DETAILED DCUMENTATION OF THE VARIOUS SUBROUTINES MAKING UP THE "ZETA14CUBIC.F90" CODE

Fortran Subroutines

1. Name **ERF**

Purpose: To calculate the complex error function value

$$\mathrm{erf}\left(ze^{-\mathrm{i}\pi/4}\right) := \frac{2}{\sqrt{\pi}}\int_0^{ze^{-\mathrm{i}\pi/4}} e^{-t^2}\,dt,$$

for $z \in \mathbb{R}$.

Outline/Method: Although most Fortran compilers support two intrinsic routines to calculate both the error function $\mathrm{erf}(x)$ and it compliment $\mathrm{erfc}(x) = 1 - \mathrm{erf}(x)$ for real x, they do not support calculations for complex values. For this work error function values along the $e^{-\mathrm{i}\pi/4}$ axis are necessary, hence the need for this purpose built routine. For small $|z|$ the standard series representation is used (AS eq. 7.1.5), whilst for large $|z|$ the asymptotic expansion (AS eq. 7.1.23) for the complimentary function is employed. However, within the range $4/5 < |z| < 21/8$ neither of these two series converges very efficiently. Calculations for $|z|$ within this range are carried out by means of seven Taylor series expansions centred on $|z| = 1.0, 1.25, 1.5, 1.75, 2.0, 2.25$ & 2.5 respectively. Then for any $|z| \in [4/5\,, 21/8]$ the series centred on the point closest to $|z|$ is used to compute $\mathrm{erf}\left(|z|e^{-\mathrm{i}\pi/4}\right)$. The symmetry relation (AS 7.1.9) is used if $z < 0$. The use of Taylor series expansions in this way ensures that $\mathrm{erf}\left(|z|e^{-\mathrm{i}\pi/4}\right)$ can be computed to an accuracy better than 6 decimal places using a maximum of $N = 6$ terms for any of the expansions adopted.

Inputs: N (short integer), Z (d.p. real)

Ranges: $0 \le N \le 6$, $Z \in (-\infty, \infty)$. Default value $N = 6$.

Outputs: ER (d.p. complex variable), the value of $\mathrm{erf}\left(ze^{-\mathrm{i}\pi/4}\right)$.

Common Blocks:

/PARS4/: $\pi, \sqrt{\pi}, 2\pi, -2\pi$ (d.p. reals); $e^{\mathrm{i}\pi/4}$ (d.p. complex).

/PARMS/: CEF, ZLAR, ZSMAL (d.p. complex arrays).

(These arrays store the various complex coefficients for the seven Taylor series, the asymptotic expansion and the standard series representation of the error function. All standard constants set when the code is initialised.)

2. Name **LGAM**

Purpose: To calculate the logarithm of the Gamma Function

$$\mathrm{Log}\big(\Gamma(x)\big) := \mathrm{Log}\left\{\int_0^\infty t^{x-1}e^{-t}\,dt\right\},$$

for $x > 0$.

Outline/Method: This uses a variant of Stirling's Formula (AS 6.1.37) for the Gamma function of the form

$$\Gamma(z+1) = (z+11/2)^{-(z+1/2)}e^{-(z+11/2)}\sqrt{2\pi}\left[c_0 + \sum_{m=1}^{6}\frac{c_m}{z+m} + \epsilon\right]$$

for $z \in \mathbb{C}$, $Re(z) > 0$. For an appropriate choice of coefficients c_m, the relative error $|\epsilon| < 2 \times 10^{-10}$ (Press et. al.). This formula is used to calculate $\mathrm{Log}\{\Gamma(z+1)\}$. Then $\mathrm{Log}\{\Gamma(z)\} = \mathrm{Log}\{\Gamma(z+1)/z\}$. This routine is used, in the main, to calculate values of the factorial function $n!$ and the binomial coefficient, which get very large for comparatively modest integer values. There is an intrinsic function that will calculate $\mathrm{Log}(n!)$, but this particular subroutine can easily be adapted to accept complex inputs and so is more flexible.

Input: GA (d.p. real)

Range: GA $\in (0, \infty)$.

Output: XX (d.p. real), the value of $\mathrm{Log}\big(\Gamma(\mathrm{GA})\big)$

Common Blocks:

/PARS4/: $\pi, \sqrt{\pi}, 2\pi, -2\pi$ (d.p. reals); $e^{\mathrm{i}\pi/4}$ (d.p. complex).

/PARS8/: GAMCOF (d.p. real array).

(The array GAMCOF stores the values of the real coefficients c_{0-6}. All the coefficients are set when the code is initialised.)

3. Name **TER**

Purpose: To calculate a single term of the initial contribution to the hybrid formula for $Z(t)$

$$2\sqrt{2}\sum_{\substack{\alpha>a\\ odd}}^{a+M}\frac{\cos\left(\frac{t}{2}\left\{log(pc)+\frac{1}{pc}+1\right\}+\frac{\pi}{8}\right)}{(\alpha^2-a^2)^{\frac{1}{4}}}, \tag{1}$$

where $a \coloneqq \sqrt{8t/\pi}$, $\alpha \in [a, a+M]$, $\alpha = 2\mathbb{N}+1$, $M = \lfloor t^{1/5} \rfloor$ *or* $\lceil t^{1/5} \rceil - 1$ so that $M \in 2\mathbb{N}$ and $pc = 2s^2\left(1+\sqrt{1-1/s^2}\right)-1$ with $s = \alpha/a$ (cf. Lewis & Brereton 2024, eq. 62).

Outline/Method: From a computational point of view, it is more efficient to rewrite a single term of this sum in the form:

$$\frac{\cos\{t(b(b-s)+log(s+b))+t+\pi/8\}}{\sqrt{b}},$$

where, $k \coloneqq \alpha$, $s = k/a$ & $b = \sqrt{s^2-1}$, which reduces the calculation of the cosine phase to just basic arithmetic operations, square roots and a log term (no exponentiation).

