NOLTA, IEICE

Paper

Verified sharp bounds for the real gamma function over the entire floating-point range

Siegfried M. Rump

Institute for Reliable Computing, Hamburg University of Technology
Schwarzenbergstraße 95, 21073 Hamburg, Germany
and
Visiting Professor at Waseda University, Faculty of Science and Engineering,
3–4–1 Okubo, Shinjuku-ku, Tokyo 169–8555, Japan
rump@tuhh.de

Received October 27, 20XX; Revised December 29, 20XX; Published July 1, 20XX

Abstract: An algorithm is presented for computing verified and accurate bounds for the value of the gamma function over the entire real double precision floating-point range. It means that for every double precision floating-point number x except the poles $-k$ for $0 \le k \in \mathbb{N}$ the true value of $\Gamma(x)$ is included within an almost maximally accurate interval with floating-point bounds. The application to interval arguments $\mathbf{x} \in \mathbb{IF}$, thus enclosing the range of Γ over $\mathbf{x}$, is discussed as well.

Key Words: Gamma function, rigorous error bounds, INTLAB.

1. Introduction and notation

For $z \in \mathbb{C}$ with positive real part, the gamma function is defined by

$$\Gamma(z) := \int_0^\infty e^{-t} t^{z-1} dt \ ,$$

and otherwise by analytic continuation. For $\mathbb{F}$ denoting the set of double precision floating-point numbers according to IEEE 754 [7, 8], we aim on verified error bounds for $\Gamma(x)$ for all real $x \in \mathbb{F}$ except the poles $-k$ for $0 \le k \in \mathbb{N}$.

Numerous publications describe methods to compute approximations of the gamma function with high accuracy, among them [1, 2, 4, 12–14, 17, 18, 22, 23]. Standard ways to approximate $\Gamma(x)$ are by continued fractions, polynomial or rational approximation, Lanczos methods, or countour integrals.

However, it seems not easy to derive rigorous and accurate error bounds based on those approximations. In particular for arguments of large absolute value and near the poles numerical difficulties arise. For example, in the newest release R2013b of Matlab [15] the built-in approximate gamma function delivers

```
>> gamma(-1+2^(-53))
ans =
```


Nonlinear Theory and Its Applications, IEICE, vol. 2, no. 3, pp. 1101–1109 DOI: 10.1588/nolta.2.1101

```
    -5.545090608933970e+15
```

whereas the true value to five places is $-9.0072 \cdot 10^{15}$. Note that the argument $-1+2^{-53}$ is the double precision successor of -1 in $\mathbb{F}$. The algorithm to be described computes

```
>> gamma(intval(-1+2^(-53)))
intval ans =
  1.0e+015 *
[  -9.00719925474100,  -9.00719925474098]
```

in executable INTLAB code [19]. The relative error of the bounds with respect to the midpoint is less than $5 \cdot 10^{-16}$ or a distance of 4 bits between the left and right bound. To my knowledge only one method is known to compute verified error bounds [11] for the gamma function. In this paper we aim to compute accurate bounds of a quality as in the example above over the entire range of $\mathbb{F}$.

We assume the reader to be familiar with the IEEE 754 floating-point standard [7, 8] and with basic concepts of interval arithmetic, cf. [16, 21].

2. Main results

We develop our algorithm in several steps. We start with identifying the range of floating-point numbers such that $\Gamma(x)$ is neither in the overflow nor underflow range, continue with argument reductions into the interval $[1, 2]$, and finally describe how to compute error bounds for $x \in [1, 2]$. Lastly, we add a remark on interval arguments.

2.1 The floating-point range

The floating-point number `xmax = hex2num('406573fae561f647')` which is about 171.624377 is the largest floating-point number $x \in \mathbb{F}$ such that $\Gamma(x)$ does not cause overflow. Hence for $x >$ `xmax` the best possible error bound of $\Gamma(x)$ is $[$`realmax`, `Inf`$]$. Note that for the built-in gamma function already `gamma(171.62430)` yields `Inf`.

