NAG C 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 C 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 arguments vary from one implementation of the Library to another, but within the context of a single implementation they are constants.

The arguments are intended for use primarily by other functions in the Library, but users of the Library may sometimes need to refer to them directly.

Most of these constants are *not* functions, but they are defined in the header file <nagx02.h>. Defined constant names are specified in upper case characters, and functions in lower case. Those machine constants which are defined as functions have also been given upper case names using #define in <nagx02.h>.

Each argument-value is returned by a separate C function. Because of the trivial nature of the functions, individual function 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 arguments, NAG uses a simplified **model** of floating-point arithmetic. The arguments 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 arguments and one logical argument. The four integer arguments 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 arguments 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 argument 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:

$$\begin{array}{lcrl} b & = & 2 & \\ p & = & 24 & \\ e_{\min} & = & -127 & \\ e_{\max} & = & 127 & \text{and ROUNDS is } \textbf{NagTrue}. \end{array}$$

For machines supporting IEEE binary double precision arithmetic:

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

For other machines, values of the model arguments 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 argument ROUNDS has been introduced to take account of machines with good rounding.)

2.1.2 Derived arguments of floating-point arithmetic

Most numerical algorithms require access, not to the basic arguments 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 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. 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 C 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** argument 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**(LOG(x)/LOG(y)) for any y

In a similar fashion we define the safe range argument 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**(LOG(w)/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 argument 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 argument 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 functions in the Library.

3 Recommendations on Choice and Use of Available Functions

Derived parameters of model of floating-point arithmetic:
- largest positive model number ..... nag_real_largest_number (X02ALC)
- machine precision ..... nag_machine_precision (X02AJC)
- safe range ..... nag_real_safe_small_number (X02AMC)
- safe range of complex floating point arithmetic ..... nag_complex_safe_small_number (X02ANC)
- smallest positive model number ..... nag_real_smallest_number (X02AKC)

Largest permissible argument for SIN and COS ..... nag_max_sine_argument (X02AHC)
Largest representable integer ..... nag_max_integer (X02BBC)
Maximum number of decimal digits that can be represented ..... nag_decimal_digits (X02BEC)
Parameters of model of floating-point arithmetic:
- b ..... nag_real_base (X02BHC)
- emax ..... nag_real_max_exponent (X02BLC)
- emin ..... nag_real_min_exponent (X02BKC)
- p ..... nag_real_base_digits (X02BJC)
- ROUNDS ..... nag_real_arithmetic_rounds (X02DJC)

Switch for taking precautions to avoid underflow ..... nag_underflow_flag (X02DAC)

4 Functions Withdrawn or Scheduled for Withdrawal

Withdrawn Function	Mark of Withdrawal	Replacement Function(s)
nag_active_set_size (X02CAC)	2	No replacement document required

5 References

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