TECHNICAL REPORT FOR ARCHER2-PROJECT-eCSE11-7

eCSE Title: ***"High Performance Algorithms for the Computation of the Hardy Function – Dissemination & Development"***

PI Name: Dr D M Lewis (University of Liverpool).

Technical Staff: Dr Mario Antonioletti (EPCC, The University of Edinburgh),

Dr A. R. Brereton (University of California, Berkeley).

Abstract

In 2011 G. A. Hiary devised a computational algorithm for the Hardy function $Z(t)$, requiring just $O\left(t^{1/3}\{log(t)\}^{\kappa}\right)$ operations. This compares to $O\left(\sqrt{t}\right)$ operations necessary for computing $Z(t)$ using the classical Riemann-Siegel formula. The methodology involved the sub-division of the Riemann-Siegel formula into sequences of quadratic Gauss sums of various lengths N. Such sums can be computed rapidly, in $O(log(N))$ operations, using a standard recursive scheme. Subsequently, the PI developed a similar algorithm for $Z(t)$ with an $O\left((t/\varepsilon_t)^{1/3}\{log(t)\}^2\right)$ operational count, accurate to a tolerance scale ε_t in the relative error. Although constructively analogous, the sub-division into quadratic sums was applied to a new asymptotic formula for $Z(t)$, giving the new algorithm an original formulation. This report presents a summary of some significant advancement of these ideas. The new asymptotic formulation for $Z(t)$ can be extended beyond quadratic Gauss sums, to encompass more complex sub-sequences of generalised, m^{th}-order, Gauss sums of progressively increasing length. This is attractive, computationally, because as the length of the respective sums increases, the number needed to express $Z(t)$ falls away. If the above recursive scheme could be adapted to compute m^{th}-order Gauss sums in an analogous manner to the quadratic case, the overall computational demand for calculating $Z(t)$ would drop substantially. The main aim of this project was to develop public, open access, source code, founded upon these theoretical insights, capable of delivering significant computational benefits to the wider mathematical and coding community. The report focuses mainly on the source code developed for the cubic $m = 3$ sum case. The specific parameterisation of the cubic sums makes them amenable to rapid computation, utilising a similar, but more powerful, recursive scheme to that implemented for the quadratic case. The net result is a computational algorithm for $Z(t)$ with a reduced $O\left((t/\varepsilon_t)^{1/4}\{log(t)\}^3\right)$ operational count for $t \in [10^{23-32}]$, to high accuracy. Sample computations demonstrate practical support to these findings.

Keywords: The Hardy function, generalized Gauss sums, Riemann zeta computations

Mathematical Foundations

The starting point for any algorithm designed to facilitate the computation of the Hardy Function $Z(t)$ is a suitable asymptotic approximation valid for large t. The classical approximation, utilised for many years, is the famous Riemann-Siegel formula (RSF)

$$Z(t) = 2\sum_{N=1}^{N_t} \frac{cos\big(\theta(t) - tlog(N)\big)}{\sqrt{N}} + O\big(t^{-1/4}\big). \tag{1}$$

Here $N_t = \left\lfloor \sqrt{t/2\pi} \right\rfloor$ and the $O\big(t^{-1/4}\big)$ term is easily computed from standard functions [1]. The RSF is an $O\big(t^{1/2}\big)$ operational method of computation (since the main term consists of $O\big(t^{1/2}\big)$ summands) and is perfectly adequate for moderately large values of $t \in [10^2, 10^{15}]$.

Beyond this range more efficient computational methods [2] are necessary. For the code developed under this project an algebraically more complex, hybrid, asymptotic formula discovered by the PI [4, 5] is utilised. This is given by

$$Z(t) = 2\sum_{N=1}^{N_C(m)} \frac{cos\big(\theta(t) - tlog(N)\big)}{\sqrt{N}} + ZP(t,m) + O(\varepsilon_t), \tag{2}$$

where $N_C(m) = \left\lfloor (t/\varepsilon_t)^{1/(m+1)} log(t)/\sqrt{\pi} \right\rfloor$, $m = 2,3,\ldots$ and $\varepsilon_t \ll 1$ is a user prescribed tolerance scale, bounding the relative error in the calculation. Notice that the first part of this approximation is formed by main term of the RSF, but the number of summands rapidly falls away as m is increased. The missing summands, lying between $[N_C(m), N_t]$, which form the bulk of the calculation, are replaced by a second partial sum $ZP(t,m)$, given by

