(1 + \frac{1}{x}a(t)\right) \cos \alpha + \frac{1}{x}b(t) \sin \alpha \right] \\ & + \frac{e^{\frac{-x}{\sqrt{2}}}}{\sqrt{2\pi x}} \left[ \left(1 + \frac{1}{x}c(t)\right) \sin \beta + \frac{1}{x}d(t) \cos \beta \right] \end{aligned}$$

$$\text{where } \alpha = \frac{x}{\sqrt{2}} - \frac{\pi}{8}, \quad \beta = \frac{x}{\sqrt{2}} + \frac{\pi}{8},$$

and  $a(t)$ ,  $b(t)$ ,  $c(t)$ , and  $d(t)$  are expansions in the variable  $t = \frac{10}{x} - 1$ .

When  $x$  is sufficiently close to zero, the result is set directly to  $\text{ber } 0 = 1.0$ .

For large  $x$ , there is a danger of the result being totally inaccurate, as the error amplification factor grows in an essentially exponential manner; therefore the routine must fail.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9.9, p. 379, 1968.

## 5. Parameters

1: **X** – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

2: **IFAIL** – *INTEGER*.

*Input/Output*

*On entry:* **IFAIL** must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* **IFAIL** = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry ABS(X) is too large for an accurate result to be returned. On soft failure, the routine returns zero.

## 7. Accuracy

Since the function is oscillatory, the absolute error rather than the relative error is important. Let  $E$  be the absolute error in the result and  $\delta$  be the relative error in the argument. If  $\delta$  is somewhat larger than the *machine precision*, then we have:

$$E \simeq \left| \frac{x}{\sqrt{2}} (\text{ber}_1 x + \text{bei}_1 x) \right| \delta$$

(provided  $E$  is within machine bounds).

For small  $x$  the error amplification is insignificant and thus the absolute error is effectively bounded by the *machine precision*.

For medium and large  $x$ , the error behaviour is oscillatory and its amplitude grows like  $\sqrt{\frac{x}{2\pi}} e^{\frac{x}{\sqrt{2}}}$ .

Therefore it is not possible to calculate the function with any accuracy when  $\sqrt{x} e^{\frac{x}{\sqrt{2}}} > \frac{\sqrt{2\pi}}{\delta}$ .

Note that this value of  $x$  is much smaller than the minimum value of  $x$  for which the function overflows.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S19AAF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5, NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S19AAF
      EXTERNAL         S19AAF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S19AAF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X              Y              IFAIL'
      WRITE (NOUT,*)
20    READ (NIN,*,END=40) X
      IFAIL = 1
*
*      Y = S19AAF(X, IFAIL)
*
```

```
        WRITE (NOUT,99999) X, Y, IFAIL
        GO TO 20
    40 STOP
*
99999 FORMAT (1X,1P,2E12.3,I7)
        END
```

## 9.2. Program Data

```
S19AAF Example Program Data
    0.1
    1.0
    2.5
    5.0
   10.0
   15.0
   60.0
   -1.0
```

## 9.3. Program Results

S19AAF Example Program Results

X	Y	IFAIL
1.000E-01	1.000E+00	0
1.000E+00	9.844E-01	0
2.500E+00	4.000E-01	0
5.000E+00	-6.230E+00	0
1.000E+01	1.388E+02	0
1.500E+01	-2.967E+03	0
6.000E+01	0.000E+00	1
-1.000E+00	9.844E-01	0

---



## S19ABF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S19ABF returns a value for the Kelvin function  $\text{bei } x$  via the routine name.

## 2. Specification

```
real FUNCTION S19ABF (X, IFAIL)
      INTEGER      IFAIL
      real         X
```

## 3. Description

This routine evaluates an approximation to the Kelvin function  $\text{bei } x$ .

**Note:**  $\text{bei } -x = \text{bei } x$ , so the approximation need only consider  $x \geq 0.0$ .

The routine is based on several Chebyshev expansions:

For  $0 \leq x \leq 5$ ,

$$\text{bei } x = \frac{x^2}{4} \sum_{r=0}^{\infty} a_r T_r(t), \quad \text{with } t = 2\left(\frac{x}{5}\right)^4 - 1;$$

For  $x > 5$ ,

$$\begin{aligned} \text{bei } x = & \frac{e^{\frac{x}{\sqrt{2}}}}{\sqrt{2\pi x}} \left[ \left(1 + \frac{1}{x} a(t)\right) \sin \alpha - \frac{1}{x} b(t) \cos \alpha \right] \\ & + \frac{e^{\frac{x}{\sqrt{2}}}}{\sqrt{2\pi x}} \left[ \left(1 + \frac{1}{x} c(t)\right) \cos \beta - \frac{1}{x} d(t) \sin \beta \right] \end{aligned}$$

$$\text{where } \alpha = \frac{x}{\sqrt{2}} - \frac{\pi}{8}, \quad \beta = \frac{x}{\sqrt{2}} + \frac{\pi}{8},$$

and  $a(t)$ ,  $b(t)$ ,  $c(t)$ , and  $d(t)$  are expansions in the variable  $t = \frac{10}{x} - 1$ .

When  $x$  is sufficiently close to zero, the result is computed as  $\text{bei } x = \frac{x^2}{4}$ . If this result would underflow, the result returned is  $\text{bei } x = 0.0$ .

For large  $x$ , there is a danger of the result being totally inaccurate, as the error amplification factor grows in an essentially exponential manner; therefore the routine must fail.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9.9, p. 379, 1968.

## 5. Parameters

1:  $X$  – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry ABS(X) is too large for an accurate result to be returned. On soft failure, the routine returns zero.

## 7. Accuracy

Since the function is oscillatory, the absolute error rather than the relative error is important. Let  $E$  be the absolute error in the function, and  $\delta$  be the relative error in the argument. If  $\delta$  is somewhat larger than the *machine precision*, then we have:

$$E \approx \left| \frac{x}{\sqrt{2}} (-\text{ber}_1 x + \text{bei}_1 x) \right| \delta$$

(provided  $E$  is within machine bounds).

For small  $x$  the error amplification is insignificant and thus the absolute error is effectively bounded by the *machine precision*.

For medium and large  $x$ , the error behaviour is oscillatory and its amplitude grows like  $\sqrt{\frac{x}{2\pi}} e^{\frac{x}{\sqrt{2}}}$ . Therefore it is impossible to calculate the functions with any accuracy when  $\sqrt{x} e^{\frac{x}{\sqrt{2}}} > \frac{\sqrt{2\pi}}{\delta}$ . Note that this value of  $x$  is much smaller than the minimum value of  $x$  for which the function overflows.

## 8. Further Comments

For details of the time taken by the routine see the Users' Note for your implementation.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S19ABF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
real           X, Y
INTEGER          IFAIL
*      .. External Functions ..
real           S19ABF
EXTERNAL        S19ABF
*      .. Executable Statements ..
WRITE (NOUT,*) 'S19ABF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
WRITE (NOUT,*)
WRITE (NOUT,*) '      X              Y              IFAIL'
WRITE (NOUT,*)
20 READ (NIN,*,END=40) X
   IFAIL = 1
*
*      Y = S19ABF(X, IFAIL)
*
```

```
        WRITE (NOUT,99999) X, Y, IFAIL
        GO TO 20
    40 STOP
*
99999 FORMAT (1X,1P,2E12.3,I7)
END
```

## 9.2. Program Data

S19ABF Example Program Data

```
0.1
1.0
2.5
5.0
10.0
15.0
60.0
-1.0
```

## 9.3. Program Results

S19ABF Example Program Results

X	Y	IFAIL
1.000E-01	2.500E-03	0
1.000E+00	2.496E-01	0
2.500E+00	1.457E+00	0
5.000E+00	1.160E-01	0
1.000E+01	5.637E+01	0
1.500E+01	-2.953E+03	0
6.000E+01	0.000E+00	1
-1.000E+00	2.496E-01	0

---





## S19ACF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised terms* and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S19ACF returns a value for the Kelvin function  $\ker x$ , via the routine name.

## 2. Specification

```
real FUNCTION S19ACF (X, IFAIL)
      INTEGER      IFAIL
      real         X
```

## 3. Description

This routine evaluates an approximation to the Kelvin function  $\ker x$ .

Note: for  $x < 0$  the function is undefined and at  $x = 0$  it is infinite so we need only consider  $x > 0$ .

The routine is based on several Chebyshev expansions:

For  $0 < x \leq 1$ ,

$$\ker x = -f(t)\log x + \frac{\pi}{16}x^2 g(t) + y(t)$$

where  $f(t)$ ,  $g(t)$  and  $y(t)$  are expansions in the variable  $t = 2x^4 - 1$ ;

For  $1 < x \leq 3$ ,

$$\ker x = \exp\left(-\frac{11}{16}x\right)q(t)$$

where  $q(t)$  is an expansion in the variable  $t = x - 2$ ;

For  $x > 3$ ,

$$\ker x = \sqrt{\frac{\pi}{2x}} e^{-\frac{x}{\sqrt{2}}} \left[ \left(1 + \frac{1}{x}c(t)\right) \cos \beta - \frac{1}{x}d(t) \sin \beta \right]$$

where  $\beta = \frac{x}{\sqrt{2}} + \frac{\pi}{8}$ , and  $c(t)$  and  $d(t)$  are expansions in the variable  $t = \frac{6}{x} - 1$ .

When  $x$  is sufficiently close to zero, the result is computed as

$$\ker x = -\gamma - \log\left(\frac{x}{2}\right) + \left(\pi - \frac{3}{8}x^2\right)\frac{x^2}{16}$$

and when  $x$  is even closer to zero, simply as

$$\ker x = -\gamma - \log\left(\frac{x}{2}\right).$$

For large  $x$ ,  $\ker x$  is asymptotically given by  $\sqrt{\frac{\pi}{2x}} e^{-\frac{x}{\sqrt{2}}}$  and this becomes so small that it cannot be computed without underflow and the routine fails.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9.9, p. 379, 1968.

## 5. Parameters

### 1: X – real.

Input

On entry: the argument  $x$  of the function.

Constraint:  $X > 0$ .

### 2: IFAIL – INTEGER.

Input/Output

On entry: IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

On exit: IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry X is too large, the result underflows. On soft failure, the routine returns zero.

IFAIL = 2

On entry  $X \leq 0$ , the function is undefined. On soft failure the routine returns zero.

## 7. Accuracy

Let  $E$  be the absolute error in the result,  $\varepsilon$  be the relative error in the result and  $\delta$  be the relative error in the argument. If  $\delta$  is somewhat larger than the *machine precision*, then we have:

$$E \simeq \left| \frac{x}{\sqrt{2}} (\ker_1 x + \text{kei}_1 x) \right| \delta,$$

$$\varepsilon \simeq \left| \frac{x \ker_1 x + \text{kei}_1 x}{\ker x} \right| \delta.$$

For very small  $x$ , the relative error amplification factor is approximately given by  $\frac{1}{|\log x|}$ , which implies a strong attenuation of relative error. However,  $\varepsilon$  in general cannot be less than the *machine precision*.

For small  $x$ , errors are damped by the function and hence are limited by the *machine precision*.

For medium and large  $x$ , the error behaviour, like the function itself, is oscillatory, and hence only the absolute accuracy for the function can be maintained. For this range of  $x$ , the amplitude

of the absolute error decays like  $\sqrt{\frac{\pi x}{2}} e^{-\frac{x}{\sqrt{2}}}$  which implies a strong attenuation of error. Eventually,

$\ker x$ , which asymptotically behaves like  $\sqrt{\frac{\pi}{2x}} e^{-\frac{x}{\sqrt{2}}}$ , becomes so small that it cannot be calculated without causing underflow, and the routine returns zero. Note that for large  $x$  the errors are dominated by those of the Fortran intrinsic function EXP.

## 8. Further Comments

Underflow may occur for a few values of  $x$  close to the zeros of  $\ker x$ , below the limit which causes a failure with IFAIL = 1.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S19ACF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S19ACF
      EXTERNAL         S19ACF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S19ACF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S19ACF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2e12.3,I7)
      END
```

### 9.2. Program Data

```
S19ACF Example Program Data
      0.0
      0.1
      1.0
      2.5
      5.0
      10.0
      15.0
      1100.0
      -1.0
```

### 9.3. Program Results

S19ACF Example Program Results

X	Y	IFAIL
0.000E+00	0.000E+00	2
1.000E-01	2.420E+00	0
1.000E+00	2.867E-01	0
2.500E+00	-6.969E-02	0
5.000E+00	-1.151E-02	0
1.000E+01	1.295E-04	0
1.500E+01	-1.514E-08	0
1.100E+03	0.000E+00	1
-1.000E+00	0.000E+00	2

---



## S19ADF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

S19ADF returns a value for the Kelvin function  $\text{kei } x$  via the routine name.

### 2. Specification

```
real FUNCTION S19ADF (X, IFAIL)
      INTEGER          IFAIL
      real             X
```

### 3. Description

This routine evaluates an approximation to the Kelvin function  $\text{kei } x$ .

**Note:** for  $x < 0$  the function is undefined, so we need only consider  $x \geq 0$ .

The routine is based on several Chebyshev expansions:

For  $0 \leq x \leq 1$ ,

$$\text{kei } x = -\frac{\pi}{4}f(t) + \frac{x^2}{4}[-g(t)\log x + v(t)]$$

where  $f(t)$ ,  $g(t)$  and  $v(t)$  are expansions in the variable  $t = 2x^4 - 1$ ;

For  $1 < x \leq 3$ ,

$$\text{kei } x = \exp\left(-\frac{9}{8}x\right)u(t)$$

where  $u(t)$  is an expansion in the variable  $t = x - 2$ ;

For  $x > 3$ ,

$$\text{kei } x = \sqrt{\frac{\pi}{2x}}e^{-\frac{x}{\sqrt{2}}}\left[\left(1+\frac{1}{x}\right)c(t)\sin\beta + \frac{1}{x}d(t)\cos\beta\right]$$

where  $\beta = \frac{x}{\sqrt{2}} + \frac{\pi}{8}$ , and  $c(t)$  and  $d(t)$  are expansions in the variable  $t = \frac{6}{x} - 1$ .

For  $x < 0$ , the function is undefined, and hence the routine fails and returns zero.

When  $x$  is sufficiently close to zero, the result is computed as

$$\text{kei } x = -\frac{\pi}{4} + \left(1-\gamma-\log\left(\frac{x}{2}\right)\right)\frac{x^2}{4}$$

and when  $x$  is even closer to zero simply as

$$\text{kei } x = -\frac{\pi}{4}.$$

For large  $x$ ,  $\text{kei } x$  is asymptotically given by  $\sqrt{\frac{\pi}{2x}}e^{-\frac{x}{\sqrt{2}}}$  and this becomes so small that it cannot be computed without underflow and the routine fails.

### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 9.9, p. 379, 1968.

## 5. Parameters

1:  $X$  – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

*Constraint:*  $X \geq 0$ .

2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry  $X$  is too large, the result underflows. On soft failure, the routine returns zero.

IFAIL = 2

On entry  $X < 0$ , the function is undefined. On soft failure the routine returns zero.

## 7. Accuracy

Let  $E$  be the absolute error in the result, and  $\delta$  be the relative error in the argument. If  $\delta$  is somewhat larger than the machine representation error, then we have:

$$E \approx \left| \frac{x}{\sqrt{2}} (-\ker_1 x + \operatorname{kei}_1 x) \right| \delta.$$

For small  $x$ , errors are attenuated by the function and hence are limited by the *machine precision*.

For medium and large  $x$ , the error behaviour, like the function itself, is oscillatory and hence only absolute accuracy of the function can be maintained. For this range of  $x$ , the amplitude of the

absolute error decays like  $\sqrt{\frac{\pi x}{2}} e^{-\frac{x}{\sqrt{2}}}$ , which implies a strong attenuation of error. Eventually,  $\operatorname{kei} x$ ,

which is asymptotically given by  $\sqrt{\frac{\pi}{2x}} e^{-\frac{x}{\sqrt{2}}}$ , becomes so small that it cannot be calculated without causing underflow and therefore the routine returns zero. Note that for large  $x$ , the errors are dominated by those of the Fortran intrinsic function EXP.

## 8. Further Comments

Underflow may occur for a few values of  $x$  close to the zeros of  $\operatorname{kei} x$ , below the limit which causes a failure with IFAIL = 1.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S19ADF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5, NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
```

```

*      .. External Functions ..
      real          S19ADF
      EXTERNAL     S19ADF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S19ADF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S19ADF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END

```

## 9.2. Program Data

```

S19ADF Example Program Data
      0.0
      0.1
      1.0
      2.5
      5.0
     10.0
     15.0
    1100.0
     -1.0

```

## 9.3. Program Results

S19ADF Example Program Results

X	Y	IFAIL
0.000E+00	-7.854E-01	0
1.000E-01	-7.769E-01	0
1.000E+00	-4.950E-01	0
2.500E+00	-1.107E-01	0
5.000E+00	1.119E-02	0
1.000E+01	-3.075E-04	0
1.500E+01	7.963E-06	0
1.100E+03	0.000E+00	1
-1.000E+00	0.000E+00	2

---





## S20ACF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S20ACF returns a value for the Fresnel Integral  $S(x)$ , via the routine name.

## 2. Specification

```

real FUNCTION S20ACF (X, IFAIL)
      INTEGER          IFAIL
      real             X

```

## 3. Description

This routine evaluates an approximation to the Fresnel Integral

$$S(x) = \int_0^x \sin\left(\frac{\pi}{2}t^2\right) dt.$$

Note:  $S(x) = -S(-x)$ , so the approximation need only consider  $x \geq 0.0$ .

The routine is based on three Chebyshev expansions:

For  $0 < x \leq 3$ ,

$$S(x) = x^3 \sum_{r=0}^{\prime} a_r T_r(t), \quad \text{with } t = 2\left(\frac{x}{3}\right)^4 - 1;$$

For  $x > 3$ ,

$$S(x) = \frac{1}{2} - \frac{f(x)}{x} \cos\left(\frac{\pi}{2}x^2\right) - \frac{g(x)}{x^3} \sin\left(\frac{\pi}{2}x^2\right),$$

$$\text{where } f(x) = \sum_{r=0}^{\prime} b_r T_r(t),$$

$$\text{and } g(x) = \sum_{r=0}^{\prime} c_r T_r(t), \quad \text{with } t = 2\left(\frac{3}{x}\right)^4 - 1.$$

For small  $x$ ,  $S(x) \simeq \frac{\pi}{6}x^3$ . This approximation is used when  $x$  is sufficiently small for the result to be correct to *machine precision*. For very small  $x$ , this approximation would underflow; the result is then set exactly to zero.

For large  $x$ ,  $f(x) \simeq \frac{1}{\pi}$  and  $g(x) \simeq \frac{1}{\pi^2}$ . Therefore for moderately large  $x$ , when  $\frac{1}{\pi^2 x^3}$  is negligible compared with  $\frac{1}{2}$ , the second term in the approximation for  $x > 3$  may be dropped. For very large  $x$ , when  $\frac{1}{\pi x}$  becomes negligible,  $S(x) \simeq \frac{1}{2}$ . However there will be considerable difficulties in calculating  $\cos\left(\frac{\pi}{2}x^2\right)$  accurately before this final limiting value can be used. Since  $\cos\left(\frac{\pi}{2}x^2\right)$  is periodic, its value is essentially determined by the fractional part of  $x^2$ . If  $x^2 = N + \theta$  where  $N$  is an integer and  $0 \leq \theta < 1$ , then  $\cos\left(\frac{\pi}{2}x^2\right)$  depends on  $\theta$  and on  $N$  modulo 4. By exploiting this fact, it is possible to retain significance in the calculation of  $\cos\left(\frac{\pi}{2}x^2\right)$  either all the way to the very large  $x$  limit, or at least until the integer part of  $\frac{x}{2}$  is equal to the maximum integer allowed on the machine.

4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 7, p. 300, 1968.

5. Parameters

1: X – *real*. *Input*  
*On entry:* the argument  $x$  of the function.

2: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

6. Error Indicators and Warnings

Errors detected by the routine:

There are no error exits from this routine. The parameter IFAIL is included for consistency with other routines in this chapter.

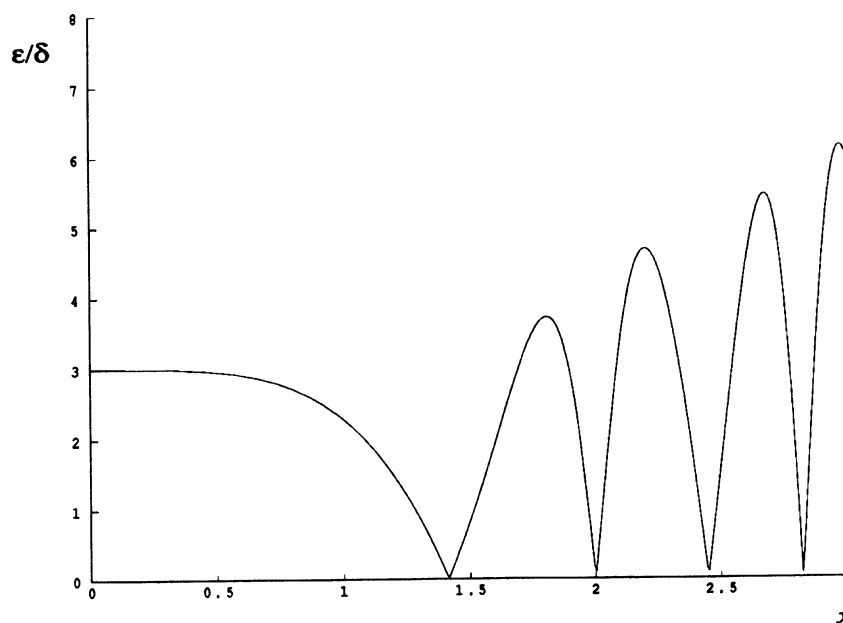
7. Accuracy

Let  $\delta$  and  $\epsilon$  be the relative errors in the argument and result respectively.

If  $\delta$  is somewhat larger than the *machine precision* (i.e. if  $\delta$  is due to data errors etc.), then  $\epsilon$  and  $\delta$  are approximately related by:

$$\epsilon \approx \left| \frac{x \sin\left(\frac{\pi x^2}{2}\right)}{S(x)} \right| \delta.$$

The following graph shows the behaviour of the error amplification factor  $\left| \frac{x \sin\left(\frac{\pi x^2}{2}\right)}{S(x)} \right|$ :



However if  $\delta$  is of the same order as the *machine precision*, then rounding errors could make  $\epsilon$  slightly larger than the above relation predicts.

For small  $x$ ,  $\varepsilon \simeq 3\delta$  and hence there is only moderate amplification of relative error. Of course for very small  $x$  where the correct result would underflow and exact zero is returned, relative error-control is lost.

For moderately large values of  $x$ ,

$$|\varepsilon| \simeq \left| 2x \sin\left(\frac{\pi}{2}x^2\right) \right| |\delta|$$

and the result will be subject to increasingly large amplification of errors. However the above relation breaks down for large values of  $x$  (i.e. when  $\frac{1}{x^2}$  is of the order of the *machine precision*); in this region the relative error in the result is essentially bounded by  $\frac{2}{\pi x}$ .

Hence the effects of error amplification are limited and at worst the relative error loss should not exceed half the possible number of significant figures.

## 8. Further Comments

None.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S20ACF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5, NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S20ACF
      EXTERNAL         S20ACF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S20ACF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20     READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S20ACF(X, IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40     STOP
*
99999  FORMAT (1X,1P,2E12.3,I7)
      END
```

**9.2. Program Data**

S20ACF Example Program Data

```
0.0
0.5
1.0
2.0
4.0
5.0
6.0
8.0
10.0
-1.0
1000.0
```

**9.3. Program Results**

S20ACF Example Program Results

X	Y	IFAIL
0.000E+00	0.000E+00	0
5.000E-01	6.473E-02	0
1.000E+00	4.383E-01	0
2.000E+00	3.434E-01	0
4.000E+00	4.205E-01	0
5.000E+00	4.992E-01	0
6.000E+00	4.470E-01	0
8.000E+00	4.602E-01	0
1.000E+01	4.682E-01	0
-1.000E+00	-4.383E-01	0
1.000E+03	4.997E-01	0

---

## S20ADF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

S20ADF returns a value for the Fresnel Integral  $C(x)$ , via the routine name.

### 2. Specification

```

real FUNCTION S20ADF (X, IFAIL)
      INTEGER      IFAIL
      real        X

```

### 3. Description

This routine evaluates an approximation to the Fresnel Integral

$$C(x) = \int_0^x \cos\left(\frac{\pi}{2}t^2\right) dt.$$

**Note:**  $C(x) = -C(-x)$ , so the approximation need only consider  $x \geq 0.0$ .

The routine is based on three Chebyshev expansions:

For  $0 < x \leq 3$ ,

$$C(x) = x \sum_{r=0}^{\prime} a_r T_r(t), \quad \text{with } t = 2\left(\frac{x}{3}\right)^4 - 1;$$

For  $x > 3$ ,

$$C(x) = \frac{1}{2} + \frac{f(x)}{x} \sin\left(\frac{\pi}{2}x^2\right) - \frac{g(x)}{x^3} \cos\left(\frac{\pi}{2}x^2\right),$$

$$\text{where } f(x) = \sum_{r=0}^{\prime} b_r T_r(t),$$

$$\text{and } g(x) = \sum_{r=0}^{\prime} c_r T_r(t), \quad \text{with } t = 2\left(\frac{3}{x}\right)^4 - 1.$$

For small  $x$ ,  $C(x) \simeq x$ . This approximation is used when  $x$  is sufficiently small for the result to be correct to *machine precision*.

For large  $x$ ,  $f(x) \simeq \frac{1}{\pi}$  and  $g(x) \simeq \frac{1}{\pi^2}$ . Therefore for moderately large  $x$ , when  $\frac{1}{\pi^2 x^3}$  is negligible compared with  $\frac{1}{2}$ , the second term in the approximation for  $x > 3$  may be dropped.

For very large  $x$ , when  $\frac{1}{\pi x}$  becomes negligible,  $C(x) \simeq \frac{1}{2}$ . However there will be considerable difficulties in calculating  $\sin\left(\frac{\pi}{2}x^2\right)$  accurately before this final limiting value can be used. Since  $\sin\left(\frac{\pi}{2}x^2\right)$  is periodic, its value is essentially determined by the fractional part of  $x^2$ . If  $x^2 = N + \theta$ , where  $N$  is an integer and  $0 \leq \theta < 1$ , then  $\sin\left(\frac{\pi}{2}x^2\right)$  depends on  $\theta$  and on  $N$  modulo 4. By exploiting this fact, it is possible to retain some significance in the calculation of  $\sin\left(\frac{\pi}{2}x^2\right)$  either all the way to the very large  $x$  limit, or at least until the integer part of  $\frac{x}{2}$  is equal to the maximum integer allowed on the machine.

#### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 7, p. 300, 1968.

#### 5. Parameters

1: X – *real*.

*Input*

*On entry:* the argument  $x$  of the function.

2: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

#### 6. Error Indicators and Warnings

Errors detected by the routine:

There are no actual failure exits from this routine. The parameter IFAIL is included for consistency with other routines in this chapter.

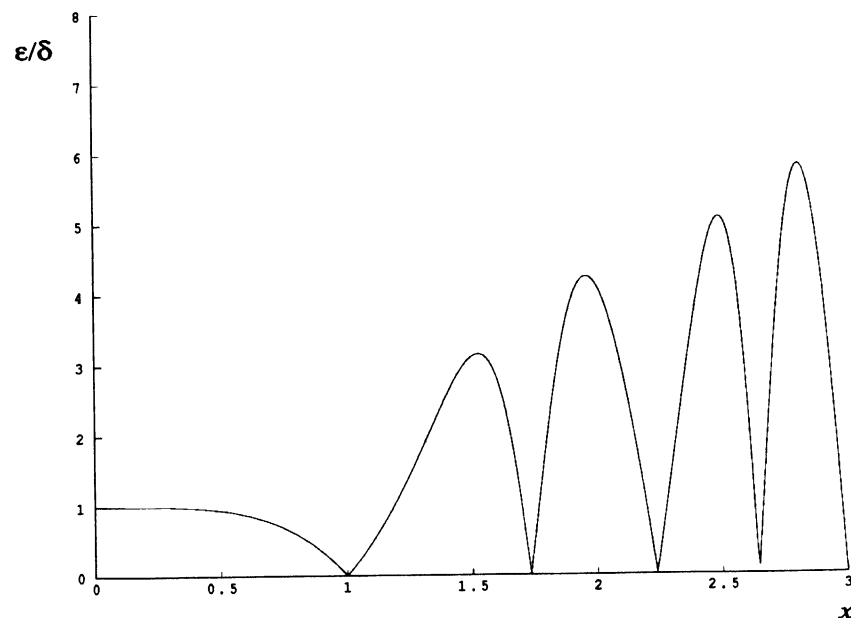
#### 7. Accuracy

Let  $\delta$  and  $\epsilon$  be the relative errors in the argument and result respectively.

If  $\delta$  is somewhat larger than the *machine precision* (i.e if  $\delta$  is due to data errors etc.), then  $\epsilon$  and  $\delta$  are approximately related by:

$$\epsilon \approx \left| \frac{x \cos\left(\frac{\pi}{2}x^2\right)}{C(x)} \right| \delta.$$

The following graph shows the behaviour of the error amplification factor  $\left| \frac{x \cos\left(\frac{\pi}{2}x^2\right)}{C(x)} \right|$ :



However if  $\delta$  is of the same order as the *machine precision*, then rounding errors could make  $\epsilon$  slightly larger than the above relation predicts.

For small  $x$ ,  $\varepsilon \approx \delta$  and there is no amplification of relative error.

For moderately large values of  $x$ ,

$$|\varepsilon| \approx \left| 2x \cos\left(\frac{\pi}{2}x^2\right) \right| |\delta|$$

and the result will be subject to increasingly large amplification of errors. However the above relation breaks down for large values of  $x$  (i.e. when  $\frac{1}{x^2}$  is of the order of the *machine precision*); in this region the relative error in the result is essentially bounded by  $\frac{2}{\pi x}$ .

Hence the effects of error amplification are limited and at worst the relative error loss should not exceed half the possible number of significant figures.

## 8. Further Comments

None.

## 9. Example

The following program reads values of the argument  $x$  from a file, evaluates the function at each value of  $x$  and prints the results.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S20ADF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real            X, Y
      INTEGER          IFAIL
*      .. External Functions ..
      real            S20ADF
      EXTERNAL         S20ADF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S20ADF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X          Y          IFAIL'
      WRITE (NOUT,*)
20    READ (NIN,*,END=40) X
      IFAIL = 1
*
      Y = S20ADF(X,IFAIL)
*
      WRITE (NOUT,99999) X, Y, IFAIL
      GO TO 20
40    STOP
*
99999 FORMAT (1X,1P,2e12.3,I7)
      END
```

## 9.2. Program Data

S20ADF Example Program Data

0.0
0.5
1.0
2.0
4.0
5.0
6.0
8.0
10.0
-1.0
1000.0

## 9.3. Program Results

S20ADF Example Program Results

X	Y	IFAIL
0.000E+00	0.000E+00	0
5.000E-01	4.923E-01	0
1.000E+00	7.799E-01	0
2.000E+00	4.883E-01	0
4.000E+00	4.984E-01	0
5.000E+00	5.636E-01	0
6.000E+00	4.995E-01	0
8.000E+00	4.998E-01	0
1.000E+01	4.999E-01	0
-1.000E+00	-7.799E-01	0
1.000E+03	5.000E-01	0

---



## S21BAF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S21BAF returns a value of an elementary integral, which occurs as a degenerate case of an elliptic integral of the first kind, via the routine name.