For non-positive integers our inclusion algorithm yields `NaN` because the gamma function has a pole with non-coinciding left and right limit. For $x \in \mathbb{F}$, $N \in \mathbb{N}$ and $-N < x < -N + 1$ it follows $N_1 \le x \le N_2$ for N_1 and N_2 denoting the floating-point successor and predecessor of $-N$ and $-N+1$, respectively. Hence

$$|\Gamma(x)| \le \max\big(\,|\Gamma(N_1)|, |\Gamma(N_2|)\big) \qquad \text{for all } x \in \mathbb{F} \text{ with } -N < x < -N + 1\ .$$

For $-N = -185$ this maximum is smaller than the smallest positive (denormalized) double precision floating-point number $T := 2^{-1074} \approx 4.94 \cdot 10^{-324}$. By the recurrence relation $\Gamma(x + 1) = x\Gamma(x)$ it follows that for all $x \in \mathbb{F}$ with $\lfloor x \rfloor \le -185$ the best possible inclusion of $\Gamma(x)$ with floating-point bounds is $[0, T]$ or $[-T, 0]$, depending on whether $\lfloor x \rfloor$ is even or odd, respectively.

Henceforth we can restrict our attention to the range $(-184,$ `xmax`$]$. We will use an argument reduction to restrict the range further to $[1, 2]$.

2.2 Argument reduction for $x > 2$

The recurrence relation $\Gamma(x + 1) = x\Gamma(x)$ implies the argument reduction

$$\Gamma(x) = \Gamma(x - k) \prod_{i=1}^{k} (x - i) \tag{1}$$

for every $k \in \mathbb{N}$, so that the particular choice $k := \lfloor x \rfloor - 1$ assures $x - k \in [1, 2)$. Computing the product in (1) with directed rounding downwards and upwards yields fairly good error bounds because the relative error of a product is bounded by the sum of the relative errors of the factors. However, for arguments near `xmax` the relative error may be 10^{-14} and larger. To reduce this, the product has to be computed more accurately.

One possibility is a compensated product as proposed by Graillat [5]. For a vector $x \in \mathbb{F}^n$ of floating-point numbers the algorithm is as follows.

Algorithm 1 Compensated product for $x \in \mathbb{F}^n$.

function $\texttt{res} = \texttt{CompProd}(x)$
  $p_1 = x_1$
  $e_1 = 0$
  for $i = 2 : n$
    $[p_i, \pi_i] = \texttt{TwoProduct}(p_{i-1}, x_i)$
    $e_i = \mathrm{fl}(e_{i-1}x_i + \pi_i)$
  end for
  $\texttt{res} = \mathrm{fl}(p_n + e_n)$

Here $\mathrm{fl}(\cdot)$ means that the expression in the parenthesis is computed in floating-point arithmetic. Moreover, $[x, y] = \texttt{TwoProduct}(a, b)$ denotes the error-free transformation $x + y = ab$ as analyzed by Knuth [10]. It is based on the algorithm $[x, y] = \texttt{Split}(a)$ which is the error-free splitting of $a \in \mathbb{F}$ into $x + y$ such that both x and y have at most 26 nonzero leading bits in the mantissa [3].

Graillat analyzes that the error of res with respect to the true product $p := \prod_{i=1}^{n} x_i$ is bounded by

$$|\texttt{res} - p| \leq \mathbf{u}|p| + \gamma_n\gamma_{2n}|p| \ , \tag{2}$$

provided no overflow or underflow occurs. Here $\mathbf{u}$ is the relative rounding error unit which is 2^{-53} in double precision, and $\gamma_k := k\mathbf{u}/(1 - k\mathbf{u})$ as usual. From (2) bounds depending on the *computed* result res are easily derived.

However, we do not need this high accuracy, a result with relative error $\mathbf{u}$ is sufficient. For a general vector $x \in \mathbb{F}^n$ this is achieved by the following algorithm.

