

# Implementation of Highly Accurate Complex Inclusion Functions in the CoStLy Library

Markus Neher\*

Institut für Angewandte Mathematik, Universität Karlsruhe, 76128 Karlsruhe, Germany

Inclusion functions for complex standard functions have been implemented in the CoStLy C++ class library. Among other functions, CoStLy includes procedures for the exponential function, the logarithm, root and power functions, trigonometric and hyperbolic functions, and their inverse functions.

Numerical examples are presented to demonstrate the high level of accuracy in computation that has been achieved with our implementation.

© 2006 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim

## 1 Introduction

In this paper, we report on a C++ class library of complex standard functions for the rigorous computation of function values and of function ranges, which has been named CoStLy (Complex Standard Functions Library) and which is distributed under the conditions of the GNU General Public License. It is available at the CoStLy website [2].

Rectangular complex interval arithmetic [1] is used for the computations. The set of all rectangular complex intervals is denoted by  $\mathbb{IC}$  in the following. In the CoStLy procedures, all truncation and roundoff errors are calculated during the course of the floating-point computation and enclosed into the result. Thus all computations yield rigorous bounds for the function values or ranges, respectively.

## 2 Verified Complex Inclusion Functions

The design of the inclusion functions included in the CoStLy library has been guided by the paradigm that range bounds must be valid in any circumstance. For a single-valued complex function  $f$ , its inclusion function  $F : \mathbb{IC} \rightarrow \mathbb{IC}$ , and a given rectangular complex interval  $Z$ , this means that  $F(Z)$  must contain the set  $\{f(z) \mid z \in Z\}$ . For a multi-valued complex function, the meaning of a *valid* enclosure is less obvious. What is, for example, the value of  $\sqrt{-1}$ ? Should the computer program return  $+i$ ,  $-i$ ,  $\{+i, -i\}$ ,  $i[-1, 1]$  or consider  $\sqrt{-1}$  as undefined? The answer to this question depends very much on the context of the computation.

For each multi-valued function  $f$  in the library, CoStLy contains an inclusion function  $F_s$  for the single-valued principal branch of  $f$ . Usually,  $F_s$  is only defined on a subset of  $\mathbb{IC}$ . Since program abortion is highly unpopular with computer users, some alternative types of inclusion functions have also been implemented in the CoStLy library for the multi-valued functions. These include inclusion functions  $F_c$  that are defined on  $\mathbb{IC}$ , but enclose function values of different branches of  $f$ . For this type of inclusion function, the user of the software must check that the respective calculation is justified by theory. Where applicable, inclusion functions  $F_a$  containing all function values of  $f$  have been implemented. For example, such inclusion functions are available for roots. For some multi-valued complex functions, however, the set of all function values is generally unbounded. In this case, no such inclusion functions are available.

## 3 Accuracy of the Inclusion Functions

For all inclusion functions in the CoStLy library, optimal range bounds are computed in exact arithmetic. The implementation of the algorithms in floating point arithmetic is generally obstructed by overflow, underflow or cancellation (OUC) in intermediate expressions. Avoiding OUC exceptions often requires many case distinctions. In the CoStLy library, all expressions subject to potential overflow have been removed to avoid program abortion or invalid results. For the sake of accuracy, we have also tried to remove all cancellation and underflow exceptions, even though these often only produce overestimations for arguments that are unlikely to appear in applications, such as arguments with very large or very small absolute values. For arguments with absolute values close to  $\max_{\text{real}} \approx 1.8\text{E}+308$ , the computation may occasionally terminate due to intermediate overflow. Removing all potential overflow situations for arguments that are unlikely to occur in a practical calculation has not been arranged, since it would generally slow down the computation. All OUC exceptions are documented in the CoStLy source code.

\* e-mail: markus.neher@math.uni-karlsruhe.de

## 4 Numerical Examples

The CoStLy library has been extensively tested for arguments with absolute values ranging from  $1.0\text{E}-300$  to  $1.0\text{E}+300$ . For most arguments, the computed bounds for function values are highly accurate. In many test cases, the observed precision of the result was about 50 correct bits (out of the 53 bits available in IEEE 754 floating point arithmetic) for point arguments, as shown in Table I.