Inputs: k (long integer), ain (d.p. real), $yphase$ (d.p. real), t (d.p. real)

Ranges: $k \in \left(a,\ a+t^{1/5}\right)$ where $a = \sqrt{8t/\pi}$ and $k = 2\mathbb{N}+1$:

$ain = a^{-1} \in (0,\infty)$:

$yphase = t+\pi/8 \ \ (mod\ 2\pi) \in [0, 2\pi)$:

$t \in (0, 10^{38})$:

Output: $term$ (d.p. real), the value of an early term in the hybrid formula for $Z(t)$.

Notes: This simple looking routine has the potential for significant errors when t is large. The first problem is that the dynamic range of Fortran long integers is limited to $(-2^{64}+1) < k < (2^{64}-1)$. Since $max(k) \approx \ a+t^{1/5}$, this means that $t < 1.3 \times 10^{38}$, otherwise some of the k inputs will begin to exceed this dynamic range, resulting in errors. It is possible to replace k with an identical real variable (as done in other parts of the code), which would resolve this issue. The other problem lies in the phase calculation itself. The largest possible value of the coefficient $[b(b-s)+log(s+b)]\sim t^{-3/10}$, which $\Rightarrow t[b(b-s)+log(s+b)]\sim t^{7/10}$. For $t \approx 10^{38}$ this is a number around 10^{26}, which is small enough to compute the phase, mod 2π, accurately, provided the calculation is done in double precision. However, for $t > 10^{38}$ even a double precision Fortran variable will not include enough decimal places in its memory to avoid potentially catastrophic round-off errors. Summing up - **if the code is to be used for calculations of $Z(t)$ for $t > 10^{38}$, this subroutine needs to be re-written in C utilising routines from the Pari library, to avoid significant round off errors.**

4. Name **START**

Purpose: To calculate the value of first index $\alpha_1 \in 2\mathbb{N}+1$ that commences the sum constituting the initial contribution (1) to the hybrid formula for $Z(t)$. It also establishes the initial value of the sum itself, $zsum = T_{a+\varepsilon}$, a transition term (cf. Lewis & Brereton 2024, eq. 129) which augments equation (1).

In most instances, $\alpha_1 \coloneqq INT_O(a)+2$ (where $INT_O(x) \equiv$ the largest-odd integer less than or equal to $x \geq 1$) and $T_{a+\varepsilon} = 0$. Very occasionally the value of $a = \sqrt{8t/\pi}$ happens to lie very 'close' to either α_1 or $\alpha_1 - 2$. Specifically, the term 'close' here implies that either $\varepsilon = (\alpha_1 - a) < 3.2t^{-1/6}$ or $\varepsilon = \big(a - (\alpha_1 - 2)\big) < 3.2t^{-1/6}$. If either of these conditions are satisfied, then $T_{a+\varepsilon} = O\big(t^{-1/6}\big)$ and this term must be specifically calculated. $T_{a+\varepsilon}$ is defined in terms of simple integral which can be calculated numerically, using Gaussian quadrature. Most of the code in this routine is devoted to that numerical calculation, although it will be rarely employed in practice. If the first of these conditions on ε is satisfied, then $\alpha_1 \coloneqq INT_O(a)+4$.

Inputs: a (d.p. real), $t6$ (d.p. real)], $yphase$ (d.p. real)

Ranges: $a \in (0,\infty)$:

$t6 \coloneqq t^{1/6} \in [0,\infty)$:

$yphase \coloneqq t + \pi/8 \in [0, 2\pi)$, only needed if $T_{a+\varepsilon} \neq 0$.

Outputs: $zsum$ (d.p. real), the value of $T_{a+\varepsilon}$. In the vast majority of cases $zsum$ will be returned as zero.
$M2$ (long integer), the value α_1.

5. Name **ALPHASUM**

Purpose: Given a specific pivot integer $k \in 2\mathbb{N}$ and a Gauss sum length $M_t^- \in \mathbb{N}$, the initial aim of this routine is to compute the coefficients that define the generalized Gauss sums which, when combined together, produce a single summand constituting the main part of the hybrid formula. In this particular code the generalized Gauss sums are ***cubics***, with lengths commencing at $M_t^- = O\big(t^{1/6}\big)$, which increase with k up to $M_t^- = O\big(t^{1/3}\big)$ as the overall calculation proceeds. When expressed in terms of cubic Gauss sums, the hybrid formula consists of $O\big(t^{1/4}\big)$ summands. Specifically, a single summand is the result of the computation of

$$\text{Real Part of}\left\{e^{\mathrm{i}\omega^+(pc)}S_{M_t^-}(\Phi^+_{1..3}) + e^{\mathrm{i}\omega^-(pc)}S_{M_t^-}(\Phi^-_{1..3})\right\}, \tag{2}$$

where the cubic Gauss sums are defined by

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

In this instance, the respective coefficients are defined as follows:

$$\omega^{\pm}(pc) = -\frac{t}{2}\left(log(pc) + \frac{1}{pc} + 1\right) - \frac{\pi}{8} + \frac{\pi}{4}\left[\frac{1}{(pc-1)} \pm \frac{a}{\sqrt{pc}}\right] \pm \frac{\pi}{3a}\frac{pc^{3/2}}{(pc-1)^3},$$

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

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

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

The pivot integer $k \in 2\mathbb{N}$ fixes the value of the "portcullis" variable in the above expressions, since $pc \coloneqq 2s^2\left(1+\sqrt{1-1/s^2}\right) - 1$ with $s = k/a$. The first part of the routine computes and stores the above coefficients in the arrays 'initialcoeff' and 'initialcoeffcc' for $\Phi_i^{+}(pc)$ and $\Phi_i^{-}(pc)$ respectively. These arrays are then passed on to the two routines COMPUTARRS and MGAUSSSUM which do the actual computation of (3). The final part of this routine is to use the values for (3) found by MGAUSSSUM to compute the corresponding summand value (2). This is returned as the output variable 'pcsum'.