Algorithm 2 Accurate product.

function $[p, q] = \texttt{AccProd}(x)$
  $[p, q] = \texttt{Split}(x_1)$
  for $k = 2 : n$
    $[b, \beta] = \texttt{Split}(x_k)$
    $[p', \pi] = \texttt{Split}(p \cdot b)$
    $q = \mathrm{fl}(p\beta + qx_k + \pi)$
    $p = p'$
  end for

The algorithm uses the fact that both p and b have at most 26 nonzero leading bits in their mantissa, so that $pb = \mathrm{fl}(pb)$ and the splitting $p' + \pi = p \cdot b$ is error-free. The only non-exact operations are those in the secoond last line of the loop. The following is true.

Theorem 1 Let $x \in \mathbb{F}^n$ with $n < 2^{25}$ be given, and let p, q be the results computed in double precision by Algorithm 2. Then

$$|\prod_{i=1}^{n} x_i - (p + q)| \leq \frac{3n^2}{1 - 2^{-25}n} 2^{-79}|p + q| \ , \tag{3}$$

provided no over- or underflow occurred.

The proof is deferred to the appendix. The theorem implies that for $n \leq 4729$ the result is of maximum accuracy, by far sufficient for our purpose. One part of the method is not to deliver one floating-point approximation as the output but to keep the approximation and error term separately.

In our case the method can be further simplified with an adapted analysis which we omit here. For $x \in \mathbb{F}$ with $x \geq 2$, the differences $x - i$ for $i = 1, 2, \ldots, \lfloor x \rfloor - 1$ do not cause a rounding error. Thus one error-free splitting $[e, f] = \texttt{Split}(x)$ is sufficient because it implies $(e - i) + f = x - i$. Moreover, both $e - i$ and f have at most 256 nonzero leading bits in the mantissa. Furthermore, the product (1) for the argument reduction cannot produce underflow.

Only overflow has to be taken care of, both in the splitting algorithm and in the argument reduction. This is because in the splitting the input is multiplied by $2^{27} + 1$, and for $x \leq \texttt{xmax}$ the product in (1) may cause overflow. Both is easily resolved by scaling with a proper power of 2.

2.3 Argument reduction for $x < 1$

For $x < 1$ the argument reduction is similar to that of the previous subsection, but additional care is necessary. The recurrence relation $\Gamma(x+1) = x\Gamma(x)$ implies the argument reduction

$$\Gamma(x) = \Gamma(x+k) \, \Big(\prod_{i=0}^{k-1} (x+i) \Big)^{-1} \tag{4}$$

for $k \in \mathbb{N}$, so that the particular choice $k := 1 - \lfloor x \rfloor$ assures $x + k \in [1, 2)$. For $x \in \mathbb{F}$ and $x < -1$ it follows that all $x + i$ are floating-point numbers for $0 \le i \le k-1$, so that no rounding error occurs in the computation of the factors $x + i$. This is not true for $-1 < x < 1$. Here we distinguish three cases.

If $-1 < x \le -0.5$, then $x + 1 \in \mathbb{F}$ by Sterbenz' lemma [6], but not necessarily $x + 2 \in \mathbb{F}$ as for $x = -1 + \mathbf{u}$. But $1 \le x + 2 < 2$, so that

$$-1 < x \le -0.5 \qquad \Rightarrow \qquad x + 1 \in \mathbb{F} \quad \text{and} \quad |\mathrm{fl}(x+2) - (x+2)| \le \mathbf{u} \; . \tag{5}$$

Similarly it follows

$$-0.5 < x < 0 \qquad \Rightarrow \qquad |\mathrm{fl}(x+1) - (x+1)| \le \mathbf{u}/2 \quad \text{and} \quad |\mathrm{fl}(x+2) - (x+2)| \le \mathbf{u} \; , \tag{6}$$

and

$$0 < x < 1 \qquad \Rightarrow \qquad |\mathrm{fl}(x+1) - (x+1)| \le \mathbf{u} \; . \tag{7}$$

We choose the simple approach to calculate $x+i$ in the cases above, i.e. for $-1 < x < 1$, with directed rounding to obtain an accurate inclusion of the product $\prod_{i=0}^{k-1} (x+i)$.