## 2. Specification

```

real FUNCTION S21BAF (X, Y, IFAIL)
      INTEGER      IFAIL
      real        X, Y

```

## 3. Description

This routine calculates an approximate value for the integral

$$R_C(x,y) = \frac{1}{2} \int_0^{\infty} \frac{dt}{\sqrt{t+x} (t+y)}$$

where  $x \geq 0$  and  $y \neq 0$ .

This function, which is related to the logarithm or inverse hyperbolic functions for  $y < x$  and to inverse circular functions if  $x < y$ , arises as a degenerate form of the elliptic integral of the first kind. If  $y < 0$ , the result computed is the Cauchy principal value of the integral.

The basic algorithm, which is due to Carlson [2] and [3], is to reduce the arguments recursively towards their mean by the system:

$$\begin{aligned} x_0 &= x, & y_0 &= y \\ \mu_n &= (x_n + 2y_n)/3, & S_n &= (y_n - x_n)/3\mu_n \\ \lambda_n &= y_n + 2\sqrt{x_n y_n} \\ x_{n+1} &= (x_n + \lambda_n)/4, & y_{n+1} &= (y_n + \lambda_n)/4. \end{aligned}$$

The quantity  $|S_n|$  for  $n = 0, 1, 2, 3, \dots$  decreases with increasing  $n$ , eventually  $|S_n| \sim 1/4^n$ . For small enough  $S_n$  the required function value can be approximated by the first few terms of the Taylor series about the mean. That is

$$R_C(x,y) = \left( 1 + \frac{3S_n^2}{10} + \frac{S_n^3}{7} + \frac{3S_n^4}{8} + \frac{9S_n^5}{22} \right) / \sqrt{\mu_n}.$$

The truncation error involved in using this approximation is bounded by  $16|S_n|^6 / (1 - 2|S_n|)$  and the recursive process is stopped when  $S_n$  is small enough for this truncation error to be negligible compared to the *machine precision*.

Within the domain of definition, the function value is itself representable for all representable values of its arguments. However, for values of the arguments near the extremes the above algorithm must be modified so as to avoid causing underflows or overflows in intermediate steps. In extreme regions arguments are pre-scaled away from the extremes and compensating scaling of the result is done before returning to the calling program.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 17, 1968.
- [2] CARLSON, B.C.  
Computing Elliptic Integrals by Duplication.  
Department of Physics, Iowa State University, Preprint 1978.

- [3] CARLSON, B.C.  
A Table of Elliptic Integrals of the Third Kind.  
Math. Comp. 51, pp. 267-280, 1988.

## 5. Parameters

- 1: X – *real*. Input  
2: Y – *real*. Input

*On entry:* the arguments  $x$  and  $y$  of the function, respectively.

*Constraint:*  $X \geq 0.0$  and  $Y \neq 0.0$ .

- 3: IFAIL – INTEGER. Input/Output

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry  $X < 0.0$ ; the function is undefined.

IFAIL = 2

On entry  $Y = 0.0$ ; the function is undefined.

On soft failure the routine returns zero.

## 7. Accuracy

In principle the routine is capable of producing full *machine precision*. However round off errors in internal arithmetic will result in slight loss of accuracy. This loss should never be excessive as the algorithm does not involve any significant amplification of round off error. It is reasonable to assume that the result is accurate to within a small multiple of the *machine precision*.

## 8. Further Comments

Users should consult the Chapter Introduction which shows the relationship of this function to the classical definitions of the elliptic integrals.

## 9. Example

This example program simply generates a small set of non-extreme arguments which are used with the routine to produce the table of low accuracy results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S21BAF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER       (NOUT=6)
*      .. Local Scalars ..
      real            RC, X, Y
      INTEGER          IFAIL, IX
*      .. External Functions ..
      real            S21BAF
      EXTERNAL         S21BAF
```

```
*      .. Executable Statements ..
WRITE (NOUT,*) 'S21BAF Example Program Results'
WRITE (NOUT,*)
WRITE (NOUT,*) '      X      Y      S21BAF  IFAIL'
WRITE (NOUT,*)
DO 20 IX = 1, 3
  X = IX*0.5e0
  Y = 1.0e0
  IFAIL = 1
*
      RC = S21BAF(X,Y,IFAIL)
*
      WRITE (NOUT,99999) X, Y, RC, IFAIL
20 CONTINUE
STOP
*
99999 FORMAT (1X,2F7.2,F12.4,I5)
END
```

## 9.2. Program Data

None.

## 9.3. Program Results

S21BAF Example Program Results

X	Y	S21BAF	IFAIL
0.50	1.00	1.1107	0
1.00	1.00	1.0000	0
1.50	1.00	0.9312	0

---



## S21BBF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S21BBF returns a value of the symmetrised elliptic integral of the first kind, via the routine name.

## 2. Specification

```
real FUNCTION S21BBF (X, Y, Z, IFAIL)
  INTEGER      IFAIL
  real        X, Y, Z
```

## 3. Description

This routine calculates an approximation to the integral

$$R_F(x,y,z) = \frac{1}{2} \int_0^{\infty} \frac{dt}{\sqrt{(t+x)(t+y)(t+z)}}$$

where  $x, y, z \geq 0$  and at most one is zero.

The basic algorithm, which is due to Carlson [2] and [3], is to reduce the arguments recursively towards their mean by the rule:

$$\begin{aligned} x_0 &= \min(x,y,z), \quad z_0 = \max(x,y,z), \\ y_0 &= \text{remaining third intermediate value argument.} \end{aligned}$$

(This ordering, which is possible because of the symmetry of the function, is done for technical reasons related to the avoidance of overflow and underflow.)

$$\begin{aligned} \mu_n &= (x_n + y_n + z_n)/3 \\ X_n &= 1 - x_n/\mu_n \\ Y_n &= 1 - y_n/\mu_n \\ Z_n &= 1 - z_n/\mu_n \\ \lambda_n &= \sqrt{x_n y_n} + \sqrt{y_n z_n} + \sqrt{z_n x_n} \\ x_{n+1} &= (x_n + \lambda_n)/4 \\ y_{n+1} &= (y_n + \lambda_n)/4 \\ z_{n+1} &= (z_n + \lambda_n)/4 \end{aligned}$$

$\epsilon_n = \max(|X_n|, |Y_n|, |Z_n|)$  and the function may be approximated adequately by a 5th order power series:

$$R_F(x,y,z) = \frac{1}{\sqrt{\mu_n}} \left( 1 - \frac{E_2}{10} + \frac{E_2^2}{24} - \frac{3E_2 E_3}{44} + \frac{E_3}{14} \right)$$

where  $E_2 = X_n Y_n + Y_n Z_n + Z_n X_n$ ,  $E_3 = X_n Y_n Z_n$ .

The truncation error involved in using this approximation is bounded by  $\epsilon_n^6/4(1-\epsilon_n)$  and the recursive process is stopped when this truncation error is negligible compared with the *machine precision*.

Within the domain of definition, the function value is itself representable for all representable values of its arguments. However, for values of the arguments near the extremes the above algorithm must be modified so as to avoid causing underflows or overflows in intermediate steps. In extreme regions arguments are pre-scaled away from the extremes and compensating scaling of the result is done before returning to the calling program.

#### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 17, 1968.
- [2] CARLSON, B.C.  
Computing Elliptic Integrals by Duplication.  
Department of Physics, Iowa State University, Preprint 1978.
- [3] CARLSON, B.C.  
A Table of Elliptic Integrals of the Third Kind.  
Math. Comp. 51, pp. 267-280, 1988.

#### 5. Parameters

- 1: X – *real*. *Input*
- 2: Y – *real*. *Input*
- 3: Z – *real*. *Input*

*On entry:* the arguments  $x$ ,  $y$  and  $z$  of the function.

*Constraint:*  $X, Y, Z \geq 0.0$  and only one of  $X, Y$  and  $Z$  may be zero.

- 4: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

#### 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry one or more of  $X, Y$  and  $Z$  is negative; the function is undefined.

IFAIL = 2

On entry two or more of  $X, Y$  and  $Z$  are zero; the function is undefined.

On soft failure, the routine returns zero.

#### 7. Accuracy

In principle the routine is capable of producing full *machine precision*. However round off errors in internal arithmetic will result in slight loss of accuracy. This loss should never be excessive as the algorithm does not involve any significant amplification of round off error. It is reasonable to assume that the result is accurate to within a small multiple of the *machine precision*.

#### 8. Further Comments

Users should consult the Chapter Introduction which shows the relationship of this function to the classical definitions of the elliptic integrals.

If two arguments are equal, the function reduces to the elementary integral  $R_C$ , computed by S21BAF.

#### 9. Example

This example program simply generates a small set of non-extreme arguments which are used with the routine to produce the table of low accuracy results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      S21BBF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER       (NOUT=6)
*      .. Local Scalars ..
      real            RF, X, Y, Z
      INTEGER          IFAIL, IX
*      .. External Functions ..
      real            S21BBF
      EXTERNAL        S21BBF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S21BBF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X      Y      Z      S21BBF  IFAIL'
      WRITE (NOUT,*)
      DO 20 IX = 1, 3
          X = IX*0.5e0
          Y = (IX+1)*0.5e0
          Z = (IX+2)*0.5e0
          IFAIL = 1
*
          RF = S21BBF(X,Y,Z,IFAIL)
*
          WRITE (NOUT,99999) X, Y, Z, RF, IFAIL
20  CONTINUE
      STOP
*
99999 FORMAT (1X,3F7.2,F12.4,I5)
      END

```

### 9.2. Program Data

None.

### 9.3. Program Results

S21BBF Example Program Results

X	Y	Z	S21BBF	IFAIL
0.50	1.00	1.50	1.0281	0
1.00	1.50	2.00	0.8260	0
1.50	2.00	2.50	0.7116	0

---





## S21BCF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

S21BCF returns a value of the symmetrised elliptic integral of the second kind, via the routine name.

### 2. Specification

*real* FUNCTION S21BCF (X, Y, Z, IFAIL)

INTEGER IFAIL  
*real* X, Y, Z

### 3. Description

This routine calculates an approximate value for the integral

$$R_D(x,y,z) = 3 \int_0^\infty \frac{dt}{\sqrt{(t+x)(t+y)(t+z)^3}}$$

where  $x, y \geq 0$ , at most one of  $x$  and  $y$  is zero, and  $z > 0$ .

The basic algorithm, which is due to Carlson [2] and [3], is to reduce the arguments recursively towards their mean by the rule:

$$\begin{aligned} x_0 &= x, y_0 = y, z_0 = z \\ \mu_n &= (x_n + y_n + 3z_n)/5 \\ X_n &= 1 - x_n/\mu_n \\ Y_n &= 1 - y_n/\mu_n \\ Z_n &= 1 - z_n/\mu_n \\ \lambda_n &= \sqrt{x_n y_n} + \sqrt{y_n z_n} + \sqrt{z_n x_n} \\ x_{n+1} &= (x_n + \lambda_n)/4 \\ y_{n+1} &= (y_n + \lambda_n)/4 \\ z_{n+1} &= (z_n + \lambda_n)/4 \end{aligned}$$

For  $n$  sufficiently large,

$$\varepsilon_n = \max(|X_n|, |Y_n|, |Z_n|) \sim \left(\frac{1}{4}\right)^n$$

and the function may be approximated adequately by a 5th order power series

$$\begin{aligned} R_D(x,y,z) &= 3 \sum_{m=0}^{n-1} \frac{4^{-m}}{(z_m + \lambda_m) \sqrt{z_m}} + \\ &\frac{4^{-n}}{\sqrt{\mu_n^3}} \left[ 1 + \frac{3}{7} S_n^{(2)} + \frac{1}{3} S_n^{(3)} + \frac{3}{22} (S_n^{(2)})^2 + \frac{3}{11} S_n^{(4)} + \frac{3}{13} S_n^{(2)} S_n^{(3)} + \frac{3}{13} S_n^{(5)} \right] \end{aligned}$$

where

$$S_n^{(m)} = (X_n^m + Y_n^m + 3Z_n^m)/2m.$$

The truncation error in this expansion is bounded by  $\frac{3\varepsilon_n^6}{\sqrt{(1-\varepsilon_n)^3}}$  and the recursive process is terminated when this quantity is negligible compared with the *machine precision*.

The routine may fail either because it has been called with arguments outside the domain of definition, or with arguments so extreme that there is an unavoidable danger of setting underflow or overflow.

Note:  $R_D(x,x,x) = x^{-1}$ , so there exists a region of extreme arguments for which the function value is not representable.

#### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 17, 1968.
- [2] CARLSON, B.C.  
Computing Elliptic Integrals by Duplication.  
Department of Physics, Iowa State University, Preprint 1978.
- [3] CARLSON, B.C.  
A Table of Elliptic Integrals of the Third Kind.  
Math. Comput. 51, pp. 267-280, 1988.

#### 5. Parameters

- 1: X – *real*. *Input*
- 2: Y – *real*. *Input*
- 3: Z – *real*. *Input*

*On entry:* the arguments  $x, y$  and  $z$  of the function.

*Constraint:*  $X, Y \geq 0.0, Z > 0.0$  and only one of X and Y may be zero.

- 4: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

#### 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, either X or Y is negative, or both X and Y are zero; the function is undefined.

IFAIL = 2

On entry,  $Z \leq 0.0$ ; the function is undefined.

IFAIL = 3

On entry, either Z is too close to zero or both X and Y are too close to zero: there is a danger of setting overflow.

IFAIL = 4

On entry, at least one of X, Y and Z is too large: there is a danger of setting underflow.

On soft failure the routine returns zero.

#### 7. Accuracy

In principle the routine is capable of producing full *machine precision*. However round-off errors in internal arithmetic will result in slight loss of accuracy. This loss should never be excessive as the algorithm does not involve any significant amplification of round-off error. It is reasonable to assume that the result is accurate to within a small multiple of the *machine precision*.

## 8. Further Comments

Users should consult the Chapter Introduction which shows the relationship of this function to the classical definitions of the elliptic integrals.

## 9. Example

This example program simply generates a small set of non-extreme arguments which are used with the routine to produce the table of low accuracy results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S21BCF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER       (NOUT=6)
*      .. Local Scalars ..
      real            RD, X, Y, Z
      INTEGER          IFAIL, IX, IY
*      .. External Functions ..
      real            S21BCF
      EXTERNAL         S21BCF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S21BCF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X      Y      Z      S21BCF  IFAIL'
      WRITE (NOUT,*)
      DO 40 IX = 1, 3
         X = IX*0.5e0
         DO 20 IY = IX, 3
            Y = IY*0.5e0
            Z = 1.0e0
            IFAIL = 1

*
            RD = S21BCF(X,Y,Z,IFAIL)
*
            WRITE (NOUT,99999) X, Y, Z, RD, IFAIL
20      CONTINUE
40      CONTINUE
      STOP
*
99999  FORMAT (1X,3F7.2,F12.4,I5)
      END
```

### 9.2. Program Data

None.

### 9.3. Program Results

S21BCF Example Program Results

X	Y	Z	S21BCF	IFAIL
0.50	0.50	1.00	1.4787	0
0.50	1.00	1.00	1.2108	0
0.50	1.50	1.00	1.0611	0
1.00	1.00	1.00	1.0000	0
1.00	1.50	1.00	0.8805	0
1.50	1.50	1.00	0.7775	0

---



## S21BDF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S21BDF returns a value of the symmetrised elliptic integral of the third kind, via the routine name.

## 2. Specification

```
real FUNCTION S21BDF (X, Y, Z, R, IFAIL)
  INTEGER      IFAIL
  real        X, Y, Z, R
```

## 3. Description

This routine calculates an approximation to the integral

$$R_J(x,y,z,\rho) = \frac{3}{2} \int_0^{\infty} \frac{dt}{(t+\rho)\sqrt{(t+x)(t+y)(t+z)}}$$

where  $x, y, z \geq 0$ ,  $\rho \neq 0$  and at most one of  $x, y$  and  $z$  is zero.

If  $p < 0$ , the result computed is the Cauchy principal value of the integral.

The basic algorithm, which is due to Carlson [2] and [3], is to reduce the arguments recursively towards their mean by the rule:

$$\begin{aligned} x_0 &= x, \quad y_0 = y, \quad z_0 = z, \quad \rho_0 = \rho \\ \mu_n &= (x_n + y_n + z_n + 2\rho_n)/5 \\ X_n &= 1 - x_n/\mu_n \\ Y_n &= 1 - y_n/\mu_n \\ Z_n &= 1 - z_n/\mu_n \\ P_n &= 1 - \rho_n/\mu_n \\ \lambda_n &= \sqrt{x_n y_n} + \sqrt{y_n z_n} + \sqrt{z_n x_n} \\ x_{n+1} &= (x_n + \lambda_n)/4 \\ y_{n+1} &= (y_n + \lambda_n)/4 \\ z_{n+1} &= (z_n + \lambda_n)/4 \\ \rho_{n+1} &= (\rho_n + \lambda_n)/4 \\ \alpha_n &= [\rho_n(\sqrt{x_n} + \sqrt{y_n} + \sqrt{z_n}) + \sqrt{x_n y_n z_n}]^2 \\ \beta_n &= \rho_n(\rho_n + \lambda_n)^2. \end{aligned}$$

For  $n$  sufficiently large,

$$\varepsilon_n = \max(|X_n|, |Y_n|, |Z_n|, |P_n|) \sim \frac{1}{4^n}$$

and the function may be approximated by a 5th order power series

$$\begin{aligned} R_J(x,y,z,\rho) &= 3 \sum_{m=0}^{n-1} 4^{-m} R_C(\alpha_m, \beta_m) \\ &+ \frac{4^{-n}}{\sqrt{\mu_n^3}} \left[ 1 + \frac{3}{7} S_n^{(2)} + \frac{1}{3} S_n^{(3)} + \frac{3}{22} (S_n^{(2)})^2 + \frac{3}{11} S_n^{(4)} + \frac{3}{13} S_n^{(2)} S_n^{(3)} + \frac{3}{13} S_n^{(5)} \right] \end{aligned}$$

$$\text{where } S_n^{(m)} = (X_n^m + Y_n^m + Z_n^m + 2P_n^m)/2m.$$

The truncation error in this expansion is bounded by  $3\varepsilon_n^6/\sqrt{(1-\varepsilon_n)^3}$  and the recursion process is terminated when this quantity is negligible compared with the *machine precision*. The routine may fail either because it has been called with arguments outside the domain of definition or with arguments so extreme that there is an unavoidable danger of setting underflow or overflow.

**Note:**  $R_J(x,x,x,x) = x^{-3}$ , so there exists a region of extreme arguments for which the function value is not representable.

#### 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 17, 1968.
- [2] CARLSON, B.C.  
Computing Elliptic Integrals by Duplication.  
Department of Physics, Iowa State University, Preprint 1978.
- [3] CARLSON, B.C.  
A Table of Elliptic Integrals of the Third Kind.  
Math. Comp. 51, pp. 267-280, 1988.

#### 5. Parameters

- |    |                   |              |
|----|-------------------|--------------|
| 1: | X – <i>real</i> . | <i>Input</i> |
| 2: | Y – <i>real</i> . | <i>Input</i> |
| 3: | Z – <i>real</i> . | <i>Input</i> |
| 4: | R – <i>real</i> . | <i>Input</i> |

*On entry:* the arguments  $x$ ,  $y$ ,  $z$  and  $\rho$  of the function.

*Constraint:*  $X, Y, Z \geq 0.0$ ,  $R \neq 0.0$  and at most one of  $X$ ,  $Y$  and  $Z$  may be zero.

- |    |                  |                     |
|----|------------------|---------------------|
| 5: | IFAIL – INTEGER. | <i>Input/Output</i> |
|----|------------------|---------------------|

*On entry:* IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

#### 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or –1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry at least one of  $X$ ,  $Y$  and  $Z$  is negative, or at least two of them are zero; the function is undefined.

IFAIL = 2

On entry  $R = 0.0$ ; the function is undefined.

IFAIL = 3

On entry either  $R$  is too close to zero, or any two of  $X$ ,  $Y$  and  $Z$  are too close to zero; there is a danger of setting overflow.

IFAIL = 4

On entry at least one of  $X$ ,  $Y$ ,  $Z$  and  $R$  is too large; there is a danger of setting underflow.

## 7. Accuracy

In principle the routine is capable of producing full *machine precision*. However round-off errors in internal arithmetic will result in slight loss of accuracy. This loss should never be excessive as the algorithm does not involve any significant amplification of round-off error. It is reasonable to assume that the result is accurate to within a small multiple of the *machine precision*.

## 8. Further Comments

Users should consult the Chapter Introduction which shows the relationship of this function to the classical definitions of the elliptic integrals.

If the argument R is equal to any of the other arguments, the function reduces to the integral  $R_D$ , computed by S21BCF.

## 9. Example

This example program simply generates a small set of non-extreme arguments which are used with the routine to produce the table of low accuracy results.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S21BDF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER       (NOUT=6)
*      .. Local Scalars ..
      real            R, RJ, X, Y, Z
      INTEGER          IFAIL, IX, IY, IZ
*      .. External Functions ..
      real            S21BDF
      EXTERNAL         S21BDF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'S21BDF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '      X      Y      Z      R      S21BDF  IFAIL'
      WRITE (NOUT,*)
      DO 60 IX = 1, 3
         X = IX*0.5e0
         DO 40 IY = IX, 3
            Y = IY*0.5e0
            DO 20 IZ = IY, 3
               Z = IZ*0.5e0
               R = 2.0e0
               IFAIL = 1
*
               RJ = S21BDF(X,Y,Z,R,IFAIL)
*
               WRITE (NOUT,99999) X, Y, Z, R, RJ, IFAIL
            20      CONTINUE
         40      CONTINUE
      60      CONTINUE
      STOP
*
99999  FORMAT (1X,4F7.2,F12.4,I5)
      END
```

**9.2. Program Data**

None.

**9.3. Program Results**

S21BDF Example Program Results

X	Y	Z	R	S21BDF	IFAIL
0.50	0.50	0.50	2.00	1.1184	0
0.50	0.50	1.00	2.00	0.9221	0
0.50	0.50	1.50	2.00	0.8115	0
0.50	1.00	1.00	2.00	0.7671	0
0.50	1.00	1.50	2.00	0.6784	0
0.50	1.50	1.50	2.00	0.6017	0
1.00	1.00	1.00	2.00	0.6438	0
1.00	1.00	1.50	2.00	0.5722	0
1.00	1.50	1.50	2.00	0.5101	0
1.50	1.50	1.50	2.00	0.4561	0

---



## S21CAF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

S21CAF evaluates the Jacobian elliptic functions  $\text{sn}$ ,  $\text{cn}$  and  $\text{dn}$ .

## 2. Specification

```
SUBROUTINE S21CAF (U, M, SN, CN, DN, IFAIL)
  INTEGER          IFAIL
  real           U, M, SN, CN, DN
```

## 3. Description

This routine evaluates the Jacobian elliptic functions of argument  $u$  and parameter  $m$ ,

$$\begin{aligned}\text{sn}(u|m) &= \sin \phi, \\ \text{cn}(u|m) &= \cos \phi, \\ \text{dn}(u|m) &= \sqrt{1-m \sin^2 \phi},\end{aligned}$$

where  $\phi$ , called the *amplitude* of  $u$ , is defined by the integral

$$u = \int_0^\phi \frac{d\theta}{\sqrt{1-m \sin^2 \theta}}.$$

The elliptic functions are sometimes written simply as  $\text{sn } u$ ,  $\text{cn } u$  and  $\text{dn } u$ , avoiding explicit reference to the parameter  $m$ .

Another nine elliptic functions may be computed via the formulae

$$\begin{aligned}\text{cd } u &= \text{cn } u / \text{dn } u \\ \text{sd } u &= \text{sn } u / \text{dn } u \\ \text{nd } u &= 1 / \text{dn } u \\ \text{dc } u &= \text{dn } u / \text{cn } u \\ \text{nc } u &= 1 / \text{cn } u \\ \text{sc } u &= \text{sn } u / \text{cn } u \\ \text{ns } u &= 1 / \text{sn } u \\ \text{ds } u &= \text{dn } u / \text{sn } u \\ \text{cs } u &= \text{cn } u / \text{sn } u\end{aligned}$$

(see Abramowitz and Stegun [1]).

S21CAF is based on a procedure given by Bulirsch [2], and uses the process of the arithmetic-geometric mean ([1], 16.9). Constraints are placed on the values of  $u$  and  $m$  in order to avoid the possibility of machine overflow.

## 4. References

- [1] ABRAMOWITZ, M. and STEGUN, I.A.  
Handbook of Mathematical Functions.  
Dover Publications, Ch. 16, 1968.
- [2] BULIRSCH, R.  
Numerical Calculation of Elliptic Integrals and Elliptic Functions.  
Num. Math., 7, pp 76-90, 1965.

## 5. Parameters

- 1: *U* – *real*. *Input*  
 2: *M* – *real*. *Input*

*On entry:* the argument *u* and the parameter *m* of the functions, respectively.

*Constraints:*  $ABS(U) \leq \sqrt{\lambda}$ , where  $\lambda = 1/X02AMF$ ,  
 $ABS(M) \leq \sqrt{\lambda}$  if  $ABS(U) < 1/\sqrt{\lambda}$ .

- 3: *SN* – *real*. *Output*  
 4: *CN* – *real*. *Output*  
 5: *DN* – *real*. *Output*

*On exit:* the values of the functions *sn u*, *cn u* and *dn u*, respectively.

- 6: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

If, on exit, IFAIL  $\neq$  0, then S21CAF returns with the value 0.0 for *sn*, *cn* and *dn*.

IFAIL = 1

On entry,  $ABS(U) > \sqrt{\lambda}$ , where  $\lambda = 1/X02AMF$ .

IFAIL = 2

On entry,  $ABS(M) > \sqrt{\lambda}$  and  $ABS(U) < 1/\sqrt{\lambda}$ .

## 7. Accuracy

In principle the routine is capable of achieving full relative precision in the computed values. However, the accuracy obtainable in practice depends on the accuracy of the Fortran intrinsic functions for elementary functions such as SIN and COS.

## 8. Further Comments

None.