Inputs: MTM (long integer), rae (d.p. real)

Ranges: $MTM \in (0, \infty)$:

$rae \in [0, \infty)$: see notes.

Outputs: $pcsum$ (d.p. real), the value of the summand given by (2)

Common Blocks:

/PARS4/: $\pi, \sqrt{\pi}, 2\pi, -2\pi$ (d.p. reals); $e^{i\pi/4}$ (d.p. complex).

/PARS6/: t, a, ain, et, ht, Y all d.p. reals, K, ip, mc all short integers

Only a, $ain = a^{-1}$, K and mc are used directly in this routine. The other variables are needed in COMPUTARRS and MGAUSSSUM. K is the 'cut-off' integer set when the code is initialised. mc gives the degree of the initial generalized Gauss sums to be calculated. This code uses **cubic** Gauss sums, so $mc = 3$ throughout.

Notes: Here the variable rae represents the pivot integer $k \in 2\mathbb{N}$, but it is stored as a d.p. real to avoid the dynamic range difficulties highlighted under subroutine **TER**. The first term of the phase values $\omega^{\pm}(pc)$, namely $t(log(pc) + 1/pc + 1)/2$, is prone to round-off error issues for large t, even if the computation were to be carried out in Fortran double precision. So instead, this term is calculated in C using the Pari library by means of the subroutine

pari_tse. The value returned $tse := t(log(pc) + 1/pc + 1)/2 \ mod(2\pi)$ will be accurate to at least 30 decimal places.

6. Name **PSI**

Purpose: To calculate the Psi function (AS chapter 5)

$$\psi(x) := \Gamma'(x)/\Gamma(x), \qquad x \neq 0, -1, -2, \ldots$$

for $x \in \mathbb{R}\backslash\{-\mathbb{N}\}$.

Outline/Method: When $x \to \infty$ one can use the Poincaŕe-type expansion

$$\psi(x) \sim log(x) - \frac{1}{2x} - \sum_{j=1}^{J_x} \frac{B_{2j}}{(2j)x^{2j}},$$

where B_{2j} are the Bernoulli numbers ($B_2 = 1/6, B_4 = -1/30, \ldots$) and $J_x \leq \pi xe$. In practice once $x > 4$ a value of $J_x = 4$ is sufficient for an approximation accurate to seven decimal places. For $0 < x \leq 4$ the Taylor series

$$\psi(1 + x) = \frac{1}{2x} - \frac{\pi}{2tan(\pi x)} + \frac{1}{x^2 - 1} + 1 - \gamma - \sum_{j=1}^{\infty} (\varsigma(2j + 1) - 1)x^{2j}, x \neq 0, \pm 1, \pm 2, \ldots$$

can be employed. However, this series is only valid for $|x| < 2$ and convergence is very slow beyond $|x| < 1/2$. Hence for $1/2 < x \leq 4$, the recurrence formula

$$\psi(x) = \frac{1}{x - 1} + \frac{1}{x - 2} + \cdots + \frac{1}{x - (n - 1)} + \frac{1}{x - n} + \psi(x - n),$$

is used, so that $0 < x - n < 1/2$. $\psi(x - n)$ can then be found rapidly using the Taylor series above. Special care must be taken for the cases $x = -\mathbb{N}$ where the Psi function is undefined. Likewise, the values of $\psi(1/2), \psi(1), \psi(2)$ and $\psi(3)$ must dealt with separately. Initially the routine calculates $\psi(|x|)$ and if $x < 0$, then the reflection formula

$$\psi(1 - x) = \psi(x) + \frac{\pi}{tan(\pi x)},$$

is employed.

Inputs: *X* (d.p. real)

Ranges: $X \in (-\infty, \infty)$. Strictly $X \notin -\mathbb{N}$ but the routine tests for these values

Outputs: *PS* (d.p. real variable), the value of $\psi(x)$

Common Blocks:

/PARS2/: GAM (d.p. real), ZET (d.p. real array)

/PARS4/: $\pi, \sqrt{\pi}, 2\pi, -2\pi$ (d.p. reals); $e^{\mathrm{i}\pi/4}$ (d.p. complex).

(PARS2 contains the values of Euler's constant γ and the values of $\varsigma(2j+1)$ used in the above Taylors series. All these coefficients are set when the code is initialised.)

7. Name **SCHEME**

Purpose: To calculate a real parameter denoted by E_m which is typically small in the sense that $|E_m| \ll 1$, given by

$$E_m = \frac{(-1)^{m+1} C_m\left(\Phi_{2,3,\ldots m:j-1}, \Phi_{m+1:j-1}=0\right)}{x_{j-1}^{2m-1}},$$

where $m \geq 3$ is the order of the $(j-1)th$ Gauss sum in the hierarchical chain of Gauss sums. The hierarchical chain links the initial cubic Gauss sum one wishes to estimate down to a vastly shorter, kernel Gauss sum, which provides the basis of the estimate. Intermediate Gauss sums link the initial cubic sum to the kernel and comprise chain. Here $x_{j-1} = 2\Phi_{2:j-1}$ and $C_m\left(\Phi_{2,3,\ldots m:\, j-1}\right)$ are increasing complex polynomial terms of the coefficients $\Phi_{2,3,\ldots m_l:\, j-1}$ that define the $(j-1)th$ Gauss sum. (Specific expressions for C_{1-8} are given in the mathematical appendix of ……)