2.4 Arguments $x \in [1, 2]$

In [20] effective methods to compute accurate error bounds for elementary standard functions are based on sample values together with some kind of addition theorems. This seems difficult because the gamma function lacks an addition theorem expressing $\Gamma(\widetilde{x} + \delta)$ in terms of $\Gamma(\widetilde{x})$ plus some error term, and a Taylor expansion needs $\Gamma'(x) = \Gamma(x)\Psi(x)$.

A polynomial approximation being accurate to the last bit requires a high degree. A simple solution to that is to compute low degree polynomials P_i for the intervals $I_i := [1+ih, 1+(i+1)h]$ for $h := 1/N$ and $0 \le i < N$. The larger N the smaller we can choose the maximum degree of the polynomials P_i, but the more polynomials are necessary. The polynomials are simply computed as interpolation polynomials at Chebyshev points. We choose the tradeoff $N = 512$ for which polynomials of degree 4 are sufficient to produce approximations with least significant bit accuracy.

Having computed the approximating polynomials P_i, the maximum error $\Gamma(x) - P_i(x)$ for $x \in I_i$ is determined by some high precision routine. This has to be done once, and lower and upper bounds `err1` and `err2` for the maximum negative and positive error, respectively, are stored. For given $x \in \mathbb{F}$ with $x \in I_i$, the final inclusion algorithm for the gamma function evaluates $P_i(x)$ with directed rounding and subtracts or adds `err1` or `err2` depending on the rounding mode, respectively.

The amount of storage for all polynomial coefficients is about 20 kByte. Note that for $x \in \mathbb{F}^k$ Matlab's vector notation allows a simple evalulation of the vector $P_i(x)$ by choosing the correct polynomial indices i depending on x_ν for $1 \le \nu \le k$.

2.5 Interval arguments

For interval input $\mathbf{x} \in \mathbb{IF}$ we first check whether there exists some integer $k \in \mathbb{Z}$ with $k \le 0$ and $k \in \mathbf{x}$. In that case $\Gamma(\mathbf{x})$ is not defined for all $\widetilde{x} \in \mathbf{x}$ and the result of $\Gamma(\mathbf{x})$ is `NaN`. Otherwise, we apply the argument reductions with directed rounding as described in the previous subsections, so that without loss of generality we may assume $\mathbf{x} \subseteq [1, 2]$. Define

$$\xi := \texttt{hex2num}(\texttt{'3ff762d86356be39'}) \in \mathbb{F} \qquad \text{with} \;\; \xi \approx 1.4616 \; .$$

Then

$$\Gamma'(\widetilde{x}) < 0 \quad \text{for} \quad 1 \le \widetilde{x} \le \xi \qquad \text{and } \Gamma'(\widetilde{x}) > 0 \quad \text{for} \quad \xi < \widetilde{x} \le 2 \ .$$

Hence an inclusion of $\{\Gamma(\mathbf{x}) : \widetilde{x} \in \mathbf{x}\}$ is easily computed with some case distinctions and the methods presented in the previous subsections. In particular the lower bound of $\Gamma(\mathbf{x})$ for $\mathbf{x} = [x_1, x_2]$ is

$$\Lambda := \texttt{hex2num}('\texttt{3fec56dc82a74aee}') \in \mathbb{F} \qquad \text{with} \qquad \Lambda \approx 0.8856$$

if $\xi \in \mathbf{x}$, and the upper bound is $\max\big(\Gamma(x_1), \Gamma(x_2)\big)$.

3. Computational results

As we have seen in Subsection 2.1 we have one-bit wide inclusions of $\Gamma(x)$ for x outside $[-184, 172]$. Thus we display computational results only for x inside this interval.

We say an interval $\mathbf{x} \in \mathbb{IF}$ has a *width of m bits* if the supremum of $\mathbf{x}$ is the m-th floating-point successor of the infimum of $\mathbf{x}$. For x outside $[-184, 172]$ the width in bits of the inclusion of $\Gamma(x)$ is always 1. This is best possible.

