User Manual for *Algorithm 1046: An improved recurrence method for the scaled complex error function*

Ian Thompson

January 2025

This file contains details of the code and other supplementary materials that accompany the article. All Fortran code is compliant with the 2008 standard. The *erfs* algorithm is in module erfs_container.f90. It can be accessed by a use statement and procedure call (see erfs_simple_test.f90, below). The algorithm and testing programs can be compiled using a makefile or a shell script (see below). Most instructions that follow are only needed for reproducing the tests described in Section 8 of the paper.

- Generation of test data
- Example data files
- Fortran code included in the supplement
- Scripts included in the supplement
- Other authors' code

Generation of test data

Place the three files listed below in the same folder. If using the Standard Worksheet Interface (as opposed to command line Maple), set the current directory to the folder containing the files. See ?currentdir in Maple.

- makeMA_package.mpl (Maple script)

  Generates a package called MachineArithmetic. Needs to be executed (once only) via the command

  ```
  read( "makeMA_package.mpl" ) ;
  ```

  before gen_test_data2.mpl can be used.

- gtd_procs2.mpl (Maple script)

  Contains some procedure code used by gen_test_data2.mpl.

- gen_test_data2.mpl (Maple script)

  Generates test data. Edit the settings at the top of the script and then in Maple execute

  ```
  read( "gen_test_data2.mpl" ) ;
  ```

  to generate a grid file. It is initially set to produce a small (101x101) rectangular grid with $0 \leq \Re z \leq 100$ and $0 \leq \Im z \leq 100$ and store this in the file 'small_grid.dat'. If

machine number mode is set to false, the output is formatted in four columns: $\Re z$, $\Im z$, $\Re w(z)$ and $\Im w(z)$. If machine number mode is set to true then there are eight columns of output: first the mantissa and exponent for $\Re z$, then the same for $\Im z$, etc. The testing program erfs_grid_test.f90 can detect and read either format.

Reproducing data from the paper

The following settings may be used to generate the grid data files used in section 8 of the paper. The meaning of each setting is described in gen_test_data2.mpl.

For all grids:

```
machine_number_mode := true :
old_nag_bug         := false :
d                   := 20    :
md                  := 4000 :
randomise_values    := false :
c1_pts              := 601 :
c2_pts              := 601 :
```

Grid R:

```
plr                 := false :
c1_bnds             := [  0 , 100.0 ] :
c2_bnds             := [  0 , 100.0 ] :
c1_bnds_include     := [ true , true ] :
c2_bnds_include     := [ true , true ] :
c1_lg               := false :
c2_lg               := false :
rl_clip             := -1  :
rl_clip_op          := `<` :
```

Grid L:

```
plr                 := false :
c1_bnds             := [  -20 , 4 ] :
c2_bnds             := [  -20 , 4 ] :
c1_bnds_include     := [ true , true ] :
c2_bnds_include     := [ true , true ] :
c1_lg               := true :
c2_lg               := true :
rl_clip             := -1  :
rl_clip_op          := `<` :
```

Grid O:

```
plr                 := true :
c1_bnds             := [  6.209 , 100.0 ] :
c2_bnds             := [  0 , Pi / 2 ] :
c1_bnds_include     := [ false , true ] :
c2_bnds_include     := [ true  , true ] :
```

```
c1_lg               := false :
c2_lg               := false :
rl_clip             := -1  :
rl_clip_op          := `<` :
```

Grid I:

```
plr                 := true :
c1_bnds             := [  0 , 6.209 ] :
c2_bnds             := [  0 , Pi / 2 ] :
c1_bnds_include     := [ false , false ] :
c2_bnds_include     := [ true  , true ] :
c1_lg               := false :
c2_lg               := false :
rl_clip             := 2.542  :
rl_clip_op          := `<` :
```

Grid T:

```
plr                 := true :
c1_bnds             := [  0 , 6.209 ] :
c2_bnds             := [  0 , Pi / 2 ] :
c1_bnds_include     := [ false , false ] :
c2_bnds_include     := [ true  , true ] :
c1_lg               := false :
c2_lg               := false :
rl_clip             := 2.542  :
rl_clip_op          := `>=` :
```

Potential issues

– *Double precision underflow.*
The set of values that can be handled by Maple includes numbers that do not possess double precision representations, because their magnitude is too small. Such values frequently appear when a grid includes z values on the real line, because there the real part of $w(z)$ is $exp(-z^2)$. The presence of such values in a data file may cause an underflow warning when the file is read by erfs_grid_test.f90, but has no other adverse effects.