Outline/Method: This is a relatively simple routine but because of the nature of the C_m terms, algebraically very complex. The situation is as follows. Suppose at the jth point in the hierarchical chain, one has a generalised Gauss sum of order m defined by

$$S_{L_{j-1}}\left(\Phi_{1..m:j}\right) = \sum_{k=0}^{L_{j-1}} exp\left\{2\pi\mathrm{i} \sum_{q=1}^{m} \Phi_{q:j} k^q\right\}. \tag{4}$$

One's aim is to calculate the above efficiently. The whole ***raison-d'etre*** behind the computational method for calculating the Hardy function is that a Gauss sum such as this can be linked to another Gauss sum with fewer summands $\lfloor \xi \rfloor < L_{j-1}$. Specifically, one has

$$S_{L_{j-1}}\left(\Phi_{1..m:j}\right) = \frac{e^{-\mathrm{i}\pi/4}}{\sqrt{2\Phi_{2:j}}} \sum_{n=H(\Phi_1)}^{\lfloor \xi \rfloor} e^{2\pi\mathrm{i} g(c)} + small\ correction\ terms. \tag{5}$$

The sum on the right-hand side of this equation forms the $(j+1)th$ Gauss sum in the hierarchical chain and obviously because it has fewer summands it is much faster to estimate

than computing the left-hand side directly. However, it may be of greater order than the current value of m. The purpose behind the calculation of E_m is to decide on the size of the new order, be it $m, m+1, m+2, \ldots$ In the above

$$g(c) = nc - \sum_{q=1}^{m} \Phi_{q:j} c^q \tag{6}$$

and c is the corresponding saddle point satisfying

$$n = \sum_{q=1}^{m} q\Phi_{q:j} c^{q-1} \qquad \text{for} \quad n \in \left[H\left(\Phi_{1,j}\right), \lfloor \xi \rfloor\right]. \tag{7}$$

In general, there is no closed form solution for c which satisfies (6). However, because the higher order coefficients are asymptotically small, in particular the initial coefficients satisfy $\Phi_{q,1} = O\left(t^{1-q/2}\right)$ for $q = 3,4, \ldots m$, it is possible define closed form approximation to c which is suitably accurate for the purposes of estimating $S_{L_{j-1}}\left(\Phi_{1..m,j}\right)$. Specifically, letting $y_n = n - \Phi_{1,j}$ and $x = 2\Phi_{2,j}$, one can define an approximation

$$c_m = \sum_{q=1}^{m} C_q\left(\Phi_{2,3,\ldots q+1:j}\right) \frac{(q+1) y_n^q}{x^{2q-1}} \tag{8}$$

for the exact value of c truncated at the m^{th} order in y. Substituting (7) into (4) gives

$$g_m(c_m) = y c_m - \sum_{q=2}^{m} \Phi_{q:j} c_m^q = \sum_{q=1}^{m-1} \frac{C_q y_n^{q+1}}{x^{2q-1}} + \sum_{q=m}^{\infty} E_q y_n^{q+1}. \tag{9}$$

Exclusion of the error term, that is the sum over E_q in (9), means that the right-hand side of (5) is also Gaussian sum of order m. However, for this conclusion to be valid, one requires that value of $|2\pi E_m y_n^{m+1}| < \varepsilon$ (with $y_n = \lfloor \xi \rfloor - \Phi_{1:j}$ its maximum value), be sufficiently small so that $e^{\mathrm{i}\varepsilon} \approx 1$. (In the code a fixed parameter value of $\varepsilon = 10^{-3}$ is employed.) If this is not the case, then the order m must be increased, successively by one in (8) & (9), until the value of $|2\pi E_m y_n^{m+1}|$ falls below ε, which then fixes the order of the new Gauss sum in (5). **The necessity of determining when this constraint is satisfied is the reason for calculations of E_m.** (N.B. the error term is a sum over infinity and although typically the leading term is very dominant, making this dominant term sufficiently small does not, it itself, guarantee the entire sum is also sufficiently small. To reach this conclusion requires that the error term satisfy certain additional convergence properties which are discussed in …..)

Inputs: m, j (both short integers)

Ranges: $m \in [3, 10]$, $j \in [1, 50]$. In this code the largest order is restricted to a value of 10.

Outputs: $small$ (d.p. real variable) — the value of E_m for the input value m.

Common Block:

/PARS1/: phicoeff (two dimensional d.p. real array), fracL (d.p. real array), xr (d.p. real array), L (long integer array), icj (short integer array), MIT (short integer), mmax (short integer).

These arrays contain the various coefficients that define the jth sum in the hierarchal chain. In particular:

1. $Phicoeff(q,j) = \Phi_{q:j}$, that is the qth coefficient of the jth sum.
2. $L(j-1) = L_{j-1}$ for $j = 1,2, \ldots MIT$

   In particular $L(j) = \lfloor \xi \rfloor$ or the integer part of ξ, where $\xi = \sum_{q=1}^{m} q\Phi_{q:j} L_{j-1}^{q-1}$.
3. $fracL(j) =$ fractional part of ξ.
4. $xr(j) = 2\Phi_{2:j}$.
5. $icj(j) = \pm 1$, which designates whether the jth sum in the hierarchal chain should take its actual value, denoted by $icj = 1$, or its complex conjugate, denoted by $icj = -1$.
6. MIT is designation of the kernel sum (the last one) in the hierarchal chain.
7. $mmax$ a user defined variable that sets the maximum value of m. So $m \in [3, mmax]$. The default value $mmax = 10$, but it is possible to set this to something less this.

/PARS7/: see (d.p. real array)

During the course of the calculation of a particular E_m parameter, closely related intermediate parameters are also calculated and these are stored in the array $see(q)$. These are defined by

$$see(q) = \frac{C_q\left(\Phi_{2,3,\ldots q,j-1}, \Phi_{q+1:j-1}\right)}{x_{j-1}^{2q-1}}, \quad for\ q = 1,2, \ldots m,\ \ with\ \Phi_{m+1} = 0\ when\ q = m.$$

The parameters $see(q)$ are needed in the next subroutine PHIFACTORS.