For each interval $I_k := [k, k+1]$ with $-184 \le k \le 171$ we generate one million random points x within I_k and compute inclusions $\mathbf{y}$ of $\Gamma(x)$. We then compute the median and the maximum width in bits of all those inclusions $\mathbf{y}$ and display them in Figure 1. The graph of the mean looks very similar to the median.

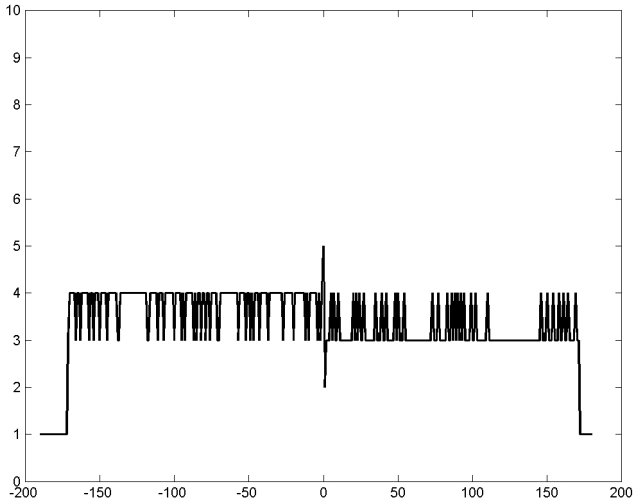

Fig. 1. Median and maximum of the width in bits of inclusions of $\Gamma(x)$ in the range $[-185, 171]$.

As can be seen in the median the width is between 3 and 4 bits, and maximally 8 bits except a small range near zero. Next compute the same pictures for 100 million random points in the range $[-2, 2]$. The results are displayed in Figure 2.

Now the median width is between 3 and 6 bits with a maximum of again 9 bits for positive arguments near zero. For arguments in $[1, 2]$ the median width is only 2 and maximally 3 bits. In Figure 3 the relative error of the Matlab built-in gamma function near $x = -1$ is displayed. For arguments close to but less than -1, the approximation is very accurate. However, in the range $(-1, -1 + 10^{-11}]$ only around 6 figures, and very close to -1, as mentioned in the introduction, no figure at all is correct.

Finally we investigate small numbers and generate 100 million logarithmically distributed test points between 10^{-320} and 1. Here the mean width is 4.46 bits, the median is 4 bits and the maximum again 9 bits; a corresponding figure is omitted.

The presented algorithm is included in INTLAB [19] Version 7.2.

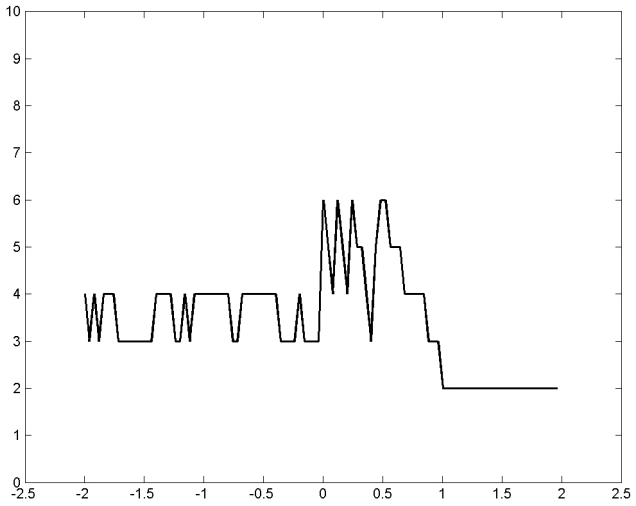

Fig. 2. Median and maximum of the width in bits of inclusions of $\Gamma(x)$ in the range $[-2, 2]$.