## 9. Example

The following program reads values of the argument *u* and parameter *m* from a file, evaluates the function and prints the results.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      S21CAF Example Program Text
*      Mark 15 Release. NAG Copyright 1991
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER       (NIN=5, NOUT=6)
*      .. Local Scalars ..
      real            CN, DN, M, SN, U
      INTEGER          IFAIL
*      .. External Subroutines ..
      EXTERNAL        S21CAF
```

```

*      .. Executable Statements ..
      WRITE (NOUT,*) 'S21CAF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*)
+ '      U          M          SN          CN          DN'
20 READ (NIN,*,END=40) U, M
*
      IFAIL = 0
      CALL S21CAF(U,M,SN,CN,DN,IFAIL)
*
      WRITE (NOUT,99999) U, M, SN, CN, DN
      GO TO 20
40 STOP
*
99999 FORMAT (3X,5E13.4)
      END

```

## 9.2. Program Data

```

S21CAF Example Program Data
  0.2   0.3
  5.0  -1.0
 -0.5  -0.1
 10.0  11.0

```

## 9.3. Program Results

S21CAF Example Program Results

U	M	SN	CN	DN
0.2000E+00	0.3000E+00	0.1983E+00	0.9801E+00	0.9941E+00
0.5000E+01	-0.1000E+01	-0.2440E+00	0.9698E+00	0.1029E+01
-0.5000E+00	-0.1000E+00	-0.4812E+00	0.8766E+00	0.1012E+01
0.1000E+02	0.1100E+02	0.2512E+00	0.9679E+00	0.5528E+00

---



## Chapter X01 – Mathematical Constants

**Note.** Please refer to the Users' Note for your implementation to check that a routine is available.

<b>Routine Name</b>	<b>Mark of Introduction</b>	<b>Purpose</b>
X01AAF	5	Provides the mathematical constant $\pi$
X01ABF	5	Provides the mathematical constant $\gamma$ (Euler's Constant)

---



# **Chapter X01**

## **Mathematical Constants**

### **Contents**

<b>1</b>	<b>Scope of the Chapter</b>	<b>2</b>
<b>2</b>	<b>Background to the Problems</b>	<b>2</b>
<b>3</b>	<b>Recommendations on Choice and Use of Available Routines</b>	<b>2</b>

## 1 Scope of the Chapter

This chapter is concerned with the provision of mathematical constants required by other routines within the Library.

It should be noted that because of the trivial nature of the routines individual routine documents are not provided.

## 2 Background to the Problems

Some Library routines require mathematical constants to maximum *machine precision*. These routines call Chapter X01 and thus lessen the number of changes that have to be made between different implementations of the Library.

## 3 Recommendations on Choice and Use of Available Routines

**Note.** Refer to the Users' Note for your implementation to check that a routine is available.

Although these routines are primarily intended for use by other routines they may be accessed directly by the user:

Constant	Fortran Specification
$\pi$	<i>real</i> FUNCTION X01AAF(X) <i>real</i> X
$\gamma$ (Euler's constant)	<i>real</i> FUNCTION X01ABF(X) <i>real</i> X

The parameter X of these routines is a dummy parameter.

---



## Chapter X02 – Machine Constants

**Note.** Please refer to the Users' Note for your implementation to check that a routine is available.

<b>Routine Name</b>	<b>Mark of Introduction</b>	<b>Purpose</b>
X02AHF	9	The largest permissible argument for sin and cos
X02AJF	12	The machine precision
X02AKF	12	The smallest positive model number
X02ALF	12	The largest positive model number
X02AMF	12	The safe range parameter
X02ANF	15	The safe range parameter for complex floating-point arithmetic
X02BBF	5	The largest representable integer
X02BEF	5	The maximum number of decimal digits that can be represented
X02BHF	12	The floating-point model parameter, $b$
X02BJF	12	The floating-point model parameter, $p$
X02BKF	12	The floating-point model parameter $e_{\min}$
X02BLF	12	The floating-point model parameter $e_{\max}$
X02DAF	8	Switch for taking precautions to avoid underflow
X02DJF	12	The floating-point model parameter ROUNDS

---



# Chapter X02

## Machine Constants

### Contents

<b>1</b>	<b>Scope of the Chapter</b>	<b>2</b>
<b>2</b>	<b>Background to the Problems</b>	<b>2</b>
2.1	Floating-Point Arithmetic . . . . .	2
2.1.1	A model of floating-point arithmetic . . . . .	2
2.1.2	Derived parameters of floating-point arithmetic . . . . .	3
2.2	Other Aspects of the Computing Environment . . . . .	4
<b>3</b>	<b>Recommendations on Choice and Use of Available Routines</b>	<b>4</b>
3.1	Historical Note . . . . .	4
3.2	Parameters of Floating-point Arithmetic . . . . .	4
3.3	Parameters of Other Aspects of the Computing Environment . . . . .	5
<b>4</b>	<b>Routines Withdrawn or Scheduled for Withdrawal</b>	<b>5</b>
<b>5</b>	<b>References</b>	<b>5</b>
<b>6</b>	<b>Example Program</b>	<b>5</b>
6.1	Example Text . . . . .	5
6.2	Example Data . . . . .	6
6.3	Example Results . . . . .	6

## 1 Scope of the Chapter

This chapter is concerned with **parameters** which characterise certain aspects of the **computing environment** in which the NAG Fortran Library is implemented. They relate primarily to floating-point arithmetic, but also to integer arithmetic, the elementary functions and exception handling. The values of the parameters vary from one implementation of the Library to another, but within the context of a single implementation they are constants.

The parameters are intended for use primarily by other routines in the Library, but users of the Library may sometimes need to refer to them directly.

Each parameter-value is returned by a separate Fortran function. Because of the trivial nature of the functions, individual routine documents are not provided; the necessary details are given in Section 3 of this Introduction.

## 2 Background to the Problems

### 2.1 Floating-Point Arithmetic

#### 2.1.1 A model of floating-point arithmetic

In order to characterise the important properties of floating-point arithmetic by means of a small number of parameters, NAG uses a simplified **model** of floating-point arithmetic. The parameters of the model can be chosen to provide a sufficiently close description of the behaviour of actual implementations of floating-point arithmetic, but not, in general, an exact description; actual implementations vary too much in the details of how numbers are represented or arithmetic operations are performed.

The model is based on that developed by Brown [1], but differs in some respects. The essential features are summarised here.

The model is characterised by four integer parameters and one logical parameter. The four integer parameters are:

- $b$ : the base
- $p$ : the precision (i.e., the number of significant base- $b$  digits)
- $e_{\min}$ : the minimum exponent
- $e_{\max}$ : the maximum exponent

These parameters define a set of numerical values of the form:

$$f \times b^e$$

where the exponent  $e$  must lie in the range  $[e_{\min}, e_{\max}]$ , and the fraction  $f$  (also called the mantissa or significand) lies in the range  $[1/b, 1)$ , and may be written:

$$f = 0.f_1f_2\dots f_p$$

Thus  $f$  is a  $p$ -digit fraction to the base  $b$ ; the  $f_i$  are the base- $b$  digits of the fraction: they are integers in the range 0 to  $b - 1$ , and the leading digit  $f_1$  must not be zero.

The set of values so defined (together with zero) are called **model numbers**. For example, if  $b = 10$ ,  $p = 5$ ,  $e_{\min} = -99$  and  $e_{\max} = +99$ , then a typical model number is  $0.12345 \times 10^{67}$ .

The model numbers must obey certain rules for the computed results of the following basic arithmetic operations: addition, subtraction, multiplication, negation, absolute value, and comparisons. The rules depend on the value of the logical parameter **ROUNDS**.

If **ROUNDS** is **true**, then the computed result must be the nearest model number to the exact result (assuming that overflow or underflow does not occur); if the exact result is midway between two model numbers, then it may be rounded either way.

If **ROUNDS** is **false**, then: if the exact result is a model number, the computed result must be equal to the exact result; otherwise, the computed result may be either of the adjacent model numbers on either side of the exact result.

For division and square root, this latter rule is further relaxed (regardless of the value of ROUNDS): the computed result may also be one of the next adjacent model numbers on either side of the permitted values just stated.

On some machines, the full set of representable floating-point numbers conforms to the rules of the model with appropriate values of  $b$ ,  $p$ ,  $e_{\min}$ ,  $e_{\max}$  and ROUNDS. For example, for DEC VAX machines in single precision:

$$\begin{aligned} b &= 2 \\ p &= 24 \\ e_{\min} &= -127 \\ e_{\max} &= 127 \quad \text{and ROUNDS is true.} \end{aligned}$$

For machines supporting IEEE binary double precision arithmetic:

$$\begin{aligned} b &= 2 \\ p &= 53 \\ e_{\min} &= -1021 \\ e_{\max} &= 1024 \quad \text{and ROUNDS is true.} \end{aligned}$$

For other machines, values of the model parameters must be chosen which define a large subset of the representable numbers; typically it may be necessary to decrease  $p$  by 1 (in which case ROUNDS is always set to **false**), or to increase  $e_{\min}$  or decrease  $e_{\max}$  by a little bit. There are additional rules to ensure that arithmetic operations on those representable numbers that are not model numbers are consistent with arithmetic on model numbers.

(**Note.** The model used here differs from that described in Brown [1] in the following respects: square-root is treated, like division, as a weakly supported operator; and the logical parameter ROUNDS has been introduced to take account of machines with good rounding.)

### 2.1.2 Derived parameters of floating-point arithmetic

Most numerical algorithms require access, not to the basic parameters of the model, but to certain derived values, of which the most important are:

$$\begin{aligned} \text{the } \mathbf{machine\ precision} \ \epsilon: &= \left(\frac{1}{2}\right) \times b^{1-p} \text{ if ROUNDS is true,} \\ &= b^{1-p} \text{ otherwise (but see Note below).} \\ \text{the smallest positive model number:} &= b^{e_{\min}-1} \\ \text{the largest positive model number:} &= (1 - b^{-p}) \times b^{e_{\max}} \end{aligned}$$

**Note.** This value is increased very slightly in some implementations to ensure that the computed result of  $1 + \epsilon$  or  $1 - \epsilon$  differs from 1. For example in IEEE binary single precision arithmetic the value is set to  $2^{-24} + 2^{-47}$ .

Two additional derived values are used in the NAG Fortran Library. Their definitions depend not only on the properties of the basic arithmetic operations just considered, but also on properties of some of the elementary functions. We define the **safe range** parameter to be the smallest positive model number  $z$  such that for any  $x$  in the range  $[z, 1/z]$  the following can be computed without undue loss of accuracy, overflow, underflow or other error:

$$\begin{aligned} &-x \\ &1/x \\ &-1/x \\ &\text{SQRT}(x) \\ &\text{LOG}(x) \\ &\text{EXP}(\text{LOG}(x)) \\ &y^{**}(\text{LOG}(x)/\text{LOG}(y)) \text{ for any } y \end{aligned}$$

In a similar fashion we define the safe range parameter for complex arithmetic as the smallest positive model number  $z$  such that for any  $x$  in the range  $[z, 1/z]$  the following can be computed without any undue loss of accuracy, overflow, underflow or other error:

$-w$   
 $1/w$   
 $-1/w$   
 $\text{SQRT}(w)$   
 $\text{LOG}(w)$   
 $\text{EXP}(\text{LOG}(w))$   
 $y^{**}(\text{LOG}(w)/\text{LOG}(y))$  for any  $y$   
 $\text{ABS}(w)$

where  $w$  is any of  $x$ ,  $ix$ ,  $x + ix$ ,  $1/x$ ,  $i/x$ ,  $1/x + i/x$ , and  $i$  is the square root of  $-1$ .

This parameter was introduced to take account of the quality of complex arithmetic on the machine. On machines with well implemented complex arithmetic, its value will differ from that of the real safe range parameter by a small multiplying factor less than 10. For poorly implemented complex arithmetic this factor may be larger by many orders of magnitude.

## 2.2 Other Aspects of the Computing Environment

No attempt has been made to characterise comprehensively any other aspects of the computing environment. The other functions in this chapter provide specific information that is occasionally required by routines in the Library.

## 3 Recommendations on Choice and Use of Available Routines

**Note.** Refer to the Users' Note for your implementation to check that a routine is available.

### 3.1 Historical Note

At Mark 12 a new set of routines was introduced to return parameters of floating-point arithmetic. The new set of routines is more carefully defined, and they do not require a dummy parameter. They are listed in Section 3.2. The older routines have since been withdrawn (see Section 4).

### 3.2 Parameters of Floating-point Arithmetic

<i>real</i> FUNCTION X02AJF()	returns the <b>machine precision</b> , i.e., $(\frac{1}{2}) \times b^{1-p}$ if ROUNDS is <b>true</b> or $b^{1-p}$ otherwise (or a value very slightly larger than this, see Section 2.1.2)
<i>real</i> FUNCTION X02AKF()	returns the smallest positive model number, i.e., $b^{e_{\min}-1}$
<i>real</i> FUNCTION X02ALF()	returns the largest positive model number, i.e., $(1 - b^{-p}) \times b^{e_{\max}}$
<i>real</i> FUNCTION X02AMF()	returns the <b>safe range</b> parameter as defined in Section 2.1.2
<i>real</i> FUNCTION X02ANF()	returns the <b>safe range</b> parameter for complex arithmetic as defined in Section 2.1.2
INTEGER FUNCTION X02BHF()	returns the model parameter $b$
INTEGER FUNCTION X02BJF()	returns the model parameter $p$
INTEGER FUNCTION X02BKF()	returns the model parameter $e_{\min}$
INTEGER FUNCTION X02BLF()	returns the model parameter $e_{\max}$
LOGICAL FUNCTION X02DJF()	returns the model parameter ROUNDS

### 3.3 Parameters of Other Aspects of the Computing Environment

<i>real</i> FUNCTION X02AHF(X) <i>real</i> X	returns the largest positive <i>real</i> argument for which the intrinsic functions SIN and COS return a result with some meaningful accuracy
INTEGER FUNCTION X02BBF(X) <i>real</i> X	returns the largest positive integer value
INTEGER FUNCTION X02BEF(X) <i>real</i> X	returns the maximum number of decimal digits which can be accurately represented over the whole range of floating-point numbers
LOGICAL FUNCTION X02DAF(X) <i>real</i> X	returns <b>false</b> if the system sets underflowing quantities to zero, without any error indication or undesirable warning or system overhead

The parameter X in these routines is a dummy parameter.

## 4 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have been withdrawn. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

X02AAF	X02ABF	X02ACF	X02ADF	X02AEF	X02AFF
X02AGF	X02BAF	X02BCF	X02BDF	X02CAF	

## 5 References

- [1] Brown W S (1981) A simple but realistic model of floating-point computation *ACM Trans. Math. Software* 7 445-480

## 6 Example Program

The example program listed below simply prints the values of all the functions in Chapter X02. Obviously the results will vary from one implementation of the Library to another. The results listed in Section 6.3 are those from a double precision implementation on a Silicon Graphics workstation.

### 6.1 Example Text

```
*      X02AJF Example Program Text
*      Mark 17 Revised.  NAG Copyright 1995.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER       (NOUT=6)
*      .. External Functions ..
      real            X02AHF, X02AJF, X02AKF, X02ALF, X02AMF, X02ANF
      INTEGER         X02BBF, X02BEF, X02BHF, X02BJF, X02BKF, X02BLF
      LOGICAL         X02DAF, X02DJF
      EXTERNAL        X02AHF, X02AJF, X02AKF, X02ALF, X02AMF, X02ANF,
+                   X02BBF, X02BEF, X02BHF, X02BJF, X02BKF, X02BLF,
+                   X02DAF, X02DJF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'X02AJF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*) '(results are machine-dependent)'
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'The basic parameters of the model'
      WRITE (NOUT,*)
      WRITE (NOUT,99999) ' X02BHF = ', X02BHF(),
```

```

+ ' (the model parameter B)'
WRITE (NOUT,99999) ' X02BJF = ', X02BJF(),
+ ' (the model parameter P)'
WRITE (NOUT,99999) ' X02BKF = ', X02BKF(),
+ ' (the model parameter EMIN)'
WRITE (NOUT,99999) ' X02BLF = ', X02BLF(),
+ ' (the model parameter EMAX)'
WRITE (NOUT,99998) ' X02DJF = ', X02DJF(),
+ ' (the model parameter ROUNDS)'
WRITE (NOUT,*)
WRITE (NOUT,*) 'Derived parameters of floating-point arithmetic'
WRITE (NOUT,*)
WRITE (NOUT,*) ' X02AJF = ', X02AJF(), ' (the machine precision)'
WRITE (NOUT,*) ' X02AKF = ', X02AKF(),
+ ' (the smallest positive model number)'
WRITE (NOUT,*) ' X02ALF = ', X02ALF(),
+ ' (the largest positive model number)'
WRITE (NOUT,*) ' X02AMF = ', X02AMF(),
+ ' (the real safe range parameter)'
WRITE (NOUT,*) ' X02ANF = ', X02ANF(),
+ ' (the complex safe range parameter)'
WRITE (NOUT,*)
WRITE (NOUT,*)
+ 'Parameters of other aspects of the computing environment'
WRITE (NOUT,*)
WRITE (NOUT,*) ' X02AHF = ', X02AHF(0.0e0),
+ ' (largest argument for SIN and COS)'
WRITE (NOUT,99997) ' X02BBF = ', X02BBF(0.0e0),
+ ' (largest positive integer)'
WRITE (NOUT,99997) ' X02BEF = ', X02BEF(0.0e0),
+ ' (precision in decimal digits)'
WRITE (NOUT,99996) ' X02DAF = ', X02DAF(0.0e0),
+ ' (indicates how underflow is handled)'
STOP
*
99999 FORMAT (1X,A,I7,A)
99998 FORMAT (1X,A,L7,A)
99997 FORMAT (1X,A,I20,A)
99996 FORMAT (1X,A,L20,A)
END

```

## 6.2 Example Data

None.

## 6.3 Example Results

X02AJF Example Program Results

(results are machine-dependent)

The basic parameters of the model

```

X02BHF =      2 (the model parameter B)
X02BJF =     53 (the model parameter P)
X02BKF =   -1021 (the model parameter EMIN)
X02BLF =    1024 (the model parameter EMAX)
X02DJF =      T (the model parameter ROUNDS)

```



## Derived parameters of floating-point arithmetic

X02AJF = 1.1102230246251600E-16 (the machine precision)  
X02AKF = 2.2250738585072107-308 (the smallest positive model number)  
X02ALF = 1.7976931348623093+308 (the largest positive model number)  
X02AMF = 2.2250738585072107-308 (the real safe range parameter)  
X02ANF = 2.2250738585072107-308 (the complex safe range parameter)

## Parameters of other aspects of the computing environment

X02AHF = 1.8014398509481900E+16 (largest argument for SIN and COS)  
X02BBF = 2147483647 (largest positive integer)  
X02BEF = 15 (precision in decimal digits)  
X02DAF = F (indicates how underflow is handled)

---



## Chapter X03 – Inner Products

**Note.** Please refer to the Users' Note for your implementation to check that a routine is available.

<b>Routine Name</b>	<b>Mark of Introduction</b>	<b>Purpose</b>
X03AAF	5	Real inner product added to initial value, basic/additional precision
X03ABF	5	Complex inner product added to initial value, basic/additional precision

---



# Chapter X03

## Inner Products

### Contents

1	Scope of the Chapter	2
2	Background to the Problems	2
3	Recommendations on Choice and Use of Available Routines	2

## 1 Scope of the Chapter

This chapter is concerned with the calculation of innerproducts required by other routines within the Library.

## 2 Background to the Problems

Some Library routines require to calculate the innerproduct

$$c + \sum_i x_i y_i,$$

preferably in additional precision, but, if this is unavailable or prohibitively expensive, then in basic precision. These routines call Chapter X03 so that machine dependencies of this type can be isolated to this chapter.

## 3 Recommendations on Choice and Use of Available Routines

**Note.** Refer to the Users' Note for your implementation to check that a routine is available.

Although these routines are primarily intended for use by other Library routines they may be accessed directly by the user:

X03AAF Calculates the innerproduct for real values  $c$ ,  $x_i$  and  $y_i$ ,

X03ABF Calculates the innerproduct for complex values  $c$ ,  $x_i$  and  $y_i$ ,

---

## X03AAF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

X03AAF calculates the value of a scalar product using *basic* or *additional precision* and adds it to a *basic* or *additional precision* initial value.

## 2. Specification

```

SUBROUTINE X03AAF (A, ISIZEA, B, ISIZEB, N, ISTEPA, ISTEPB, C1, C2,
1              D1, D2, SW, IFAIL)
INTEGER       ISIZEA, ISIZEB, N, ISTEPA, ISTEPB, IFAIL
real        A(ISIZEA), B(ISIZEB), C1, C2, D1, D2
LOGICAL       SW

```

## 3. Description

The routine calculates the scalar product of two *real* vectors and adds it to an initial value  $c$  to give a correctly rounded result  $d$ :

$$d = c + \sum_{i=1}^n a_i b_i.$$

If  $n < 1$ ,  $d = c$ .

The vector elements  $a_i$  and  $b_i$  are stored in selected elements of the one-dimensional array parameters A and B, which in the (sub)program from which X03AAF is called may be identified with parts of possibly multi-dimensional arrays according to the standard Fortran rules. For example, the vectors may be parts of a row or column of a matrix. See Section 5 for details, and Section 9 for an example.

Both the initial value  $c$  and the result  $d$  are defined by a pair of *real* variables, so that they may take either *basic* or *additional precision* values.

- (a) If SW = .TRUE., the products are accumulated in *additional precision*, and on exit the result is available either in *basic precision*, correctly rounded, or in *additional precision*.
- (b) If SW = .FALSE., the products are accumulated in *basic precision*, and the result is returned in *basic precision*.

This routine is designed primarily for use as an auxiliary routine by other routines in the NAG Fortran Library, especially those in the chapters on Linear Algebra.

## 4. References

None.

## 5. Parameters

1: A(ISIZEA) – *real* array. *Input*

*On entry:* the elements of the first vector.

The  $i$ th vector element is stored in the array element A(( $i-1$ )×ISTEPA+1). In the user's (sub)program from which X03AAF is called, A can be part of a multi-dimensional array and the actual argument must be the array element containing the first vector element.

2: ISIZEA – INTEGER. *Input*

*On entry:* the dimension of array A inside the routine.

The upper bound for ISIZEA is found by multiplying together the dimensions of A as declared in the user's (sub)program from which X03AAF is called, subtracting the starting position and adding 1.

*Constraint:*  $ISIZEA \geq (N-1) \times ISTEPA + 1$ .

- 3: B(ISIZEB) – *real* array. *Input*  
*On entry:* the elements of the second vector.  
 The *i*th vector element is stored in the array element  $B((i-1) \times ISTEPB + 1)$ . In the user's (sub)program from which X03AAF is called, B can be part of a multi-dimensional array and the actual argument must be the array element containing the first vector element.
- 4: ISIZEB – INTEGER. *Input*  
*On entry:* the dimension of array B inside the routine.  
 The upper bound for ISIZEB is found by multiplying together the dimensions of B as declared in the (sub)program from which X03AAF is called, subtracting the starting position and adding 1.  
*Constraint:*  $ISIZEB \geq (N-1) \times ISTEPB + 1$ .
- 5: N – INTEGER. *Input*  
*On entry:* the number of elements in the scalar product, *n*.
- 6: ISTEPA – INTEGER. *Input*  
*On entry:* the step length between elements of the first vector in array A.  
*Constraint:*  $ISTEPA > 0$ .
- 7: ISTEPB – INTEGER. *Input*  
*On entry:* the step length between elements of the second vector in array B.  
*Constraint:*  $ISTEPB > 0$ .
- 8: C1 – *real*. *Input*  
 9: C2 – *real*. *Input*  
*On entry:* C1 and C2 must specify the initial value *c*:  $c = C1 + C2$ . Normally, if *c* is in **additional precision**, C1 specifies the most significant part and C2 the least significant part; if *c* is in **basic precision**, then C1 specifies *c* and C2 must have the value 0.0. Both C1 and C2 must be defined on entry.
- 10: D1 – *real*. *Output*  
 11: D2 – *real*. *Output*  
*On exit:* the result *d*.  
 If the calculation is in **additional precision** (SW = .TRUE.),  
      $D1 = d$  rounded to **basic precision**,  
      $D2 = d - D1$ ;  
 thus D1 holds the correctly rounded **basic precision** result and the sum  $D1 + D2$  gives the result in **additional precision**. D2 may have the opposite sign to D1.  
 If the calculation is in **basic precision** (SW = .FALSE.),  
      $D1 = d$ ,  
      $D2 = 0.0$ .
- 12: SW – LOGICAL. *Input*  
*On entry:* the precision to be used in the calculation.  
     SW = .TRUE., – **additional precision**;  
     SW = .FALSE., – **basic precision**.



## 13: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, ISTEPA  $\leq$  0,  
or ISTEPB  $\leq$  0.

IFAIL = 2

On entry, ISIZEA  $<$  (N-1) $\times$ ISTEPA + 1,  
or ISIZEB  $<$  (N-1) $\times$ ISTEPB + 1.

## 7. Accuracy

If the calculation is an *additional precision*, the rounded *basic precision* result D1 is correct to full implementation accuracy, provided that exceptionally severe cancellation does not occur in the summation. If the calculation is in *basic precision*, such accuracy cannot be guaranteed.

## 8. Further Comments

The time taken by the routine is approximately proportional to  $n$  and also depends on whether *basic* or *additional precision* is used.

On exit the variables D1 and D2 may be used directly to supply a *basic* or *additional precision* initial value for a subsequent call of X03AAF.

## 9. Example

To calculate the scalar product of the second column of the matrix  $A$  and the vector  $B$ , and add it to an initial value 1.0 where

$$A = \begin{pmatrix} -2 & -3 & 7 \\ 2 & -5 & 3 \\ -9 & 1 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 8 \\ -4 \\ -2 \end{pmatrix}.$$

## 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      X03AAF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER                N
      PARAMETER              (N=3)
      INTEGER                NIN, NOUT
      PARAMETER              (NIN=5, NOUT=6)
*      .. Local Scalars ..
      real                   C1, C2, D1, D2
      INTEGER                I, IFAIL, ISIZEA, ISIZEB, ISTEPA, ISTEPB, J
      LOGICAL                SW
*      .. Local Arrays ..
      real                   A(N,N), B(N)
*      .. External Subroutines ..
      EXTERNAL               X03AAF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'X03AAF Example Program Results'
```

```
*      Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) ((A(I,J),J=1,N),I=1,N), (B(I),I=1,N)
      C1 = 1.0e0
      C2 = 0.0e0
      ISIZEA = N
      ISIZEB = N
      ISTEPA = 1
      ISTEPB = 1
      SW = .TRUE.
      IFAIL = 0

*
      CALL X03AAF(A(1,2),ISIZEA,B,ISIZEB,N,ISTEPA,ISTEPB,C1,C2,D1,D2,SW,
+             IFAIL)

*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'D1 = ', D1, ' D2 = ', D2
      STOP

*
99999 FORMAT (1X,A,F4.1,A,F4.1)
      END
```

## 9.2. Program Data

X03AAF Example Program Data

```
-2  -3  7
 2  -5  3
-9   1  0
 8  -4 -2
```

## 9.3. Program Results

X03AAF Example Program Results

```
D1 = -5.0 D2 = 0.0
```

---

## X03ABF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X03ABF calculates the value of a *complex* scalar product using *basic* or *additional precision* and adds it to a *complex* initial value.

### 2. Specification

```

SUBROUTINE X03ABF (A, ISIZEA, B, ISIZEB, N, ISTEPA, ISTEPB, CX, DX,
1                SW, IFAIL)
INTEGER          ISIZEA, ISIZEB, N, ISTEPA, ISTEPB, IFAIL
complex        A(ISIZEA), B(ISIZEB), CX, DX
LOGICAL          SW

```

### 3. Description

The routine calculates the scalar product of two *complex* vectors and adds it to an initial value  $c$  to give a correctly rounded result  $d$ :

$$d = c + \sum_{i=1}^n a_i b_i.$$

If  $n < 1$ ,  $d = c$ .

The vector elements  $a_i$  and  $b_i$  are stored in selected elements of the one-dimensional array parameters A and B, which in the (sub)program from which X03ABF is called may be identified with parts of possibly multi-dimensional arrays according to the standard Fortran rules. For example, the vectors may be parts of a row or column of a matrix. See Section 5 for details, and Section 9 for an example.

The products are accumulated in *basic* and *additional precision* depending on the parameter SW. This routine has been designed primarily for use as an auxiliary routine by other routines in the NAG Fortran Library, especially those in the chapters on Linear Algebra.

### 4. References

None.

### 5. Parameters

- 1: A(ISIZEA) – *complex* array. *Input*  
*On entry:* the elements of the first vector.  
 The  $i$ th vector element is stored in the array element A(( $i-1$ )×ISTEPA+1). In the user's (sub)program from which X03ABF is called, A can be part of a multi-dimensional array and the actual argument must be the array element containing the first vector element.
- 2: ISIZEA – INTEGER. *Input*  
*On entry:* the dimension of array A inside the routine.  
 The upper bound for ISIZEA is found by multiplying together the dimensions of A as declared in the (sub)program from which X03ABF is called, subtracting the starting position and adding 1.  
*Constraint:* ISIZEA ≥ (N-1)×ISTEPA + 1.

