lerchphi User's Guide

Sergej V. Aksenov[1,2] and Ulrich D. Jentschura[3]

[1] Department of Microbiology and Immunology,
University of Michigan, Ann Arbor, MI 48109, USA

[2] Current address: Department of Zoology,
University of Cambridge, Downing Street, Cambridge, CB2 3EJ, UK

[3] Institute of Theoretical Physics,
Dresden University of Technology, 01062 Dresden, Germany

Email: sa354@cam.ac.uk, jentschura@physik.tu-dresden.de

May 1, 2002

Abstract This document describes the usage of a C program that calculates Lerch's Φ transcendent of real arguments using convergence acceleration methods (nonlinear sequence transformations and the combined nonlinear-condensation transformation).

Contents

1 Conditions of Use

- This document and software (file `lerchphi.c`) described herein is copyright by Sergej V. Aksenov and Ulrich D. Jentschura.
- This software and its constituent parts come with no warranty, whether expressed or implied, that it is free of errors or suitable for any specific purpose. It must not be used to solve any problem, whose incorrect solution could result in injury to a person, institution, or property. It is user's own risk to use this software or its parts and the authors disclaim all liability for such use.
- This software is distributed "as is". In particular, no maintenance, support, troubleshooting, or subsequent upgrade is implied.
- The use of this software must be acknowledged in any publication that contains results obtained with it; please cite [1].
- The free use of this software and its parts is restricted for research purposes; commercial use require permission and licensing from Sergej V. Aksenov and Ulrich D. Jentschura.

2 Availability

The source of the LerchPhi program `lerchphi.c`, the example driver file `test.c`, example makefile, and a sample output of a run of `test.c`, as well as this documentation can be downloaded from the web pages of the authors [2].

3 Purpose

The C program contained in the file `lerchphi.c` calculates Lerch's transcendent which is defined by the following infinite series:

$$\Phi(z,s,v) = \sum_{n=0}^{\infty} \frac{z^n}{(n+v)^s}\,, \qquad |z|<1\,, \qquad v \neq 0,-1,\ldots\,. \tag{1}$$

[See Eq. (1) on p. 27 of Ref. [3]]. We consider the case of real parameters z, s, and v only.

The C language implementation of the combined nonlinear-condensation transformation (CNCT) described here is motivated by the virtual absence of freely available and transportable C code for Lerch's transcendent for real arguments.

4 Basic formulas

Assume that the elements of the sequence $\{s_n\}_{n=0}^{\infty}$ represent partial sums

$$s_n = \sum_{k=0}^{n} a(k) \tag{2}$$

of an infinite series

$$\sum_{k=0}^{\infty} a(k)\,, \quad a(k) \geq 0\,, \tag{3}$$

e.g. that in Eq. (1). We denote the limit $s \equiv \sum_{k=0}^{\infty} a(k)$.

The first step of the CNCT is the Van Wijngaarden transformation [5] of the nonalternating input series (3), whose partial sums are given by (2), into an alternating series

$$s = \sum_{j=0}^{\infty} (-1)^j \mathbf{A}_j\,. \tag{4}$$

The quantities $\mathbf{A}_j$ are defined according to

$$\mathbf{A}_j \;=\; \sum_{k=0}^{\infty} \mathbf{b}_k^{(j)}\,, \tag{5}$$

where

$$\mathbf{b}_k^{(j)} \;=\; 2^k\, a(2^k\,(j+1)-1)\,. \tag{6}$$

The second step is to apply an appropriate sequence transformation to the series of partial sums

$$\mathbf{S}_n \;=\; \sum_{j=0}^{n} (-1)^j\, \mathbf{A}_j \tag{7}$$

of the Van Wijngaarden transformed series (4) of the first step. We use the delta sequence transformation [6, 7] which is constructed according to [Eq. (8.4-4) of [7]]

$$\delta_k^{(0)}\,(1, \mathbf{S}_n) = \frac{\displaystyle\sum_{j=0}^{k} (-1)^j \binom{k}{j} \frac{(j+1)_{k-1}}{(k+1)_{k-1}} \frac{\mathbf{S}_j}{\omega_j}}{\displaystyle\sum_{j=0}^{k} (-1)^j \binom{k}{j} \frac{(j+1)_{k-1}}{(k+1)_{k-1}} \frac{1}{\omega_j}}\,. \tag{8}$$

The quantities appearing in Eq. (8) have the following interpretation: $\mathbf{S}_n$ are the elements of an alternating sequence whose convergence is to be accelerated, and the ω_n are remainder estimates, i.e. estimates for the truncation error $\omega_n \approx r_n$ defined by $r_n = \mathbf{S}_n - s$. Remainder estimates are calculated as $\omega_n = \mathbf{S}_{n+1} - \mathbf{S}_n$.

