Documentation of ReLIADiff. A C++ Software Package For Real Laplace transform Inversion based on Algorithmic Differentiation [*]

LUISA D'AMORE†, ROSANNA CAMPAGNA†, VALERIA MELE†
ALMERICO MURLI‡

† University of Naples Federico II, Via Cintia, Naples, Italy
‡ CMCC -Lecce, Italy and SPACI

RELIADIFF SOFTWARE

[*]Accompanying the paper in [1]

Contents

1 Purpose **2**

2 Outline of use **2**

3 Specification **4**

4 Arguments **4**
4.1 Input Parameters 4
4.2 Output Parameters 6
4.3 Function Return Value 7

5 Calling sequence **7**

6 An example of Calling Program **7**
6.1 Output of the Calling Program Example 10

7 How To run the Calling Program Example **10**
7.1 On a Linux system, with a g++ compiler installed 10
7.2 On a Windows system, with Bloodshed DEV-C++ IDE v. 4.9.9.2 installed . . . 11

8 Analysis of the Diagnostic parameters **11**

9 Remarks on the behavior **13**

10 Content of the directory C++_files/src **13**

1 Purpose

RELIADIFF is a fully automatic software which computes the inverse (f) of a Laplace transform function (F) computable on the real axis. It uses the software `FADBAD/TADIFF` [2] implementing the Algorithmic Differentiation. The mathematical background and general information about its performance are described in the accompanying paper [1].

RELIADIFF is a software package written in C++. To compile RELIADIFF and any software using RELIADIFF you need a C++ compiler. If you do not have one:

- On a Linux system we suggest to install GCC (GNU Compiler Collection, `http://www.gnu.org/software/gcc/gcc-4.6/`).
- On a Windows system we suggest to install Bloodshed DEV-C++ IDE (v. 4.9.9.2, `http://www.bloodshed.net/dev/devcpp.html`).

2 Outline of use

The package consists essentially of the routine RELIADIFF.

Last section illustrates all the files.

To use the package, the user must provide

- a context (such as a Main program) calling RELIADIFF;
- a function subprogram evaluating the transform $F(s)$, a double precision (in sense of `TADIFF` [2]) valued function of a double precision (in sense of `TADIFF`) argument.

The C/C++ function implementing F(s) shall take in input a T<double> value [2] and return a T<double> value, that will contain the required function value and related Taylor coefficients, calculated by Taylor arithmetic.
To obtain this, all the floating point variables inside the function shall be of T<double> type (not the constants).

Users have to include the header `RELIADIFF.h`.

Next sections illustrate the calling sequence and a main program.

The routine RELIADIFF needs as input

- the function to invert F;
- the value of $x \geq 0$ where the inverse function f has to be computed;

User may tune the work of the software providing also (optionally):

- the value of $sigma0$ (or an approximated value of it), the convergence abscissa of F;
- a tolerance to the accuracy on $f(x)$;
- the maximum number of series coefficients;
- the singularity at zero of F;
- the choice of printing all the used series expansion coefficients.

The software provides as output

- the inverse Laplace function f evaluated at the input point x;
- the estimate of the absolute error on f;
- the estimate of the relative error on f;

WARNING: Returned errors are just errors estimate, they are not errors bounds

- some diagnostics parameters:
  - the "optimal" number of series coefficients to use;
  - the number of series coefficients used to compute the "optimal" value;
  - a flag needed to interpret the work of the software.

In the directory ***C++_files\demos-Linux_g++*** *users finds demos to test easy the RELIADIFF software on functions from a provided database or written by the user, on a Linux system. Consult the document DEMOS_useguide.pdf.*

3 Specification

```
int RELIADIFF  (  double x,
                  T<double> (*fz)(T<double>),
                  char* szero,
                  char* pcoeff,
                  char* sigma0,
                  char* tol,
                  char* nmax,
                  double *ILf,
                  double *absesterr,
                  double *relesterr,
                  int *flag,
                  int *nopt,
                  int *ncalc,
               )
```

4 Arguments

4.1 Input Parameters

x : double precision
it contains value at which the Inverse Laplace Transform is required.
It has to be greater or equal to zero.

fz : (`TADIFF`) double precision[1] function pointer
it contains the name of the Laplace Transform function.
This function shall be written in C++

Next input parameters are optional: each one can contain the character "n" to mean that user doesn't want to provide it.

szero: string
It contains

- a parameter, interpreted as integer, specifying if the transform has a singularity at zero.
  IT CAN BE 0 (there is nott) OR GREATER (there is).

or

- the character "n", to mean that the user does not want to provide it.
  It will be posed to the default value (`szero`= 0).