- 3: **B**(ISIZEB) – *complex* array. *Input*  
*On entry:* the elements of the second vector.  
 The *i*th vector element is stored in the array element  $B((i-1)\times\text{ISTEPB}+1)$ . In the (sub)program from which X03ABF is called, B can be part of a multi-dimensional array and the actual argument must be the array element containing the first vector element.
- 4: ISIZEB – INTEGER. *Input*  
*On entry:* the dimension of array B inside the routine.  
 The upper bound for ISIZEB is found by multiplying together the dimensions of B as declared in the user's (sub)program from which X03ABF is called, subtracting the starting position and adding 1.  
*Constraint:*  $\text{ISIZEB} \geq (N-1)\times\text{ISTEPB} + 1$ .
- 5: N – INTEGER. *Input*  
*On entry:* the number of elements in the scalar product, *n*.
- 6: ISTEPA – INTEGER. *Input*  
*On entry:* the step length between elements of the first vector in array A.  
*Constraint:*  $\text{ISTEPA} > 0$ .
- 7: ISTEPB – INTEGER. *Input*  
*On entry:* the step length between elements of the second vector in array B.  
*Constraint:*  $\text{ISTEPB} > 0$ .
- 8: CX – *complex*. *Input*  
*On entry:* the initial value *c*.
- 9: DX – *complex*. *Output*  
*On exit:* the result *d*.
- 10: SW – LOGICAL. *Input*  
*On entry:* the precision to be used.  
 If SW = .TRUE., *additional precision*.  
 If SW = .FALSE., *basic precision*.
- 11: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry,  $\text{ISTEPA} \leq 0$ ,  
 or  $\text{ISTEPB} \leq 0$ .

IFAIL = 2

On entry,  $\text{ISIZEA} < (N-1)\times\text{ISTEPA} + 1$ ,  
 or  $\text{ISIZEB} < (N-1)\times\text{ISTEPB} + 1$ .

## 7. Accuracy

If the calculation is in *additional precision*, the result is correct to full implementation accuracy provided that exceptionally severe cancellation does not occur in the summation. If the calculation is in *basic precision*, such accuracy cannot be guaranteed.

## 8. Further Comments

The time taken by the routine is approximately proportional to  $n$  and also depends on whether *basic* or *additional precision* is used.

## 9. Example

To calculate the scalar product of the second column of the matrix  $A$  and the vector  $B$ , and add it to an initial value of  $1 + i$ , where

$$A = \begin{pmatrix} -1 & -i & 1 \\ 2 + 3i & i & 2i \\ 0 & -1 - i & 1 - 2i \end{pmatrix} \quad B = \begin{pmatrix} i \\ 1 - i \\ -i \end{pmatrix}.$$

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*
* X03ABF Example Program Text
* Mark 14 Revised. NAG Copyright 1989.
* .. Parameters ..
  INTEGER N
  PARAMETER (N=3)
  INTEGER NIN, NOUT
  PARAMETER (NIN=5, NOUT=6)
* .. Local Scalars ..
  complex CX, DX
  INTEGER I, IFAIL, ISIZEA, ISIZEB, ISTEPA, ISTEPB, J
  LOGICAL SW
* .. Local Arrays ..
  complex A(N,N), B(N)
* .. External Subroutines ..
  EXTERNAL X03ABF
* .. Executable Statements ..
  WRITE (NOUT,*) 'X03ABF Example Program Results'
* Skip heading in data file
  READ (NIN,*)
  READ (NIN,*) ((A(I,J),J=1,N),I=1,N), (B(I),I=1,N)
  CX = (1.0e0,1.0e0)
  ISIZEA = N
  ISIZEB = N
  ISTEPA = 1
  ISTEPB = 1
  SW = .TRUE.
  IFAIL = 0
*
  CALL X03ABF(A(1,2), ISIZEA, B, ISIZEB, N, ISTEPA, ISTEPB, CX, DX, SW, IFAIL)
*
  WRITE (NOUT,*)
  WRITE (NOUT,99999) 'Result = ', DX
  STOP
*
99999 FORMAT (1X,A,' (',F3.0,',',F3.0,')')
END
```

**9.2. Program Data**

```
X03ABF Example Program Data
(-1.0, 0.0) ( 0.0, -1.0) (1.0,  0.0)
( 2.0, 3.0) ( 0.0,  1.0) (0.0,  2.0)
( 0.0, 0.0) (-1.0, -1.0) (1.0, -2.0)
( 0.0, 1.0) ( 1.0, -1.0) (0.0, -1.0)
```

**9.3. Program Results**

```
X03ABF Example Program Results
```

```
Result = ( 2., 3.)
```

---

## Chapter X04 – Input/Output Utilities

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

<b>Routine Name</b>	<b>Mark of Introduction</b>	<b>Purpose</b>
X04AAF	7	Return or set unit number for error messages
X04ABF	7	Return or set unit number for advisory messages
X04BAF	12	Write formatted record to external file
X04BBF	12	Read formatted record from external file
X04CAF	14	Print a real general matrix (easy-to-use)
X04CBF	14	Print a real general matrix (comprehensive)
X04CCF	14	Print a real packed triangular matrix (easy-to-use)
X04CDF	14	Print a real packed triangular matrix (comprehensive)
X04CEF	14	Print a real packed banded matrix (easy-to-use)
X04CFF	14	Print a real packed banded matrix (comprehensive)
X04DAF	14	Print a complex general matrix (easy-to-use)
X04DBF	14	Print a complex general matrix (comprehensive)
X04DCF	14	Print a complex packed triangular matrix (easy-to-use)
X04DDF	14	Print a complex packed triangular matrix (comprehensive)
X04DEF	14	Print a complex packed banded matrix (easy-to-use)
X04DFE	14	Print a complex packed banded matrix (comprehensive)
X04EAF	14	Print an integer matrix (easy-to-use)
X04EBF	14	Print an integer matrix (comprehensive)

---





# Chapter X04

## Input/Output Utilities

### Contents

<b>1</b>	<b>Scope of the Chapter</b>	<b>2</b>
<b>2</b>	<b>Background to the Problems</b>	<b>2</b>
2.1	Output from NAG Library Routines . . . . .	2
2.2	Matrix Printing Routines . . . . .	2
<b>3</b>	<b>Recommendations on Choice and Use of Available Routines</b>	<b>2</b>
<b>4</b>	<b>Index</b>	<b>3</b>

## 1 Scope of the Chapter

This chapter contains utility routines concerned with input and output to or from an external file.

## 2 Background to the Problems

### 2.1 Output from NAG Library Routines

Output from NAG library routines to an external file falls into two categories.

- (a) **Error messages**  
which are always associated with an error exit from a routine, that is, with a non-zero value of IFAIL as specified in Section 6 of the routine document.
- (b) **Advisory messages**  
which include output of final results, output of intermediate results to monitor the course of a computation, and various warning or informative messages.

Each category of output is written to its own Fortran output unit – the **error message unit** or the **advisory message unit**. In practice these may be the same unit number. Default unit numbers are provided for each implementation of the Library (see the Users' Note for your implementation); they may be changed by users. Output of error messages may be controlled by the setting of IFAIL (see the Essential Introduction or Chapter P01). Output of advisory messages may usually be controlled by the setting of some other parameter (e.g. MSGLVL) (or in some routines also by IFAIL). An alternative mechanism for completely suppressing output is to set the relevant unit number < 0.

At present only formatted records are output from the Library. All formatted output to an external file from within the Library is performed by X04BAF. Similarly, all formatted input from an external file is performed by X04BBF.

For further information about error and advisory messages, see Chapter P01.

### 2.2 Matrix Printing Routines

Routines are provided to allow formatted output of

- (a) general matrices stored in a two-dimensional array (real, complex and integer data types);
- (b) triangular matrices stored in a packed one-dimensional array (real and complex data types);
- (c) band matrices stored in a packed two-dimensional array (real and complex data types).

Routines in (b) and (c) allow printing of matrices stored in formats used in particular by Chapters F06 and F07 of the Library.

By appropriate choice of arguments the user can specify titles, labels, maximum output record length, and the format of individual matrix elements. All output is directed to the unit number for output of advisory messages, which may be altered by a call to X04ABF.

## 3 Recommendations on Choice and Use of Available Routines

**Note.** Refer to the Users' Note for your implementation to check that a routine is available.

Apart from the obvious utility of the matrix printing routines, users of the Library may need to call routines in Chapter X04 for the following purposes.

If the default unit number for error messages (given in the Users' Note for your implementation) is not satisfactory, it may be changed to a new value NERR by the statement

```
CALL X04AAF(1,NERR)
```

Similarly the unit number for advisory messages may be changed to a new value NADV by the statement

```
CALL X04ABF(1,NADV)
```

## 4 Index

Accessing external formatted file:	
reading a record	X04BBF
writing a record	X04BAF
Accessing unit number:	
of advisory message unit	X04ABF
of error message unit	X04AAF
Printing matrices:	
Comprehensive routines:	
general complex matrix	X04DBF
general integer matrix	X04EBF
general real matrix	X04CBF
packed complex band matrix	X04DFB
packed real band matrix	X04CFB
packed complex triangular matrix	X04DDF
packed real triangular matrix	X04CDF
Easy-to-use routines:	
general complex matrix	X04DAF
general integer matrix	X04EAF
general real matrix	X04CAF
packed complex band matrix	X04DEF
packed real band matrix	X04CEF
packed complex triangular matrix	X04DCF
packed real triangular matrix	X04CCF

---



## X04AAF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04AAF returns the value of the current error message unit number, or sets the current error message unit number to a new value.

### 2. Specification

```
SUBROUTINE X04AAF ( IFLAG, NERR )
  INTEGER          IFLAG, NERR
```

### 3. Description

This routine enables those library routines which output error messages, to determine the number of the output unit to which the error messages are to be sent; in this case X04AAF is called with IFLAG = 0. X04AAF may also be called with IFLAG = 1 to set the unit number to a specified value. Otherwise a default value (stated in the Users' Note for your implementation) is returned.

Records written to this output unit by other library routines are at most 80 characters long (including a line-printer carriage control character).

Note that if the unit number is set < 0, no messages will be output.

### 4. References

None.

### 5. Parameters

1: IFLAG – INTEGER.

*Input*

*On entry:* the action to be taken (see NERR).

*Constraint:* IFLAG = 0 or 1.

2: NERR – INTEGER.

*Input/Output*

*On entry:*

if IFLAG = 0, NERR need not be set;

if IFLAG = 1, NERR must specify the new error message unit number.

*On exit:*

if IFLAG = 0, NERR is set to the current error message unit number,

if IFLAG = 1, NERR is unchanged.

Note that Fortran unit numbers must be positive or zero. If NERR is set < 0, output of error messages is totally suppressed.

### 6. Error Indicators and Warnings

None.

### 7. Accuracy

Not applicable.

### 8. Further Comments

The time taken by the routine is negligible.

## 9. Example

In this example X04AAF is called by the user's main program to make the error message from the routine DUMMY appear on the same unit as the rest of the output (unit 6). Normally a NAG Fortran Library routine with an IFAIL parameter (see Chapter P01) would take the place of DUMMY.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      X04AAF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
          INTEGER          NOUT
          PARAMETER        (NOUT=6)
*      .. External Subroutines ..
          EXTERNAL         DUMMY, X04AAF
*      .. Executable Statements ..
          WRITE (NOUT,*) 'X04AAF Example Program Results'
*
          CALL X04AAF(1,NOUT)
          CALL DUMMY
*
          STOP
          END
*
          SUBROUTINE DUMMY
*      .. Local Scalars ..
          INTEGER          NERR
*      .. External Subroutines ..
          EXTERNAL         X04AAF
*      .. Executable Statements ..
          CALL X04AAF(0,NERR)
          WRITE (NERR,*)
          WRITE (NERR,*) 'This is a dummy error message'
          RETURN
          END
```

### 9.2. Program Data

None.

### 9.3. Program Results

X04AAF Example Program Results

This is a dummy error message

---

## X04ABF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04ABF returns the value of the current advisory message unit number, or sets the current advisory message unit number to a new value.

### 2. Specification

```
SUBROUTINE X04ABF ( IFLAG, NADV )
      INTEGER      IFLAG, NADV
```

### 3. Description

This routine enables those library routines which output advisory messages, to determine the number of the output unit to which the advisory messages are to be sent; in this case X04ABF is called with IFLAG = 0. X04ABF may also be called with IFLAG = 1 to set the unit number to a specified value. Otherwise a default value (stated in the User's Note for your implementation) is returned.

Records written to this output unit by other library routines are at most 120 characters long (including a line-printer carriage control character), unless those library routines allow users to specify longer records.

Note that if the unit number is set < 0, no messages will be output.

### 4. References

None.

### 5. Parameters

1: IFLAG – INTEGER. *Input*

*On entry:* the action to be taken (see NADV).

*Constraint:* IFLAG = 0 or 1.

2: NADV – INTEGER. *Input/Output*

*On entry:*

if IFLAG = 0, NADV need not be set;

if IFLAG = 1, NADV must specify the new advisory message unit number.

*On exit:*

if IFLAG = 0, NADV is set to the current advisory message unit number;

if IFLAG = 1, NADV is unchanged.

Note that Fortran unit numbers must be positive or zero. If NADV is set < 0, output of advisory messages is totally suppressed.

### 6. Error Indicators and Warnings

None.

### 7. Accuracy

Not applicable.

### 8. Further Comments

The time taken by this routine is negligible.

## 9. Example

In this example X04ABF is called by the user's main program to make the advisory message from the routine DUMMY appear on the same unit as the rest of the output (unit 6). Normally a NAG Fortran Library routine with an IFAIL parameter (see Chapter P01) would take the place of DUMMY.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      X04ABF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NOUT
PARAMETER        (NOUT=6)
*      .. External Subroutines ..
EXTERNAL         DUMMY, X04ABF
*      .. Executable Statements ..
WRITE (NOUT,*) 'X04ABF Example Program Results'

*
CALL X04ABF(1,NOUT)
CALL DUMMY

*
STOP
END

*
SUBROUTINE DUMMY
*      .. Local Scalars ..
INTEGER          NADV
*      .. External Subroutines ..
EXTERNAL         X04ABF
*      .. Executable Statements ..
CALL X04ABF(0,NADV)
WRITE (NADV,*)
WRITE (NADV,*) 'This is a dummy advisory message'
RETURN
END
```

### 9.2. Program Data

None.

### 9.3. Program Results

X04ABF Example Program Results

This is a dummy advisory message

---



## X04BAF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04BAF writes a single formatted record to an external file.

### 2. Specification

```
SUBROUTINE X04BAF (NOUT, REC)
      INTEGER      NOUT
      CHARACTER*(*) REC
```

### 3. Description

X04BAF is used by NAG Fortran Library routines to write formatted records to an external file. All formatted output to an external file from NAG Fortran Library routines is performed by calls to X04BAF.

### 4. References

None.

### 5. Parameters

1: NOUT – INTEGER.

*Input*

*On entry:* the Fortran unit number which identifies the file to be written to. If NOUT < 0 (not a valid Fortran unit number), then no output occurs. Within the NAG Fortran Library NOUT is always determined by a call to X04AAF or X04ABF.

2: REC – CHARACTER\*(\*) .

*Input*

*On entry:* a character-string. This is written to the external file as a single record. Trailing blanks are not output, except that if REC is entirely blank, a single blank character is output. If the record is printed, the first character is treated as a carriage-control character.

### 6. Error Indicators and Warnings

System-dependent errors may occur if the unit specified by NOUT is not connected to an external file, or if REC exceeds the maximum record-length for that file.

### 7. Accuracy

Not applicable.

### 8. Further Comments

None.

### 9. Example

This example program simply illustrates how a formatted record is output from the NAG Fortran Library, by first writing it to the character-string REC, used as an internal file, and then passing the character-string to X04BAF.

## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      X04BAF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
          INTEGER          NOUT
          PARAMETER        (NOUT=6)
*      .. Local Scalars ..
          CHARACTER*40     REC
*      .. External Subroutines ..
          EXTERNAL         X04BAF
*      .. Executable Statements ..
          WRITE (NOUT,*) 'X04BAF Example Program Results'
          WRITE (NOUT,*)
          WRITE (REC,99999) 'This record was output by X04BAF'

*
          CALL X04BAF(NOUT,REC)
*
          STOP
*
99999  FORMAT (1X,A)
          END
```

## 9.2. Program Data

None.

## 9.3. Program Results

X04BAF Example Program Results

This record was output by X04BAF

---

## X04BBF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04BBF reads a single formatted record from an external file.

### 2. Specification

```
SUBROUTINE X04BBF (NIN, REC, IFAIL)
  INTEGER      NIN, IFAIL
  CHARACTER*(*) REC
```

### 3. Description

X04BBF is used by NAG Fortran Library routines to read formatted records from an external file. All formatted input from an external file by NAG Fortran Library routines is performed by calls to X04BBF.

### 4. References

None.

### 5. Parameters

1: NIN – INTEGER.

*Input*

*On entry:* the Fortran unit number which identifies the file to be read from. If  $NIN < 0$  (not a valid Fortran unit number), then no input occurs.

2: REC – CHARACTER\*(\*) .

*Output*

*On exit:* the first  $LEN(REC)$  characters of the record read from unit NIN, padded with trailing blanks if the record was shorter than  $LEN(REC)$ .

3: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

### 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

An end-of-file was detected by the READ statement.

System-dependent errors may also occur if the unit specified by NIN is not connected to an external file, or if a read error occurs.

### 7. Accuracy

Not applicable.

### 8. Further Comments

None.

## 9. Example

This example program simply illustrates how a formatted record is read from the NAG Fortran Library, by first reading it into the character-string REC, used as an internal file, by X04BBF and then reading the internal file.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      X04BBF Example Program Text
*      Mark 14 Revised.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*      .. Local Scalars ..
      real             X
      INTEGER          I, IFAIL
      CHARACTER*40     REC
*      .. External Subroutines ..
      EXTERNAL         X04BBF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'X04BBF Example Program Results'
*      Skip heading in data file
      READ (NIN,*)
      WRITE (NOUT,*)

*
*      Read in values of I and X.
*
      CALL X04BBF(NIN,REC,IFAIL)
*
      READ (REC,99999) I, X
*
*      Write out I and X.
      WRITE (NOUT,99998) I, X
      STOP
*
99999 FORMAT (I3,F7.3)
99998 FORMAT (1X,I5,F11.3)
      END
```

### 9.2. Program Data

```
X04BBF Example Program Data
20 2.996
```

### 9.3. Program Results

```
X04BBF Example Program Results

20      2.996
```

---

## X04CAF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04CAF is an easy-to-use routine to print a *real* matrix stored in a two-dimensional array.

### 2. Specification

```

SUBROUTINE X04CAF (MATRIX, DIAG, M, N, A, LDA, TITLE, IFAIL)
  INTEGER          M, N, LDA, IFAIL
  real           A(LDA,*)
  CHARACTER*1     MATRIX, DIAG
  CHARACTER*(*)  TITLE

```

### 3. Description

X04CAF prints a *real* matrix. It is an easy-to-use driver for X04CBF. The routine uses default values for the format in which numbers are printed, for labelling the rows and columns, and for output record length.

X04CAF will choose a format code such that numbers will be printed with either an F8.4, F11.4 or a 1PE13.4 format. The F8.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 1.0. The F11.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 9999.9999. Otherwise the 1PE13.4 code is chosen.

The matrix is printed with integer row and column labels, and with a maximum record length of 80.

The matrix is output to the unit defined by X04ABF.

### 4. References

None.

### 5. Parameters

#### 1: MATRIX – CHARACTER\*1.

*Input*

*On entry:* indicates the part of the matrix to be printed, as follows:

MATRIX = 'G' or 'g' (General), the whole of the rectangular matrix.

MATRIX = 'L' or 'l' (Lower), the lower triangle of the matrix, or the lower trapezium if the matrix has more rows than columns.

MATRIX = 'U' or 'u' (Upper), the upper triangle of the matrix, or the upper trapezium if the matrix has more columns than rows.

*Constraint:* MATRIX must be one of 'G', 'g', 'L', 'l', 'U' or 'u'.

#### 2: DIAG – CHARACTER\*1.

*Input*

*On entry:* unless MATRIX = 'G' or 'g', DIAG must specify whether the diagonal elements of the matrix are to be printed, as follows:

DIAG = 'B' or 'b' (Blank), the diagonal elements of the matrix are not referenced and not printed.

DIAG = 'U' or 'u' (Unit diagonal), the diagonal elements of the matrix are not referenced, but are assumed all to be unity, and are printed as such.

DIAG = 'N' or 'n' (Non-unit diagonal), the diagonal elements of the matrix are referenced and printed.

If MATRIX = 'G' or 'g', then DIAG need not be set.

*Constraint:* If MATRIX  $\neq$  'G' or 'g', then DIAG must be one of 'B', 'b', 'U', 'u', 'N' or 'n'.

- 3: M – INTEGER. *Input*  
 4: N – INTEGER. *Input*

*On entry:* the number of rows and columns of the matrix, respectively, to be printed.

If either of M or N is less than 1, X04CAF will exit immediately after printing TITLE; no row or column labels are printed.

- 5: A(LDA,\*) – *real* array. *Input*

The second dimension of A must be at least  $\max(1, N)$ .

*On entry:* the matrix to be printed. Only the elements that will be referred to, as specified by parameters MATRIX and DIAG, need be set.

- 6: LDA – INTEGER. *Input*

*On entry:* the first dimension of the array A as declared in the (sub)program from which X04CAF is called.

*Constraint:*  $LDA \geq M$ .

- 7: TITLE – CHARACTER\*(\*). *Input*

*On entry:* a title to be printed above the matrix. If TITLE = ' ', no title (and no blank line) will be printed.

If TITLE contains more than 80 characters, the contents of TITLE will be wrapped onto more than one line, with the break after 80 characters.

Any trailing blank characters in TITLE are ignored.

- 8: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, MATRIX  $\neq$  'G', 'g', 'L', 'l', 'U' or 'u'.

IFAIL = 2

On entry, MATRIX = 'L', 'l', 'U' or 'u', but DIAG  $\neq$  'N', 'n', 'U', 'u', 'B' or 'b'.

IFAIL = 3

On entry,  $LDA < M$ .

## 7. Accuracy

Not applicable.

## 8. Further Comments

A call to X04CAF is equivalent to a call to X04CBF with the following argument values:

```

NCOLS = 80
INDENT = 0
LABROW = 'I'
LABCOL = 'I'
FORMAT = ' '

```

## 9. Example

This example program calls X04CAF twice, first to print a 3 by 5 rectangular matrix, and then to print a 5 by 5 lower triangular matrix.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      X04CAF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NOUT
PARAMETER        (NOUT=6)
INTEGER          NMAX, LDA
PARAMETER        (NMAX=5,LDA=NMAX)
*      .. Local Scalars ..
INTEGER          I, IFAIL, J
*      .. Local Arrays ..
real            A(LDA,NMAX)
*      .. External Subroutines ..
EXTERNAL         X04CAF
*      .. Executable Statements ..
WRITE (NOUT,*) 'X04CAF Example Program Results'
WRITE (NOUT,*)
*      Generate an array of data
DO 40 J = 1, NMAX
    DO 20 I = 1, LDA
        A(I,J) = 10*I + J
    20 CONTINUE
40 CONTINUE
*
    IFAIL = 0
*
*      Print 3 by 5 rectangular matrix
CALL X04CAF('General',' ',3,5,A,LDA,'Example 1:',IFAIL)
*
WRITE (NOUT,*)
*
*      Print 5 by 5 lower triangular matrix
CALL X04CAF('Lower','Non-unit',5,5,A,LDA,'Example 2:',IFAIL)
*
STOP
END

```

### 9.2. Program Data

None.

**9.3. Program Results**

## X04CAF Example Program Results

## Example 1:

	1	2	3	4	5
1	11.0000	12.0000	13.0000	14.0000	15.0000
2	21.0000	22.0000	23.0000	24.0000	25.0000
3	31.0000	32.0000	33.0000	34.0000	35.0000

## Example 2:

	1	2	3	4	5
1	11.0000				
2	21.0000	22.0000			
3	31.0000	32.0000	33.0000		
4	41.0000	42.0000	43.0000	44.0000	
5	51.0000	52.0000	53.0000	54.0000	55.0000

---



## X04CBF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04CBF prints a *real* matrix stored in a two-dimensional array.

### 2. Specification

```
SUBROUTINE X04CBF (MATRIX, DIAG, M, N, A, LDA, FORMAT, TITLE, LABROW,
                  RLABS, LABCOL, CLABS, NCOLS, INDENT, IFAIL)
```

```
INTEGER          M, N, LDA, NCOLS, INDENT, IFAIL
real            A(LDA, *)
```

```
CHARACTER*1     MATRIX, DIAG, LABROW, LABCOL
CHARACTER*(*)   FORMAT, TITLE, RLABS(*), CLABS(*)
```

### 3. Description

X04CBF prints a *real* matrix, or part of it, using a format specifier supplied by the user. The matrix is output to the unit defined by X04ABF.

### 4. References

None.

### 5. Parameters

1: MATRIX – CHARACTER\*1.

*Input*

*On entry:* indicates the part of the matrix to be printed, as follows:

MATRIX = 'G' or 'g' (General), the whole of the rectangular matrix.

MATRIX = 'L' or 'l' (Lower), the lower triangle of the matrix, or the lower trapezium if the matrix has more rows than columns.

MATRIX = 'U' or 'u' (Upper), the upper triangle of the matrix, or the upper trapezium if the matrix has more columns than rows.

*Constraint:* MATRIX must be one of 'G', 'g', 'L', 'l', 'U' or 'u'.

2: DIAG – CHARACTER\*1.

*Input*

*On entry:* unless MATRIX = 'G' or 'g', DIAG must specify whether the diagonal elements of the matrix are to be printed, as follows:

DIAG = 'B' or 'b' (Blank), the diagonal elements of the matrix are not referenced and not printed.

DIAG = 'U' or 'u' (Unit diagonal), the diagonal elements of the matrix are not referenced, but are assumed all to be unity, and are printed as such.

DIAG = 'N' or 'n' (Non-unit diagonal), the diagonal elements of the matrix are referenced and printed.

If MATRIX = 'G' or 'g', then DIAG need not be set.

*Constraint:* If MATRIX ≠ 'G' or 'g', then DIAG must be one of 'B', 'b', 'U', 'u', 'N' or 'n'.

3: M – INTEGER.

*Input*

4: N – INTEGER.

*Input*

*On entry:* the number of rows and columns of the matrix, respectively, to be printed.

If either of M or N is less than 1, X04CBF will exit immediately after printing TITLE; no row or column labels are printed.

- 5: A(LDA,\*) – *real* array. *Input*  
 The second dimension of A must be at least max(1,N).  
*On entry:* the matrix to be printed. Only the elements that will be referred to, as specified by parameters MATRIX and DIAG, need be set.
- 6: LDA – INTEGER. *Input*  
*On entry:* the first dimension of the array A as declared in the (sub)program from which X04CBF is called.  
*Constraint:* LDA ≥ M.
- 7: FORMAT – CHARACTER\*(\*). *Input*  
*On entry:* a valid Fortran format code. This may be any format code allowed on the system, whether it is standard Fortran or not. FORMAT is used to print elements of the matrix A. It may or may not be enclosed in brackets. Examples of valid values for FORMAT are '(F11.4)', '1PE13.5', 'G14.5'.  
 In addition, there are two special codes which force X04CBF to choose its own format code:  
 FORMAT = ' ' means that X04CBF will choose a format code such that numbers will be printed with either an F8.4, an F11.4 or a 1PE13.4 format. The F8.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 1.0. The F11.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 9999.9999. Otherwise the 1PE13.4 code is chosen.  
 FORMAT = '\*' means that X04CBF will choose a format code such that numbers will be printed to as many significant digits as are necessary to distinguish between neighbouring machine numbers. Thus any two numbers that are stored with different internal representations should look different on output. Whether they do in fact look different will depend on the run-time library of the Fortran compiler in use.  
*Constraint:* the character length of FORMAT must be ≤ 80.
- 8: TITLE – CHARACTER\*(\*). *Input*  
*On entry:* a title to be printed above the matrix. If TITLE = ' ', no title (and no blank line) will be printed.  
 If TITLE contains more than NCOLS characters, the contents of TITLE will be wrapped onto more than one line, with the break after NCOLS characters.  
 Any trailing blank characters in TITLE are ignored.
- 9: LABROW – CHARACTER\*1. *Input*  
*On entry:* indicates the type of labelling to be applied to the rows of the matrix, as follows:  
 If LABROW = 'N' or 'n', X04CBF prints no row labels.  
 If LABROW = 'I' or 'i', X04CBF prints integer row labels.  
 If LABROW = 'C' or 'c', X04CBF prints character labels, which must be supplied in array RLABS.  
*Constraint:* LABROW must be one of 'N', 'n', 'I', 'i', 'C', or 'c'.
- 10: RLABS(\*) – CHARACTER\*(\*) array. *Input*  
*On entry:* if LABROW = 'C' or 'c', RLABS must be dimensioned at least of length M and must contain labels for the rows of the matrix, otherwise RLABS may be dimensioned of length 1.  
 Labels are right justified when output, in a field which is as wide as necessary to hold the longest row label. Note that this field width is subtracted from the number of usable columns, NCOLS.