5 Usage of the `lerchphi` Program

The program consists in a self-contained file `lerchphi.c` written in ISO C. The file `lerchphi.c` should be compiled with any user written main program. A test program `test.c` is provided which calculates Lerch's transcendent for several combinations of parameters, as well as a make-file (written for Unix systems) to build the executable `test`. We built and ran an executable

from this makefile on a Macintosh computer with the PowerPC G3 processor, Mac OS X v10.1.3, Darwin Kernel v5.3, and on an INTEL–based LINUX system running SuSE Linux 7.3, as well as a Sun workstation running SunOS. However, one should rather view these makefiles as templates and modify for the particular operating system and compiler.

Further useful explanations are provided in the `README` file which forms part of the current distribution.

The main program should call `lerchphi()` as follows

```
flag = lerchphi(&z, &s, &v, &acc, &result, &iter);
```

Here, the variables should be defined in the main program and have the following types:

```
int flag, iter;
```

```
double z, s, v, acc, result;
```

The meaning of these variables is as follows: `z`, `s`, `v` carry the arguments of Lerch's transcendent $\Phi(z, s, v)$; `acc` stores the value of the desired relative accuracy of the result; `result` holds the calculated value of Lerch's transcendent; and `iter` is the number of iterations in the convergence acceleration loop needed to reach the accuracy `acc`. A positive (nonzero) value of `flag` signifies that an internal error occurred in the calculation. The error flags are explained in Table 1 below.

The range of valid arguments z, s and v of the C program is determined by the properties of the defining series representation (1): we require $-1 < z < 1$, and s, v to be real. The case of negative v is problematic. The C program stops with an error flag if v is a negative integer [one of the terms of the defining series of Lerch's transcendent has a vanishing denominator for negative integer v]. Finally, if v is negative non-integer, then we require s to be integer (for negative non-integer v and non-integer s, the C function `pow` is not well-defined).

6 Details of the Implementation

The use of the CNCT for positive z, z close to unity, removes the principal numerical difficulties associated with the slow convergence of the series (1) in this parameter region. For negative z, the partial sums of the – in this case alternating – series (1) are directly used as input data for the delta transformation (8), and the Van Wijngaarden step (4) – (6) of the CNCT may be skipped. This corresponds to the standard use of nonlinear sequence transformations as efficient accelerators for alternating series [7].

The LerchPhi program has two subroutines: `aj()` which calculates the Van Wijngaarden terms $\mathbf{A}_j$ in Eq. (5), and `lerchphi()` that calculates the sequence of CNC transforms (8). Table 1 summarizes certain special conditions and genuine error conditions handled by `lerchphi()` and lists the flag values generated.

The program checks for parameters z, s, and v to be in the domain of validity of the series representation of Lerch's transcendent (1). The program also refuses to calculate if $v < 0$ and s is not an integer, which is problematic for the standard C function `pow()`. Note that the program can determine only up to machine accuracy whether the parameters v and s are integer.

Special case	Program action	Flag
$z = 0$	Return $1/v^{s\dagger}$	0
$z < -0.5$	Use (2) as input for (8)‡	0
$\lvert z\rvert \leq 0.5$	Use term-by-term summation of (1)	0
$v < 0$, $v \notin \mathbb{Z} \setminus \mathbb{N}$, $s \in \mathbb{Z}$	Use (9) with $m = -[\![v]\!]$	0
$\lvert z\rvert \geq 1$	Error exit	1
$v < 0$, $v \in \mathbb{Z} \setminus \mathbb{N}$	Error exit	2
$v < 0$, $v \notin \mathbb{Z} \setminus \mathbb{N}$, $s \notin \mathbb{Z}$	Error exit	3
Overflow in index of $a(k)$ [Eq. (6)]	Return current iterate	4
Underflow in ω_j [Eq. (8)]	Return current iterate	5
Over max. iterations	Return current iterate	6

†This is just the first term of the series (1)
if we use the definition $0^0 = 1$.