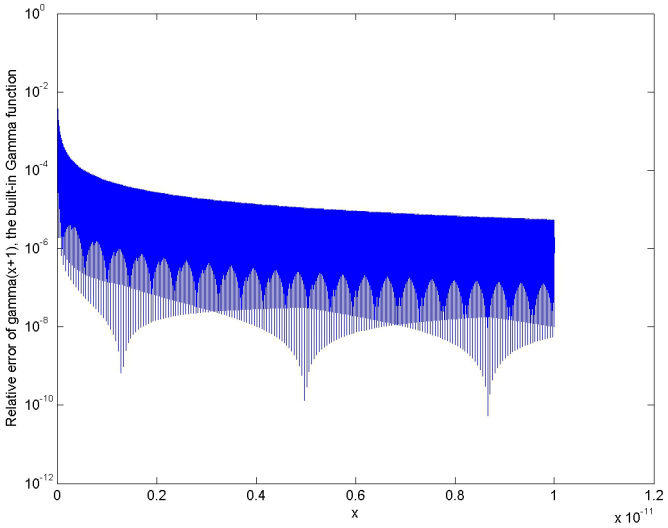

Fig. 3. Relative error of the Matlab built-in gamma function in the range $(-1, -1 + 10^{-11}]$.

4. Appendix

To prove Theorem 1 we start with an important property of the `Split`-function. As has been noted, for $a \in \mathbb{F}$ the call $[x, y] = \texttt{Split}(a)$ produces $x, y \in \mathbb{F}$ with $x + y = a$ such that the binary expansion of both x and y have at most 26 nonzero leading bits. This means that for $[u, v] = \texttt{Split}(b)$ all products xu, xv, yu and yv are floating-point numbers so that the multiplications do not produce any rounding error. Moreover, we frequently need

$$[x, y] = \texttt{Split}(a) \qquad \Rightarrow \qquad |y| \leq 2^{-26}|a| \tag{8}$$

in the following. For the analysis, we first rewrite Algorithm 2 (`AccProd`) with indices for each intermediate variable.

$[p_1, \pi_1] = \text{Split } (x_1)$
$q_1 = \pi_1$
for $k = 2 : n$
$\quad [b_k, \beta_k] = \text{Split } (x_k) \qquad \%\ b_k + \beta_k = x_k$
$\quad [p_k, \pi_k] = \text{Split } (p_{k-1} \cdot b_k) \qquad \%\ p_k + \pi_k = p_{k-1} \cdot b_k$
$\quad \widetilde{q}_k = \text{fl}(p_{k-1}\beta_k + \widetilde{q}_{k-1}x_k + \pi_k)$

Note that in the last line $\widetilde{q}_k$ is the result of the expression in any order of evaluation. Since it is a dot product we know by [9, Proposition 4.1] that

$$|\widetilde{q}_k - q_k| \leq 3\mathbf{u}\big(|p_{k-1}\beta_k| + |\widetilde{q}_{k-1}x_k| + |\pi_k|\big) \qquad \text{for} \quad q_k := p_{k-1}\beta_k + \widetilde{q}_{k-1}x_k + \pi_k \ . \tag{9}$$

The mentioned properties of the `Split`-function and in particular (8) imply $p_{k-1}b_k \in \mathbb{F}$, such that

$$p_k + \pi_k = p_{k-1}b_k \quad \text{and} \quad |\beta_k| \leq \varphi|x_k| \quad \text{and} \quad |\pi_k| \leq \varphi|p_{k-1}b_k| \qquad \text{for } \ \varphi := 2^{-26} \ . \tag{10}$$

Define

$$P_k := p_k + \widetilde{q}_k \quad \text{and} \quad \Delta_k := P_k - \prod_{i=1}^{k} x_i \ . \tag{11}$$

Then

$$\begin{aligned} P_k &= p_{k-1}b_k - \pi_k + \widetilde{q}_k - q_k + p_{k-1}\beta_k + \widetilde{q}_{k-1}x_k + \pi_k \\ &= p_{k-1}(b_k + \beta_k) + \widetilde{q}_{k-1}x_k + \widetilde{q}_k - q_k \\ &= P_{k-1}x_k + \widetilde{q}_k - q_k \ . \end{aligned} \tag{12}$$

