

in_gamma_sequence documentation

Ian Thompson

Contents

1	Introduction	2
1.1	Files provided	2
2	Compiling	2
3	Testing and using the code	3
4	Generating additional data	3
5	Maple code	3
5.1	binary_float.mpl	3
5.2	in_gamma.mpl	4
5.3	in_gamma_precomp.mpl	5
5.4	in_gamma_test.mpl	5
6	Fortran code	6
6.1	in_gamma_sequence.f90	6
6.2	in_gamma_test.f90	7

1 Introduction

This manual describes the implementation of the algorithm in [1], which computes a sequence of values $S_{n_0}(x), \dots, S_{n_1}(x)$, where $x \geq 0$, and n_0 and n_1 are arbitrary integers with $n_1 \geq n_0$. The function $S_n(x)$ is related to the lower incomplete gamma function via

$$\gamma(n + \tfrac{1}{2}, -x) = e^x i(-1)^n x^{n+1/2} S_n(x). \quad (1)$$

Since $S_n(x)$ is real, $\gamma(n + \tfrac{1}{2}, -x)$ is purely imaginary, the upper incomplete gamma function can be computed using the identity

$$\Gamma(n + \tfrac{1}{2}, x) = \Gamma(n + \tfrac{1}{2}) - \gamma(n + \tfrac{1}{2}, x), \quad (2)$$

without any loss of significant digits.

1.1 Files provided

- `in_gamma_sequence.f90`
- `in_gamma_test.f90`
- `binary_float.mpl`
- `in_gamma.mpl`
- `in_gamma_precomp.mpl`
- `in_gamma_test.mpl`
- `in_gamma_precomp.dat`
- `in_gamma_test.dat`

2 Compiling

The Fortran 2003 code for the algorithm is contained in the file `in_gamma_sequence.f90`. This contains no main program, and so a flag (usually `-c`) is required to prevent the compiler from attempting to produce an executable. Some compilers treat the features introduced in Fortran 2003 as extensions; it is desirable suppress warnings concerning these. Two directives are used to strip out diagnostic code at compile time, and so a flag may be required to invoke the preprocessor. Alternatively, the file extension can be changed (usually to `.F90` or `.ff90`), so that the preprocessor is invoked automatically. Full details of the preprocessor and the means of invoking it are provided in the compiler documentation.¹ The recommended flags for compiling `in_gamma_sequence.f90` are as follows:

- NAG Fortran compiler: `-c -f2003 -fpp`
- Intel Fortran compiler: `-c -stand=f03 -fpp=1`
- Gnu Fortran compiler (gfortran): `-c -std=f2003 -cpp`

¹Intel: <http://software.intel.com/en-us/articles/intel-fortran-composer-xe-documentation>
 NAG: http://www.nag.co.uk/nagware/np/r53_doc/index
 Gnu: <http://gcc.gnu.org/onlinedocs/gfortran/>

The flags `-DDIAGNOSTIC=true` and `-DSUPERDIAGNOSTIC=true` can be used to activate the diagnostic code. These options cause the program to output information during calculations, and have no effect on the results. The testing program `in_gamma_test.f90` contains no preprocessor directives, and so only the second flag (i.e. `-f2003` or the equivalent) is recommended.

3 Testing and using the code

Sequences of values of the function $S_n(x)$ for $n = n_0, \dots, n_1$ can be generated by calling the routine `scaled_in_gamma` (§6.1). For each $n < 0$, there is a unique positive number x_n such that $S_n(x) = 0$, and the algorithm requires that the machine numbers closest to these values, which we denote by \tilde{x}_n , are provided in the file `in_gamma_precomp.dat`, along with the associated values of $S_n(x)$. If data is available for $n = -1, \dots, -M$, then the maximum allowable value for x is $M - 0.18$. This restriction does not apply if $n_0 \geq 0$. The file `in_gamma_precomp.dat` that accompanies the code provides data for $M = -201$.

The program `in_gamma_test` provides a simple front end which can be used to test the algorithm. It prompts the user to choose whether to enter values for n and x manually, or to perform an exhaustive test using data from the file `in_gamma_test.dat`, which is generated by the Maple program `in_gamma_test.mpl` (§5.4).

4 Generating additional data

The maximum allowable value for x can be increased by calculating additional data using the Maple program `in_gamma_precomp.mpl` (§5.3). Likewise, additional testing data can be generated using the Maple program `in_gamma_test.mpl` (§5.4). Before using these programs, both `binary_float.mpl` (§5.1) and `in_gamma.mpl` (§5.2) must be executed. All Maple code should be placed in the same directory, and the current working directory for Maple should be set to this location. The `currentdir` command can be used to set the current working directory.

5 Maple code

5.1 `binary_float.mpl`

This module provides mechanisms for converting Maple's software decimal floats to and from binary floats, and for manipulating binary floats within Maple. A binary float is a number of the form

$$B = \text{sg} \times \text{mantis} \times 2^{\text{xp}+1-\text{mantis_length}}, \quad (3)$$

where $\text{sg} = 1$ or -1 , and `mantis_length` is a fixed parameter. Inside the module, binary floats are represented as ordered triples of the form `(sg,xp,mantis)`.

Parameters

- `nbits`

The number of bits used in representing a real number as a binary float.

- `mantis_length`

The length of the mantissa, including the hidden bit.

- **emax**

The maximum permitted value for the exponent.