8. Name **PHIFACTORS**

Purpose: Given the coefficients $\Phi_{01,2,3,\ldots m:\,j}$, of the jth Gauss sum in a respective hierarchal chain this routine calculates the coefficients $\Phi_{01,2,3,\ldots m:\,j+1}$ which define the $(j+1)th$ Gauss sum (the next generation) down the hierarchal chain. The new coefficients are given by

$$\Phi_{q:j+1} = \sum_{i=max(q,2)}^{m} \binom{i}{q} \left(-\Phi_{1:j}\right)^{i-q} see(q-1), \quad for\ q = 0,1,2 \ldots ., m$$

The values of $see(q)$ are computed in **SCHEME** and communicated via the common block **/PARS7/**. N.B. in the above equation the two Gauss sums are assumed to be of the same order m. The order of the $(j+1)th$ Gauss sum might well be larger than the previous one in which case extra coefficients need to be computed. This is arranged in the routine **COMPUTARRS**, which is discussed below.

Outline/Method: Very straightforward. The only potential problem is the size of the factorials making up the binomial coefficient in the above equation. For large $m \geq 12$ it is better to compute the binomial coefficient as exp $[log(m!) - \log((m-q)!) - \log(q!)]$, since individual factorials will overflow long before the coefficient itself. However, for relatively small m the individual factorials are computed directly.

Inputs: m, j (both short integers)

Ranges: $m \in [3, 10]$, $j \in [1, 50]$. In this code the largest order is restricted to a value of 10.

Outputs: No direct outputs. However, the new $(j+1)th$ Gauss sum coefficients $\Phi_{0..m,j+1}$ are stored in the array $phicoeff$ and communicated to other routines via the common block **/PARS1/**.

Common Blocks:

/PARS1/: phicoeff (two dimensional d.p. real array), fracL (d.p. real array), xr (d.p. real array), L (long integer array), icj (short integer array), MIT (short integer), mmax (short integer).

/PARS7/: see (d.p. real array)

9. Name **COMPUTECNEWTON**

Purpose: Given the coefficients $\Phi_{01,2,3,\ldots m:\,j}$, of the jth Gauss sum in a respective hierarchal chain this routine uses a standard Newton-Raphson scheme to find c the saddle point satisfying (4) above, although it is rewritten in the form

$$f(c) = m - \Phi_{1:j} - \sum_{q=2}^{n} q\Phi_{q:j}c^{q-1} = 0 \qquad \text{for} \quad m \in \left[H\left(\Phi_{1:j}\right), \lfloor \xi \rfloor\right]. \tag{10}$$

Outline/Method: A standard Newton-Raphson root finding scheme is employed. Given an estimate of the root c_i, a revised estimate is computed in the form

$$c_{i+1} = c_i - f(c_i)/f'(c_i).$$

The method is repeated until $|c_{i+1} - c_i| < tol$, where tol is some user defined tolerance level. No special coding is necessary here because the initial estimate of the root c_0 will always lie

relatively close to the actual answer and so all that is required is a bit of 'tidying up'. The reason for this lies in the fact that the coefficients Φ_2 and Φ_3 dominate the others, so one can always make an accurate initial guess c_0 by assuming (10) is approximately quadratic in c. Various values of c are needed to establish values for the 'small correction terms' that appear on the RHS of equation (1). These are the subject of subroutine **Q**, which is discussed below. All calls to **COMPUTECNEWTON** originate from subroutine **Q**.

Inputs: n, nit (both short integers); $con1, tol, c$ (all d.p. reals)

Ranges: $n \in [3, 10]$, the order of the current sum: $nit \in [1, 50]$ the position of the current sum in the hierarchal chain: $con1 = m - \Phi_1 \in (-1/2\,, \lfloor \xi \rfloor + 1/2\,)$: $tol = 10^{-7}$ (default value; can be reset by the user in routine **Q**): $c > 0$, on input $c = c_0$ the initial estimate of the root.

Outputs: c (d.p. real variable) — the final value of the root.

Common Block:

/PARS1/: phicoeff (two dimensional d.p. real array), fracL (d.p. real array), xr (d.p. real array), L (long integer array), icj (short integer array), MIT (short integer), mmax (short integer).

10. Name **MGAUSSUM**

Purpose: Given the various coefficients that define the kernel sum which terminates the hierarchal chain (specifically the MIT^{th} sum in the chain with coefficients $\Phi_{0,1,2,3,\ldots m:\,MIT}$ and number of summands $L(MIT-1)$) , the aim of this routine is to, in essence, reverse equation (5) and compute estimates for all the larger sums in the chain. The final step in the computation which estimates the 1st sum in the chain, contributes towards the calculation of the main term (equation ….) making up the Hardy function $Z(t)$.

Outline/Method: The kernel sum takes the basic form (cf. equation 1)

$$e^{-\mathrm{i}\pi/4} \sum_{n=H(\Phi_1)}^{L(MIT-1)} \frac{e^{2\pi \mathrm{i} g(c_m)}}{\sqrt{\sum_{q=2}^{m} q(q-1)\Phi_{q:MIT} c_m^{q-2}}}. \tag{11}$$

Using equations (8) and (9), this can be rewritten explicitly in terms of the sum index n as follows:

$$\frac{e^{-\mathrm{i}\pi/4 + 2\pi \mathrm{i} \Phi_{0:MIT}}}{\sqrt{2\Phi_{2:MIT}}} \sum_{n=H(\Phi_{1:\,MIT-1})}^{L(MIT-1)} e^{-2\pi \mathrm{i} \sum_{q=1}^{m} \Phi_{q:MIT} n^q} \Big/ \sqrt{1 + \sum_{q=0}^{m} r_q n^q}. \tag{12}$$