- 11: LABCOL – CHARACTER\*1. *Input*  
*On entry:* indicates the type of labelling to be applied to the columns of the matrix, as follows:  
 If LABCOL = 'N' or 'n', X04CBF prints no column labels.  
 If LABCOL = 'I' or 'i', X04CBF prints integer column labels.  
 If LABCOL = 'C' or 'c', X04CBF prints character labels, which must be supplied in array CLABS.  
*Constraint:* LABCOL must be one of 'N', 'n', 'I', 'i', 'C', 'c'.
- 12: CLABS(\*) – CHARACTER\*(\*) array. *Input*  
*On entry:* if LABCOL = 'C' or 'c', CLABS must be dimensioned at least of length N and must contain labels for the columns of the matrix, otherwise CLABS may be dimensioned of length 1.  
 Labels are right-justified when output. Any label that is too long for the column width, which is determined by FORMAT, is truncated.
- 13: NCOLS – INTEGER. *Input*  
*On entry:* the maximum output record length. If the number of columns of the matrix is too large to be accommodated in NCOLS characters, the matrix will be printed in parts, containing the largest possible number of matrix columns, and each part separated by a blank line.  
 NCOLS must be large enough to hold at least one column of the matrix using the format specifier in FORMAT. If a value less than 0 or greater than 132 is supplied for NCOLS, then the value 80 is used instead.
- 14: INDENT – INTEGER. *Input*  
*On entry:* the number of columns by which the matrix (and any title and labels) should be indented. The effective value of NCOLS is reduced by INDENT columns. If a value less than 0 or greater than NCOLS is supplied for INDENT, the value 0 is used instead.
- 15: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, MATRIX ≠ 'G', 'g', 'L', 'l', 'U' or 'u'.

IFAIL = 2

On entry, MATRIX = 'L', 'l', 'U' or 'u', but DIAG ≠ 'N', 'n', 'U', 'u', 'B' or 'b'.

IFAIL = 3

On entry, LDA < M.

IFAIL = 4

On entry, variable FORMAT is more than 80 characters long.

IFAIL = 5

The code supplied in FORMAT cannot be used to output a number. FORMAT probably has too wide a field width or contains an illegal edit descriptor.

IFAIL = 6

On entry, either LABROW or LABCOL  $\neq$  'N', 'n', 'T', 't', 'C' or 'c'.

IFAIL = 7

The quantity NCOLS – INDENT – LABWID (where LABWID is the width needed for the row labels), is not large enough to hold at least one column of the matrix.

## 7. Accuracy

Not applicable.

## 8. Further Comments

X04CBF may be used to print a vector, either as a row or as a column. The following code fragment illustrates possible calls.

```

      real A(4)
      CHARACTER*1 RLABS(1), CLABS(1)
C   Print vector A as a column vector.
      LDA = 4
      IFAIL = 0
      CALL X04CBF('G', 'X', 1, 4, A, LDA, ' ', ' ', 'I', RLABS,
*              'N', CLABS, 80, 0, IFAIL)
C   Print vector A as a row vector.
      LDA = 1
      IFAIL = 0
      CALL X04CBF('G', 'X', 4, 1, A, LDA, ' ', ' ', 'N', RLABS,
*              'I', CLABS, 80, 0, IFAIL)

```

## 9. Example

This example program calls X04CBF twice, first to print a 3 by 5 rectangular matrix, and then to print a 5 by 5 upper triangular matrix, various options for labelling and formatting are illustrated.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*   X04CBF Example Program Text
*   Mark 14 Release.  NAG Copyright 1989.
*   .. Parameters ..
      INTEGER          NOUT
      PARAMETER       (NOUT=6)
      INTEGER          NMAX, LDA
      PARAMETER       (NMAX=5, LDA=NMAX)
*   .. Local Scalars ..
      INTEGER          I, IFAIL, INDENT, J, NCOLS
*   .. Local Arrays ..
      real             A(LDA, NMAX)
      CHARACTER*7      CLABS(NMAX), RLABS(NMAX)
*   .. External Subroutines ..
      EXTERNAL         X04CBF
*   .. Data statements ..
      DATA            CLABS/'Un', 'Deux', 'Trois', 'Quatre', 'Cinq'/
      DATA            RLABS/'Uno', 'Duo', 'Tre', 'Quattro', 'Cinque'/
*   .. Executable Statements ..
      WRITE (NOUT,*) 'X04CBF Example Program Results'
      WRITE (NOUT,*)

```

```

*      Generate an array of data
      DO 40 J = 1, NMAX
        DO 20 I = 1, LDA
          A(I,J) = 10*I + J
20      CONTINUE
40     CONTINUE
      NCOLS = 80
      INDENT = 0
      IFAIL = 0
*
*      Print 3 by 5 rectangular matrix with default format and integer
*      row and column labels
      CALL X04CBF('General', ' ', 3, 5, A, LDA, ' ', 'Example 1:', 'Integer',
+              RLABS, 'Integer', CLABS, NCOLS, INDENT, IFAIL)
*
      WRITE (NOUT, *)
*
*      Print 5 by 5 upper triangular matrix with user-supplied format
*      and row and column labels
      CALL X04CBF('Upper', 'Non-unit', 5, 5, A, LDA, 'F8.2', 'Example 2:',
+              'Character', RLABS, 'Character', CLABS, NCOLS, INDENT,
+              IFAIL)
*
      STOP
      END

```

## 9.2. Program Data

None.

## 9.3. Program Results

X04CBF Example Program Results

Example 1:

	1	2	3	4	5
1	11.0000	12.0000	13.0000	14.0000	15.0000
2	21.0000	22.0000	23.0000	24.0000	25.0000
3	31.0000	32.0000	33.0000	34.0000	35.0000

Example 2:

	Un	Deux	Trois	Quatre	Cinq
Uno	11.00	12.00	13.00	14.00	15.00
Duo		22.00	23.00	24.00	25.00
Tre			33.00	34.00	35.00
Quattro				44.00	45.00
Cinque					55.00

---



## X04CCF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04CCF is an easy-to-use routine to print a *real* triangular matrix stored in a packed one-dimensional array.

### 2. Specification

```

SUBROUTINE X04CCF (UPLO, DIAG, N, A, TITLE, IFAIL)
  INTEGER          N, IFAIL
  real            A(*)
  CHARACTER*1      UPLO, DIAG
  CHARACTER*(*)    TITLE

```

### 3. Description

X04CCF prints a *real* triangular matrix stored in packed form. It is an easy-to-use driver for X04CDF. The routine uses default values for the format in which numbers are printed, for labelling the rows and columns, and for output record length. The matrix must be packed by column.

X04CCF will choose a format code such that numbers will be printed with either an F8.4, F11.4 or a 1PE13.4 format. The F8.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 1.0. The F11.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 9999.9999. Otherwise the 1PE13.4 code is chosen.

The matrix is printed with integer row and column labels, and with a maximum record length of 80.

The matrix is output to the unit defined by X04ABF.

### 4. References

None.

### 5. Parameters

1: UPLO – CHARACTER\*1.

*Input*

*On entry:* indicates the type of the matrix to be printed, as follows:

UPLO = 'L' or 'l' (Lower), the matrix is lower triangular. In this case, the packed array A holds the matrix elements in the following order: (1,1), (2,1),..., (N,1), (2,2), (3,2),..., (N,2), etc.

UPLO = 'U' or 'u' (Upper), the matrix is upper triangular. In this case, the packed array A holds the matrix elements in the following order: (1,1), (1,2), (2,2), (1,3), (2,3), (3,3), (1,4), etc.

*Constraint:* UPLO must be one of 'L', 'l', 'U' or 'u'.

2: DIAG – CHARACTER\*1.

*Input*

*On entry:* indicates whether the diagonal elements of the matrix are to be printed, as follows:

DIAG = 'B' or 'b' (Blank), the diagonal elements of the matrix are not referenced and not printed.

DIAG = 'U' or 'u' (Unit diagonal), the diagonal elements of the matrix are not referenced, but are assumed all to be unity, and are printed as such.

DIAG = 'N' or 'n' (Non-unit diagonal), the diagonal elements of the matrix are referenced and printed.

*Constraint:* DIAG must be one of 'B', 'b', 'U', 'u', 'N' or 'n'.

3: N – INTEGER. *Input*

*On entry:* the order of the matrix to be printed.

If N is less than 1, X04CCF will exit immediately after printing TITLE; no row or column labels are printed.

4: A(\*) – *real* array. *Input*

The dimension of A must be at least  $\max(1, N*(N+1)/2)$ .

*On entry:* the matrix to be printed. Note that A must have space for the diagonal elements of the matrix, even if these are not stored.

5: TITLE – CHARACTER\*(\*). *Input*

*On entry:* a title to be printed above the matrix. If TITLE = ' ', no title (and no blank line) will be printed.

If TITLE contains more than 80 characters, the contents of TITLE will be wrapped onto more than one line, with the break after 80 characters.

Any trailing blank characters in TITLE are ignored.

6: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, UPLO  $\neq$  'L', 'l', 'U' or 'u'.

IFAIL = 2

On entry, DIAG  $\neq$  'N', 'n', 'U', 'u', 'B', or 'b'.

## 7. Accuracy

Not applicable.

## 8. Further Comments

A call to X04CCF is equivalent to a call to X04CDF with the following argument values:

```

NCOLS = 80
INDENT = 0
LABROW = 'I'
LABCOL = 'I'
FORMAT = ' '
```

## 9. Example

This example program calls X04CCF twice, first to print a 4 by 4 lower triangular matrix, and then to print a 5 by 5 upper triangular matrix.



## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      X04CCF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NMAX, LA
      PARAMETER        (NMAX=5, LA=(NMAX*(NMAX+1))/2)
*      .. Local Scalars ..
      INTEGER          I, IFAIL
*      .. Local Arrays ..
      real            A(LA)
*      .. External Subroutines ..
      EXTERNAL         X04CCF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'X04CCF Example Program Results'
      WRITE (NOUT,*)
*
*      Generate an array of data
      DO 20 I = 1, LA
          A(I) = I
20 CONTINUE
*
      IFAIL = 0
*
*      Print order 4 lower triangular matrix
      CALL X04CCF('Lower', 'Unit', 4, A, 'Example 1:', IFAIL)
*
      WRITE (NOUT,*)
*
*      Print order 5 upper triangular matrix
      CALL X04CCF('Upper', 'Non-unit', 5, A, 'Example 2:', IFAIL)
*
      STOP
      END

```

## 9.2. Program Data

None.

## 9.3. Program Results

X04CCF Example Program Results

Example 1:

	1	2	3	4
1	1.0000			
2	2.0000	1.0000		
3	3.0000	6.0000	1.0000	
4	4.0000	7.0000	9.0000	1.0000

Example 2:

	1	2	3	4	5
1	1.0000				
2		2.0000	4.0000	7.0000	11.0000
3		3.0000	5.0000	8.0000	12.0000
4			6.0000	9.0000	13.0000
5				10.0000	14.0000
					15.0000

---



## X04CDF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04CDF prints a *real* triangular matrix stored in a packed one-dimensional array.

### 2. Specification

```

SUBROUTINE X04CDF (UPLO, DIAG, N, A, FORMAT, TITLE, LABROW, RLABS,
1                 LABCOL, CLABS, NCOLS, INDENT, IFAIL)
    INTEGER          N, NCOLS, INDENT, IFAIL
    real            A(*)
    CHARACTER*1      UPLO, DIAG, LABROW, LABCOL
    CHARACTER*(*)    FORMAT, TITLE, RLABS(*), CLABS(*)

```

### 3. Description

X04CDF prints a *real* triangular matrix stored in packed form, using a format specifier supplied by the user. The matrix must be packed by column. The matrix is output to the unit defined by X04ABF.

### 4. References

None.

### 5. Parameters

- 1: UPLO – CHARACTER\*1. *Input*
- On entry:* indicates the type of the matrix to be printed, as follows:
- UPLO = 'L' or 'l' (Lower), the matrix is lower triangular. In this case, the packed array A holds the matrix elements in the following order: (1,1), (2,1),..., (N,1), (2,2), (3,2),..., (N,2), etc.
- UPLO = 'U' or 'u' (Upper), the matrix is upper triangular. In this case, the packed array A holds the matrix elements in the following order: (1,1), (1,2), (2,2), (1,3), (2,3), (3,3), (1,4), etc.
- Constraint:* UPLO must be one of 'L', 'l', 'U' or 'u'.
- 2: DIAG – CHARACTER\*1. *Input*
- On entry:* indicates whether the diagonal elements of the matrix are to be printed, as follows:
- DIAG = 'B' or 'b' (Blank), the diagonal elements of the matrix are not referenced and not printed.
- DIAG = 'U' or 'u' (Unit diagonal), the diagonal elements of the matrix are not referenced, but are assumed all to be unity, and are printed as such.
- DIAG = 'N' or 'n' (Non-unit diagonal), the diagonal elements of the matrix are referenced and printed.
- Constraint:* DIAG must be one of 'B', 'b', 'U', 'u', 'N' or 'n'.
- 3: N – INTEGER. *Input*
- On entry:* the order of the matrix to be printed.
- If N is less than 1, X04CDF will exit immediately after printing TITLE; no row or column labels are printed.

- 4: A(\*) – *real* array. *Input*  
 The dimension of A must be at least  $\max(1, N*(N+1)/2)$ .  
*On entry:* the matrix to be printed. Note that A must have space for the diagonal elements of the matrix, even if these are not stored.
- 5: FORMAT – CHARACTER\*(\*). *Input*  
*On entry:* a valid Fortran format code. This may be any format code allowed on the system, whether it is standard Fortran or not. FORMAT is used to print elements of the matrix A. It may or may not be enclosed in brackets. Examples of valid values for FORMAT are '(F11.4)', '1PE13.5', 'G14.5'.  
 In addition, there are two special codes which force X04CDF to choose its own format code:  
 FORMAT = ' ' means that X04CDF will choose a format code such that numbers will be printed with either an F8.4, an F11.4 or a 1PE13.4 format. The F8.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 1.0. The F11.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 9999.9999. Otherwise the 1PE13.4 code is chosen.  
 FORMAT = '\*' means that X04CDF will choose a format code such that numbers will be printed to as many significant digits as are necessary to distinguish between neighbouring machine numbers. Thus any two numbers that are stored with different internal representations should look different on output. Whether they do in fact look different will depend on the run-time library of the Fortran compiler in use.  
*Constraint:* the character length of FORMAT must be  $\leq 80$ .
- 6: TITLE – CHARACTER\*(\*). *Input*  
*On entry:* a title to be printed above the matrix. If TITLE = ' ', no title (and no blank line) will be printed.  
 If TITLE contains more than NCOLS characters, the contents of TITLE will be wrapped onto more than one line, with the break after NCOLS characters.  
 Any trailing blank characters in TITLE are ignored.
- 7: LABROW – CHARACTER\*1. *Input*  
*On entry:* indicates the type of labelling to be applied to the rows of the matrix, as follows:  
 If LABROW = 'N' or 'n', X04CDF prints no row labels.  
 If LABROW = 'I' or 'i', X04CDF prints integer row labels.  
 If LABROW = 'C' or 'c', X04CDF prints character labels, which must be supplied in array RLABS.  
*Constraint:* LABROW must be one of 'N', 'n', 'I', 'i', 'C' or 'c'.
- 8: RLABS(\*) – CHARACTER\*(\*) array. *Input*  
*On entry:* if LABROW = 'C' or 'c', RLABS must be dimensioned at least of length N and must contain labels for the rows of the matrix, otherwise RLABS may be dimensioned of length 1.  
 Labels are right justified when output, in a field which is as wide as necessary to hold the longest row label. Note that this field width is subtracted from the number of usable columns, NCOLS.

- 9: LABCOL – CHARACTER\*1. *Input*  
*On entry:* indicates the type of labelling to be applied to the columns of the matrix, as follows:  
 If LABCOL = 'N' or 'n', X04CDF prints no column labels.  
 If LABCOL = 'I' or 'i', X04CDF prints integer column labels.  
 If LABCOL = 'C' or 'c', X04CDF prints character labels, which must be supplied in array CLABS.  
*Constraint:* LABCOL must be one of 'N', 'n', 'I', 'i', 'C' or 'c'.
- 10: CLABS(\*) – CHARACTER\*(\*) array. *Input*  
*On entry:* if LABCOL = 'C' or 'c', CLABS must be dimensioned at least of length N and must contain labels for the columns of the matrix, otherwise CLABS may be dimensioned of length 1.  
 Labels are right-justified when output. Any label that is too long for the column width, which is determined by FORMAT, is truncated.
- 11: NCOLS – INTEGER. *Input*  
*On entry:* the maximum output record length. If the number of columns of the matrix is too large to be accommodated in NCOLS characters, the matrix will be printed in parts, containing the largest possible number of matrix columns, and each part separated by a blank line.  
 NCOLS must be large enough to hold at least one column of the matrix using the format specifier in FORMAT. If a value less than 0 or greater than 132 is supplied for NCOLS, then the value 80 is used instead.
- 12: INDENT – INTEGER. *Input*  
*On entry:* the number of columns by which the matrix (and any title and labels) should be indented. The effective value of NCOLS is reduced by INDENT columns. If a value less than 0 or greater than NCOLS is supplied for INDENT, the value 0 is used instead.
- 13: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, UPLO ≠ 'L', 'l', 'U' or 'u'.

IFAIL = 2

On entry, DIAG ≠ 'N', 'n', 'U', 'u', 'B', or 'b'.

IFAIL = 3

On entry, variable FORMAT is more than 80 characters long.

IFAIL = 4

The code supplied in FORMAT cannot be used to output a number. FORMAT probably has too wide a field width or contains an illegal edit descriptor.

IFAIL = 5

On entry, either LABROW or LABCOL  $\neq$  'N', 'n', 'T', 't', 'C' or 'c'.

IFAIL = 6

The quantity NCOLS – INDENT – LABWID (where LABWID is the width needed for the row labels) is not large enough to hold at least one column of the matrix.

## 7. Accuracy

Not applicable.

## 8. Further Comments

None.

## 9. Example

This example program calls X04CDF twice, first to print a 4 by 4 lower triangular matrix, and then to print a 5 by 5 upper triangular matrix; various options for labelling and formatting are illustrated.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      X04CDF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NOUT
PARAMETER       (NOUT=6)
INTEGER          NMAX, LA
PARAMETER       (NMAX=5, LA=(NMAX*(NMAX+1))/2)
*      .. Local Scalars ..
INTEGER          I, IFAIL, INDENT, NCOLS
*      .. Local Arrays ..
real           A(LA)
CHARACTER*7     CLABS(NMAX), RLABS(NMAX)
*      .. External Subroutines ..
EXTERNAL        X04CDF
*      .. Data statements ..
DATA           CLABS/'Un', 'Deux', 'Trois', 'Quatre', 'Cinq'/
DATA           RLABS/'Uno', 'Duo', 'Tre', 'Quattro', 'Cinque'/
*      .. Executable Statements ..
WRITE (NOUT,*) 'X04CDF Example Program Results'
WRITE (NOUT,*)

*      Generate an array of data
DO 20 I = 1, LA
    A(I) = I
20 CONTINUE
*
    NCOLS = 80
    INDENT = 0
    IFAIL = 0
*
*      Print order 4 lower triangular matrix with default format and
*      integer row and column labels
CALL X04CDF('Lower', 'Non-unit', 4, A, ' ', 'Example 1:', 'Integer',
+         RLABS, 'Integer', CLABS, NCOLS, INDENT, IFAIL)
*
WRITE (NOUT,*)
*

```

```

*   Print order 5 upper triangular matrix with user-supplied format
*   and row and column labels
*   CALL X04CDF('Upper','Unit',5,A,'F8.2','Example 2:','Character',
+             RLABS,'Character',CLABS,NCOLS,INDENT,IFAIL)
*
      STOP
      END

```

## 9.2. Program Data

None.

## 9.3. Program Results

X04CDF Example Program Results

Example 1:

	1	2	3	4
1	1.0000			
2	2.0000	5.0000		
3	3.0000	6.0000	8.0000	
4	4.0000	7.0000	9.0000	10.0000

Example 2:

	Un	Deux	Trois	Quatre	Cinq
Uno	1.00	2.00	4.00	7.00	11.00
Duo		1.00	5.00	8.00	12.00
Tre			1.00	9.00	13.00
Quattro				1.00	14.00
Cinque					1.00

---





## X04CEF - NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04CEF is an easy-to-use routine to print a *real* band matrix stored in a packed two-dimensional array.

### 2. Specification

```

SUBROUTINE X04CEF (M, N, KL, KU, A, LDA, TITLE, IFAIL)
  INTEGER          M, N, KL, KU, LDA, IFAIL
  real           A(LDA, *)
  CHARACTER*(*)   TITLE

```

### 3. Description

X04CEF prints a *real* band matrix stored in a packed two-dimensional array. It is an easy-to-use driver for X04CFF. The routine uses default values for the format in which numbers are printed, for labelling the rows and columns, and for output record length.

X04CEF will choose a format code such that numbers will be printed with either an F8.4, F11.4 or a 1PE13.4 format. The F8.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 1.0. The F11.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 9999.9999. Otherwise the 1PE13.4 code is chosen.

The matrix is printed with integer row and column labels, and with a maximum record length of 80.

The matrix is output to the unit defined by X04ABF.

### 4. References

None.

### 5. Parameters

1: M - INTEGER.

*Input*

2: N - INTEGER.

*Input*

*On entry:* the number of rows and columns of the band matrix, respectively, to be printed.

If either of M or N is less than 1, X04CEF will exit immediately after printing TITLE; no row or column labels are printed.

3: KL - INTEGER.

*Input*

*On entry:* the number of sub-diagonals of the band matrix A.

*Constraint:*  $KL \geq 0$ .

4: KU - INTEGER.

*Input*

*On entry:* the number of super-diagonals of the band matrix A.

*Constraint:*  $KU \geq 0$ .

5: A(LDA,\*) - *real* array.

*Input*

**Note:** the second dimension of the array A must be at least  $\max(1, \min(M+KU, N))$ .

*On entry:* the band matrix to be printed. The leading  $(KL+KU+1)$  by  $\min(M+KU, N)$  part of array A must contain the matrix, packed column by column, with the leading diagonal of the matrix in row  $(KU+1)$  of the array, the first super-diagonal starting at position 2 in row KU, the first sub-diagonal starting at position 1 in row  $(KU+2)$ , and so on. Elements in the

array A that do not correspond to elements in the band matrix (such as the top left KU by KU triangle) are not referenced, and need not be set.

- 6: LDA – INTEGER. *Input*  
*On entry:* the first dimension of the array A as declared in the (sub)program from which X04CEF is called.  
*Constraint:*  $LDA \geq KL + KU + 1$ .
- 7: TITLE – CHARACTER\*(\*). *Input*  
*On entry:* a title to be printed above the matrix. If TITLE = ' ', no title (and no blank line) will be printed.  
 If TITLE contains more than 80 characters, the contents of TITLE will be wrapped onto more than one line, with the break after 80 characters.  
 Any trailing blank characters in TITLE are ignored.
- 8: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry,  $KL < 0$ .

IFAIL = 2

On entry,  $KU < 0$ .

IFAIL = 3

On entry,  $LDA < KL + KU + 1$ .

## 7. Accuracy

Not applicable.

## 8. Further Comments

A call to X04CEF is equivalent to a call to X04CFF with the following argument values:

```

NCOLS = 80
INDENT = 0
LABROW = 'I'
LABCOL = 'I'
FORMAT = ' / '

```

## 9. Example

This example program calls X04CEF to print a 5 by 5 band matrix with one sub-diagonal and one super-diagonal.

## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      X04CEF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NMAX, LDA
      PARAMETER        (NMAX=5, LDA=NMAX)
*      .. Local Scalars ..
      INTEGER          I, IFAIL, J
*      .. Local Arrays ..
      real            A(LDA,NMAX)
*      .. External Subroutines ..
      EXTERNAL         X04CEF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'X04CEF Example Program Results'
      WRITE (NOUT,*)

*
*      Generate an array of data
      DO 40 J = 1, NMAX
        DO 20 I = 1, LDA
          A(I,J) = 10*I + J
20      CONTINUE
40      CONTINUE

*
      IFAIL = 0

*
*      Print 5 by 5 band matrix with 1 sub-diagonal and 1 super-diagonal
      CALL X04CEF(5,5,1,1,A,LDA,'Band Matrix:',IFAIL)
*

      STOP
      END

```

## 9.2. Program Data

None.

## 9.3. Program Results

X04CEF Example Program Results

Band Matrix:

	1	2	3	4	5
1	21.0000	12.0000			
2	31.0000	22.0000	13.0000		
3		32.0000	23.0000	14.0000	
4			33.0000	24.0000	15.0000
5				34.0000	25.0000

---



## X04CFF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

## 1. Purpose

X04CFF prints a *real* band matrix stored in a packed two-dimensional array.

## 2. Specification

```

SUBROUTINE X04CFF (M, N, KL, KU, A, LDA, FORMAT, TITLE, LABROW,
1                RLABS, LABCOL, CLABS, NCOLS, INDENT, IFAIL)
    INTEGER          M, N, KL, KU, LDA, NCOLS, INDENT, IFAIL
    real            A(LDA, *)
    CHARACTER*1     LABROW, LABCOL
    CHARACTER*(*)   FORMAT, TITLE, RLABS(*), CLABS(*)

```

## 3. Description

X04CFF prints a *real* band matrix stored in a packed two-dimensional array, using a format specifier supplied by the user. The matrix is output to the unit defined by X04ABF.

## 4. References

None.

## 5. Parameters

- 1: M – INTEGER. *Input*
- 2: N – INTEGER. *Input*
- On entry:* the number of rows and columns of the band matrix, respectively, to be printed. If either of M or N is less than 1, X04CFF will exit immediately after printing TITLE; no row or column labels are printed.
- 3: KL – INTEGER. *Input*
- On entry:* the number of sub-diagonals of the band matrix A.  
*Constraint:*  $KL \geq 0$ .
- 4: KU – INTEGER. *Input*
- On entry:* the number of super-diagonals of the band matrix A.  
*Constraint:*  $KU \geq 0$ .
- 5: A(LDA,\*) – *real* array. *Input*
- The second dimension of A must be at least  $\max(1, \min(M+KU, N))$ .  
*On entry:* the band matrix to be printed. The leading  $(KL+KU+1)$  by  $\min(M+KU, N)$  part of array A must contain the matrix, packed column by column, with the leading diagonal of the matrix in row  $(KU+1)$  of the array, the first super-diagonal starting at position 2 in row KU, the first sub-diagonal starting at position 1 in row  $(KU+2)$ , and so on. Elements in the array A that do not correspond to elements in the band matrix (such as the top left KU by KU triangle) are not referenced, and need not be set.
- 6: LDA – INTEGER. *Input*
- On entry:* the first dimension of the array A as declared in the (sub)program from which X04CFF is called.  
*Constraint:*  $LDA \geq KL + KU + 1$ .

## 7: FORMAT – CHARACTER\*(\*).

*Input*

*On entry:* a valid Fortran format code. This may be any format code allowed on the system, whether it is standard Fortran or not. FORMAT is used to print elements of the matrix *A*. It may or may not be enclosed in brackets. Examples of valid values for FORMAT are '(F11.4)', '1PE13.5', 'G14.5'.

In addition, there are two special codes which force X04CFF to choose its own format code:

FORMAT = ' ' means that X04CFF will choose a format code such that numbers will be printed with either an F8.4, an F11.4 or a 1PE13.4 format. The F8.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 1.0. The F11.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 9999.9999. Otherwise the 1PE13.4 code is chosen.

FORMAT = '\*' means that X04CFF will choose a format code such that numbers will be printed to as many significant digits as are necessary to distinguish between neighbouring machine numbers. Thus any two numbers that are stored with different internal representations should look different on output. Whether they do in fact look different will depend on the run-time library of the Fortran compiler in use.

*Constraint:* the character length of FORMAT must be ≤ 80.

## 8: TITLE – CHARACTER\*(\*).

*Input*

*On entry:* a title to be printed above the matrix. If TITLE = ' ', no title (and no blank line) will be printed.

If TITLE contains more than NCOLS characters, the contents of TITLE will be wrapped onto more than one line, with the break after NCOLS characters.

Any trailing blank characters in TITLE are ignored.

## 9: LABROW – CHARACTER\*1.

*Input*

*On entry:* indicates the type of labelling to be applied to the rows of the matrix, as follows:

If LABROW = 'N' or 'n', X04CFF prints no row labels.

If LABROW = 'I' or 'i', X04CFF prints integer row labels.

If LABROW = 'C' or 'c', X04CFF prints character labels, which must be supplied in array RLABS.

*Constraint:* LABROW must be one of 'N', 'n', 'I', 'i', 'C' or 'c'.

## 10: RLABS(\*) – CHARACTER\*(\*) array.

*Input*

*On entry:* if LABROW = 'C' or 'c', RLABS must be dimensioned at least of length M and must contain labels for the rows of the matrix, otherwise RLABS may be dimensioned of length 1.

Labels are right justified when output, in a field which is as wide as necessary to hold the longest row label. Note that this field width is subtracted from the number of usable columns, NCOLS.

## 11: LABCOL – CHARACTER\*1.

*Input*

*On entry:* indicates the type of labelling to be applied to the columns of the matrix, as follows:

If LABCOL = 'N' or 'n', X04CFF prints no column labels.

If LABCOL = 'I' or 'i', X04CFF prints integer column labels.

If LABCOL = 'C' or 'c', X04CFF prints character labels, which must be supplied in array CLABS.

*Constraint:* LABCOL must be one of 'N', 'n', 'I', 'i', 'C' or 'c'.

- 12: CLABS(\*) – CHARACTER\*(\*) array. *Input*

*On entry:* if LABCOL = 'C' or 'c', CLABS must be dimensioned at least of length N and must contain labels for the columns of the matrix, otherwise CLABS may be dimensioned of length 1.

