NAG Fortran Library Chapter Introduction

X02 – Machine Constants

Contents

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.

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 (1981), but differs in some respects. The essential features are summarized 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

$$f = 0.f_1 f_2 \cdots f_p$$

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 0.12345×10^{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 machines supporting IEEE binary double precision arithmetic:

$$\begin{aligned} b &= 2 \\ p &= 53 \\ e_{\min} &= -1021 \\ e_{\max} &= 1024 \quad \text{and ROUNDS is } \textbf{true}. \end{aligned}$$

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 (1981) 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:

the ***machine precision*** ϵ:	$= \left(\frac{1}{2}\right) \times b^{1-p}$ if ROUNDS is **true**,
	$= b^{1-p}$ otherwise (but see the Note below).
the smallest positive model number:	$= b^{e_{\min}-1}$
the largest positive model number:	$= (1 - b^{-p}) \times b^{e_{\max}}$

Note: the value of ϵ is increased very slightly in some implementations to ensure that the computed result of $1 + \epsilon$ or $1 - \epsilon$ differs from 1.

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:

$-x$

$1/x$

$-1/x$

SQRT(x)

LOG(x)

EXP(LOG(x))

$y**(\text{LOG}(x)/\text{LOG}(y))$ for any y

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:

$-w$

$1/w$

$-1/w$

SQRT(w)

LOG(w)

EXP(LOG(w))

$y**(\text{LOG}(w)/\text{LOG}(y))$ for any y

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: please 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 Index

Derived parameters of model of floating-point arithmetic:
- largest positive model number X02ALF
- machine precision X02AJF
- safe range X02AMF
- safe range of complex floating point arithmetic X02ANF
- smallest positive model number X02AKF

Largest permissible argument for SIN and COS X02AHF
Largest representable integer X02BBF
Maximum number of decimal digits that can be represented X02BEF
Parameters of model of floating-point arithmetic:
- b X02BHF
- emax X02BLF
- emin X02BKF
- p X02BJF
- ROUNDS X02DJF

Switch for taking precautions to avoid underflow X02DAF

4 Routines Withdrawn or Scheduled for Withdrawal

Withdrawn Routine	Mark of Withdrawal	Replacement Routine(s)
X02AAF	16	X02AJF
X02ABF	16	X02AKF
X02ACF	16	X02ALF
X02ADF	14	X02AJF and X02AKF
X02AEF	14	X02AMF

X02AFF	14	X02AMF
X02AGF	16	X02AMF
X02BAF	14	X02BHF
X02BCF	14	X02AMF
X02BDF	14	X02AMF
X02CAF	17	Not needed except with F01BTF and F01BXF

5 References

Brown W S (1981) A simple but realistic model of floating-point computation *ACM Trans. Math. Software* **7** 445–480