[1]See [2] to know about the class `T<double>`, that extends the C++ type `double`.

pcoeff: string
it contains

- a parameter, interpreted as integer, specifying the printing or not of the coefficients in a file at the end of work.
IT CAN BE 0 (do not print) OR GREATER (print).

or

- the character “n”, specifying that the user does not want to provide it.
It will be posed to the default value (`pcoeff`$= 0$).

sigma0 : string
It contains

- the abscissa of convergence of f: it will be interpreted as double precision.
- If it is less than zero, it will be posed to zero.

or

- the character “n”, specifying that the user does not want to provide it.
It will be posed to the default value (`sigma0`$= 1$).

tol : string
It contains

- the required accuracy on f, interpreted as double precision;
  - if it is greater than 1 it will be posed to the default value ($tol = 10^{-3}$);
  - if it is less than the machine precision (single precision), it will be posed to the value of the machine precision.

or

- the character “n”, specifying that the user does not want to provide it.
It will be posed to the default value (`tol`$= 10^{-3}$).

nmax: string
It contains

- the maximum number of Laguerre series terms, interpreted as integer.
  - if $nmax < 8$, it is posed equal to a default value (`nmax` $= 2000$)
  - if $nmax > MaxLength = 5000$, it is posed at `MaxLength`$= 5000$.

or

- the character “n”, specifying that the user does not want to provide it. It will be posed equal to the default value (`nmax`= 2000).

4.2 Output Parameters

szero:	string It contains the value actually used by the software in computation, it is interpreted as integer.
pcoeff:	string It contains the value actually used by the software in computation, it is interpreted as integer.
sigma0 :	string It contains the value actually used by the software in computation, it is interpreted as double precision.
tol :	string It contains the value actually used by the software in computation, it is interpreted as double precision.
nmax:	string It contains the value actually used by the software in computation, it is interpreted as integer.
ILf :	double precision It contains the computed value of f at x.
absesterr :	double precision It contains the estimate of the absolute error on f.
relesterr:	double precision It contains the estimate of the relative error on f.
flag:	integer (DIAGNOSTIC) It contains an information on the obtained accuracy.
nopt:	integer (DIAGNOSTIC) It contains the found “optimal” number of Laguerre series terms.
ncalc	integer (DIAGNOSTIC) It contains the total number of terms calculated by the software to find the “optimal” one.

4.3 Function Return Value

Function return:	integer. the function returns a diagnostic on the input data. if it is equal to 1, it means that the routine ended without any output because the input x was less than 0.

5 Calling sequence

The variable declarations should be as follows:

```
double      x;
T<double>   (*fz)(T<double>);
char        szero[];
char        pcoeff[];
char        sigma0[];
char        tol[];
char        nmax[];
double      ILf;
double      absesterr;
double      relesterr;
int         flag;
int         nopt;
int         ncalc;
int         ierr;
```

The RELIADIFF call is then:

```
 ierr= RELIADIFF (x, fz, pcoeff, sigma0, tol, nmax, szero, &ILf,
&absesterr, &relesterr, &flag, &nopt, &ncalc);
```