Labels are right-justified when output. Any label that is too long for the column width, which is determined by FORMAT, is truncated.

- 13: NCOLS – INTEGER. *Input*

*On entry:* the maximum output record length. If the number of columns of the matrix is too large to be accommodated in NCOLS characters, the matrix will be printed in parts, containing the largest possible number of matrix columns, and each part separated by a blank line.

NCOLS must be large enough to hold at least one column of the matrix using the format specifier in FORMAT. If a value less than 0 or greater than 132 is supplied for NCOLS, then the value 80 is used instead.

- 14: INDENT – INTEGER. *Input*

*On entry:* the number of columns by which the matrix (and any title and labels) should be indented. The effective value of NCOLS is reduced by INDENT columns. If a value less than 0 or greater than NCOLS is supplied for INDENT, the value 0 is used instead.

- 15: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, KL < 0.

IFAIL = 2

On entry, KU < 0.

IFAIL = 3

On entry, LDA < KL + KU + 1.

IFAIL = 4

On entry, variable FORMAT is more than 80 characters long.

IFAIL = 5

The code supplied in FORMAT cannot be used to output a number. FORMAT probably has too wide a field width or contains an illegal edit descriptor.

IFAIL = 6

On entry, either LABROW or LABCOL ≠ 'N', 'n', 'T', 't', 'C' or 'c'.

IFAIL = 7

The quantity NCOLS – INDENT – LABWID (where LABWID is the width needed for the row labels) is not large enough to hold at least one column of the matrix.

## 7. Accuracy

Not applicable.

## 8. Further Comments

None.

## 9. Example

This example program calls X04CFF twice, to print 5 by 5 matrices of different bandwidths; various options for labelling and formatting are illustrated.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      X04CFF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NOUT
PARAMETER       (NOUT=6)
INTEGER          NMAX, LDA
PARAMETER       (NMAX=5, LDA=NMAX)
*      .. Local Scalars ..
INTEGER          I, IFAIL, INDENT, J, NCOLS
*      .. Local Arrays ..
real           A(LDA,LDA)
CHARACTER*7     CLABS(NMAX), RLABS(NMAX)
*      .. External Subroutines ..
EXTERNAL        X04CFF
*      .. Data statements ..
DATA           CLABS/'Un', 'Deux', 'Trois', 'Quatre', 'Cinq'/
DATA           RLABS/'Uno', 'Duo', 'Tre', 'Quattro', 'Cinque'/
*      .. Executable Statements ..
WRITE (NOUT,*) 'X04CFF Example Program Results'
WRITE (NOUT,*)

*
*      Generate an array of data
DO 40 J = 1, NMAX
  DO 20 I = 1, LDA
    A(I,J) = 10*I + J
20  CONTINUE
40  CONTINUE

*
  NCOLS = 80
  INDENT = 0
  IFAIL = 0

*
*      Print 5 by 5 band matrix with 1 sub-diagonal, 1 super-diagonal,
*      default format and integer row and column labels
CALL X04CFF(5,5,1,1,A,LDA,' ','Example 1:', 'Integer',RLABS,
+          'Integer',CLABS,NCOLS,INDENT,IFAIL)
*
  WRITE (NOUT,*)

*
*      Print 5 by 5 band matrix with 1 sub-diagonal, 2 super-diagonals,
*      user-supplied format and row and column labels
CALL X04CFF(5,5,1,2,A,LDA,'F8.2', 'Example 2:', 'Character',RLABS,
+          'Character',CLABS,NCOLS,INDENT,IFAIL)
*
  STOP
  END

```

### 9.2. Program Data

None.



**9.3. Program Results**

## X04CFF Example Program Results

## Example 1:

	1	2	3	4	5
1	21.0000	12.0000			
2	31.0000	22.0000	13.0000		
3		32.0000	23.0000	14.0000	
4			33.0000	24.0000	15.0000
5				34.0000	25.0000

## Example 2:

	Un	Deux	Trois	Quatre	Cinq
Uno	31.00	22.00	13.00		
Duo	41.00	32.00	23.00	14.00	
Tre		42.00	33.00	24.00	15.00
Quattro			43.00	34.00	25.00
Cinque				44.00	35.00

---



## X04DAF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised terms* and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04DAF is an easy-to-use routine to print a *complex* matrix stored in a two-dimensional array.

### 2. Specification

```

SUBROUTINE X04DAF (MATRIX, DIAG, M, N, A, LDA, TITLE, IFAIL)
  INTEGER          M, N, LDA, IFAIL
  complex        A(LDA, *)
  CHARACTER*1     MATRIX, DIAG
  CHARACTER*(*)   TITLE

```

### 3. Description

X04DAF prints a *complex* matrix. It is an easy-to-use driver for X04DBF. The routine uses default values for the format in which numbers are printed, for labelling the rows and columns, and for output record length.

X04DAF will choose a format code such that numbers will be printed with either an F8.4, F11.4 or a 1PE13.4 format. The F8.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 1.0. The F11.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 9999.9999. Otherwise the 1PE13.4 code is chosen. The chosen code is used to print each complex element of the matrix with the real part above the imaginary part.

The matrix is printed with integer row and column labels, and with a maximum record length of 80.

The matrix is output to the unit defined by X04ABF.

### 4. References

None.

### 5. Parameters

1: MATRIX – CHARACTER\*1.

*Input*

*On entry:* indicates the part of the matrix to be printed, as follows:

MATRIX = 'G' or 'g' (General), the whole of the rectangular matrix.

MATRIX = 'L' or 'l' (Lower), the lower triangle of the matrix, or the lower trapezium if the matrix has more rows than columns.

MATRIX = 'U' or 'u' (Upper), the upper triangle of the matrix, or the upper trapezium if the matrix has more columns than rows.

*Constraint:* MATRIX must be one of 'G', 'g', 'L', 'l', 'U' or 'u'.

2: DIAG – CHARACTER\*1.

*Input*

*On entry:* unless MATRIX = 'G' or 'g', DIAG must specify whether the diagonal elements of the matrix are to be printed, as follows:

DIAG = 'B' or 'b' (Blank), the diagonal elements of the matrix are not referenced and not printed.

DIAG = 'U' or 'u' (Unit diagonal), the diagonal elements of the matrix are not referenced, but are assumed all to be unity, and are printed as such.

DIAG = 'N' or 'n' (Non-unit diagonal), the diagonal elements of the matrix are referenced and printed.

If MATRIX = 'G' or 'g', then DIAG need not be set.

*Constraint:* If MATRIX ≠ 'G' or 'g', then DIAG must be one of 'B', 'b', 'U', 'u', 'N' or 'n'.

- 3: M – INTEGER. *Input*  
 4: N – INTEGER. *Input*  
*On entry:* the number of rows and columns of the matrix, respectively, to be printed.  
 If either of M or N is less than 1, X04DAF will exit immediately after printing TITLE; no row or column labels are printed.
- 5: A(LDA,\*) – *complex* array. *Input*  
 The second dimension of A must be at least max(1,N).  
*On entry:* the matrix to be printed. Only the elements that will be referred to, as specified by parameters MATRIX and DIAG, need be set.
- 6: LDA – INTEGER. *Input*  
*On entry:* the first dimension of the array A as declared in the (sub)program from which X04DAF is called.  
*Constraint:* LDA ≥ M.
- 7: TITLE – CHARACTER\*(\*). *Input*  
*On entry:* a title to be printed above the matrix. If TITLE = ' ', no title (and no blank line) will be printed.  
 If TITLE contains more than 80 characters, the contents of TITLE will be wrapped onto more than one line, with the break after 80 characters.  
 Any trailing blank characters in TITLE are ignored.
- 8: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, MATRIX ≠ 'G', 'g', 'L', 'l', 'U' or 'u'.

IFAIL = 2

On entry, MATRIX = 'L', 'l', 'U' or 'u', but DIAG ≠ 'N', 'n', 'U', 'u', 'B' or 'b'.

IFAIL = 3

On entry, LDA < M.

## 7. Accuracy

Not applicable.

## 8. Further Comments

A call to X04DAF is equivalent to a call to X04DBF with the following argument values:

```

NCOLS = 80
INDENT = 0
LABROW = 'I'
LABCOL = 'I'
FORMAT = ' '
USEFRM = 'A'

```

## 9. Example

This example program calls X04DAF twice, first to print a 4 by 3 rectangular matrix, and then to print a 4 by 4 lower triangular matrix.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      X04DAF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER       (NOUT=6)
      INTEGER          NMAX, LDA
      PARAMETER       (NMAX=4, LDA=NMAX)
*      .. Local Scalars ..
      real            AA
      INTEGER          I, IFAIL, J
*      .. Local Arrays ..
      complex         A(LDA,NMAX)
*      .. External Subroutines ..
      EXTERNAL        X04DAF
*      .. Intrinsic Functions ..
      INTRINSIC       cmplx
*      .. Executable Statements ..
      WRITE (NOUT,*) 'X04DAF Example Program Results'
      WRITE (NOUT,*)
*      Generate an array of data
      DO 40 J = 1, NMAX
         DO 20 I = 1, LDA
            AA = 10*I + J
            A(I,J) = cmplx(AA,-AA)
        20    CONTINUE
      40    CONTINUE
*
      IFAIL = 0
*
*      Print 4 by 3 rectangular matrix
      CALL X04DAF('General',' ',4,3,A,LDA,'Example 1:',IFAIL)
*
      WRITE (NOUT,*)
*
*      Print 4 by 4 lower triangular matrix
      CALL X04DAF('Lower','Non-unit',4,4,A,LDA,'Example 2:',IFAIL)
*
      STOP
      END

```

### 9.2. Program Data

None.

## 9.3. Program Results

## X04DAF Example Program Results

## Example 1:

	1	2	3
1	11.0000 -11.0000	12.0000 -12.0000	13.0000 -13.0000
2	21.0000 -21.0000	22.0000 -22.0000	23.0000 -23.0000
3	31.0000 -31.0000	32.0000 -32.0000	33.0000 -33.0000
4	41.0000 -41.0000	42.0000 -42.0000	43.0000 -43.0000

## Example 2:

	1	2	3	4
1	11.0000 -11.0000			
2	21.0000 -21.0000	22.0000 -22.0000		
3	31.0000 -31.0000	32.0000 -32.0000	33.0000 -33.0000	
4	41.0000 -41.0000	42.0000 -42.0000	43.0000 -43.0000	44.0000 -44.0000

---

## X04DBF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04DBF prints a *complex* matrix stored in a two-dimensional array.

### 2. Specification

```

SUBROUTINE X04DBF (MATRIX, DIAG, M, N, A, LDA, USEFRM, FORMAT,
1              TITLE, LABROW, RLABS, LABCOL, CLABS, NCOLS,
2              INDENT, IFAIL)
    INTEGER      M, N, LDA, NCOLS, INDENT, IFAIL
    complex     A(LDA, *)
    CHARACTER*1  MATRIX, DIAG, USEFRM, LABROW, LABCOL
    CHARACTER*(*) FORMAT, TITLE, RLABS(*), CLABS(*)

```

### 3. Description

X04DBF prints a *complex* matrix, or part of it, using a format specifier supplied by the user. The matrix is output to the unit defined by X04ABF.

### 4. References

None.

### 5. Parameters

1: MATRIX – CHARACTER\*1.

*Input*

*On entry:* indicates the part of the matrix to be printed, as follows:

MATRIX = 'G' or 'g' (General), the whole of the rectangular matrix.

MATRIX = 'L' or 'l' (Lower), the lower triangle of the matrix, or the lower trapezium if the matrix has more rows than columns.

MATRIX = 'U' or 'u' (Upper), the upper triangle of the matrix, or the upper trapezium if the matrix has more rows than columns.

*Constraint:* MATRIX must be one of 'G', 'g', 'L', 'l', 'U' or 'u'.

2: DIAG – CHARACTER\*1.

*Input*

*On entry:* unless MATRIX = 'G' or 'g', DIAG must specify whether the diagonal elements of the matrix are to be printed, as follows:

DIAG = 'B' or 'b' (Blank), the diagonal elements of the matrix are not referenced and not printed.

DIAG = 'U' or 'u' (Unit diagonal), the diagonal elements of the matrix are not referenced, but are assumed all to be unity, and are printed as such.

DIAG = 'N' or 'n' (Non-unit diagonal), the diagonal elements of the matrix are referenced and printed.

If MATRIX = 'G' or 'g', then DIAG need not be set.

*Constraint:* if MATRIX ≠ 'G' or 'g', then DIAG must be one of 'B', 'b', 'U', 'u', 'N' or 'n'.

3: M – INTEGER.

*Input*

4: N – INTEGER.

*Input*

*On entry:* the number of rows and columns of the matrix, respectively, to be printed.

If either of M or N is less than 1, X04DBF will exit immediately after printing TITLE; no row or column labels are printed.

- 5: A(LDA,\*) – *complex* array. *Input*  
 The second dimension of A must be at least  $\max(1, N)$ .  
*On entry:* the matrix to be printed. Only the elements that will be referred to, as specified by parameters MATRIX and DIAG, need be set.
- 6: LDA – INTEGER. *Input*  
*On entry:* the first dimension of the array A as declared in the (sub)program from which X04DBF is called.  
*Constraint:*  $LDA \geq M$ .
- 7: USEFRM – CHARACTER\*1. *Input*  
*On entry:* indicates how the value of FORMAT is to be used to print matrix elements.  
 USEFRM = 'A' or 'a' (Above), the format code in FORMAT is assumed to contain a single real edit-descriptor which is to be used to print the real and imaginary parts of each *complex* number one above the other. Each row of the matrix is separated by a blank line, and any row labels are attached only to the real parts. This option means that about twice as many columns can be fitted into NCOLS characters than if any other USEFRM option is used. A typical value of FORMAT for this USEFRM option might be `FORMAT = 'E13.4', '*', ' '`.  
 USEFRM = 'B' or 'b' (Bracketed), the format code in FORMAT is assumed to contain a single edit-descriptor such as 'E13.4', '\*', or ' ', which is used to print the real and imaginary parts of each *complex* number separated by a comma, and surrounded by brackets. Thus a matrix element printed with this USEFRM option might look like this:  
 ( 12.345, -11.323)  
 USEFRM = 'D' or 'd' (Direct), the format code in FORMAT is used unaltered to print a *complex* number. This USEFRM option allows the user flexibility to specify exactly how the number is printed. With this option for USEFRM and a suitable value for FORMAT it is possible, for example, to print a *complex* number in the form  $(0.123 + 3.214i)$  or  $(0.123E-02, 0.234E-01)$ . See Section 9 for an example illustrating this option.  
*Constraint:* USEFRM must be one of 'A', 'a', 'B', 'b', 'D' or 'd'.
- 8: FORMAT – CHARACTER\*(\*). *Input*  
*On entry:* a valid Fortran format code. This may be any format code allowed on the system, whether it is standard Fortran or not. FORMAT is used in conjunction with parameter USEFRM, described above, to print elements of the matrix A. It may or may not be enclosed in brackets. Examples of valid values for FORMAT are 'F11.4', '1P,2E13.5'.  
 In addition, there are two special codes which force X04DBF to choose its own format code:  
 FORMAT = ' ' means that X04DBF will choose a format code such that numbers will be printed with either an F8.4, an F11.4 or a 1PE13.4 format. The F8.4 code is chosen if the sizes of the real and imaginary parts of all the matrix elements to be printed lie between 0.001 and 1.0. The F11.4 code is chosen if the sizes of all the numbers to be printed lie between 0.001 and 9999.9999. Otherwise the 1PE13.4 code is chosen.  
 FORMAT = '\*' means that X04DBF will choose a format code such that numbers will be printed to as many significant digits as are necessary to distinguish between neighbouring machine numbers. Thus any two numbers that are stored with different internal representations should look different on output. Whether they do in fact look different will depend on the run-time library of the Fortran compiler in use.  
 More complicated values of FORMAT, to print a *complex* number in a desired form, may be used. See the description of parameter USEFRM above for more details.  
*Constraint:* the character length of FORMAT must be  $\leq 80$ .



- 9: **TITLE – CHARACTER\*(\*)**. *Input*  
*On entry:* a title to be printed above the matrix. If TITLE = ' ', no title (and no blank line) will be printed.  
 If TITLE contains more than NCOLS characters, the contents of TITLE will be wrapped onto more than one line, with the break after NCOLS characters.  
 Any trailing blank characters in TITLE are ignored.
- 10: **LABROW – CHARACTER\*1**. *Input*  
*On entry:* the type of labelling to be applied to the rows of the matrix, as follows:  
 If LABROW = 'N' or 'n', X04DBF prints no row labels.  
 If LABROW = 'I' or 'i', X04DBF prints integer row labels.  
 If LABROW = 'C' or 'c', X04DBF prints character labels, which must be supplied in array RLABS.  
*Constraint:* LABROW must be one of 'N', 'n', 'I', 'i', 'C' or 'c'.
- 11: **RLABS(\*) – CHARACTER\*(\*) array**. *Input*  
*On entry:* if LABROW = 'C' or 'c', RLABS must be dimensioned at least of length M and must contain labels for the rows of the matrix, otherwise RLABS may be dimensioned of length 1.  
 Labels are right justified when output, in a field which is as wide as necessary to hold the longest row label. Note that this field width is subtracted from the number of usable columns, NCOLS.
- 12: **LABCOL – CHARACTER\*1**. *Input*  
*On entry:* the type of labelling to be applied to the columns of the matrix, as follows:  
 If LABCOL = 'N' or 'n', X04DBF prints no column labels.  
 If LABCOL = 'I' or 'i', X04DBF prints integer column labels.  
 If LABCOL = 'C' or 'c', X04DBF prints character labels, which must be supplied in array CLABS.  
*Constraint:* LABCOL must be one of 'N', 'n', 'I', 'i', 'C' or 'c'.
- 13: **CLABS(\*) – CHARACTER\*(\*) array**. *Input*  
*On entry:* if LABCOL = 'C' or 'c', CLABS must be dimensioned at least of length N and must contain labels for the columns of the matrix, otherwise CLABS may be dimensioned of length 1.  
 Labels are right-justified when output. Any label that is too long for the column width, which is determined by FORMAT, is truncated.
- 14: **NCOLS – INTEGER**. *Input*  
*On entry:* the maximum output record length. If the number of columns of the matrix is too large to be accommodated in NCOLS characters, the matrix will be printed in parts, containing the largest possible number of matrix columns, and each part separated by a blank line.  
 NCOLS must be large enough to hold at least one column of the matrix using the format specifier in FORMAT. If a value less than 0 or greater than 132 is supplied for NCOLS, then the value 80 is used instead.
- 15: **INDENT – INTEGER**. *Input*  
*On entry:* the number of columns by which the matrix (and any title and labels) should be indented. The effective value of NCOLS is reduced by INDENT columns. If a value less than 0 or greater than NCOLS is supplied for INDENT, the value 0 is used instead.

## 16: IFAIL – INTEGER.

*Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, MATRIX  $\neq$  'G', 'g', 'L', 'l', 'U' or 'u'.

IFAIL = 2

On entry, MATRIX = 'L', 'l', 'U' or 'u', but DIAG  $\neq$  'N', 'n', 'U', 'u', 'B', or 'b'.

IFAIL = 3

On entry, LDA < M.

IFAIL = 4

On entry, USEFRM  $\neq$  'A', 'a', 'B', 'b', 'D' or 'd'.

IFAIL = 5

On entry, variable FORMAT is more than 80 characters long.

IFAIL = 6

The code supplied in FORMAT cannot be used to output a number. FORMAT probably has too wide a field width or contains an illegal edit descriptor.

IFAIL = 7

On entry, either LABROW or LABCOL  $\neq$  'N', 'n', 'I', 'i', 'C' or 'c'.

IFAIL = 8

The quantity NCOLS – INDENT – LABWID (where LABWID is the width needed for the row labels) is not large enough to hold at least one column of the matrix.

## 7. Accuracy

Not applicable.

## 8. Further Comments

X04DBF may be used to print a vector, either as a row or as a column. The following code fragment illustrates possible calls.

```

      complex A(4)
      CHARACTER*1 RLABS(1), CLABS(1)
C   Print vector A as a column vector.
      LDA = 4
      IFAIL = 0
      CALL X04DBF('G', 'X', 1, 4, A, LDA, 'B', ' ', ' ', ' ', 'I', RLABS,
*              'N', CLABS, 80, 0, IFAIL)
C   Print vector A as a row vector.
      LDA = 1
      IFAIL = 0
      CALL X04DBF('G', 'X', 4, 1, A, LDA, 'B', ' ', ' ', ' ', 'N', RLABS,
*              'I', CLABS, 80, 0, IFAIL)

```

## 9. Example

This example program calls X04DBF twice, first to print a 3 by 4 rectangular matrix, and then to print a 4 by 4 upper triangular matrix; various options for labelling and formatting are illustrated.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      X04DBF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NMAX, LDA
      PARAMETER        (NMAX=4, LDA=NMAX)
*      .. Local Scalars ..
      real            AA
      INTEGER          I, IFAIL, INDENT, J, NCOLS
*      .. Local Arrays ..
      complex         A(LDA,NMAX)
      CHARACTER*7      CLABS(NMAX), RLABS(NMAX)
*      .. External Subroutines ..
      EXTERNAL         X04DBF
*      .. Intrinsic Functions ..
      INTRINSIC        cmplx
*      .. Data statements ..
      DATA             CLABS/'Un', 'Deux', 'Trois', 'Quatre'/
      DATA             RLABS/'Uno', 'Duo', 'Tre', 'Quattro'/
*      .. Executable Statements ..
      WRITE (NOUT,*) 'X04DBF Example Program Results'
      WRITE (NOUT,*)
*      Generate an array of data
      DO 40 J = 1, NMAX
        DO 20 I = 1, LDA
          AA = 10*I + J
          A(I,J) = cmplx(AA,-AA)
20      CONTINUE
40      CONTINUE
      NCOLS = 80
      INDENT = 0
      IFAIL = 0
*
*      Print 3 by 4 rectangular matrix with default format and integer
*      row and column labels, and bracketed complex elements
      CALL X04DBF('General', ' ', 3, 4, A, LDA, 'Bracketed', ' ', 'Example 1:',
+              'Integer', RLABS, 'Integer', CLABS, NCOLS, INDENT, IFAIL)
*
      WRITE (NOUT,*)
*
*      Print 4 by 4 upper triangular matrix with user-supplied format
*      and row and column labels, and complex elements with real part
*      above imaginary part
      CALL X04DBF('Upper', 'Non-unit', 4, 4, A, LDA, 'Above', 'F8.2',
+              'Example 2:', 'Character', RLABS, 'Character', CLABS,
+              NCOLS, INDENT, IFAIL)
*
      STOP
      END
```

### 9.2. Program Data

None.

9.3. Program Results

X04DBF Example Program Results

Example 1:

			1			2
1	(	11.0000,	-11.0000)	(	12.0000,	-12.0000)
2	(	21.0000,	-21.0000)	(	22.0000,	-22.0000)
3	(	31.0000,	-31.0000)	(	32.0000,	-32.0000)
			3			4
1	(	13.0000,	-13.0000)	(	14.0000,	-14.0000)
2	(	23.0000,	-23.0000)	(	24.0000,	-24.0000)
3	(	33.0000,	-33.0000)	(	34.0000,	-34.0000)

Example 2:

	Un	Deux	Trois	Quatre
Uno	11.00	12.00	13.00	14.00
	-11.00	-12.00	-13.00	-14.00
Duo		22.00	23.00	24.00
		-22.00	-23.00	-24.00
Tre			33.00	34.00
			-33.00	-34.00
Quattro				44.00
				-44.00

---

## X04DCF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04DCF is an easy-to-use routine to print a *complex* triangular matrix stored in a packed one-dimensional array.

### 2. Specification

```
SUBROUTINE X04DCF (UPLO, DIAG, N, A, TITLE, IFAIL)
  INTEGER          N, IFAIL
  complex        A(*)
  CHARACTER*1     UPLO, DIAG
  CHARACTER*(*)   TITLE
```

### 3. Description

X04DCF prints a *complex* triangular matrix stored in packed form. It is an easy-to-use driver for X04DDF. The routine uses default values for the format in which numbers are printed, for labelling the rows and columns, and for output record length. The matrix must be packed by column.

X04DCF will choose a format code such that numbers will be printed with either an F8.4, F11.4 or a 1PE13.4 format. The F8.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 1.0. The F11.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 9999.9999. Otherwise the 1PE13.4 code is chosen. The chosen code is used to print each complex element of the matrix with the real part above the imaginary part.

The matrix is printed with integer row and column labels, and with a maximum record length of 80.

The matrix is output to the unit defined by X04ABF.

### 4. References

None.

### 5. Parameters

1: UPLO – CHARACTER\*1.

*Input*

*On entry:* indicates the type of the matrix to be printed, as follows:

UPLO = 'L' or 'l' (Lower), the matrix is lower triangular. In this case, the packed array A holds the matrix elements in the following order: (1,1), (2,1),..., (N,1), (2,2), (3,2),..., (N,2), etc.

UPLO = 'U' or 'u' (Upper), the matrix is upper triangular. In this case, the packed array A holds the matrix elements in the following order: (1,1), (1,2), (2,2), (1,3), (2,3), (3,3), (1,4), etc.

*Constraint:* UPLO must be one of 'L', 'l', 'U' or 'u'.

2: DIAG – CHARACTER\*1.

*Input*

*On entry:* indicates whether the diagonal elements of the matrix are to be printed, as follows:

DIAG = 'B' or 'b' (Blank), the diagonal elements of the matrix are not referenced and not printed.

DIAG = 'U' or 'u' (Unit diagonal), the diagonal elements of the matrix are not referenced, but are assumed all to be unity, and are printed as such.

DIAG = 'N' or 'n' (Non-unit diagonal), the diagonal elements of the matrix are referenced and printed.

*Constraint:* DIAG must be one of 'B', 'b', 'U', 'u', 'N' or 'n'.

3: N – INTEGER. *Input*

*On entry:* the order of the matrix to be printed.

If N is less than 1, X04DCF will exit immediately after printing TITLE; no row or column labels are printed.

4: A(\*) – *complex* array. *Input*

The dimension of A must be at least  $\max(1, N*(N+1)/2)$ .

*On entry:* the matrix to be printed. Note that A must have space for the diagonal elements of the matrix, even if these are not stored.

5: TITLE – CHARACTER\*(\*). *Input*

*On entry:* a title to be printed above the matrix. If TITLE = ' ', no title (and no blank line) will be printed.

If TITLE contains more than 80 characters, the contents of TITLE will be wrapped onto more than one line, with the break after 80 characters.

Any trailing blank characters in TITLE are ignored.

6: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, UPLO ≠ 'L', 'l', 'U' or 'u'.

IFAIL = 2

On entry, DIAG ≠ 'N', 'n', 'U', 'u', 'B' or 'b'.

## 7. Accuracy

Not applicable.

## 8. Further Comments

A call to X04DCF is equivalent to a call to X04DDF with the following argument values:

```

NCOLS = 80
INDENT = 0
LABROW = ' I '
LABCOL = ' I '
FORMAT = ' '
USEFRM = ' A '

```

## 9. Example

This example program calls X04DCF twice, first to print a 3 by 3 lower triangular matrix, and then to print a 4 by 4 upper triangular matrix.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      X04DCF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NMAX, LA
      PARAMETER        (NMAX=4, LA=(NMAX*(NMAX+1))/2)
*      .. Local Scalars ..
      real            AA
      INTEGER          I, IFAIL
*      .. Local Arrays ..
      complex        A(LA)
*      .. External Subroutines ..
      EXTERNAL         X04DCF
*      .. Intrinsic Functions ..
      INTRINSIC        cmplx
*      .. Executable Statements ..
      WRITE (NOUT,*) 'X04DCF Example Program Results'
      WRITE (NOUT,*)
*
*      Generate an array of data
      DO 20 I = 1, LA
          AA = I
          A(I) = cmplx(AA, -AA)
20 CONTINUE
*
      IFAIL = 0
*
*      Print order 3 lower triangular matrix
      CALL X04DCF('Lower', 'Unit', 3, A, 'Example 1:', IFAIL)
*
      WRITE (NOUT,*)
*
*      Print order 4 upper triangular matrix
      CALL X04DCF('Upper', 'Non-unit', 4, A, 'Example 2:', IFAIL)
*
      STOP
      END

```

### 9.2. Program Data

None.

### 9.3. Program Results

X04DCF Example Program Results

Example 1:

	1	2	3
1	1.0000 0.0000		
2	2.0000 -2.0000	1.0000 0.0000	
3	3.0000 -3.0000	5.0000 -5.0000	1.0000 0.0000

Example 2:

	1	2	3	4
1	1.0000 -1.0000	2.0000 -2.0000	4.0000 -4.0000	7.0000 -7.0000
2		3.0000 -3.0000	5.0000 -5.0000	8.0000 -8.0000
3			6.0000 -6.0000	9.0000 -9.0000
4				10.0000 -10.0000

---



## X04DDF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised terms* and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04DDF prints a *complex* triangular matrix stored in a packed one-dimensional array.