The initial part of routine calculates the first three parameters of the denominator term r_{0-2}. Beyond $q = 2$ the values of $|r_q n^q|$ are negligibly small for all n and their contribution can be ignored. The first loop then computes the kernel sum (9) long hand (adjusting the starting index to $n = 1$ if $\Phi_{1:MIT-1} > 0$ or $n = 0$ if $\Phi_{1:MIT-1} < 0$). The result is stored in the variable denoted by $c1$. The final loop then recursively moves back up the hierarchal chain repeatedly invoking equation (5). Before each iteration up the chain there is a call to the subroutine **Q** which computes the "*small correction terms*" on the right-hand side of (5). These are discussed in detail in the documentation for **Q** below. [*N.B. for all the sums in an individual chain, other than the kernel, the parameters of the denominator term in (12) are set to zero, i.e.* $r_{0-m} = 0$. *Strictly speaking this is only true for the 1st sum in the chain. Across a single hierarchal chain this assumption introduces a very small error term into the final answer. However, for an entire calculation of* $Z(t)$*, which is carried out across many such chains, the accumulation of these error terms is negligible compared to the error bound governed by the user prescribed precision parameter* ε *- see* ***error notes***.]

Inputs: m, ip (both short integers). Both these inputs are simple passed on for direct use in subroutine **Q** and are not used directly in this routine. All the direct inputs for this routine are communicated via the common blocks.

Ranges: $m = 3$ the order of the initial cubic sum and $ip \in [2, 4]$.

Output: $csum$ (d.p complex variable) contains the final estimate of the initial cubic sum prescribed in routine ALPHASUM.

Common Blocks:

/PARS1/: phicoeff (two dimensional d.p. real array), fracL (d.p. real array), xr (d.p. real array), L (long integer array), icj (short integer array), MIT (short integer), mmax (short integer).

/PARS4/: $\pi, \sqrt{\pi}, 2\pi, -2\pi$ (d.p. reals); $e^{i\pi/4}$ (d.p. complex).

/PARS5/ m1 (long integer array)

The array $m1(j)$ stores the order of the jth sum in the hierarchal chain. The initial sum is always a cubic so $m1(1) = 3$. Otherwise $m1(j > 1) \in [3, 10]$.

11. Name **COMPUTARRS**

Purpose: Given a set of coefficients $\Phi_{0-3:1}$ and summand length $L(0)$ prescribing the initial cubic Gauss sum (set in routine ALPHASUM), this routine computes all the corresponding coefficients $\Phi_{0-m:2-MIT}$ and $L(MIT-1) < L(MIT-2) < \cdots < L(1) < L(0)$ for the subsequent Gauss sums making up the corresponding hierarchal chain. It also fixes whether it is the actual value of the Gauss sum ($icj(j) = +1$), or the corresponding complex conjugate value ($icj(j) = -1$) which forms the jth link in the chain.

Outline/Method: The routine commences by storing the initial coefficients, established in ALPHASUM, in such a way that $\Phi_{2:1} > 0$, $x_1 = 2\Phi_{2:1} \in [0, 1/2]$ and $\Phi_{1:1} \in (-1/2, 1/2)$. These are necessary conditions, which must be satisfied by all the sums forming a hierarchal chain. Standard translation (book-keeping) operations applied to $\Phi_{1,2}$ transform the phase,

modulo 2π, to give an identical sum with a set of coefficients satisfying these conditions. The summand length of each new sum satisfies the recursion relation

$$L(j) = \left\lfloor \sum_{q=1}^{m} q\Phi_{q:j}\{L(j-1)\}^{q-1} \right\rfloor \quad for\ j = 1, \dots, MIT - 1. \tag{13}$$

The contribution $2\Phi_2 L(j-1)$ comprises the dominant term in (10) and since $2\Phi_{2:1} \in [0, 1/2]$ this ensures the next sum in the chain is at most half the length of the previous one (on average, it is closer to one third the previous length).

Once this is done the main loop then initiates a recursive procedure with call to routines SCHEME and PHIFACTORS (described above) to calculate the coefficients characterising the next sum in the chain, assuming it is the of the same order as the previous one. The first sum in the chain is a cubic, so $m = 3$ initially. The standard book-keeping operations are then applied to ensure the new coefficients satisfy the conditions given above. At this point a validation test is conducted. The value of $|E_m y_n^{m+1}|$ (with $y_n = \lfloor \xi \rfloor - \Phi_{1:j} = L(j-1) - \Phi_{1:j}$) is then computed and stored in variable *smalle*. The coefficient E_m has previously been computed by the call to SCHEME (see notes on that routine, especially equation five and the subsequent discussion). If the variable *smalle* is sufficiently small, then the code moves on to compute the next sum in the hierarchal chain. However, if it exceeds the threshold criterion $\varepsilon = 0.001$ this indicates that the current order of this sum is too small to accommodate a summand length $L(j-1)$. In which case, the order m is first increased to $m+1$, the coefficients of the previous sum are augmented by an additional parameter $\Phi_{m+1:j-1} = 0$ and new calls to PHIFACTORS and SCHEME are made to calculate the new $\Phi_{0..m+1:j}$ coefficients corresponding to a sum of increased order. The code moves back to carry out the necessary book-keeping operations and a further validation test is conducted. If the latter still fails, then the order is increased repeatedly until a validation test is successful or the maximum order permitted by the user is reached. For this code the absolute maximum order is set at $mmax = 10$, although the user can reduce this to a number in the range $[3, 10]$ (see notes on the code initialisation).