### 4.1 Function Values for Point Arguments

Bounds for function values for point arguments are computed for selected complex functions and arguments. In the last column of Table I, the number of coinciding bits for the computed lower and upper bounds for the real and imaginary parts of the respective function value are given. % is used to denote zero values, for which the chosen measure of accuracy is not applicable.

$f, z$	$f(z)$	CoStLy 2.0 precision (bits)
$\ln(Z_1)$	$-6.9\text{E}+2 + \imath 7.9\text{E}-1$	49.9 / 48.7
$\ln(Z_2)$	$6.9\text{E}+2 + \imath 1.0\text{E}-300$	49.9 / 52.4
$\tan(Z_1)$	$1.0\text{E}-300 + \imath 1.0\text{E}-300$	49.0 / 48.2
$\tan(Z_3)$	$0.0 + \imath 1.0$	% / 52.4
$\text{asin}(Z_1)$	$1.0\text{E}-300 + \imath 1.0\text{E}-300$	51.0 / 50.2
$\text{asin}(Z_2)$	$1.6 + \imath 6.9\text{E}+2$	52.6 / 48.6
$\text{atan}(Z_1)$	$1.0\text{E}-300 + \imath 1.0\text{E}-300$	51.4 / 49.5
$\text{atan}(Z_2)$	$1.6 + \imath 0.0$	52.6 / %

Table I. Selected function values for  $Z_1 = 1.0\text{E}-300 + \imath 1.0\text{E}-300$ ,  
 $Z_2 = 1.0\text{E}+300 + \imath 1.0$ ,  $Z_3 = 1.0 + \imath 1.0\text{E}+15$ .

### 4.2 Range Bounds for Function Compositions

Here we consider range enclosures for  $f(z) = \cot(\text{acot}(\tan(\text{atan}(z))))$ ,  $Z = [1 - c, 1 + c] + \imath[1 - c, 1 + c]$ . The particular value of  $c$  is shown in the first column of Table II. The widths of the range enclosures quickly grow with increasing  $c$  due to the dependency problem of interval arithmetic. Nevertheless, the evaluation of  $f$  with CoStLy still succeeds for  $c = 1 - 1.0^{-15}$ . For  $c \geq 1$ ,  $f$  has a singularity in  $Z$  and the interval arithmetic evaluation of  $f$  is undefined.

$c$	CoStLy 2.0
0.1	$[ 7.2\text{E}-01, 1.4 ] + \imath [ 6.3\text{E}-01, 1.3 ]$
0.5	$[ 1.2\text{E}-01, 7.0 ] + \imath [ 2.8\text{E}-02, 3.7 ]$
0.9	$[ 8.3\text{E}-07, 9.1\text{E}+03 ] + \imath [ 5.5\text{E}-05, 1.1\text{E}+04 ]$
$1-1.0^{-6}$	$[ 3.9\text{E}-37, 1.3\text{E}+19 ] + \imath [ 4.1\text{E}-20, 2.5\text{E}+19 ]$
$1-1.0^{-15}$	$[ -1.7\text{E}-16, 1.6\text{E}+46 ] + \imath [ 3.2\text{E}-47, 3.2\text{E}+46 ]$

Table II. Evaluation of  $f(z) = \cot(\text{acot}(\tan(\text{atan}(z))))$   
for  $Z = [1 - c, 1 + c] + \imath[1 - c, 1 + c]$ .

## Conclusion

We have presented CoStLy, a C++ class library for the validated computation of several complex standard functions. CoStLy is distributed under the GNU general public license. Future work will concentrate on maintaining the library with regard to the continuing development of compilers.

## References

- [1] G. Alefeld and J. Herzberger, Introduction to Interval Computations (Academic Press, New York, 1983).
- [2] CoStLy website, <http://www.uni-karlsruhe.de/~Markus.Neher/CoStLy.html> [April 2006].
- [3] M. Neher and I. Eble, CoStLy: A validated library for complex functions, PAMM **4**, 594-595 (2004).