By $b_k + \beta_k = x_k$ and (10) we know $p_k = p_{k-1}b_k(1+\varphi_k)$ and $b_k = x_k(1+\widetilde{\varphi}_k)$ for some $|\varphi_k|, |\widetilde{\varphi}_k| \leq \varphi$, so that

$$p_k = p_{k-1}x_k(1+\widetilde{\varphi}_k)(1+\varphi_k) \ . \tag{13}$$

Putting things together it follows

$$\left| p_k - \prod_{i=1}^{k} x_i \right| \leq \left[(1+\varphi)^{2k-1} - 1\right] \left| \prod_{i=1}^{k} x_i \right| \leq \frac{(2k-1)\varphi}{1-(2k-1)\varphi} \left| \prod_{i=1}^{k} x_i \right| \tag{14}$$

and

$$|p_k| \leq \left(1 + \frac{(2k-1)\varphi}{1-(2k-1)\varphi}\right) \left| \prod_{i=1}^{k} x_i \right| = (1-(2k-1)\varphi)^{-1} \left| \prod_{i=1}^{k} x_i \right| \ . \tag{15}$$

Using (9), (10) and (8) we have

$$\begin{aligned} |\widetilde{q}_k| &\leq |q_k| + |\widetilde{q}_k - q_k| \leq (1+3\mathbf{u})\big[\,|p_{k-1}\beta_k| + |\widetilde{q}_{k-1}x_k| + |\pi_k|\,\big] \\ &\leq (1+3\mathbf{u})[\varphi|p_{k-1}x_k| + |\widetilde{q}_{k-1}x_k| + \varphi|p_{k-1}b_k|] \\ &\leq (1+3\mathbf{u})\big[\varphi|p_{k-1}| + |\widetilde{q}_{k-1}| + \varphi(1+\varphi)|p_{k-1}|\big]|x_k| \\ &= (1+3\mathbf{u})\big[\varphi(2+\varphi)|p_{k-1}| + |\widetilde{q}_{k-1}|\big]|x_k| \\ &\leq (1+3\mathbf{u})\left[\frac{\varphi(2+\varphi)}{1-(2k-3)\varphi}\left|\prod_{i=1}^{k} x_i\right| + |\widetilde{q}_{k-1}||x_k|\right] \quad \text{for} \quad k \geq 2 \ . \end{aligned} \tag{16}$$

An induction argument using $\widetilde{q}_1 = q_1 = \pi_1$ and therefore $|\widetilde{q}_1| \leq \varphi|x_1|$ together with $3\mathbf{u} = 1.5\varphi^2$ yields

$$|\widetilde{q}_k| \leq \frac{(2k-1)\varphi}{1-(2k-1)\varphi}\left|\prod_{i=1}^{k} x_i\right| \quad \text{for} \quad k \geq 1 \ . \tag{17}$$

Hence the same argument as in (16) gives

$$|\widetilde{q}_k - q_k| \le \frac{3\mathbf{u}\varphi(2k-1+\varphi)}{1-(2k-3)\varphi} \left| \prod_{i=1}^{k} x_i \right| . \tag{18}$$

Hence (11) and (12) give

$$\Delta_k = P_k - \prod_{i=1}^{k} x_i = P_{k-1}x_k - \prod_{i=1}^{k} x_i + \widetilde{q}_k - q_k = \Delta_{k-1}x_k + \widetilde{q}_k - q_k \tag{19}$$

and therefore

$$|\Delta_k| \le |\Delta_{k-1}||x_k| + |\widetilde{q}_k - q_k| \le |\Delta_{k-1}||x_k| + \frac{3\mathbf{u}\varphi(2k-1+\varphi)}{1-(2k-3)\varphi} \left| \prod_{i=1}^{k} x_i \right| . \tag{20}$$