$$ZP(t,m) = T_{a+\varepsilon} + \mathcal{H}(t)2\sqrt{2}\left\{\sum_{\substack{\alpha>a \\ odd}}^{a+b} \frac{\cos\left(\frac{t}{2}\left\{log(pc) + \frac{1}{pc} + 1\right\} + \frac{\pi}{8}\right)}{(\alpha^2 - a^2)^{\frac{1}{4}}}[1 + O(t^{-1/8})]\right\} +$$

$$\sum_{\substack{\alpha_E>a+b \\ step\ 4M_t^-(m)+4}}^{\alpha_C(m)} \frac{\mathcal{H}(t)2\sqrt{2}}{(\alpha_E^2 - a^2)^{\frac{1}{4}}} \mathrm{Re}\left\{e^{i\omega^+(pc)} S_{M_t^-(m)}(\Phi_{1..m}^+) + e^{i\omega^-(pc)} S_{M_t^-(m)}(\Phi_{1..m}^-)\right\}[1 + O(\varepsilon_t)]. \tag{3}$$

Here $a = \sqrt{8t/\pi}$, $b = 2\lfloor t^{1/(m+2)} \rfloor$, $\alpha \in 2\mathbb{N}+1$, $\alpha_E \in 2\mathbb{N}$, $\mathcal{H}(t) \approx 1 + \frac{1}{32t^2}$, $\alpha_C(m) \approx \frac{(\varepsilon_t t^m)^{1/(m+1)}}{\sqrt{\pi} log(t)}$ and $pc(\alpha) = 2(\alpha/a)^2\left[1 + (1 - (a/\alpha)^2)^{1/2}\right] - 1 \in [1, \infty)$. The *significant* part of this formula, from a computational point of view, are the generalised Gauss sums that make up the real part of the second term. Generalised Gauss sums are defined by

$$S_{M_t^-}(\Phi_{1..m}) = \sum_{k=0}^{M_t^-} exp[2\pi i(\Phi_1 k + \Phi_2 k^2 + \Phi_3 k^3 + \cdots + \Phi_m k^m)]. \tag{4}$$

The number of summands in (3) is defined (to the nearest integer) by

$$M_t^-(pc, m) = \frac{1}{2\sqrt{\pi}}\left[\frac{\varepsilon_t (pc^2 - 1)^m t^{(m-1)/2}}{2^{2(m-1)} pc^{(3m-1)/2}}\right]^{1/(m+1)}. \tag{5}$$

Notice that this number *increases* with m, which means that the number of independent Gauss sums in (3) *falls off* with m, which potentially increases the speed of computing $ZP(t,m)$.

The crux of the algorithms developed for this project is a methodology (see below) for the rapid computation of Gauss sums (4) using $O\left(log(M_t^-)\right)$ operations for the cases $m = 3$ cubic and $m = 4$ quartic respectively. This is a vast saving compared to computing (4) longhand and drastically reduces the number of operations required to evaluate second partial sum (3). When combined with (2) the resulting algorithms allow for the computation of $Z(t)$ using only $O\left(t^{1/4}\left(log(t)\right)^3\right)$ standard operations [6], which is much more efficient than the RSF, or indeed the previous state of the art $O\left(t^{1/3}\left(log(t)\right)^\kappa\right)$ algorithm developed by [2] (which in essence is a method utilising $m = 2$ quadratic sums). Ideally one would like to apply this rapid computational procedure to Gauss sums defined for any value of m, which would produce an $O\left(t^{1/(m+1)}\left(log(t)\right)^\kappa\right)$ operational algorithm for calculating $Z(t)$ itself. Unfortunately, technical difficulties rapidly reduce the utility of the procedure beyond the quartic Gauss sum case.