‡The Van Wijngaarden step (4) – (6) is not needed because
the series is already alternating [replace $\mathbf{S}_n \to s_n$ in Eq. (8)].

Table 1: Special cases and error conditions in the C program `lerchphi()` which calculates Lerch's transcendent.

7 Algorithms Used

We use the following relation [Eq. (2) on p. 27 of Ref. [3]]

$$\Phi(z, s, v) = z^m \, \Phi(z, s, m + v) + \sum_{n=0}^{m-1} \frac{z^n}{(n + v)^s} \, . \tag{9}$$

to transform from negative to positive values of v.

In the term-by-term summation of the defining series (1) for $|z| \leq 0.5$, we use the following recursion relationship for the terms $a(k)$ of Lerch's power series:

$$\frac{a(k+1)}{a(k)} = z \left(\frac{v + k}{v + k + 1} \right)^s \, . \tag{10}$$

For the calculation of the Van Wijngaarden condensed series (5) of odd index, we use the following recursion [1, 8]:

$$\mathbf{A}_{i+1} = 1/2 \, (\mathbf{A}_{i/2} - a(i/2)) \, , \tag{11}$$

where i in the main loop of the program is an even integer.

The final output of the CNC transformation is a sequence of approximants $\mathcal{T}_n \equiv \mathcal{T}_{\mathrm{CNC}}(n)$ that converge to the value of $\Phi(z, s, v)$. We define the ratio of two consecutive differences of approximants as

$$x_n = \left| \frac{\mathcal{T}_n - \mathcal{T}_{n-1}}{\mathcal{T}_{n-1} - \mathcal{T}_{n-2}} \right| . \tag{12}$$

The x_n behave asymptotically, as shown in [1] for a class of model problems, like a geometric series (within the region of convergence). Therefore, a good estimate for the truncation error $\mathcal{T}_n - \mathcal{T}_\infty$ can be obtained by summing the geometric series $\sum_{k=1}^{\infty} \bar{\rho}^k \, |\mathcal{T}_n - \mathcal{T}_{n-1}|$ where our best estimate for $\bar{\rho}$ is $\bar{\rho} \approx x_n$. Therefore, we use the following convergence criterion to terminate the calculation of the CNC transforms:

$$\frac{2}{x_n} \left[\frac{1}{1 - x_n} \left| \frac{\mathcal{T}_n - \mathcal{T}_{n-1}}{\mathcal{T}_n} \right| \right] < \texttt{acc} \, . \tag{13}$$

Here, `acc` is the specified desired *relative* accuracy of the result. The factor $2/x_n$ in (13) is a heuristic "safeguard factor" introduced with the notion of avoiding a premature termination of the calculation of successive transforms in the problematic case of two consecutive transforms accidentally assuming values very close to each other. Such a situation may arise *before* the asymptotic, geometric convergence sets in. The term in square brackets in (13) represents the remainder estimate based on the geometric model $x_n \approx \bar{\rho}$ [see Eq. (12)].

Please note that `lerchphi.c` relies on the header file `float.h` which contains floating-point specifications for the minimum representable double precision number and machine epsilon on a system. This header file may not be present on all computer systems on which `lerchphi.c` is compiled. If it is desired to code the relevant arithmetic constants explicitly, then one should modify lines 50 and 51 of `lerchphi.c`.

Finally, we would like to point out that the current implementation of `lerchphi` relies on only two algorithms (direct summation of the defining power series and convergence acceleration), and that one cannot expect to obtain optimal performance in all parameter regions, let alone analytic continuations for those cases where the power series (1) diverges. As regards the evaluation of special functions with very large (excessive) parameter values, it is known that asymptotic expansions can provide optimal methods of evaluation. These are not implemented in the current version of `lerchphi`.

Additionally, the use of the defining power series (1) entails *eo ipso* numerical problems, when only double-precision arithmetic is used. Consider the following two representative parameter combinations:

$$\begin{aligned} &\text{(case 1:)} \quad z = 0.99999 \, , \quad s = 2 \, , \quad v = 10^3 \, , \\ &\text{(case 2:)} \quad z = 0.0003 \, , \quad s = 2 \, , \quad v = -3.00000\ 00000\ 0001 \, . \end{aligned} \tag{14}$$

When using the series (1) in double precision, problems result for both cases.