An induction argument using $\Delta_1 = P_1 - x_1 = p_1 + \pi_1 - x_1 = 0$ and (11) yield

$$|\Delta_k| \le \psi_k \left| \prod_{i=1}^{k} x_i \right| \le \psi_k |\Delta_k| + \psi_k P_k \qquad \text{with} \quad \psi_k := \frac{3k^2\varphi\mathbf{u}}{1-(2k-1)\varphi} \quad \text{for} \quad k \ge 1 , \tag{21}$$

so that finally a computation using $k < 2^{25}$ and thus $3k^2\mathbf{u} < 1$ yields

$$|\Delta_k| \le \frac{3k^2\varphi\mathbf{u}}{1-2k\varphi} |P_k| \qquad \text{for} \quad k \ge 1 . \tag{22}$$

By $P_k = p_k + \widetilde{q}_k$ we proved

$$\prod_{i=1}^{k} x_i \ \in \ p_k + \widetilde{q}_k \ \pm \ \frac{3k^2\varphi}{1-2k\varphi}\mathbf{u}|p_k + \widetilde{q}_k| \tag{23}$$

and thus Theorem 1.

References

[1] M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions.* Dover publications, New York, 1968.

[2] W. J. Cody. Performance evaluation of programs related to the real gamma function. *ACM Trans. Math. Softw.*, 17(1):46–54, 1991.

[3] T.J. Dekker. A floating-point technique for extending the available precision. *Numerische Mathematik*, 18:224–242, 1971.

[4] P. Godfrey. A not on the computation of the convergent Lanczos complex Gamma approximation, 2001. e-mail conversation.

[5] S. Graillat. Accurate floating-point product. In *Proceedings of the REC08*, Georgia Tech, 2008.

[6] N. J. Higham. *Accuracy and stability of numerical algorithms.* SIAM Publications, Philadelphia, 2nd edition, 2002.

[7] IEEE, New York. *ANSI/IEEE 754-2008: IEEE Standard for Floating-Point Arithmetic*, 2008.

[8] *ANSI/IEEE 754-1985: IEEE Standard for Binary Floating-Point Arithmetic.* New York, 1985.

[9] C.-P. Jeannerod and S.M. Rump. Improved error bounds for inner products in floating-point artihmetic. *SIAM. J. Matrix Anal. & Appl. (SIMAX)*, 34(2):338–344, 2013.

[10] D.E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison Wesley, Reading, Massachusetts, 1969.

[11] W. Krämer. Computation of the gamma function Γ(x) for real point and interval arguments. *Z. Angew. Math. Mech. (ZAMM)*, 70(6):581–584, 1990.

[12] Hirondo Kuki. Complex gamma function with error control. *Commun. ACM*, 15(4):262–267, 1972.

[13] C. Lanczos. A Precision Approximation of the Gamma Function. *SIAM J. Numer. Anal. (SINUM)*, 1:86–96, 1964.

[14] Y. L. Luke. Algorithms for the computation of mathematical functions, 1977.

[15] MATLAB. User's Guide, Version 2013b, the MathWorks Inc., 2013.

[16] A. Neumaier. *Interval Methods for Systems of Equations*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1990.

[17] E.W. Ng. A Comparison of Computational Methods and Algorithms for the Complex Gamma Function. *ACM Trans. Math. Software*, 1(1):56–70, 1975.

[18] Frank W. Olver, Daniel W. Lozier, Ronald F. Boisvert, and Charles W. Clark. *NIST Handbook of Mathematical Functions*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[19] S.M. Rump. INTLAB - INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999.

[20] S.M. Rump. Rigorous and portable standard functions. *BIT Numerical Mathematics*, 41(3):540–562, 2001.

[21] S.M. Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 19:287–449, 2010.

[22] Thomas Schmelzer and Lloyd N. Trefethen. Computing the Gamma Function Using Contour Integrals and Rational Approximations. *Siam Journal on Numerical Analysis*, 45:558–571, 2007.

[23] J.L. Spouge. Computation of the Gamma, Digamma, and Trigamma Functions. *SIAM J. Numer. Anal. (SINUM)*, 31(3):931–944, 1994.