Since most of the source code developed under this project and posted in the GitHub repository

https://github.com/dml2391/Hardy-function-fastcodes

utilises cubic Gauss sums, it is worth stating the initial values of the parameters Φ_1, Φ_2 and Φ_3 which define (4).

$$\Phi_1^{\pm}(pc) = \frac{1}{2(pc-1)} \pm \frac{a}{4\sqrt{pc}} \pm \frac{pc^{3/2}}{a(pc-1)^3},$$

$$\Phi_2^{\pm}(pc) = \frac{1}{2(pc-1)} \pm \frac{2pc^{3/2}}{a(pc-1)^3},$$

$$\Phi_3^{\pm}(pc) = \pm \frac{4pc^{3/2}}{3a(pc-1)^3}. \qquad (6a,b,c)$$

Simple symmetry properties and complex conjugation enable any values of the parameters $\Phi_1^{\pm}(pc)$ and $\Phi_2^{\pm}(pc)$ to be shifted, so that $\Phi_1 \in (-1/2, 1/2)$ and $\Phi_2 \in (0, 1/4)$ respectively. The important point to note is that the value of $\Phi_3 = O\left(t^{-1/2}\right)$ is always very much smaller than either Φ_1 or Φ_2. This observation is crucial to what follows.

Methodology for the Rapid Computation of Gauss sums with small higher order coefficients $\boldsymbol{\Phi_j = O\left(t^{1-j/2}\right)}$

Using a variation of the Euler-Maclaurin Summation method [e.g. 8], it is possible formulate an exact expression for any Gauss sum in the form

$$S_{L_j}\left(\Phi_{1..m:j}\right) = \frac{e^{-\mathrm{i}\pi/4}}{\sqrt{2\Phi_{2:j}}} \sum_{n=H(\Phi_1)}^{\lfloor\xi\rfloor} e^{2\pi \mathrm{i} g(c)} + small\ corrections. \qquad (7)$$

The "*small corrections*" are algebraically complex $O(1)$ terms, but relatively simple first order estimates are easy to compute. (They are derived in [6] and specifically listed in the detailed coding documentation [7], under subroutine Q, posted the repository.) The subscript value $j = 1$ denotes the parameters of the initial Gauss sum to be estimated. The secondary sum on the right-hand side of (7) consists of $L_{j+1} = \lfloor\xi\rfloor \approx \sum_{q=2}^{m} q\Phi_q L_j^{q-1} \approx 2\Phi_2 L_j < L_j$ summands, since $2\Phi_2 < 1/2$. Hence it is a shorter sum than the left-hand side. The exponent term in (7) is defined by

$$g(c) = nc - \sum_{q=1}^{m} \Phi_q c^q \quad with \quad n - \Phi_1 = \sum_{q=2}^{m} q\Phi_q c^{q-1}. \qquad (8a,b)$$

Now *provided* the saddle point c, satisfying (8b), can be expressed in closed form as a function of the parameters n and $\Phi_{1..m:j}$, then the exponent (8a) would reduce to a polynomial of degree m in n, with a new set of coefficients $\Phi_{1..m:j+1}$. In which case (7) would generate the relationship

$$S_{L_j}\left(\Phi_{1..m:j}\right) = \frac{e^{-\mathrm{i}\pi/4}}{\sqrt{2\Phi_{2:j}}} S_{L_{j+1}}\left(\Phi_{1..m:j+1}\right) + small\ corrections, \qquad (9)$$

linking the longer Gauss sum on the left-hand side to a shorter Gauss sum on the right-hand side. In isolation, this result is of little import, since L_{j+1} would still be a large integer. However, if (9) is applied recursively, a hierarchal chain of Gauss sums is formed, connecting the original Gauss sum of length L_1 to a vastly smaller, kernel Gauss sum, of length K, via at most $j \approx log(L_1/K)/log(2)$ links. Hence, computation of the kernel sum, followed by the recursive substitution of each intermediate sum value in the hierarchal chain leads, after $O(K\ log(L_1/K)/log(2))$ operations, to an estimate for the initial Gauss sum as desired. Since $K \ll L_1$, this provides a vast computational saving compared to $O(L_1)$ operations needed to compute the initial sum directly.

For the quadratic case $m = 2$ this recursive scheme can be applied *ad infinitum*. That is because the saddle point solution of (8b) is linear in n, meaning that (8a) is just another quadratic [9] in n. For cubic sums $m = 3$ and above this is not the case. However, because in this instance the initial parameter value of Φ_3 is so very small, the sums themselves remain sufficiently quadratic in character for the recursive scheme (9) to be applied multiple times. Specifically, one defines

$$c_{m-1} = \sum_{q=1}^{m-1} C_q\left(\Phi_{2,3,\dots q+1}\right)\frac{(q+1)y^q}{x^{2q-1}}, \tag{10}$$