### 2. Specification

```

SUBROUTINE X04DDF (UPLO, DIAG, N, A, USEFRM, FORMAT, TITLE, LABROW,
1                RLABS, LABCOL, CLABS, NCOLS, INDENT, IFAIL)
    INTEGER          N, NCOLS, INDENT, IFAIL
    complex        A(*)
    CHARACTER*1     UPLO, DIAG, USEFRM, LABROW, LABCOL
    CHARACTER*(*)   FORMAT, TITLE, RLABS(*), CLABS(*)

```

### 3. Description

X04DDF prints a *complex* triangular matrix stored in packed form, using a format specifier supplied by the user. The matrix must be packed by column. The matrix is output to the unit defined by X04ABF.

### 4. References

None.

### 5. Parameters

- 1: UPLO – CHARACTER\*1. *Input*
- On entry:* indicates the type of the matrix to be printed, as follows:
- UPLO = 'L' or 'l' (Lower), the matrix is lower triangular. In this case, the packed array A holds the matrix elements in the following order: (1,1), (2,1),..., (N,1), (2,2), (3,2),..., (N,2), etc.
- UPLO = 'U' or 'u' (Upper), the matrix is upper triangular. In this case, the packed array A holds the matrix elements in the following order: (1,1), (1,2), (2,2), (1,3), (2,3), (3,3), (1,4), etc.
- Constraint:* UPLO must be one of 'L', 'l', 'U' or 'u'.
- 2: DIAG – CHARACTER\*1. *Input*
- On entry:* indicates whether the diagonal elements of the matrix are to be printed, as follows:
- DIAG = 'B' or 'b' (Blank), the diagonal elements of the matrix are not referenced and not printed.
- DIAG = 'U' or 'u' (Unit diagonal), the diagonal elements of the matrix are not referenced, but are assumed all to be unity, and are printed as such.
- DIAG = 'N' or 'n' (Non-unit diagonal), the diagonal elements of the matrix are referenced and printed.
- Constraint:* DIAG must be one of 'B', 'b', 'U', 'u', 'N' or 'n'.
- 3: N – INTEGER. *Input*
- On entry:* the number of rows and columns of the matrix to be printed.
- If N is less than 1, X04DDF will exit immediately after printing TITLE; no row or column labels are printed.

- 4: **A(\*) – *complex* array.** *Input*  
 The dimension of A must be at least  $\max(1, N*(N+1)/2)$ .  
*On entry:* the matrix to be printed. Note that A must have space for the diagonal elements of the matrix, even if these are not stored.
- 5: **USEFRM – CHARACTER\*1.** *Input*  
*On entry:* indicates how the value of FORMAT is to be used to print matrix elements.  
 USEFRM = 'A' or 'a' (Above), the format code in FORMAT is assumed to contain a single real edit-descriptor which is to be used to print the real and imaginary parts of each *complex* number one above the other. Each row of the matrix is separated by a blank line, and any row labels are attached only to the real parts. This option means that about twice as many columns can be fitted into NCOLS characters than if any other USEFRM option is used. A typical value of FORMAT for this USEFRM option might be FORMAT = 'E13.4', '\*' or ' '.  
 USEFRM = 'B' or 'b' (Bracketed), the format code in FORMAT is assumed to contain a single edit-descriptor such as 'E13.4', '\*' or ' ', which is used to print the real and imaginary parts of each *complex* number separated by a comma, and surrounded by brackets. Thus a matrix element printed with this USEFRM option might look like this:  
 ( 12.345, -11.323 )  
 USEFRM = 'D' or 'd' (Direct), the format code in FORMAT is used unaltered to print a *complex* number. This USEFRM option allows the user flexibility to specify exactly how the number is printed. With this option for USEFRM and a suitable value for FORMAT it is possible, for example, to print a *complex* number in the form (0.123 + 3.214i) or (0.123E-02,0.234E-01). See Section 9 for an example illustrating this option.  
*Constraint:* USEFRM must be one of 'A', 'a', 'B', 'b', 'D' or 'd'.
- 6: **FORMAT – CHARACTER\*(\*).** *Input*  
*On entry:* a valid Fortran format code. This may be any format code allowed on the system, whether it is standard Fortran or not. FORMAT is used in conjunction with parameter USEFRM, described above, to print elements of the matrix A. It may or may not be enclosed in brackets. Examples of valid values for FORMAT are '(F11.4)', '1P,2E13.5'.  
 In addition, there are two special codes which force X04DDF to choose its own format code:  
 FORMAT = ' ' means that X04DDF will choose a format code such that numbers will be printed with either an F8.4, an F11.4 or a 1PE13.4 format. The F8.4 code is chosen if the sizes of the real and imaginary parts of all the matrix elements to be printed lie between 0.001 and 1.0. The F11.4 code is chosen if the sizes of all the numbers to be printed lie between 0.001 and 9999.9999. Otherwise the 1PE13.4 code is chosen.  
 FORMAT = '\*' means that X04DDF will choose a format code such that numbers will be printed to as many significant digits as are necessary to distinguish between neighbouring machine numbers. Thus any two numbers that are stored with different internal representations should look different on output. Whether they do in fact look different will depend on the run-time library of the Fortran compiler in use.  
 More complicated values of FORMAT, to print a *complex* number in a desired form, may be used. See the description of parameter USEFRM above for more details.  
*Constraint:* the character length of FORMAT must be  $\leq 80$ .
- 7: **TITLE – CHARACTER\*(\*).** *Input*  
*On entry:* a title to be printed above the matrix. If TITLE = ' ', no title (and no blank line) will be printed.  
 If TITLE contains more than NCOLS characters, the contents of TITLE will be wrapped onto more than one line, with the break after NCOLS characters.  
 Any trailing blank characters in TITLE are ignored.

- 8: LABROW – CHARACTER\*1. *Input*  
*On entry:* indicates the type of labelling to be applied to the rows of the matrix, as follows:  
 If LABROW = 'N' or 'n', X04DDF prints no row labels.  
 If LABROW = 'I' or 'i', X04DDF prints integer row labels.  
 If LABROW = 'C' or 'c', X04DDF prints character labels, which must be supplied in array RLABS.  
*Constraint:* LABROW must be one of 'N', 'n', 'I', 'i', 'C' or 'c'.
- 9: RLABS(\*) – CHARACTER\*(\*) array. *Input*  
*On entry:* if LABROW = 'C' or 'c', RLABS must be dimensioned at least of length N and must contain labels for the rows of the matrix, otherwise RLABS may be dimensioned of length 1.  
 Labels are right justified when output, in a field which is as wide as necessary to hold the longest row label. Note that this field width is subtracted from the number of usable columns, NCOLS.
- 10: LABCOL – CHARACTER\*1. *Input*  
*On entry:* indicates the type of labelling to be applied to the columns of the matrix, as follows:  
 If LABCOL = 'N' or 'n', X04DDF prints no column labels.  
 If LABCOL = 'I' or 'i', X04DDF prints integer column labels.  
 If LABCOL = 'C' or 'c', X04DDF prints character labels, which must be supplied in array CLABS.  
*Constraint:* LABCOL must be one of 'N', 'n', 'I', 'i', 'C' or 'c'.
- 11: CLABS(\*) – CHARACTER\*(\*) array. *Input*  
*On entry:* if LABCOL = 'C' or 'c', CLABS must be dimensioned at least of length N and must contain labels for the columns of the matrix, otherwise CLABS may be dimensioned of length 1.  
 Labels are right-justified when output. Any label that is too long for the column width, which is determined by FORMAT, is truncated.
- 12: NCOLS – INTEGER. *Input*  
*On entry:* the maximum output record length. If the number of columns of the matrix is too large to be accommodated in NCOLS characters, the matrix will be printed in parts, containing the largest possible number of matrix columns, and each part separated by a blank line.  
 NCOLS must be large enough to hold at least one column of the matrix using the format specifier in FORMAT. If a value less than 0 or greater than 132 is supplied for NCOLS, then the value 80 is used instead.
- 13: INDENT – INTEGER. *Input*  
*On entry:* the number of columns by which the matrix (and any title and labels) should be indented. The effective value of NCOLS is reduced by INDENT columns. If a value less than 0 or greater than NCOLS is supplied for INDENT, the value 0 is used instead.
- 14: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, UPLO ≠ 'L', 'l', 'U' or 'u'.

IFAIL = 2

On entry, DIAG ≠ 'N', 'n', 'U', 'u', 'B', or 'b'.

IFAIL = 3

On entry, USEFRM ≠ 'A', 'a', 'B', 'b', 'D', or 'd'.

IFAIL = 4

On entry, variable FORMAT is more than 80 characters long.

IFAIL = 5

The code supplied in FORMAT cannot be used to output a number. FORMAT probably has too wide a field width or contains an illegal edit descriptor.

IFAIL = 6

On entry, either LABROW or LABCOL ≠ 'N', 'n', 'I', 'i', 'C' or 'c'.

IFAIL = 7

The quantity NCOLS – INDENT – LABWID (where LABWID is the width needed for the row labels) is not large enough to hold at least one column of the matrix.

## 7. Accuracy

Not applicable.

## 8. Further Comments

None.

## 9. Example

This example program calls X04DDF twice, first to print a 4 by 4 lower triangular matrix, and then to print a 4 by 4 triangular matrix, various options for labelling and formatting are illustrated.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      X04DDF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER       (NOUT=6)
      INTEGER          NMAX, LA
      PARAMETER       (NMAX=4, LA=(NMAX*(NMAX+1))/2)
*      .. Local Scalars ..
      real            AA
      INTEGER          I, IFAIL, INDENT, NCOLS
      CHARACTER*19    FORMAT
```

```

*      .. Local Arrays ..
*      complex          A(LA)
CHARACTER*7          CLABS(NMAX), RLABS(NMAX)
*      .. External Subroutines ..
EXTERNAL            X04DDF
*      .. Intrinsic Functions ..
INTRINSIC          cmplx
*      .. Data statements ..
DATA              CLABS/'Un', 'Deux', 'Trois', 'Quatre'/
DATA              RLABS/'Uno', 'Duo', 'Tre', 'Quattro'/
*      .. Executable Statements ..
WRITE (NOUT,*) 'X04DDF Example Program Results'
WRITE (NOUT,*)

*
*      Generate an array of data
DO 20 I = 1, LA
    AA = I
    A(I) = cmplx(AA, -AA)
20 CONTINUE
*
    NCOLS = 80
    INDENT = 0
    IFAIL = 0
    FORMAT = ' '
*
*      Print order 4 lower triangular matrix with default format and
*      integer row and column labels, and bracketed complex elements
CALL X04DDF('Lower', 'Non-unit', 4, A, 'Bracketed', FORMAT,
+          'Example 1:', 'Integer', RLABS, 'Integer', CLABS, NCOLS,
+          INDENT, IFAIL)
*
WRITE (NOUT,*)
FORMAT = 'SS,F7.1,SP,F6.1,','i''
*
*      Print order 4 upper triangular matrix with user-supplied format
*      and row and column labels, using the supplied format directly
CALL X04DDF('Upper', 'Unit', 4, A, 'Direct', FORMAT, 'Example 2:',
+          'Character', RLABS, 'Character', CLABS, NCOLS, INDENT,
+          IFAIL)
*
    STOP
    END

```

## 9.2. Program Data

None.

## 9.3. Program Results

X04DDF Example Program Results

Example 1:

			1		2
1	(	1.0000,	-1.0000)		
2	(	2.0000,	-2.0000)	(	5.0000, -5.0000)
3	(	3.0000,	-3.0000)	(	6.0000, -6.0000)
4	(	4.0000,	-4.0000)	(	7.0000, -7.0000)
			3		4
1					
2					
3	(	8.0000,	-8.0000)		
4	(	9.0000,	-9.0000)	(	10.0000, -10.0000)

Example 2:

	Un		Deux		Trois		Quatre	
Uno	1.0	+0.0i	2.0	-2.0i	4.0	-4.0i	7.0	-7.0i
Duo			1.0	+0.0i	5.0	-5.0i	8.0	-8.0i
Tre					1.0	+0.0i	9.0	-9.0i
Quattro							1.0	+0.0i

---

## X04DEF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04DEF is an easy-to-use routine to print a *complex* band matrix stored in a packed two-dimensional array.

### 2. Specification

```

SUBROUTINE X04DEF (M, N, KL, KU, A, LDA, TITLE, IFAIL)
  INTEGER          M, N, KL, KU, LDA, IFAIL
  complex        A(LDA,*)
  CHARACTER*(*)   TITLE

```

### 3. Description

X04DEF prints a *complex* band matrix stored in a packed two-dimensional array. It is an easy-to-use driver for X04DEF. The routine uses default values for the format in which numbers are printed, for labelling the rows and columns, and for output record length.

X04DEF will choose a format code such that numbers will be printed with either an F8.4, F11.4 or a 1PE13.4 format. The F8.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 1.0. The F11.4 code is chosen if the sizes of all the matrix elements to be printed lie between 0.001 and 9999.9999. Otherwise the 1PE13.4 code is chosen. The chosen code is used to print each complex element of the matrix with the real part above the imaginary part.

The matrix is printed with integer row and column labels, and with a maximum record length of 80.

The matrix is output to the unit defined by X04ABF.

### 4. References

None.

### 5. Parameters

- 1: M – INTEGER. *Input*  
 2: N – INTEGER. *Input*

*On entry:* the number of rows and columns of the band matrix, respectively, to be printed.

If either of M or N is less than 1, X04DEF will exit immediately after printing TITLE; no row or column labels are printed.

- 3: KL – INTEGER. *Input*

*On entry:* the number of sub-diagonals of the band matrix A.

*Constraint:*  $KL \geq 0$ .

- 4: KU – INTEGER. *Input*

*On entry:* the number of super-diagonals of the band matrix A.

*Constraint:*  $KU \geq 0$ .

- 5: A(LDA,\*) – *complex* array. *Input*

The second dimension of A must be at least  $\max(1, \min(M+KU, N))$ .

*On entry:* the band matrix to be printed. The leading  $(KL+KU+1)$  by  $\min(M+KU, N)$  part of array A must contain the matrix, packed column by column, with the leading diagonal of the matrix in row  $(KU+1)$  of the array, the first super-diagonal starting at position 2 in row KU, the first sub-diagonal starting at position 1 in row  $(KU+2)$ , and so on. Elements in the array A that do not correspond to elements in the band matrix (such as the top left KU by KU triangle) are not referenced, and need not be set.

- 6: LDA – INTEGER. *Input*

*On entry:* the first dimension of the array A as declared in the (sub)program from which X04DEF is called.

*Constraint:*  $LDA \geq KL + KU + 1$ .

- 7: TITLE – CHARACTER\*(\*). *Input*

*On entry:* a title to be printed above the matrix. If TITLE = ' ', no title (and no blank line) will be printed.

If TITLE contains more than 80 characters, the contents of TITLE will be wrapped onto more than one line, with the break after 80 characters.

Any trailing blank characters in TITLE are ignored.

- 8: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry,  $KL < 0$ .

IFAIL = 2

On entry,  $KU < 0$ .

IFAIL = 3

On entry,  $LDA < KL + KU + 1$ .

## 7. Accuracy

Not applicable.

## 8. Further Comments

A call to X04DEF is equivalent to a call to X04DFE with the following argument values:

```

NCOLS = 80
INDENT = 0
LABROW = ' I '
LABCOL = ' I '
FORMAT = ' '
USEFRM = ' A '

```



## 9. Example

This example program calls X04DEF to print a 5 by 5 band matrix with one sub-diagonal and one super-diagonal.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      X04DEF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NOUT
PARAMETER        (NOUT=6)
INTEGER          NMAX, LDA
PARAMETER        (NMAX=5, LDA=NMAX)
*      .. Local Scalars ..
real           AA
INTEGER          I, IFAIL, J
*      .. Local Arrays ..
complex       A(LDA, NMAX)
*      .. External Subroutines ..
EXTERNAL         X04DEF
*      .. Intrinsic Functions ..
INTRINSIC        cmplx
*      .. Executable Statements ..
WRITE (NOUT,*) 'X04DEF Example Program Results'
WRITE (NOUT,*)

*
*      Generate an array of data
DO 40 J = 1, NMAX
  DO 20 I = 1, LDA
    AA = 10*I + J
    A(I, J) = cmplx(AA, -AA)
20  CONTINUE
40  CONTINUE

*
*      IFAIL = 0

*
*      Print 5 by 5 band matrix with 1 sub-diagonal and 1 super-diagonal
CALL X04DEF(5, 5, 1, 1, A, LDA, 'Band Matrix:', IFAIL)

*
*      STOP
*      END
```

### 9.2. Program Data

None.

9.3. Program Results

X04DEF Example Program Results

Band Matrix:

	1	2	3	4	5
1	21.0000 -21.0000	12.0000 -12.0000			
2	31.0000 -31.0000	22.0000 -22.0000	13.0000 -13.0000		
3		32.0000 -32.0000	23.0000 -23.0000	14.0000 -14.0000	
4			33.0000 -33.0000	24.0000 -24.0000	15.0000 -15.0000
5				34.0000 -34.0000	25.0000 -25.0000

---

## X04DFF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04DFF prints a *complex* band matrix stored in a packed two-dimensional array.

### 2. Specification

```

SUBROUTINE X04DFF (M, N, KL, KU, A, LDA, USEFRM, FORMAT, TITLE,
1                 LABROW, RLABS, LABCOL, CLABS, NCOLS, INDENT,
2                 IFAIL)
INTEGER          M, N, KL, KU, LDA, NCOLS, INDENT, IFAIL
complex        A(LDA, *)
CHARACTER*1      USEFRM, LABROW, LABCOL
CHARACTER*(*)    FORMAT, TITLE, RLABS(*), CLABS(*)

```

### 3. Description

X04DFF prints a *complex* band matrix stored in a packed two-dimensional array, using a format specifier supplied by the user. The matrix is output to the unit defined by X04ABF.

### 4. References

None.

### 5. Parameters

1: M – INTEGER. *Input*  
 2: N – INTEGER. *Input*

*On entry:* the number of rows and columns of the band matrix, respectively, to be printed.  
 If either of M or N is less than 1, X04DFF will exit immediately after printing TITLE; no row or column labels are printed.

3: KL – INTEGER. *Input*

*On entry:* the number of sub-diagonals of the band matrix A.  
*Constraint:*  $KL \geq 0$ .

4: KU – INTEGER. *Input*

*On entry:* the number of super-diagonals of the band matrix A.  
*Constraint:*  $KU \geq 0$ .

5: A(LDA,\*) – *complex* array. *Input*

The second dimension of A must be at least  $\max(1, \min(M+KU, N))$ .  
*On entry:* the band matrix to be printed. The leading  $(KL+KU+1)$  by  $\min(M+KU, N)$  part of array A must contain the matrix, packed column by column, with the leading diagonal of the matrix in row  $(KU+1)$  of the array, the first super-diagonal starting at position 2 in row KU, the first sub-diagonal starting at position 1 in row  $(KU+2)$ , and so on. Elements in the array A that do not correspond to elements in the band matrix (such as the top left KU by KU triangle) are not referenced, and need not be set.

6: LDA – INTEGER. *Input*

*On entry:* the first dimension of the array A as declared in the (sub)program from which X04DFF is called.  
*Constraint:*  $LDA \geq KL + KU + 1$ .

## 7: USEFRM – CHARACTER\*1.

*Input*

*On entry:* indicates how the value of FORMAT is to be used to print matrix elements.

USEFRM = 'A' or 'a' (Above), the format code in FORMAT is assumed to contain a single real edit-descriptor which is to be used to print the real and imaginary parts of each *complex* number one above the other. Each row of the matrix is separated by a blank line, and any row labels are attached only to the real parts. This option means that about twice as many columns can be fitted into NCOLS characters than if any other USEFRM option is used. A typical value of FORMAT for this USEFRM option might be FORMAT = 'E13.4', '\*' or ' '.

USEFRM = 'B' or 'b' (Bracketed), the format code in FORMAT is assumed to contain a single edit-descriptor such as 'E13.4', '\*' or ' ', which is used to print the real and imaginary parts of each *complex* number separated by a comma, and surrounded by brackets. Thus a matrix element printed with this USEFRM option might look like this:  
( 12.345, -11.323)

USEFRM = 'D' or 'd' (Direct), the format code in FORMAT is used unaltered to print a *complex* number. This USEFRM option allows the user flexibility to specify exactly how the number is printed. With this option for USEFRM and a suitable value for FORMAT it is possible, for example, to print a *complex* number in the form (0.123 + 3.214i) or (0.123E-02,0.234E-01). See Section 9 for an example illustrating this option.

*Constraint:* USEFRM must be one of 'A', 'a', 'B', 'b', 'D' or 'd'.

## 8: FORMAT – CHARACTER\*(\*).

*Input*

*On entry:* a valid Fortran format code. This may be any format code allowed on the system, whether it is standard Fortran or not. FORMAT is used in conjunction with parameter USEFRM, described above, to print elements of the matrix A. It may or may not be enclosed in brackets. Examples of valid values for FORMAT are '(F11.4)', '1P,2E13.5'.

In addition, there are two special codes which force X04DFF to choose its own format code:

FORMAT = ' ' means that X04DFF will choose a format code such that numbers will be printed with either an F8.4, an F11.4 or a 1PE13.4 format. The F8.4 code is chosen if the sizes of the real and imaginary parts of all the matrix elements to be printed lie between 0.001 and 1.0. The F11.4 code is chosen if the sizes of all the numbers to be printed lie between 0.001 and 9999.9999. Otherwise the 1PE13.4 code is chosen.

FORMAT = '\*' means that X04DFF will choose a format code such that numbers will be printed to as many significant digits as are necessary to distinguish between neighbouring machine numbers. Thus any two numbers that are stored with different internal representations should look different on output. Whether they do in fact look different will depend on the run-time library of the Fortran compiler in use.

More complicated values of FORMAT, to print a *complex* number in a desired form, may be used. See the description of parameter USEFRM above for more details.

*Constraint:* the character length of FORMAT must be  $\leq 80$ .

## 9: TITLE – CHARACTER\*(\*).

*Input*

*On entry:* a title to be printed above the matrix. If TITLE = ' ', no title (and no blank line) will be printed.

If TITLE contains more than NCOLS characters, the contents of TITLE will be wrapped onto more than one line, with the break after NCOLS characters.

Any trailing blank characters in TITLE are ignored.

## 10: LABROW – CHARACTER\*1.

*Input*

*On entry:* the type of labelling to be applied to the rows of the matrix, as follows:

If LABROW = 'N' or 'n', X04DFF prints no row labels.

If LABROW = 'I' or 'i', X04DFF prints integer row labels.

If LABROW = 'C' or 'c', X04DFF prints character labels, which must be supplied in array RLABS.

*Constraint:* LABROW must be one of 'N', 'n', 'I', 'i', 'C' or 'c'.

11: RLABS(\*) – CHARACTER\*(\*) array. *Input*

*On entry:* if LABROW = 'C' or 'c', RLABS must be dimensioned at least of length M and must contain labels for the rows of the matrix, otherwise RLABS may be dimensioned of length 1.

Labels are right justified when output, in a field which is as wide as necessary to hold the longest row label. Note that this field width is subtracted from the number of usable columns, NCOLS.

12: LABCOL – CHARACTER\*1. *Input*

*On entry:* the type of labelling to be applied to the columns of the matrix, as follows:

If LABCOL = 'N' or 'n', X04DFF prints no column labels.

If LABCOL = 'I' or 'i', X04DFF prints integer column labels.

If LABCOL = 'C' or 'c', X04DFF prints character labels, which must be supplied in array CLABS.

*Constraint:* LABCOL must be one of 'N', 'n', 'I', 'i', 'C' or 'c'.

13: CLABS(\*) – CHARACTER\*(\*) array. *Input*

*On entry:* if LABCOL = 'C' or 'c', CLABS must be dimensioned at least of length N and must contain labels for the columns of the matrix, otherwise CLABS may be dimensioned of length 1.

Labels are right-justified when output. Any label that is too long for the column width, which is determined by FORMAT, is truncated.

14: NCOLS – INTEGER. *Input*

*On entry:* the maximum output record length. If the number of columns of the matrix is too large to be accommodated in NCOLS characters, the matrix will be printed in parts, containing the largest possible number of matrix columns, and each part separated by a blank line.

NCOLS must be large enough to hold at least one column of the matrix using the format specifier in FORMAT. If a value less than 0 or greater than 132 is supplied for NCOLS, then the value 80 is used instead.

15: INDENT – INTEGER. *Input*

*On entry:* the number of columns by which the matrix (and any title and labels) should be indented. The effective value of NCOLS is reduced by INDENT columns. If a value less than 0 or greater than NCOLS is supplied for INDENT, the value 0 is used instead.

16: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry,  $KL < 0$ .

IFAIL = 2

On entry,  $KU < 0$ .

IFAIL = 3

On entry,  $LDA < KL + KU + 1$ .

IFAIL = 4

On entry,  $USEFRM \neq 'A', 'a', 'B', 'b', 'D'$  or  $'d'$ .

IFAIL = 5

On entry, variable **FORMAT** is more than 80 characters long.

IFAIL = 6

The code supplied in **FORMAT** cannot be used to output a number. **FORMAT** probably has too wide a field width or contains an illegal edit descriptor.

IFAIL = 7

On entry, either **LABROW** or **LABCOL**  $\neq 'N', 'n', 'T', 't', 'C'$  or  $'c'$ .

IFAIL = 8

The quantity  $NCOLS - INDENT - LABWID$  (where **LABWID** is the width needed for the row labels) is not large enough to hold at least one column of the matrix.

## 7. Accuracy

Not applicable.

## 8. Further Comments

None.