6 An example of Calling Program

A sample calling program to invert the function

$$F(z) = \frac{1}{z^4 - a^4}, \qquad a = \frac{3}{5}$$

at the point $x = 2.5$, without giving any optional input and printing results at screen is listed below.

```
#include "RELIADIFF.h"
T<double> fzTest(T<double> z) {
        return 1./(pow(z,4)−pow(3./5.,4));
}
int main(int argc,char **argv){
/*RELIADIFF INPUT*/
          double  x=2.5;  //Inverse Function evaluation point(s)
        T<double> (*fz)(T<double>);  //Function F Pointer

/*RELIADIFF OPTIONAL INPUT: we don't give any*/
        char szero[10]="n";
        char  pcoeff[10]="n";
        char  sigma0[10]="n";
        char   tol[10]="n";
        char  nmax[10]="n";

/*RELIADIFF OUTPUT*/
        double  absesterr; //absolute error estimate
        double  relesterr; //relative error estimate
        double  ILf;  //Inverse Function f computed
        int  flag; //diagnostics on the result
        int  nopt; //diagnostics on the software work
        int  ncalc; //diagnostics on the software work
        int  ierr; //diagnostics on the software work

/*AUXILIARY VARIABLE*/
        char name[20]="";

        fz = fzTest;

        /********************************** CALLING RELIADIFF ****************************************/
        ierr=RELIADIFF(x,fz,szero,pcoeff,sigma0,tol,nmax,&ILf,&absesterr,&relesterr,&flag,&nopt,&ncalc);
        /****************************************************************************************/

        switch(ierr){
        case 1:
                printf("\nx=%f − RELIADIFF stopped: x<0.0!\n It must be x>=0\n\n",x);
                break;
        default:
                printf("Used Tolerance on accuracy: %e;\n",atof(tol));
                printf("Used abscissa of convergence on F: %e;\n",atof(sigma0));
                printf("Used maximum number of Laguerre series coefficients: %d;\n",atoi(nmax));
                printf("Singularity in zero: ");

                if(atoi(szero)) printf(" yes;\n");
                else printf(" no;\n");
                printf("Used Taylor coefficients printed: ");
                if(atoi(pcoeff)){
                        sprintf(name,"coeff_x%.1f.txt",x);
                        printf(" yes, they are in file %s;\n",name);
                }
                else printf(" no;\n");
                printf("\n TABLE\n");
                printf("−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−\n");
                printf("| x | f_comp | estabserr | estrelerr | Nopt | Ncal |FLAG|\n");
                printf("−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−\n");

                printf("| %4.2e | %14.8e | %9.3e | %9.3e | %5d | %5d | %2d |\n", x, ILf, absesterr, relesterr, nopt, ncalc, flag);
                break;
        } /*endswitch*/
        printf("\n************************************************************************\n");
        printf("   WARNING\n");
        printf("−−−−Calculated errors are just errors estimate, they are not errors bounds−−−\n");
        printf("************************************************************************\n");
        printf("\n\nProgram terminated. Please press a key to exit."); getchar();
        return 0;
}
/*END OF MAIN*/
```

6.1 Output of the Calling Program Example

Running the example program users will obtain at screen the following output.

```
Used Tolerance on accuracy: 1.000000e-03;
Used abscissa of convergence on F: 1.000000e+00;
Used maximum number of Laguerre series coefficients: 2000;
Singularity in zero:  no;
Used Taylor coefficients printed:  no;

                  TABLE
---------------------------------------------------------------------------
|     x    |      f_comp     | estabserr | estrelerr |  Nopt |  Ncal |FLAG|
---------------------------------------------------------------------------
| 2.50e+00 | 2.61987232e+00 | 3.210e-04 | 1.225e-04 |    12 |    12 |  0 |

*******************************************************************************
                                    WARNING
----Calculated errors are just errors estimate, they are not errors bounds-----
*******************************************************************************

Program termineted. Please press a key to exit.
```

7 How To run the Calling Program Example

The package directory is C++_files/src and it contains the above Calling Program Example, and two directory to execute it on either a Linux or a Windows system.

See the last section to know the content of the directory.

7.1 On a Linux system, with a g++ compiler installed

To easy obtain the previous output, user can execute the script
CallingProgramExecution-Linux_g++/compiling_a_CallingProgram.sh.
The script will

1. create a RELIADIFF library with the command make
2. compile the calling program,
3. link it to the RELIADIFF library
4. execute the executable file.

7.2 On a Windows system, with Bloodshed DEV-C++ IDE v. 4.9.9.2 installed

(To download DEV, visit: `http://www.bloodshed.net/dev/devcpp.html`[2])

To easy obtain the previous output, user can open the file
CallingProgramExecution-Windows_DevC++5/ExampleCallingRELIADIFFproject.dev
then click on Execute → Compile & Run.
It will create in its directory the following files:

- `ExampleCallingRELIADIFFproject.layout`
- `ExampleCallingRELIADIFFproject.exe` (executable)
- `Makefile.win`
- `phi.o`
- `RELIADIFF.o`
- `SimpleCallingProgramExample.o`

8 Analysis of the Diagnostic parameters

$flag$: it is about the obtained accuracy.