The code completes the coefficient calculations and exits the main loop in a number of different ways. *By far and away the most usual scenario* is that the code completes a series of successful validation tests until the number of summands $L(j) < L(j-1) < KCUT$, the user defined cut-off integer. The index specifying the final sum in the hierarchal chain is fixed by setting $MIT = j$. This final sum forms the kernel, specified by coefficients $\Phi_{0-m:MIT}$ and number of summands $L(MIT-1)$. There are a couple of caveats to this:

a) It is best not to make $L(MIT-1)$ too small, otherwise the calculation of the "$small\ correction\ terms$" in (1) becomes somewhat problematic. So, a minimum value of $L(MIT-1) = 6$ is applied. If $L(MIT-1) < 6$ then the kernel sum is moved one place back up the hierarchal chain by setting $MIT = j - 1$. True this may now mean the number of summands is greater than $KCUT$, but in practice this scenario is *extremely rare* and makes no significant difference to the run time of the code.

b) As one moves down the chain the number of summands reduces, but the absolute value of $\Phi_{3:j}$ increases, proportionally, at a faster rate. This means that the relative importance of the cubic term making up the Gauss sum phase (4) increases vis-à-vis the quadratic term. Eventually a point is reached where the gradient of the phase ceases to be predominantly linear in character and becomes more quadratic, leading to the complication of "double saddles" - two solutions to equations (7, 10). Dealing with this issue requires more sophisticated root finding methodology (at the expense of run time) and is a feature of the higher order codes under development. But in this code no such provision is made and it *essential* that the kernel sum does not contain any "double saddle" terms. Usually, the choice of cut-off integer $KCUT$ will be sufficiently large to avoid this issue but very occasionally, especially for large $t > 10^{31}$ values, this may not be enough. So, an extra check is included. "Double saddles" only occur if the second derivative of the phase is negative, i.e. when $2\Phi_{2:j} + 6\Phi_{3:j}L(j-1) < 0$ with $\Phi_{3:j} < 0$. If this scenario does occur the kernel sum is moved one place back up the hierarchal chain as in a) above.

The next most likely scenario is that the maximum order has been reached and no successful validation test has been carried out. This means it is no longer possible to construct any more sums in the chain either because they are of too high an order, or possibly because the absolute value of the error term E_m, found in SCHEME, is diverging. Either way the kernel sum is moved one place back up the hierarchal chain by setting $MIT = j - 1$ as in a) and b) above. These various tests should cover all possible scenarios which terminate the hierarchal chain calculations and exit of the main loop. But just in case something slips through, two extra checks are carried out to ensure if $L(j) < 0$ and if $L(j) > L(j-1)$, the chain terminates at that point.

Inputs: m, (short integer); N and $KCUT$ (both long integers); $initialcoeff$ (two dimensional d.p. array)

Ranges: $m = 3$ for all calls to COMPUTARRS. $N > 0, KCUT > 0$. $initialcoeff$ contains the initial Gauss sum parameters $\Phi_{0-3;1}$ calculated in ALPHASUM.

Outputs: None direct outputs. All the outputs are stored in the arrays which are communicated to other routines via the common blocks PARS1 and PARS5. For common block PARS1 see under the notes for routine SCHEME (above) for specific details.

Common Blocks:

/PARS1/: phicoeff (two dimensional d.p. real array), fracL (d.p. real array), xr (d.p. real array), L (long integer array), icj (short integer array), MIT (short integer), mmax (short integer).

/PARS5/ m1 (long integer array).

The array $m1(j)$ stores the order of the jth sum in the hierarchal chain. The initial sum is always a cubic so $m1(1) = 3$. Otherwise $m1(j > 1) \in [3, 10]$.

12. Name **Q**

Purpose: This routine calculates "$small\ correction\ terms$" that appear in (5) and are derived from the Euler-Maclaurin summation formula, which forms the crux of the recursion scheme. (Their derivation is discussed in much more detail Lewis & Brereton 2025, cf. the terms designated W_{1-5} in the discussion leading up to equation 96). In the code itself the five terms making up the correction are denoted by $t1, t2, t3, t4$ & $t5$ respectively. Specifically:

$$\overline{t1} = \frac{\mathrm{i}}{2\pi}\left[e^{2\pi \mathrm{i} f(L)}[\psi(\xi+1) - \psi(\lceil\xi\rceil - \xi)] - [\psi(\Phi_1 + 1) - \psi(\lceil\xi\rceil - \Phi_1)]\right], \tag{14}$$

$$\overline{t2} = \mathrm{i}e^{-2\pi \mathrm{i} f(L)}\left[\frac{e^{-\mathrm{i}\pi/4}}{2\sqrt{2\Phi_2}} e^{-\mathrm{i}\pi(\lceil\xi\rceil-\xi)^2/2\Phi_2} \mathrm{erfc}\left[e^{-\mathrm{i}\pi/4}\sqrt{\frac{\pi}{2\Phi_2}}(\lceil\xi\rceil - \xi)\right] - \frac{1}{2\pi(\lceil\xi\rceil - \xi)}\right], \tag{15}$$

$$t3 = \frac{\left[1 + e^{2\pi \mathrm{i} f(L)}\right]}{2}, \tag{16}$$

$$\overline{t4} = \left[\frac{\mathrm{i}e^{-\mathrm{i}\pi/4}}{2\sqrt{2\Phi_2}} e^{-\mathrm{i}\pi(1+\Phi_1)^2/2\Phi_2} \mathrm{erfc}\left[e^{-\mathrm{i}\pi/4}\sqrt{\frac{\pi}{2\Phi_2}}(1+\Phi_1)\right] - \frac{\mathrm{i}}{2\pi(1+\Phi_1)}\right], \quad \Phi_1 < 0, \tag{17a}$$

$$\overline{t4} = \left[\frac{\mathrm{i}e^{-\mathrm{i}\pi/4}}{2\sqrt{2\Phi_2}} e^{-\mathrm{i}\pi\Phi_1^2/2\Phi_2} \mathrm{erfc}\left[e^{-\mathrm{i}\pi/4}\sqrt{\frac{\pi}{2\Phi_2}}\Phi_1\right] - \frac{\mathrm{i}e^{-2\pi \mathrm{i} f(L)}}{2\pi\xi}\right], \quad \Phi_1 > 0\,. \tag{17b}$$