where $x = 2\Phi_2 \in (0, 1/2)$ and $y = n - \Phi_1 > 0$, to be the solution to (8b) truncated at the m^{th}-order in y. The constant coefficients $C_q\left(\Phi_{2,3,\dots q+1}\right)$ are complicated functions of the Φ_j, but are relatively easy to calculate using any reasonably efficient computer algebra package. See the Mathematical Appendix of [6]. Substituting (10) into (8a) one obtains the approximation

$$g_m(c_m) = yc_m - \sum_{q=2}^{m} \Phi_q c_m^q = \sum_{q=1}^{m-1} \frac{C_q y^{q+1}}{x^{2q-1}} + \sum_{q=m}^{\infty} E_q\,, \tag{11}$$

for the exponent $g(c)$, truncated at the $(m+1)^{th}$- order in y. The error, given by the secondary sum in (11), is usually small enough to be neglected. In which case (9) links two Gauss sums of the same order. Specifically, for the cubic case the parameters of the j^{th} and $(j-1)^{th}$ sums satisfy

$$\Phi_{0,j} = \frac{\Phi_{1,j-1}^2}{2x_{j-1}} + \frac{\Phi_{1,j-1}^3\Phi_{3,j-1}}{\left(x_{j-1}\right)^3}, \qquad \Phi_{1,j} = -\frac{\Phi_{1,j-1}}{x_{j-1}} - \frac{3\Phi_{1,j-1}^2\Phi_{3,j-1}}{\left(x_{j-1}\right)^3},$$

$$\Phi_{2,j} = \frac{x_j}{2} = \frac{1}{2x_{j-1}} + \frac{3\Phi_{1,j-1}\Phi_{3,j-1}}{\left(x_{j-1}\right)^3}, \qquad \Phi_{3,j} = -\frac{\Phi_{3,j-1}}{\left(x_{j-1}\right)^3}\,. \tag{12}$$

As each intermediate Gauss sum in the hierarchal chain is set up, the value of the secondary sum in (11) is monitored. If this error term exceeds some user prescribed threshold, this indicates that the m^{th}- order approximation solution to (8b) is insufficiently accurate. So instead, the $(m+1)^{th}$- order approximation is substituted. Hence a cubic sum would now move up to become a quartic sum. If the new error term lies below the threshold, then the computation moves on to the next link in the hierarchal chain. If not, then the order is increased again and again, until the error criterion is satisfied. This procedure continues until the length of the jth sum of the chain $L_j < K$ which defines the length of the kernel sum (K is a "cut-off" integer derived from the user prescribed values of t and ε_t), at which point the hierarchal chain is complete. The kernel sum is then computed exactly and the resulting value iterated back up the hierarchal chain to establish an estimate for the initial Gauss sum as desired. These operations form the basis of the following algorithm.

Algorithm MGS *An algorithm for the recursive computation of a generalized m^{th}-order Gauss Sum $S_L(\Phi_{1..m})$, with $L \gg 1$, $m \geq 3$ and coefficients $\Phi_{3..m}$ satisfying asymptotical small conditions such as (6c) or extensions thereof.*

Replace $S_L(\Phi_{1..m})$ by $S_{L_1}^{s_1}\left(\Phi_{1..m_1,1}\right)$ so that $\Phi_{1..m_1,1} \in (-1/2\,, 1/2)$, using symmetry conditions. Let $s_j = \pm 1$ to indicate if the complex conjugate sum is to be used.

Fix positive integer parameter $K > 30$ and small real $\epsilon > 0$ (typically $\epsilon \leq 10^{-2}$ is suitable). Let $\omega = e^{i\pi/4}$.

Compute for $j = 2, 3, \ldots$

a) *Set* $m_j = m_{j-1}$. *Compute* $L_j = \left\lfloor \sum_{q=2}^{m_j} q\Phi_q L_{j-1}^{q-1} \right\rfloor$. *Set* $y_{max} = L_j - \Phi_{1,j-1}$.

b) *Compute* $\Phi_{0\ldots m_j,j}$ *from* $\Phi_{0\ldots m_j,j-1}$, *using (12).*

c) $x_j = 2\Phi_{2,j}$; $x_j = x_j - \lfloor x_j \rceil$; $\Phi_{2,j} = x_j/2$; $s_j = \mathrm{sgn}(-x_j)$; *Adjust each* $\Phi_{1,2\ldots m_j,j}$ *using symmetry conditions to ensure they all lie in the range* $(-1/2\,, 1/2]$ *and multiply each one by* $\mathrm{sgn}(x_j)$, *so in particular* $\Phi_{2,j} > 0$.