$flag = 0$: corresponds to:
- $absesterr < tol$
- $relesterr < tol$

Both the absolute and the relative error estimates are smaller than tolerance, so the software fully satisfies the required accuracy.
This means that the software can obtain more accurate results if user requires a smaller value for the tolerance.

$flag = 1$: corresponds to the case of output:
- *absesterr* is the minimum obtained value, but greater than tolerance.
- *relesterr* is the minimum obtained value, but greater than tolerance.

This means that within *nmax* terms of the series expansion, the algorithm cannot satisfy the required accuracy. So, it provides the numerical result within the maximum attainable accuracy with no more than *nmax* terms, and *nopt* will be the number of terms at which such minimum is reached (eventually different from *ncalc*, that is the total number of calculated coefficients).
Moreover, this means that the series seems to converge too slowly or to diverge.
So, user can try to obtain a result more accurate tuning some of the optional parameters: $sinf$, $sigma0$, $nmax$.
User is also invited to verify if the Laplace transform function satisfies algorithm's requirements.

[2]Bloodshed Dev-C++ is a full-featured Integrated Development Environment (IDE) for the C/C++ programming language. It uses Mingw port of GCC (GNU Compiler Collection) as its compiler. It creates native Win32 executables, either console or GUI. Dev-C++ can also be used in combination with Cygwin.
Dev-C++ is Free Software (also referred as Open Source), and is written in Delphi [`http://www.bloodshed.net/dev/`].

$flag = 2$: corresponds to the case of output:

- *absesterr* $< tol$.
- *relesterr* is the minimum obtained value, but greater than tolerance.

Only the absolute error estimate is smaller than the user's required accuracy.
This means that within *nmax* terms of the series expansion, the algorithm cannot satisfy the required accuracy. So, it provides the numerical result within the maximum attainable accuracy with no more than *nmax* terms, and *nopt* will be the number of terms at which such minimum is reached (eventually different from *ncalc*, that is the total number of calculated coefficients).
Moreover, this means that the inverse function f rapidly decreases towards zero.
User can try to obtain a result more accurate tuning some of the optional parameters: $sinf$, $sigma0$, *nmax*.

$flag = 3$: corresponds to the case of output:

- *absesterr* is the minimum obtained value, but greater than tolerance.
- *relesterr* $< tol$.

Only the relative error estimate is smaller than the user's required accuracy.
This means that within *nmax* terms of the series expansion, the algorithm can satisfy the required accuracy, but not for the relative error. This means also that the inverse function f increases rapidly.

- *RETURN VALUE*:

  $= 1$: $x < 0$, the run stopped without working.

  $= 0$: RELIADIFF worked properly.

- *ncalc/nopt*:
  - *ncalc* is the maximum number of terms of the Laguerre series expansion calculated by the algorithm.
  - *nopt* is the number of terms of the Laguerre series expansion that gives the numerical result within the maximum attainable accuracy, less or equal to *nmax* (the required maximum number of terms).

  You can find one of three cases:

  a. *nopt* $=$ *ncalc* $<$ *nmax*

     The computed value of the inverse Laplace function agrees with the true one within $\log(tol)$ significant and decimal digits. This value is obtained calculating *nopt* terms of the Laguerre series expansion. It should correspond to $flag = 0$ or $flag = 3$.

  b. *nopt* $<$ *ncalc* $=$ *nmax*

     Within *nmax* terms of the Laguerre series expansion, the algorithm cannot satisfy the user's required accuracy, and the series seems to diverge. The algorithm provides a numerical result within the maximum attainable accuracy, and *nopt* is the number of terms at which such maximum is reached. It should correspond to $flag = 1$ or $flag = 2$.

  c. *nopt* $=$ *ncalc* $=$ *nmax*

     This occurs if:

     Within *nmax* terms of the Laguerre series expansion, the algorithm cannot satisfy the user's required accuracy, but the series could converge, even if very slowly, or diverge: the algorithm provides numerical result with the maximum attainable accuracy reached within *nmax* terms of the Laguerre series expansion. If the series diverges, *nopt* accidentally corresponds to *nmax*. It should correspond to $flag = 1$ or $flag = 2$.

or if:

the algorithm can satisfy the user's required accuracy, within exactly *nmax* terms of the Laguerre expansion, so the series seems to converge, even if quite slowly: the algorithm provides numerical result within the required accuracy, reached within *nmax* terms of the Laguerre series expansion. It should be $flag = 0$ or $flag = 3$.