$$\begin{aligned} t5 = {} & \frac{\mathrm{i}}{2\pi}\left[\psi(\lceil\xi\rceil - \Phi_1) - \psi(H(\Phi_1) - \Phi_1) + e^{2\pi \mathrm{i} f(N)}[\psi(H(\Phi_1) - \xi) - \psi(\lceil\xi\rceil - \xi)]\right] \\ & - \left[\frac{e^{2\pi \mathrm{i} g(c) - \mathrm{i}\pi/4}}{2\sqrt{2\Phi_2}} \mathrm{erfc}\left\{2\sqrt{\pi\Phi_2}c\right\} + \frac{\Phi_2}{2\pi^2(n-\Phi_1)^3} + \frac{\mathrm{i}e^{-2\pi(n-\Phi_1)c}}{2\pi(n-\Phi_1)} - \right. \\ & \left. \frac{e^{-2\pi(n-\Phi_1)c}}{2\pi^2}\left(\frac{\Phi_2\{2\pi^2(n-\Phi_1)^2c^2 + 2\pi(n-\Phi_1)c + 1\}}{(n-\Phi_1)^3}\right)\right]_{n \geq H(\Phi_1)} \\ & - \left[\frac{e^{2\pi \mathrm{i} g(c) - \mathrm{i}\pi/4}}{2\sqrt{2\Phi_2}} \mathrm{erfc}\left\{2\sqrt{\pi\Phi_2}(L-c)\right\} + \frac{\Phi_2 e^{2\pi \mathrm{i} f(N)}}{2\pi^2(\xi-n)^3} + \frac{\mathrm{i}e^{2\pi \mathrm{i} f(L)} e^{-2\pi(\xi-n)(L-c)}}{2\pi(\xi-n)}\right. \\ & \left. - \frac{e^{2\pi \mathrm{i} f(L)} e^{-2\pi(\xi-n)(L-c)}}{2\pi^2}\left(\frac{\Phi_2\{2\pi^2(\xi-n)^2(L-c)^2 + 2\pi(\xi-n)(L-c) + 1\}}{(\xi-n)^3}\right)\right]_{n \leq \lfloor\xi\rfloor} \end{aligned} \tag{18}$$

In these formulae $\Phi_{1,2}, L \equiv \Phi_{1,2:j}, L_{j-1}$, i.e. the parameters for the jth sum in the hierarchal chain, $\xi = f(L) = \sum_q q\Phi_{qj}L^{q-1}$ and $L_j = \lfloor \xi \rfloor$. Notice that only one or other of (14a, b) needs to be calculated, depending upon the sign of Φ_1. In (18) the value of the saddle point c satisfies (7 & 10). The computation of c utilises subroutine COMPUTECNEWTON. The term $t5$ is by far the most complicated and difficult to evaluate. In theory the value of n runs from zero or one to L_j. However, in practice the terms in square brackets die very rapidly and can be ignored except when n is close to the respective endpoints. The input parameter ip fixes the range of values of n which are calculated i.e. $n \in [0 \; or \; 1, ip] \cup [L_j - ip, L_j]$. Values of $ip = 3$ or $ip = 4$ are more than sufficient.

Outline/Method: All the terms utilise the routines ERF and PSI which are described above. One way in which to improve overall computational speed would be to integrate intrinsic routines from the PARI-library to replace the purpose-built routines ERF and PSI as currently constituted. They would, in all probability, work faster.

Inputs: m, mit, ip (all short integers).

Ranges: $m \in [3, 10]$, order of the current sum; $nit \in [1, 50]$, the position of this sum in its hierarchal chain; $ip \in [3, 4]$.

Output: qq (d.p. complex variable), the sum of terms $\overline{t1} + \overline{t2} + t3 + \overline{t4} + t5$ as defined above.

Common Blocks:

/PARS1/: phicoeff (two dimensional d.p. real array), fracL (d.p. real array), xr (d.p. real array), L (long integer array), icj (short integer array), MIT (short integer), mmax (short integer).

/PARS4/: $\pi, \sqrt{\pi}, 2\pi, -2\pi$ (d.p. reals); $e^{i\pi/4}$ (d.p. complex).

/PARS5/ m1 (long integer array).

PARI SUBROUTINES

The main routines used are as follows:

1. Name **PARI_CALC**

Purpose: To calculate the value of the term denoted by $RSremainder(t)$ which is defined by

$$RSremainder(t) = 2\sum_{N=1}^{N_c} \frac{\cos(\theta(t) - tlog(N))}{\sqrt{N}}.$$

Here $\theta(t)$ is the Riemann-Siegel theta function and N_c is the cut-off integer. This is a calculation prone to significant round-off error using Fortran routines for large t. Hence the use of high precision PARI routines to evaluate it.

2. Name **t_grabber**
   Purpose: To convert the t value, which is read into the code from the input data file as a character string, into a high precision C variable.

3. Name **real2pari**

   Purpose: To convert a Fortran real variable r into a high precision C variable denoted by r_pari. Frequently used to enable large scale $mod(2\pi)$ calculations, which require very high precision.

4. Name **pari_tse**

   Purpose: To calculate the first part of the $\omega^{\pm}(pc)$ phases $mod(2\pi)$ defined in routine ALPHASUM by equation (3). An example of a calculation where Fortran double precision is insufficiently accurate.

5. Name **pari_phases**
   Purpose: To calculate $\theta(t)$ the Riemann-Siegel theta function $mod(2\pi)$ and $(t + \pi/8)$ $mod(2\pi)$. Two phase calculations that require very high precision.