d) *Compute the max* $\left|2\pi E_{m_j}\right|$.

e) *If* $\left|2\pi E_{m_j}\right| < \epsilon$ *carry on to f). Otherwise set* $m_j = m_j + 1$, $\Phi_{m_j,j-1} = 0$ *and repeat b)-e) using the next order extension of (12).*

f) *If* $L_{j+1} < L_j < K$ *and* $L_j > \lceil \Phi_{1,j-1} \rceil$, *set* $j_K = j$, *otherwise set* $j_K = j - 1$. *If* $L_{j+1} < 0 < K < L_j$, *set* $j_K = j$. *If* $K < L_j < L_{j+1}$, *set* $j_K = j$. *If none of these, repeat a)-f).*

g) *Compute the sum* $S_{L_{j_K}}^{s_{j_K}}\left(\Phi_{1..m_{j_K},j_K}\right)$ *exactly. N.B. if* $\Phi_{1,lj_K-1} > 0$ *the summands commence at* $n = 1$ *rather than* $n = 0$, *to account for the step function in (7). This adjustment must also be after each iteration prescribed in h) below.*

h) *From this starting point compute, for* $j = j_K - 1, j_K - 2, \ldots, 1$, *the following iterations*

$$S_{L_j}^{s_j}\left(\Phi_{1..m_j,j}\right) = \left\{ \frac{e^{2\pi \mathrm{i}\Phi_{0,j+1}} S_{L_{j+1}}^{s_{j+1}}\left(\Phi_{1..m_{j+1},j+1}\right)}{\omega\left|x_j\right|^{1/2}} + \textit{small corrections} \right\}^{s_j}.$$

i) *The final iteration* $S_{L_1}^{s_1}(\Phi_{1..m_1,1})$ *gives an estimate to* $S_L(\Phi_{1..m}) = S_{L_1}^{s_1}(\Phi_{1..m_1,1}) + \delta(K, L, \Phi_{1..m})$, *where* $\delta(K, L, \Phi_{1..m})$ *is the error term.*

Table 1 shows an illustration of the workings of ***Algorithm MGS*** for a cubic Gauss sum with $t = 10^{28}$ and other parameters derived from (5) and (6). The tolerance scale for the relative error was set at $\varepsilon_t = 0.02$. This bounds the size of the relative error in the final estimate, deriving from $\delta(K, L, \Phi_{1..m})$. The algorithm first creates a hierarchal chain of sums with five links down to a kernel of length $K = 273$. Notice in the first link of the chain the order new sum remains cubic. But for the next link the error term in (11) exceeds the designated threshold $\epsilon = 10^{-2}$, which prompts the algorithm to increase the order of the sums from three to four. A further increase in order, from four to six, is necessary to create the final link. Another link would still be possible, creating a kernel sum of length $L_6 \approx 10$. However, it is best to avoid making the kernel sum too small since its magnitude will start to become comparable with the "*small corrections*" in (9), and any discrepancy in computing the latter will manifest itself in a relatively large error in the final estimate, potentially exceeding ε_t. The kernel sum is now computed, and this value used to calculate estimates for all the sums in the hierarchal chain. Notice that whilst the absolute value of the error in each estimate increases with each iteration back up the chain, the corresponding relative error quickly stabilises to a value of $0.0017 \ll \varepsilon_t = 0.02$ as expected. The bulk of the relative error always