9 Remarks on the behavior

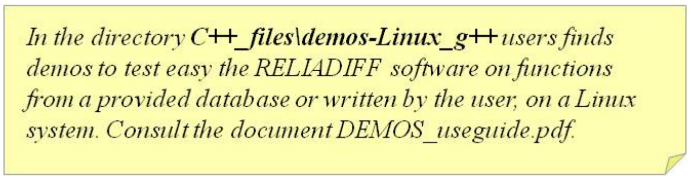
(1) The algorithm works optimally if F is

- such that it can be expressed as $F(z) = z^{-1}G(z)$, where G is analytic at infinity,
- without a singularity at zero neither a singularity at infinity,
- with abscissa of convergence $sigma0$,
- such that it may be evaluated on the real axis with a preassigned limited precision, at most equals to the machine precision.

(2) The algorithm can work good even if:

- F has a singularity at zero,
- there is a small error on the estimate of $sigma0$.

(3) The algorithm works optimally if the Inverse function (f) is infinitely differentiable for all $x > 0$. If user knows its expression, it is reasonable to verify this requirement.



In the directory ***C++_files\demos-Linux_g++*** *users finds demos to test easy the RELIADIFF software on functions from a provided database or written by the user, on a Linux system. Consult the document DEMOS_useguide.pdf.*

10 Content of the directory C++_files/src

The package directory is C++_files/src and it contains the following files:

`RELIADIFF.c` main routine for the inversion.

`phi.c` containing the routines needed to compute the *Phi* function [1].

`RELIADIFF.h` containing some header inclusions (*stdio*, *math*, *tadiff* ...), a constant definition and the prototypes of the function defined in `phi.c` and `RELIADIFF.c`.

`fadbad` a directory containing the software `FADBAD/TADIFF` headers (authors Ole Stauning and Claus Bendtsen [2]), used to implement the Algorithmic Differentiation.

`Makefile` to create a library to link to when compiling an application calling RELIADIFF (on a Linux system).

- The command `make` creates the library `libreliadiff.a`
- The command `make clean` deletes the `libreliadiff.a` file

`SimpleCallingProgramExample.c` containing a simple example of calling program for the RELIADIFF routine.

`CallingProgramExecution-Linux_g++` a directory containing a shell script to compile and execute the provided example of calling program, on a Linux system with a g++ compiler.

- `compiling_a_CallingProgram.sh` script containing an example of how to compile and execute a C/C#/C++ code using the RELIADIFF routine.

`CallingProgramExecution-Windows_DevC++5` a directory containing a DEV-C++ project to compile and execute the provided example of calling program, on a Windows system with *Bloodshed DEV-C++ IDE v. 4.9.9.2* installed.

- `ExampleCallingRELIADIFFproject.dev` dev-project containing an example of how to compile and execute a C/C#/C++ code using the RELIADIFF routine. Option given to the project:
  - **Type**: Win32 console
  - **Executable Output Directory**:
    `..\CallingProgramExecution-Windows_DevC++5`
  - **Object File Output Directory**:
    `..\CallingProgramExecution-Windows_DevC++5`

References

[1] A. Murli, L. D'Amore, V. Mele, R. Campagna, *ReLIADiff. A C++ Software Package For Real Laplace transform Inversion based on Algorithmic Differentiation*, ACM Transactions on Mathematical Software, 0, 0, Article 0 (0000), 20 pages.

[2] C. Bendtsen, O. Stauning, *Tadiff, a flexible C++ package for Automatic Differentiation using Taylor series expansion*, technical report IMM-REP-1997-07, Department of Mathematical Modelling, Technical University of Denmark, 2800 Lyngby, Denmark, 1997.