Case 1. The use of the CNCT implies that terms of very large index of the input series have to be evaluated, while avoiding the necessity to evaluate *all* terms until convergence is reached. In case 1, the argument z is very close to unity, but this should *a priori* not be a problem – the CNCT is

designed to work with this problematic case. However, since v is large, the contribution of terms of large index as compared to the terms of small index in (1) becomes numerically more significant. This entails a loss of precision when trying to evaluate expressions like 0.99999^L where L is a very large integer. This expression can be written as $\exp(L \ln 0.99999)$, and the numerical cancellations encountered in the evaluation of $\ln 0.99999 \approx -0.00001$ eventually become significant when L is excessively large. In the considered case, for a specified desired relative accuracy of 10^{-14}, the double-precision version `lerchphi.c` yields a value of $9.59714\,89709\,9796 \times 10^{-4}$ which has to be compared to the true value of $9.59714\,89709\,9654\ldots \times 10^{-4}$ obtained with arithmetic of enhanced accuracy. The relative accuracy of the result obtained with double-precision arithmetic is only 10^{-12} and thus smaller than the specified accuracy goal.

Case 2. The term with $n = 3$ in Eq. (1) dominates the sum, and its calculation entails considerable loss of numerical significance in forming the difference $3-3.00000\,00000\,0001$, corresponding to a loss of 14 significant decimals. The double-precision version yields a value of $2.58\ldots \times 10^{17}$, which has to be compared to the value $2.70\ldots \times 10^{17}$ obtained with the extended-precision version. It is natural that after a loss of 14 decimals at an intermediate stage of a calculation, the final answer will provide not more than two significant decimals if double-precision arithmetic (roughly 16 decimals) is used.

Consequently, both above cases find a solution in the extended-precision version of the code which is discussed in the next section.

8 Extended Precision

The file `lerchphimp.cc`, written in C++, contains a version of `lerchphi` that uses multiprecision software libraries, e.g. [9, 10], to perform all floating-point arithmetic. The `lerchphimp.cc`-code and the testing program `testmp.cc` use the multiprecision library [9]; running the tests reveals that calculations in extended precision give correct results for the two problematic cases listed in Eq. (14).

For the 8 representative example cases considered in the testing routine, we have also carried out calculations with our own Mathematica-based [11] implementation of the CNCT (file lerchphi.m) and we compared with Mathematica's built-in `LerchPhi` routine. The speed comparison of `lerchphimp.cc` and Mathematica's built-in `LerchPhi` is in favour of `lerchphimp.cc` for the 8 example calculations considered, and final results are in mutual agreement.

Reports on further tests and comparisons are always welcome to the authors.

References

[1] S. V. Aksenov, M. A. Savageau, U. D. Jentschura, J. Becher, G. Soff, and P. J. Mohr, *Application of the Combined Nonlinear-Condensation Transformation to Problems in Statistical Analysis and Theoretical Physics*, Comput. Phys. Commun. **150**, 1 (2003), e-print math.NA/0207086.

[2] S. V. Aksenov and U. D. Jentschura, C and Mathematica Programs for Calculation of Lerch's Transcendent,
available at *http://aksenov.freeshell.org/* and
http://tqd1.physik.uni-freiburg.de/~ulj.

[3] H. Bateman, *Higher Transcendental Functions* (McGraw-Hill, New York, NY, 1953), Vol. 1.

[4] U. D. Jentschura, P. J. Mohr, G. Soff, and E. J. Weniger, Comput. Phys. Commun. **116**, 28 (1999).

[5] A. van Wijngaarden, in *Cursus: Wetenschappelijk Rekenen B, Process Analyse* (Stichting Mathematisch Centrum, Amsterdam, 1965), pp. 51–60.

[6] A. Sidi, Math. Comput. **33**, 315 (1979).

[7] E. J. Weniger, Comput. Phys. Rep. **10**, 189 (1989).

[8] J. W. Daniel, Math. Comput. **23**, 91 (1969).

[9] D. H. Bailey *et al.*, High precision arithmetic software,
available at *http://www.nersc.gov/~dhbailey/mpdist/mpdist.html*.

[10] K. M. Briggs, Double-double floating point arithmetic,
available at *http://www.btexact.com/people/briggsk2/doubledouble.html*.

[11] S. Wolfram, *Mathematica-A System for Doing Mathematics by Computer* (Addison-Wesley, Reading, MA, 1988).