l	L_l	m_l	$Generalized\ Gauss\ sum\ factors\ \Phi_{0-m_l}$	s_l
1	2100836	3	$\Phi_0 = 0.0$ $\Phi_1 = 4.577951577 \times 10^{-1}$ $\Phi_2 = 4.149941700 \times 10^{-2}$ $\Phi_3 = 2.251742112 \times 10^{-16}$	-1
2	174367	3	$\Phi_0 = 1.262526209 \times 10^{0}$ $\Phi_1 = 4.843182053 \times 10^{-1}$ $\Phi_2 = 2.418101414 \times 10^{-2}$ $\Phi_3 = -3.93824450 \times 10^{-13}$	-1
3	8433	4	$\Phi_0 = 2.42508567 \times 10^{0}$ $\Phi_1 = -4.85568797 \times 10^{-1}$ $\Phi_2 = 1.613103826 \times 10^{-1}$ $\Phi_3 = -3.481682424 \times 10^{-9}$ $\Phi_4 = -2.63812477 \times 10^{-18}$	$+1$
4	2719	4	$\Phi_0 = 3.654090031 \times 10^{-1}$ $\Phi_1 = 5.076154596 \times 10^{-3}$ $\Phi_2 = 4.980739785 \times 10^{-2}$ $\Phi_3 = 1.036841408 \times 10^{-7}$ $\Phi_4 = 1.585086768 \times 10^{-14}$	-1
5	273	6	$\Phi_0 = 1.293349449 \times 10^{-4}$ $\Phi_1 = -5.09578458 \times 10^{-2}$ $\Phi_2 = 1.933629021 \times 10^{-2}$ $\Phi_3 = -1.048917155 \times 10^{-4}$ $\Phi_4 = 4.770974356 \times 10^{-9}$ $\Phi_5 = -2.89012763 \times 10^{-13}$ $\Phi_6 = 2.042123848 \times 10^{-17}$	-1
l	Estimate of $S_{L_l}^{s_l}(\Phi_{1..m_l,l})$	Exact Value $S_{L_l}^{s_l}(\Phi_{1..m_l,l})$	$\lvert error \rvert$	Relative error
5	As exact value	1.57504 + 14.86040i	0.0	0.0
4	38.69952 − 29.34762i	38.73012 − 29.23127i	0.1203	0.0025
3	55.47567 + 65.28449i	55.34872 + 65.34816i	0.1420	0.0017
2	−361.23492 − 148.67401i	−361.27895 − 148.02249i	0.6530	0.0017
1	−421.06045 + 1288.25289i	−422.86094 + 1286.81109i	2.3066	0.0017

Table 1. Illustration of the reduction and computation of an initial 3^{rd}- order cubic Gaussian sum achieved using ***Algorithm MGS***. The initial sum is prescribed by the parameter values: $t = 10^{28}$, pivot integer $ae = 310304270951266.0 \equiv pc = 13.048362$ ....length L_1 set from (5) and $\varepsilon_t = 0.02$. The algorithm creates a hierarchal chain of five sums, finally reaching a 6^{th}- order kernel sum of length $L_5 = 273$. The kernel $Z(t)$, via equations (2 & 3). Since this is a vastly larger calculation, it is obviously desirable to keep the operational expenditure down to an absolute minimum. For the Hardy function is then computed exactly and the value iterated back up the chain to produce the desired estimate.

accrues with the first iteration and always arises from errors occurring in the evaluation of the "$small\ corrections$". In principle these can be computed to arbitrary accuracy, but this involves additional operational expenditure. If ***Algorithm MGS*** was being employed in isolation, this would be of little or no consequence. However, in this application ***Algorithm MGS*** must be engaged millions of times in order compute the final value of $ZP(t,m)$ via (3), and any additional operational expenditure must be kept to a minimum. This trade-off between operational expenditure and computational accuracy is resolved by presetting the cut-off integer K from the user prescribed variables t and ε_t, to ensure the relative error in the calculation of $Z(t)$ always lies below the designated tolerance scale.

Sample Computations of $Z(t)$ for large t values

Utilising ***Algorithm MGS*** it is now possible to develop codes to compute the Hardy function from equations (2 & 3) much more efficiently than previously possible. The main source code developed under this project is for the cubic case with $m = 3$. Table 2 shows some $Z(t)$ calculations carried out at some very particular t values where $|Z(t)|$ is unusually large. Large $|Z(t)|$ are of interest theoretically, since hypothetical violations of the famous Riemann Hypothesis would manifest themselves in the immediate vicinity of a large peak in the value of $|Z(t)|$. But in this instance, they provide a useful practical testing ground for the repository source code, since the largest *absolute* errors (as opposed to the relative error) in the $Z(t)$ calculations will be found at these points. Table 2 shows the results of some of these calculations for values of $t > 10^{28}$, with the relative error tolerance scale set to $\varepsilon_t = 0.005$ throughout.