For double precision, `nbits = 64`, `mantis_length = 53` and `emax = 1023` [2]. The minimum exponent value is always $1 - \text{emax}$ [2]. The `binary_float` module does not compute subnormal numbers, and so the smallest positive number is obtained by setting all visible bits in the significand to zero, and setting `xp = 1 - emax`.

Procedures

- `nearestBinaryFloat(x)`

Returns the binary float that is closest to the real number `x`.

- `toFraction(sg , xp , mantis)`

Converts the binary float represented by the ordered triple `(sg,xp,mantis)` to a fraction using the formula (3).

- `stepBinaryFloat(sg , xp , mantis, direction)`

If `direction = 1`, this function returns the smallest binary float that exceeds the number represented by the ordered triple `(sg,xp,mantis)`. If `direction = -1`, the largest binary float that is smaller than `(sg,xp,mantis)` is returned.

- `transfer(sg , xp , mantis)`

Mimics the effect of the Fortran function call `transfer(B , 1_li)`, where `B` is a real number given by (3), and `li` is a kind type parameter that denotes an `nbits` bit integer. The return value is an integer `t` whose representation as a sequence of `nbits` binary bits is identical to that of the real number `B`. It is assumed that `t` is stored in two's complement format, `B` is stored in the format specified by [2], and that the endianness for integers and reals is the same.

5.2 in_gamma.mpl

Provides routines for computing $S_n(x)$ to arbitrary precision and locating its zeros.

Parameters

- `d_step`

The number of additional significant digits to include when it is necessary to increase the accuracy of a computation.

Procedures

- `S(n , x , d)`

Attempts to compute $S_n(x)$ to `d` significant decimal digits using [1, (28)].

- `xS(n , x , d)`

Attempts to compute $e^x S_n(x)$ to `d` significant decimal digits using [1, (12)].

- `double_calc_S(n , x , d)`

Repeatedly computes $S_n(x)$ using `S(n , x , d)` and `xS(n , x , d)`, increasing the number of significant digits used until the two agree to `d` significant decimal digits.

- `S_root(n , d)`

Returns an approximation to the location of the point at which $S_n(x)$ evaluates to zero, accurate to at least `d` significant decimal digits.

5.3 `in_gamma_precomp.mpl`

This program computes data for use by the Fortran subroutine `s_by_continuation` (see §6.1), and outputs this to the file `in_gamma_precomp.dat`, which will be overwritten if it already exists. See [1, §7] for further details.

Parameters

- `nmin`

The minimum value of `n` for which data is generated. The maximum value is -1 .

5.4 `in_gamma_test.mpl`

This program computes $S_n(x)$ for a range of parameter values. The resulting data is converted to integer form using `transfer` (§5.1) and output to the file `in_gamma_test.dat`, which will be overwritten if it already exists. The data contained in `in_gamma_test.dat` is used by the Fortran program `in_gamma_test.f90` (§6.2).

Parameters

- `n0`

The minimum value of `n` for which data is generated.

- `n1`

The maximum value of `n` for which data is generated.

- `xmin`

The minimum value of `x` for which data is generated.

- `xmax`

The maximum value of `x` for which data is generated.

- `xsteps`

The number of steps in `x`.

6 Fortran code

6.1 in_gamma_sequence.f90

This module implements the algorithm described in [1]. The only public entity is the subroutine `scaled_in_gamma`, and it should be noted that the other procedures are intended for internal use only, and therefore they perform no checks on the validity of their arguments.

Parameters

- `dp`
Kind type parameter for real numbers. The number of binary bits used to store a variable of type `real (dp)` must be equal to the parameter `nbits` in the Maple code `binary_float.mpl` (§5.1).
- `li`
Kind type parameter for integers. The number of bits used to represent variables of types `integer (li)` and `real (dp)` must be equal.

Procedures

- subroutine `scaled_in_gamma(n0 , n1 , x , res)`

```
integer    , intent (in) :: n0 , n1
```

```
real (dp)  , intent (in) :: x
```

```
real (dp)  , intent (out) , allocatable , dimension (:) :: res
```

Computes a sequence of values of $S_n(x)$ for $n = n_0, \dots, n_1$, for $n_1 \geq n_0$ and $x \geq 0$, and returns the result in the allocatable array `res`. If $n_0 < 0$, then x must not exceed $M - 0.18$ (see §3).

- subroutine `error(proc_name , msg)`

```
character (*) , intent (in) :: proc_name , msg
```

Reports that procedure `proc_name` has encountered an error, outputs the message `msg` and terminates execution.

- real (dp) function `sm1_series1(x)`

```
real (dp)  , intent (in) :: x
```

Calculates $S_{-1}(x)$ using [1, (12)], under the assumption that $0 \leq x \leq 1$.

- real (dp) function `s_series2(n , x)`

```
integer    , intent (in) :: n
```

```
real (dp)  , intent (in) :: x
```

Calculates $S_n(x)$ using [1, (28)].

- `real (dp) function s_by_continuation(n , x)`

`integer , intent (in) :: n`

`real (dp) , intent (in) :: x`

Uses analytic continuation to compute $S_n(x)$, as described in [1, §5].

- `subroutine read_precomp_data()`

Reads the data contained in the file `in_gamma_precomp.dat` into memory.

6.2 `in_gamma_test.f90`

A simple testing program. See §3 for details.

References

- [1] I. Thompson. Computation of incomplete gamma functions with negative arguments. Submitted for consideration with this code, 2011.
- [2] IEEE Standard for Floating-Point Arithmetic. Technical report, Microprocessor Standards Committee of the IEEE Computer Society, 3 Park Avenue, New York, NY 10016-5997, USA, August 2008.