## 9. Example

This example program calls X04DFF twice, to print band matrices of different orders and bandwidths; various options for labelling and formatting are illustrated.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      X04DFF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NOUT
PARAMETER        (NOUT=6)
INTEGER          NMAX, LDA
PARAMETER        (NMAX=5, LDA=NMAX)
*      .. Local Scalars ..
real           AA
INTEGER          I, IFAIL, INDENT, J, NCOLS
CHARACTER*19     FORMAT
*      .. Local Arrays ..
complex        A(LDA, LDA)
CHARACTER*7      CLABS(NMAX), RLABS(NMAX)
*      .. External Subroutines ..
EXTERNAL         X04DFF
```

```

*      .. Intrinsic Functions ..
      INTRINSIC      cmplx
*      .. Data statements ..
      DATA          CLABS/'Un', 'Deux', 'Trois', 'Quatre', 'Cinq'/
      DATA          RLABS/'Uno', 'Duo', 'Tre', 'Quattro', 'Cinque'/
*      .. Executable Statements ..
      WRITE (NOUT,*) 'X04DFF Example Program Results'
      WRITE (NOUT,*)
*
*      Generate an array of data
      DO 40 J = 1, NMAX
        DO 20 I = 1, LDA
          AA = 10*I + J
          A(I,J) = cmplx(AA,-AA)
20      CONTINUE
40     CONTINUE
*
      NCOLS = 80
      INDENT = 0
      IFAIL = 0
      FORMAT = ' '
*
*      Print 5 by 5 band matrix with 1 sub-diagonal, 1 super-diagonal,
*      default format, bracketed complex numbers, and integer row and
*      column labels
      CALL X04DFF(5,5,1,1,A,LDA,'Bracketed',FORMAT,'Example 1:',
+              'Integer',RLABS,'Integer',CLABS,NCOLS,INDENT,IFAIL)
*
      WRITE (NOUT,*)
      FORMAT = 'SS,F7.1,SP,F6.1,''i'''
*
*      Print 4 by 4 band matrix with 1 sub-diagonal, 2 super-diagonals,
*      user-supplied format and row and column labels
      CALL X04DFF(4,4,1,2,A,LDA,'Direct',FORMAT,'Example 2:',
+              'Character',RLABS,'Character',CLABS,NCOLS,INDENT,
+              IFAIL)
*
      STOP
      END

```

## 9.2. Program Data

None.

## 9.3. Program Results

X04DFF Example Program Results

Example 1:

```

          1
1 ( 21.0000, -21.0000) ( 12.0000, -12.0000)
2 ( 31.0000, -31.0000) ( 22.0000, -22.0000)
3 ( 32.0000, -32.0000) ( 32.0000, -32.0000)
4
5

```

```

          3          4
1
2 ( 13.0000, -13.0000)
3 ( 23.0000, -23.0000) ( 14.0000, -14.0000)
4 ( 33.0000, -33.0000) ( 24.0000, -24.0000)
5 ( 34.0000, -34.0000)

```

```

          5
1
2
3
4 ( 15.0000, -15.0000)
5 ( 25.0000, -25.0000)

```

Example 2:

	Un		Deux		Trois		Quatre	
Uno	31.0	-31.0i	22.0	-22.0i	13.0	-13.0i		
Duo	41.0	-41.0i	32.0	-32.0i	23.0	-23.0i	14.0	-14.0i
Tre			42.0	-42.0i	33.0	-33.0i	24.0	-24.0i
Quattro					43.0	-43.0i	34.0	-34.0i

---



## X04EAF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised terms* and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04EAF is an easy-to-use routine to print an integer matrix stored in a two-dimensional array.

### 2. Specification

```

SUBROUTINE X04EAF (MATRIX, DIAG, M, N, A, LDA, TITLE, IFAIL)
  INTEGER          M, N, A(LDA,*), LDA, IFAIL
  CHARACTER*1      MATRIX, DIAG
  CHARACTER*(*)    TITLE

```

### 3. Description

X04EAF prints an integer matrix. It is an easy-to-use driver for X04EBF. The routine uses default values for the format in which numbers are printed, for labelling the rows and columns, and for output record length.

X04EAF will choose a format code such that numbers will be printed with the smallest I edit descriptor that is large enough to hold all the numbers to be printed.

The matrix is printed with integer row and column labels, and with a maximum record length of 80.

The matrix is output to the unit defined by X04ABF.

### 4. References

None.

### 5. Parameters

1: MATRIX – CHARACTER\*1.

*Input*

*On entry:* indicates the part of the matrix to be printed, as follows:

MATRIX = 'G' or 'g' (General), the whole of the rectangular matrix.

MATRIX = 'L' or 'l' (Lower), the lower triangle of the matrix, or the lower trapezium if the matrix has more rows than columns.

MATRIX = 'U' or 'u' (Upper), the upper triangle of the matrix, or the upper trapezium if the matrix has more columns than rows.

*Constraint:* MATRIX must be one of 'G', 'g', 'L', 'l', 'U' or 'u'.

2: DIAG – CHARACTER\*1.

*Input*

*On entry:* unless MATRIX = 'G' or 'g', DIAG must specify whether the diagonal elements of the matrix are to be printed, as follows:

DIAG = 'B' or 'b' (Blank), the diagonal elements of the matrix are not referenced and not printed.

DIAG = 'U' or 'u' (Unit diagonal), the diagonal elements of the matrix are not referenced, but are assumed all to be unity, and are printed as such.

DIAG = 'N' or 'n' (Non-unit diagonal), the diagonal elements of the matrix are referenced and printed.

If MATRIX = 'G' or 'g', then DIAG need not be set.

*Constraint:* If MATRIX ≠ 'G' or 'g', then DIAG must be one of 'B', 'b', 'U', 'u', 'N' or 'n'.

- 3: M – INTEGER. *Input*  
 4: N – INTEGER. *Input*  
*On entry:* the number of rows and columns of the matrix, respectively, to be printed.  
 If either of M or N is less than 1, X04EAF will exit immediately after printing TITLE; no row or column labels are printed.
- 5: A(LDA,\*) – INTEGER array. *Input*  
 The second dimension of A must be at least max(1,N).  
*On entry:* the matrix to be printed. Only the elements that will be referred to, as specified by parameters MATRIX and DIAG, need be set.
- 6: LDA – INTEGER. *Input*  
*On entry:* the first dimension of the array A as declared in the (sub)program from which X04EAF is called.  
*Constraint:* LDA ≥ M.
- 7: TITLE – CHARACTER\*(\*). *Input*  
*On entry:* a title to be printed above the matrix. If TITLE = ' ', no title (and no blank line) will be printed.  
 If TITLE contains more than 80 characters, the contents of TITLE will be wrapped onto more than one line, with the break after 80 characters.  
 Any trailing blank characters in TITLE are ignored.
- 8: IFAIL – INTEGER. *Input/Output*  
*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.  
*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, MATRIX ≠ 'G', 'g', 'L', 'l', 'U' or 'u'.

IFAIL = 2

On entry, MATRIX = 'L', 'l', 'U' or 'u', but DIAG ≠ 'N', 'n', 'U', 'u', 'B' or 'b'.

IFAIL = 3

On entry, LDA < M.

## 7. Accuracy

Not applicable.

## 8. Further Comments

A call to X04EAF is equivalent to a call to X04EBF with the following argument values:

```

NCOLS = 80
INDENT = 0
LABROW = 'I'
LABCOL = 'I'
FORMAT = ' '
```

## 9. Example

This example program calls X04EAF twice, first to print a 3 by 5 rectangular matrix, and then to print a 5 by 5 triangular matrix.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicized* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      X04EAF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NMAX, LDA
      PARAMETER        (NMAX=5, LDA=NMAX)
*      .. Local Scalars ..
      INTEGER          I, IFAIL, J
*      .. Local Arrays ..
      INTEGER          A(LDA,NMAX)
*      .. External Subroutines ..
      EXTERNAL         X04EAF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'X04EAF Example Program Results'
      WRITE (NOUT,*)
*      Generate an array of data
      DO 40 J = 1, NMAX
        DO 20 I = 1, LDA
          A(I,J) = 10*I + J
20      CONTINUE
40     CONTINUE
*
      IFAIL = 0
*
*      Print 3 by 5 rectangular matrix
      CALL X04EAF('General', ' ', 3, 5, A, LDA, 'Example 1:', IFAIL)
*
      WRITE (NOUT,*)
*
*      Print 5 by 5 lower triangular matrix
      CALL X04EAF('Lower', 'Non-unit', 5, 5, A, LDA, 'Example 2:', IFAIL)
*
      STOP
      END
```

### 9.2. Program Data

None.

### 9.3. Program Results

X04EAF Example Program Results

Example 1:

```
  1  2  3  4  5
1  11 12 13 14 15
2  21 22 23 24 25
3  31 32 33 34 35
```

Example 2:

```
  1  2  3  4  5
1  11
2  21 22
3  31 32 33
4  41 42 43 44
5  51 52 53 54 55
```



## X04EBF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X04EBF prints an integer matrix stored in a two-dimensional array.

### 2. Specification

```

SUBROUTINE X04EBF (MATRIX, DIAG, M, N, A, LDA, FORMAT, TITLE, LABROW,
1                RLABS, LABCOL, CLABS, NCOLS, INDENT, IFAIL)
    INTEGER      M, N, A(LDA,*), LDA, NCOLS, INDENT, IFAIL
    CHARACTER*1  MATRIX, DIAG, LABROW, LABCOL
    CHARACTER*(*) FORMAT, TITLE, RLABS(*), CLABS(*)

```

### 3. Description

X04EBF prints an integer matrix, or part of it, using a format specifier supplied by the user. The matrix is output to the unit defined by X04ABF.

### 4. References

None.

### 5. Parameters

1: MATRIX – CHARACTER\*1.

*Input*

*On entry:* indicates the part of the matrix to be printed, as follows:

MATRIX = 'G' or 'g' (General), the whole of the rectangular matrix.

MATRIX = 'L' or 'l' (Lower), the lower triangle of the matrix, or the lower trapezium if the matrix has more rows than columns.

MATRIX = 'U' or 'u' (Upper), the upper triangle of the matrix, or the upper trapezium if the matrix has more columns than rows.

*Constraint:* MATRIX must be one of 'G', 'g', 'L', 'l', 'U' or 'u'.

2: DIAG – CHARACTER\*1.

*Input*

*On entry:* unless MATRIX = 'G' or 'g', DIAG must specify whether the diagonal elements of the matrix are to be printed, as follows:

DIAG = 'B' or 'b' (Blank), the diagonal elements of the matrix are not referenced and not printed.

DIAG = 'U' or 'u' (Unit diagonal), the diagonal elements of the matrix are not referenced, but are assumed all to be unity, and are printed as such.

DIAG = 'N' or 'n' (Non-unit diagonal), the diagonal elements of the matrix are referenced and printed.

If MATRIX = 'G' or 'g', then DIAG need not be set.

*Constraint:* If MATRIX ≠ 'G' or 'g', then DIAG must be one of 'B', 'b', 'U', 'u', 'N' or 'n'.

3: M – INTEGER.

*Input*

4: N – INTEGER.

*Input*

*On entry:* the number of rows and columns of the matrix, respectively, to be printed.

If either of M or N is less than 1, X04EBF will exit immediately after printing TITLE; no row or column labels are printed.

- 5: A(LDA,\*) – INTEGER array. *Input*  
 The second dimension of A must be at least  $\max(1,N)$ .  
*On entry:* the matrix to be printed. Only the elements that will be referred to, as specified by parameters MATRIX and DIAG, need be set.
- 6: LDA – INTEGER. *Input*  
*On entry:* the first dimension of the array A as declared in the (sub)program from which X04EBF is called.  
*Constraint:*  $LDA \geq M$ .
- 7: FORMAT – CHARACTER\*(\*) . *Input*  
*On entry:* a valid Fortran format code. This may be any format code allowed on the system, whether it is standard Fortran or not. FORMAT is used to print elements of the matrix A. It may or may not be enclosed in brackets. Examples of valid values for FORMAT are '(I6)', 'I4,2X'.  
 In addition, there is a special code which forces X04EBF to choose its own format code: FORMAT = ' ' means that X04EBF will choose a format code such that numbers will be printed using the smallest edit descriptor that is large enough to hold all the numbers to be printed.  
*Constraint:* the character length of FORMAT must be  $\leq 80$ .
- 8: TITLE – CHARACTER\*(\*) . *Input*  
*On entry:* a title to be printed above the matrix. If TITLE = ' ', no title (and no blank line) will be printed.  
 If TITLE contains more than NCOLS characters, the contents of TITLE will be wrapped onto more than one line, with the break after NCOLS characters.  
 Any trailing blank characters in TITLE are ignored.
- 9: LABROW – CHARACTER\*1. *Input*  
*On entry:* indicates the type of labelling to be applied to the rows of the matrix, as follows:  
 If LABROW = 'N' or 'n', X04EBF prints no row labels.  
 If LABROW = 'I' or 'i', X04EBF prints integer row labels.  
 If LABROW = 'C' or 'c', X04EBF prints character labels, which must be supplied in array RLABS.  
*Constraint:* LABROW must be one of 'N', 'n', 'I', 'i', 'C' or 'c'.
- 10: RLABS(\*) – CHARACTER\*(\*) array. *Input*  
*On entry:* if LABROW = 'C' or 'c', RLABS must be dimensioned at least of length M and must contain labels for the rows of the matrix, otherwise RLABS may be dimensioned of length 1.  
 Labels are right justified when output, in a field which is as wide as necessary to hold the longest row label. Note that this field width is subtracted from the number of usable columns, NCOLS.
- 11: LABCOL – CHARACTER\*1. *Input*  
*On entry:* indicates the type of labelling to be applied to the columns of the matrix, as follows:  
 If LABCOL = 'N' or 'n', X04EBF prints no column labels.  
 If LABCOL = 'I' or 'i', X04EBF prints integer column labels.

If LABCOL = 'C' or 'c', X04EBF prints character labels, which must be supplied in array CLABS.

*Constraint:* LABCOL must be one of 'N', 'n', 'T', 't', 'C' or 'c'.

12: CLABS(\*) – CHARACTER\*(\*) array. *Input*

*On entry:* if LABCOL = 'C' or 'c', CLABS must be dimensioned at least of length N and must contain labels for the columns of the matrix, otherwise CLABS may be dimensioned of length 1.

Labels are right-justified when output. Any label that is too long for the column width, which is determined by FORMAT, is truncated.

13: NCOLS – INTEGER. *Input*

*On entry:* the maximum output record length. If the number of columns of the matrix is too large to be accommodated in NCOLS characters, the matrix will be printed in parts, containing the largest possible number of matrix columns, and each part separated by a blank line.

NCOLS must be large enough to hold at least one column of the matrix using the format specifier in FORMAT. If a value less than 0 or greater than 132 is supplied for NCOLS, then the value 80 is used instead.

14: INDENT – INTEGER. *Input*

*On entry:* the number of columns by which the matrix (and any title and labels) should be indented. The effective value of NCOLS is reduced by INDENT columns. If a value less than 0 or greater than NCOLS is supplied for INDENT, the value 0 is used instead.

15: IFAIL – INTEGER. *Input/Output*

*On entry:* IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

*On exit:* IFAIL = 0 unless the routine detects an error (see Section 6).

## 6. Error Indicators and Warnings

Errors detected by the routine:

If on entry IFAIL = 0 or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

IFAIL = 1

On entry, MATRIX  $\neq$  'G', 'g', 'L', 'l', 'U' or 'u'.

IFAIL = 2

On entry, MATRIX = 'L', 'l', 'U' or 'u', but DIAG  $\neq$  'N', 'n', 'U', 'u', 'B' or 'b'.

IFAIL = 3

On entry, M > LDA.

IFAIL = 4

On entry, variable FORMAT is more than 80 characters long.

IFAIL = 5

The code supplied in FORMAT cannot be used to output a number. FORMAT probably has too wide a field width or contains an illegal edit descriptor.

IFAIL = 6

On entry, either LABROW or LABCOL  $\neq$  'N', 'n', 'T', 't', 'C' or 'c'.

IFAIL = 7

The quantity  $\text{NCOLS} - \text{INDENT} - \text{LABWID}$  (where LABWID is the width needed for the row labels) is not large enough to hold at least one column of the matrix.

## 7. Accuracy

Not applicable.

## 8. Further Comments

X04EBF may be used to print a vector, either as a row or as a column. The following code fragment illustrates possible calls.

```

      INTEGER A(4)
      CHARACTER*1 RLABS(1), CLABS(1)
C   Print vector A as a column vector.
      LDA = 4
      IFAIL = 0
      CALL X04EBF('G', 'X', 1, 4, A, LDA, ' ', ' ', 'I', RLABS,
*              'N', CLABS, 80, 0, IFAIL)
C   Print vector A as a row vector.
      LDA = 1
      IFAIL = 0
      CALL X04EBF('G', 'X', 4, 1, A, LDA, ' ', ' ', 'N', RLABS,
*              'I', CLABS, 80, 0, IFAIL)

```

## 9. Example

This example program calls X04EBF twice, first to print a 3 by 5 rectangular matrix, and then to print a 5 by 5 upper triangular matrix; various options for labelling and formatting are illustrated.

### 9.1. Program Text

**Note:** the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*   X04EBF Example Program Text
*   Mark 14 Release.  NAG Copyright 1989.
*   .. Parameters ..
      INTEGER          NOUT
      PARAMETER       (NOUT=6)
      INTEGER          NMAX, LDA
      PARAMETER       (NMAX=5, LDA=NMAX)
*   .. Local Scalars ..
      INTEGER          I, IFAIL, INDENT, J, NCOLS
*   .. Local Arrays ..
      INTEGER          A(LDA, NMAX)
      CHARACTER*7      CLABS(NMAX), RLABS(NMAX)
*   .. External Subroutines ..
      EXTERNAL         X04EBF
*   .. Data statements ..
      DATA            CLABS/'Un', 'Deux', 'Trois', 'Quatre', 'Cinq'/
      DATA            RLABS/'Uno', 'Duo', 'Tre', 'Quattro', 'Cinque'/
*   .. Executable Statements ..
      WRITE (NOUT,*) 'X04EBF Example Program Results'
      WRITE (NOUT,*)
*   Generate an array of data
      DO 40 J = 1, NMAX
         DO 20 I = 1, LDA
            A(I,J) = 10*I + J
20      CONTINUE
40     CONTINUE
      NCOLS = 80
      INDENT = 0
      IFAIL = 0
*

```



```

*   Print 3 by 5 rectangular matrix with default format and integer
*   row and column labels
*   CALL X04EBF('General',' ',3,5,A,LDA,' ','Example 1:','Integer',
+             RLABS,'Integer',CLABS,NCOLS,INDENT,IFAIL)
*
*   WRITE (NOUT,*)
*
*   Print 5 by 5 upper triangular matrix with user-supplied format
*   and row and column labels
*   CALL X04EBF('Upper','Non-unit',5,5,A,LDA,'I8','Example 2:',
+             'Character',RLABS,'Character',CLABS,NCOLS,INDENT,
+             IFAIL)
*
*   STOP
*   END

```

## 9.2. Program Data

None.

## 9.3. Program Results

X04EBF Example Program Results

Example 1:

```

  1  2  3  4  5
1  11 12 13 14 15
2  21 22 23 24 25
3  31 32 33 34 35

```

Example 2:

	Un	Deux	Trois	Quatre	Cinq
Uno	11	12	13	14	15
Duo		22	23	24	25
Tre			33	34	35
Quattro				44	45
Cinque					55

---



## Chapter X05

### Date and Time Utilities

#### 1. Scope of the Chapter

This chapter provides routines to obtain the current real time, and the amount of processor time used.

#### 2. Background to the Problems

##### 2.1. Real Time

Routines are provided to obtain the current time in two different formats, and to compare two such times.

##### 2.2. Processor Time

A routine is provided to return the current amount of processor time used. This allows the timing of a particular routine or section of code.

#### 3. Recommendations on Choice and Use of Available Routines

**Note:** refer to the Users' Note for your implementation to check that a routine is available.

X05AAF returns the current date/time in integer format.  
X05ABF converts from integer to character string date/time.  
X05ACF compares two date/time character strings.  
X05BAF returns the amount of processor time used.



## Chapter X05 – Date and Time Utilities

**Note.** Please refer to the Users' Note for your implementation to check that a routine is available.

<b>Routine Name</b>	<b>Mark of Introduction</b>	<b>Purpose</b>
X05AAF	14	Return date and time as an array of integers
X05ABF	14	Convert array of integers representing date and time to character string
X05ACF	14	Compare two character strings representing date and time
X05BAF	14	Return the CPU time

---



# Chapter X05

## Date and Time Utilities

### Contents

<b>1</b>	<b>Scope of the Chapter</b>	<b>2</b>
<b>2</b>	<b>Background to the Problems</b>	<b>2</b>
2.1	Real Time . . . . .	2
2.2	Processor Time . . . . .	2
<b>3</b>	<b>Recommendations on Choice and Use of Available Routines</b>	<b>2</b>

## 1 Scope of the Chapter

This chapter provides routines to obtain the current real time, and the amount of processor time used.

## 2 Background to the Problems

### 2.1 Real Time

Routines are provided to obtain the current time in two different formats, and to compare two such times.

### 2.2 Processor Time

A routine is provided to return the current amount of processor time used. This allows the timing of a particular routine or section of code.

## 3 Recommendations on Choice and Use of Available Routines

**Note.** Refer to the Users' Note for your implementation to check that a routine is available.

X05AAF returns the current date/time in integer format.

X05ABF converts from integer to character string date/time.

X05ACF compares two date/time character strings.

X05BAF returns the amount of processor time used.

---



## X05AAF – NAG Fortran Library Routine Document

**Note.** Before using this routine, please read the Users' Note for your implementation to check the interpretation of bold italicised terms and other implementation-dependent details.

### 1 Purpose

X05AAF returns the current date and time.

### 2 Specification

```
SUBROUTINE X05AAF(ITIME)
INTEGER          ITIME(7)
```

### 3 Description

X05AAF returns the current date and time as a set of seven integers.

### 4 References

None.

### 5 Parameters

1: ITIME(7) — INTEGER array

*Output*

*On exit:* the current date and time, as follows:

ITIME(1) contains the current year.

ITIME(2) contains the current month, in the range 1–12.

ITIME(3) contains the current day, in the range 1–31.

ITIME(4) contains the current hour, in the range 0–23.

ITIME(5) contains the current minute, in the range 0–59.

ITIME(6) contains the current second, in the range 0–59.

ITIME(7) contains the current millisecond, in the range 0–999.

### 6 Error Indicators and Warnings

None.

### 7 Accuracy

The accuracy of this routine depends on the accuracy of the host machine. In particular, on some machines it may not be possible to return a value for the current millisecond, for example. In this case, the value returned will be zero.

### 8 Further Comments

None.

## 9 Example

This program prints out the vector `ITIME` after a call to `X05AAF`.

### 9.1 Program Text

**Note.** The listing of the example program presented below uses bold italicised terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      X05AAF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*      .. Local Arrays ..
      INTEGER          ITIME(7)
*      .. External Subroutines ..
      EXTERNAL         X05AAF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'X05AAF Example Program Results'
*
      CALL X05AAF(ITIME)
*
      WRITE (NOUT,99999) '      Year : ', ITIME(1)
      WRITE (NOUT,99999) '      Month : ', ITIME(2)
      WRITE (NOUT,99999) '      Day : ', ITIME(3)
      WRITE (NOUT,99999) '      Hour : ', ITIME(4)
      WRITE (NOUT,99999) '      Minute : ', ITIME(5)
      WRITE (NOUT,99999) '      Second : ', ITIME(6)
      WRITE (NOUT,99999) ' Millisecond : ', ITIME(7)
      STOP
*
99999 FORMAT (1X,A,I4)
      END

```

### 9.2 Program Data

None.

### 9.3 Program Results

```

X05AAF Example Program Results
      Year : 1997
      Month :   5
      Day :   21
      Hour :    9
      Minute :  36
      Second :  13
      Millisecond : 511

```

---

## X05ABF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X05ABF converts from a seven-integer format time and date, as returned by X05AAF, into a character string, returned via the routine name.

### 2. Specification

```
CHARACTER*30 FUNCTION X05ABF ( ITIME )
INTEGER          ITIME ( 7 )
```

### 3. Description

X05ABF returns a character string of length 30 which contains the date and time as supplied in argument ITIME. On exit, the character string has the following format:

```
' DAY XXTH MTH YEAR HR:MN:SC.MIL' ,
```

where DAY is one of 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat',

XX is an integer denoting the day of the month,

TH is one of 'st', 'nd', 'rd', 'th',

MTH is one of 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',  
'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec',

YEAR is the year as a four digit integer,

HR is the hour,

MN is the minute,

SC is the second,

MIL is the millisecond.

If on entry the date in ITIME is invalid, the string returned is ' \*\* Illegal date/time \*\* '

### 4. References

None.

### 5. Parameters

1: ITIME(7) – INTEGER array.

*Input*

*On entry:* a date and time in the format returned by X05AAF, as follows:

ITIME(1) must contain the year as a positive integer.

ITIME(2) must contain the month, in the range 1-12.

ITIME(3) must contain the day, in the range 1 to  $p$ , where  $p = 28, 29, 30$  or  $31$ , depending on the month and year.

ITIME(4) must contain the hour, in the range 0-23.

ITIME(5) must contain the minute, in the range 0-59.

ITIME(6) must contain the second, in the range 0-59.

ITIME(7) must contain the millisecond, in the range 0-999.

### 6. Error Indicators and Warnings

None.

## 7. Accuracy

The day name included as part of the character string returned by this routine is calculated assuming that the date is part of the Gregorian calendar. This calendar has been in operation in Europe since October the 15th 1582, and in Great Britain since September the 14th 1752. Entry to this routine with a date earlier than these will therefore not return a day name that is historically accurate.

## 8. Further Comments

Two dates stored in character string format, as returned by this routine, may be compared by X05ACF.

## 9. Example

This program initialises a time in ITIME, and converts it to character format by a call to X05ABF.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      X05ABF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
          INTEGER          NIN, NOUT
          PARAMETER        (NIN=5,NOUT=6)
*      .. Local Scalars ..
          CHARACTER*30      CTIME
*      .. Local Arrays ..
          INTEGER          ITIME(7)
*      .. External Functions ..
          CHARACTER*30      X05ABF
          EXTERNAL          X05ABF
*      .. Executable Statements ..
          WRITE (NOUT,*) 'X05ABF Example Program Results'
          Skip heading in data file
          READ (NIN,*)
          READ (NIN,*) ITIME

          CTIME = X05ABF(ITIME)

          WRITE (NOUT,99999) CTIME
          STOP
*
99999 FORMAT (1X,A)
          END
```

### 9.2. Program Data

```
X05ABF Example Program Data
1789 7 14 13 11 48 320
```

### 9.3. Program Results

```
X05ABF Example Program Results
Tue 14th Jul 1789 13:11:48.320
```

---

## X05ACF – NAG Fortran Library Routine Document

**Note:** before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X05ACF compares two date/time character strings, each stored in the format returned by X05ABF.

### 2. Specification

```
INTEGER FUNCTION X05ACF (CTIME1, CTIME2)
CHARACTER*(*) CTIME1, CTIME2
```

### 3. Description

X05ACF compares two date/time character strings, and returns an integer that specifies which one is the earliest. The result is an integer returned through the routine name, with meaning as follows:

X05ACF = -1 : the first date/time string is earlier than the second.  
X05ACF = 0 : the two date/time strings are equivalent.  
X05ACF = 1 : the first date/time string is later than the second.

### 4. References

None.

### 5. Parameters

- 1: CTIME1 – CHARACTER\*(\*). *Input*  
2: CTIME2 – CHARACTER\*(\*). *Input*

*On entry:* the date/time strings to be compared. These are expected to be in the format returned by X05ABF, although X05ACF will still attempt to interpret the strings if they vary slightly from this format. See Section 8 for further details.

### 6. Error Indicators and Warnings

None.

### 7. Accuracy

Not applicable.

### 8. Further Comments

For flexibility, X05ACF will accept various formats for the two date/time strings CTIME1 and CTIME2.

The strings do not have to be the same length. It is permissible, for example, to enter with one or both of the strings truncated to a smaller length, in which case missing fields are treated as zero.

Each character string may be of any length, but everything after character 80 is ignored.

Each string may or may not include an alphabetic day name, such as 'Wednesday', at its start. These day names are ignored, and no check is made that the day name corresponds correctly to the rest of the date.

The month name may contain any number of characters provided it uniquely identifies the month, however all characters that are supplied are significant.

Each field in the character string must be separated by one or more spaces.

The case of all alphabetic characters is insignificant.

Any field in a date time string that is indecipherable according to the above rules will be converted to a zero value internally. Thus two strings that are completely indecipherable will compare equal.

According to these rules, all the following date/time strings are equivalent:

```
'Thursday 10th July 1958 12:43:17.320'
'THU 10th JULY 1958 12:43:17.320'
'10th Jul 1958 12:43:17.320'
```

## 9. Example

This program initialises two date/time strings, and compares them by a call to X05ACF.

### 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
*      X05ACF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
INTEGER          NIN, NOUT
PARAMETER       (NIN=5,NOUT=6)
*      .. Local Scalars ..
INTEGER          K
CHARACTER*50     CTIME1, CTIME2
*      .. External Functions ..
INTEGER          X05ACF
EXTERNAL         X05ACF
*      .. Executable Statements ..
WRITE (NOUT,*) 'X05ACF Example Program Results'
*      Skip heading in data file
READ (NIN,*)
READ (NIN,*) CTIME1, CTIME2
*
*      K = X05ACF(CTIME1,CTIME2)
*
IF (K.LT.0) THEN
  WRITE (NOUT,99999) CTIME1
  WRITE (NOUT,99999) 'is earlier than'
  WRITE (NOUT,99999) CTIME2
ELSE IF (K.EQ.0) THEN
  WRITE (NOUT,99999) CTIME1
  WRITE (NOUT,99999) 'is equivalent to'
  WRITE (NOUT,99999) CTIME2
ELSE
  WRITE (NOUT,99999) CTIME1
  WRITE (NOUT,99999) 'is later than'
  WRITE (NOUT,99999) CTIME2
END IF
STOP
*
99999 FORMAT (1X,A)
END
```

### 9.2. Program Data

```
X05ACF Example Program Data
'Thu 27th April 1989 13:15:21.320'
'Wed 26th April 1989 11:23:14.130'
```

### 9.3. Program Results

```
X05ACF Example Program Results
Thu 27th April 1989 13:15:21.320
is later than
Wed 26th April 1989 11:23:14.130
```

---





## X05BAF – NAG Fortran Library Routine Document

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of *bold italicised* terms and other implementation-dependent details. The routine name may be precision-dependent.

### 1. Purpose

X05BAF returns the amount of processor time used since an unspecified previous time, via the routine name.

### 2. Specification

```
real FUNCTION X05BAF ( )
```

### 3. Description

X05BAF returns the number of seconds of processor time used since some previous time. The previous time is system dependent, but may be, for example, the time the current job or the current program started running.

If the system clock of the host machine is inaccessible for any reason, X05BAF returns the value zero.

### 4. References

None.

### 5. Parameters

None.

### 6. Error Indicators and Warnings

None.

### 7. Accuracy

The accuracy of the value returned depends on the accuracy of the system clock on the host machine.

### 8. Further Comments

Since the value returned by X05BAF is the amount of processor time since some unspecified earlier time, no significance should be placed on the value other than as a marker to be compared with some later figure returned by X05BAF. The amount of processor time that has elapsed between two calls of X05BAF can be simply calculated as the earlier value subtracted from the later value.

### 9. Example

This program makes a call to X05BAF, performs some computations, makes another call to X05BAF, and gives the time used by the computations as the difference between the two returned values.

## 9.1. Program Text

Note: the listing of the example program presented below uses *bold italicised* terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```

*      X05BAF Example Program Text
*      Mark 14 Release.  NAG Copyright 1989.
*      .. Parameters ..
      INTEGER          NOUT
      PARAMETER       (NOUT=6)
      real            ONE
      PARAMETER       (ONE=1.0e0)
*      .. Local Scalars ..
      real            CPTIME, E, S1, S2, T
      INTEGER          N
*      .. External Functions ..
      real            X05BAF
      EXTERNAL        X05BAF
*      .. Executable Statements ..
      WRITE (NOUT,*) 'X05BAF Example Program Results'
*
      S1 = X05BAF()
*
      E = ONE
      T = ONE
      DO 20 N = 1, 10000
          T = T/N
          E = E + T
20  CONTINUE
*
      S2 = X05BAF()
*
      CPTIME = S2 - S1
      WRITE (NOUT,99999) 'It took', CPTIME, ' seconds to compute e =', E
      STOP
*
99999 FORMAT (1X,A,1P,e10.2,A,e13.5)
      END

```

## 9.2. Program Data

None.

## 9.3. Program Results

```

X05BAF Example Program Results
It took  8.66E-01 seconds to compute e =  2.71828E+00

```

---