Those jobs in the table for $t \leq 3.96 \times 10^{30}$ were run on ARCHER2 using four nodes. Typical run times ranged from 1.4 to 9.2 hours (5.5-36.8 Computational Resources CUs, or 704-4710 Core Hours CPUhs respectively). The larger jobs, those spanning $t \in [6.08 \times 10^{30}, 5.00 \times 10^{31}]$, were run using 8 nodes and took between 4.7 and 10.5 hours (38-84 CUs or 4864-10752 CPUhs). The three largest jobs in the table were run, partly, using the resources made available during the ARCHER2 capability days staged in September 2024. These sessions allowed users free access to many nodes for the submission of large-scale jobs, to test out their respective code's performance to the maximum. The jobs shown in the table utilised 1024 nodes for one hour, but they did not run quite as fast as anticipated and the calculations had to be completed subsequently using a standard 8 node setting. The capability days proved very useful for the code development of this project, highlighting certain small, easily correctable, errors in the restart procedures that only manifested themselves at the highest values of t.

The accuracy of the computations is highlighted by comparison with similar calculations to be found in the published databases in [3] and [10] and is extremely high. All the relative errors are less than 10^{-5}, five hundred times less than the prescribed relative error tolerance scale $\varepsilon_t = 0.005$. The absolute errors are typically good to one or two decimal places, which is excellent when considering they are at their maximum when $|Z(t)|$ is large. For most t values the errors will be much smaller than this. Of course, if a user requires a higher degree of accuracy in their calculations, then this can be achieved by specifying a smaller tolerance scale ε_t. This will produce a more accurate run, albeit for a slight increase in run time. (Other aspects of the operational run time of the cubic code are illustrated in the one-page summary section of the accompanying final report on this project.)

t value	$Z(t)$ Cubic $t^{1/4}$ code from this project	$Z(t)$ $t^{1/3}$ algorithm of [2, 7]; $\pm 5 \times 10^{-4}$	Relative error $\lvert\varepsilon\rvert$
2.059936512320112518074691006892E28	3803.86727	3803.86539	4.30×10^{-7}
2.483871744715102768284803284373E28	−1517.12399	−1517.11159	8.17×10^{-6}
3.177469531676391818363765436511E28	−9549.91673	−9549.88868	2.93×10^{-6}
4.670914185466097236850548903274E28	9587.87895	9587.85455	2.54×10^{-6}
1.083566071010277256157201115 3186E29	7297.78564	7297.75658	3.98×10^{-6}
2.892860767193253077183805 4905026E29	10907.57778	10907.55189	2.37×10^{-6}
5.521664100099312888868086 3234651E29	−13541.70579	−13541.67744	2.09×10^{-6}
6.981562889715199161359429 4046033E29	11187.59076	11187.56821	2.02×10^{-6}
8.036257285923443631238142 1877820E29	10282.41019	10282.40085	9.13×10^{-7}
1.420608056968699501169509 00347991E30	5045.59913	5045.59024	1.76×10^{-6}
1.907915287180786223131860 60719755E30	10242.78033	10242.78831	7.78×10^{-7}
2.409972808744819410276834 55628022E30	−9268.18171	−9268.20237	2.23×10^{-6}
3.805475614378624044873696 32961007E30	4547.78163	4547.80731	5.64×10^{-6}
3.962231703483297661331732 96323307E30	−7483.25069	−7483.25629	7.49×10^{-7}
6.083028695276545807063248 34685591E30	9729.73977	9729.76572	2.66×10^{-6}
6.632378187823588974002457 91070660E30	12010.61542	12010.63042	1.24×10^{-6}
9.832284408046499500622869 54013174E30	−10123.63767	−10123.62451	1.30×10^{-6}
1.472969364244678383540127 734056387E31	−7707.03675	−7706.96360	9.49×10^{-6}
3.557586000421470624922724 880597724E31	13337.02398	13337.12691	7.71×10^{-6}
3.924676458989430915525116 928410405E31	16244.39356	16244.47429	4.96×10^{-6}
4.589001484792927188496155 886462819E31	9967.75442	9967.80244	4.80×10^{-6}
5.005475723107396211588045 467161740E31	−10621.48019	−10621.51326	3.12×10^{-6}
† 9.066617388021913893282064 021886284E31	15601.555	15601.620	4.14×10^{-6}
† 2.860941159455196369121404 6991910957E32	16093.291	16093.351	3.73×10^{-6}
† 3.106788336290839656675405 7659368205E32	16874.148	16874.202	3.20×10^{-6}

Table 2. Illustrative calculations of the Hardy function for $t > 10^{28}$ showing the values obtained using the cubic code developed under this eCSE11-7 project, compared to the corresponding values listed in [3, 10]. At these very particular t values the modulus of $Z(t)$ is unusually large. The relative error tolerance scale was set to $\varepsilon_t = 0.005$ for all the calculations. † These calculations were carried out using the extended resources made available during ARCHER2 2024 capability days.

Finally, the repository contains code based upon the $m = 4$ quartic Gauss sum representation of (3). In theory this should yield an $O\left(t^{1/5}\left(log(t)\right)^3\right)$ algorithm, very much faster than the cubic case. However, certain technical difficulties render this impossible. The problem concerns the size of the cubic parameter Φ_3 in the relation to the sum length L as one moves down the hierarchal chain. The cubic parameter increases (as can be seen from Table 1) faster than the sum length declines. At some point in the chain, a condition is reached when $3\Phi_3 L > 2\Phi_2$ for $\Phi_3 < 0$. At this point the solutions to saddle point equation (8b) cease to be approximately linear and the recursive relationship (9) breaks down. It is still possible to formulate an expression for the kernel sum by finding numerical, as opposed to analytical, solutions to (8b). But this comes at a significant operational cost, slowing down the final algorithm. The upshot is that the quartic code, as it stands, simply represents a somewhat faster version of the cubic code (it typically runs 25% faster than the corresponding cubic code for $t \leq 10^{28}$) rather than the order of magnitude improvement promised if this problem could be overcome and (9) extended indefinitely. The repository also contains a third code, which is a multi-value version of the cubic code discussed above. Whilst the cubic code calculates one value of $Z(t)$ at a specified value of t, the multi-value code computes up to 15 separate $Z(t)$ values simultaneously, with the cavate that the t points must lie close together. Typically, an interval spacing of ± 0.01 is adopted, around a user specified central t value. The extra operational cost of running this multi-valued code is only some 20% over and above running the single value code, rather than the fifteen times one might expect to be the case. This offers a user to a rapid means of estimating the position of one of the famous *zeta-zeros*, which are the most important feature of the Hardy function. This information in turn can be used to facilitate Riemann Hypothesis validation calculations across short intervals of t.

This work was funded under the embedded CSE programme of the ARCHER2 UK National Supercomputing Service (http://www.archer2.ac.uk).

References

[1] Edwards, H. M. 1974 *Riemann's zeta function,* Dover Publications, INC. New York.

[2] Hiary G. A. 2011 Fast methods to compute the Riemann zeta function. *Annals of Mathematics* **174**, 891-946.

[3] Hiary G. A. 2017 *Fast methods to compute the Riemann zeta function*, **https://people.math.osu.edu/hiary.1/fastmethods.html**

[4] Lewis, D. M. 2015 The development of a hybrid asymptotic expansion for the Hardy function $Z(t)$, consisting $[2\sqrt{2}-2]\sqrt{t/2\pi}$ main terms, some 17% less than the celebrated Riemann-Siegel formula, http://arxiv.org/abs/1502.06903

[5] Lewis, D. M. 2017 A computational algorithm for the Hardy function $Z(t)$, utilising sub-sequences of generalised quadratic Gauss sums, with an overall complexity operational complexity $O\left((t/\varepsilon_t)^{1/3}\{log(t)\}^{2+o(1)}\right)$, http://arxiv.org/abs/1711.01928

[6] Lewis, D. M. & Brereton, A. R. 2025 A computational algorithm for the Hardy function $Z(t)$, utilising sub-sequences of generalised cubic Gauss sums, with an overall operational complexity $O\left((t/\varepsilon_t)^{1/4}\{log(t)\}^3\right)$ for $t \in [10^{15}, 10^{35}]$, manuscript in preparation for submission to Open Mathematics, see PI.

[7] Lewis, D. M. 2025 Hardy Code-Detailed Documentation, GitHub repository https://github.com/dml2391/Hardy-function-fastcodes

[8] Olver, F. W. J., Lozier, D. W., Boisvert, R. F. & Clark, C. W. 2010 *NIST Handbook of Mathematical Functions,* Cambridge University Press.

[9] Paris, R. B. 2008 An Asymptotic Expansion for the Generalised Quadratic Gauss Sum. *Applied Mathematical Sciences* **2**(12), 577-592.

[10] Tihanyi, N. 2019 Numerical computing of extremely large values of the Riemann-Siegel Z-function. PhD thesis, Doctoral School of Informatics, Numeric and Symbolic Computations, Eötvös Loŕand University.