– *Double precision overflow.*
The set of values that can be handled by Maple includes numbers that do not possess double precision representations, because their magnitude is too large. Such values may appear when a grid includes z values with arguments in the range $(-3\pi/4, -\pi/4)$ where the function $w(z)$ grows exponentially with $\mid z \mid$. If $\mid z \mid$ or (more likely) $\mid w(z) \mid$ is too large to possess a double precision representation, then the pair $z, w(z)$ is discarded; it is not written to the output file.

– *Maple overflow*
If z or (more likely) $w(z)$ causes an overflow in Maple, then the pair z, $w(z)$ is discarded; it is not written to the output file. This issue is rare, as the range of values

that can be handled by Maple is very large (see ?programmingguide,chapter07; in Maple), but it may occur if a data grid includes points where $|\, z \,|$ is very large and $\arg z$ lies in the range $(-3\pi/4, -\pi/4)$.

– *Too many digits used.*
The maximum number of (decimal) digits used by Maple when running gen_test_data2.mpl is limited by the parameter *md*. If Maple cannot achieve the required accuracy when computing $w(z)$ without exceeding this limit, the pair $z, w(z)$ will be discarded; it is not written to the output file. The issue may be resolved by increasing *md* and rerunning. It should be noted that the maximum number of digits that can be used by Maple is system dependent; it can be obtained by executing *kernelopts*(*maxdigits*). Allowing the number of digits used to approach this value leads to extremely poor performance, but in practice *md* can be much smaller. Data for the paper was produced with $md = 4000$, and this issue never arose.

Example data files

– z_vals.dat

A simple data file that is read by erfs_simple_test.f90 and erfs_comparison_test.f90. This must be placed in the same directory as the executables. Each line should contain two floating point values; the first is the real part of z and the second is the imaginary part. For example, if the contents of the file are

```
0 0
1 0
0 1
```

then the programs will compute $w(0)$, $w(1)$ and $w(i)$.

– small_grid.dat

A small (101×101) data file generated by gen_test_data2.mpl running in machine number mode. Regularly spaced data, with $0 \le \Re z \le 100$ and $0 \le \Im z \le 100$.

– small_grid_nm.dat

As above, but generated with gen_test_data2.mpl not running in machine number mode.

Fortran code included in the supplement

– ab_initio_wrapper.f90 (Fortran module)

An interface module for use with the MIT Ab-Initio group's Faddeeva C code (see below). Post Fortran 95 extensions used:

* use , intrinsic statement (Fortran 2003)
* bind(c) attribute (Fortran 2003)
* import statement (Fortran 2003)
* value attribute (Fortran 2003)

– azah.f90 (Fortran module)

  A module containing a Fortran implementation of the algorithm developed by Al Azah and Chandler-Wilde (2021). Post Fortran 95 extensions used:

  * Brackets [ ] for array constructor (Fortran 2003)
  * Complex part designators $\%re$ and $\%im$ (Fortran 2008)
  * Various intrinsic functions in constant expressions (Fortran 2003)

– abrarov.f90 (Fortran module)

  A module containing a Fortran implementation of the algorithm developed by Abrarov et al. (2018). Post Fortran 95 extensions used:

  * Brackets [ ] for array constructor (Fortran 2003)
  * Complex part designators $\%re$ and $\%im$ (Fortran 2008)
  * Various intrinsic functions in constant expressions (Fortran 2003)

– alg363.f90 (Fortran module)

  A module containing a modern Fortran implementation of Gautschi's original algorithm 363 with some minor improvements (see comments in the code itself). Post Fortran 95 extensions used:

  * Various intrinsic functions in constant expressions (Fortran 2003)

– erfs_container.f90 (Fortran module)

  A module containing a Fortran implementation of the *erfs* algorithm described in the paper. Post Fortran 95 extensions used:

  * Brackets [ ] for array constructor (Fortran 2003)
  * Various intrinsic functions in constant expressions (Fortran 2003)
  * Intrinsic function Gamma (Fortran 2008)
  * Complex part designators $\%re$ and $\%im$ (Fortran 2008)

– erfs_simple_test.f90 (Fortran program)

  A simple program that computes and displays $w(z)$ for each z value found in the file z_vals.dat, which must be located in the same directory as the executable. Post Fortran 95 extensions used:

  * newunit specifier in open statement (Fortran 2008)

– erfs_comparison_test.f90 (Fortran program)

  A simple program that reads z values from the file z_vals.dat, which must be located in the same directory as the executable and computes corresponding values for $w(z)$ by various algorithms. Requires the preprocessor flag (usually *-fpp* or *-cpp*). By default, the algorithms due to Gautschi, Al Azah & Chandler-Wilde and Abrarov et al. are included, alongside *erfs*. Others can be added by following the instructions in the final section of this file. Post Fortran 95 extensions used:

  * newunit specifier in open statement (Fortran 2008)

– erfs_grid_test.f90 (Fortran program)

A Fortran program to test *erfs* against other algorithms, using data grids generated by gen_test_data2.mpl (see above). Requires the preprocessor flag (usually *-fpp* or *-cpp*). By default, the algorithms due to Gautschi, Al Azah & Chandler-Wilde and Abrarov et al. are included, along with *erfs*. Others can be added by following the instructions in the final section of this file. The executable reads up to two arguments from the command line. The first is mandatory and provides the name of the data file. If this includes a path name, it may be advisable to use quotes, for example

```
erfs_grid_test "path/file.dat"
```

rather than

```
erfs_grid_test path/file.dat
```

The second argument is optional, and specifies the number of times to repeat calculations in the timing tests. The default value is 100; to use a different value use

```
erfs_grid_test file.dat n
```

where n is a positive integer. In addition to displaying results on the screen, this program also writes its timing results into a file 'erfs_grid_test_record.dat' in a format that is useful for pasting into a spreadsheet. Post Fortran 95 extensions used:

* use , intrinsic statement (Fortran 2003)
* ISO_fortran_env intrinsic module (Fortran 2003)
* IEEE_exceptions intrinsic module (Fortran 2003)
* IEEE_set_halting_mode, IEEE_set_flag, IEEE_get_flag, IEEE_underflow, IEEE_overflow (Fortran 2003)
* allocatable scalar character variables (Fortran 2003)
* compiler_version (Fortran 2008)
* command_argument_count (Fortran 2003)
* get_command_argument (Fortran 2003)
* newunit specifier in open statement (Fortran 2008)
* Complex part designators $\%re$ and $\%im$ (Fortran 2008)
* 64 bit integer arguments to system_clock intrinsic (Fortran 2003)

Scripts included in the supplement

– makefile (unix make file)

A makefile that compiles the testing programs with the algorithms due to Gautschi, Al Azah & Chandler-Wilde and Abrarov et al. and will automatically detect and include the additional codes discussed in the section below. It is set to use gfortran, but can be changed to another compiler by editing the first line. The makefile must be manually edited to include the NAG library. The relevant lines for this are 67-74. The flag *-DNAG* must be added to instruct the preprocessor

to include the relevant code for the NAG library in erfs_comparison_test.f90 and erfs_grid_test.f90. Additional flags are needed to enable the compiler to locate the .mod files for the NAG library and to load the library itself. These are system dependent.

- simple_makefile (unix make file)

  A simple makefile that compiles the test programs with the algorithms due to Gautschi, Al Azah & Chandler-Wilde and Abrarov et al. It is set to use gfortran, but can be changed to another compiler by editing the first line.

- compile (tcsh script)

  A script that uses nagfor or gfortran to compile the test programs with the algorithms due to Gautschi, Al Azah & Chandler-Wilde and Abrarov et al. This script will also detect and compile the additional codes discussed in the section below. The script assumes the flags needed for the NAG library are

  ```
  -I${NAGINCLUDEPATH} -lnag
  ```

  These may need to be edited depending on system configuration.

- simple_compile (shell script)

  A simple script that uses gfortran to compile the test programs with the algorithms due to Gautschi, Al Azah & Chandler-Wilde and Abrarov et al.

- windows_compile.bat (DOS batch file)

  A simple script that uses gfortran to compile the test programs, with the algorithms due to Gautschi, Al Azah & Chandler-Wilde and Abrarov et al. Poppe & Wijers, Zaghloul and Johnson.

- simple_windows_compile.bat (DOS batch file)

  A simple script that uses gfortran to compile the testing programs, with the algorithms due to Gautschi, Al Azah & Chandler-Wilde and Abrarov et al.

Other authors' code

Code for three additional algorithms is included in the folder 'other_authors_code'. To include these algorithms in test data, simply copy the source files into the folder containing the test programs and re-execute the makefile or compile script (not the simple versions). The additional algorithms will be detected and included automatically; the makefile or script will insert appropriate preprocessor flags as described below.

- 680.f (Fortran 77 subroutine)

  The original code for TOMS algorithm 680, downloaded from the CALGO. This file is unchanged aside from the bug fixes given in Table 4 of the paper. To include this algorithm, the testing programs erfs_comparison_test.f90 and erfs_grid_test.f90 must be compiled with the preprocessing flag *-DPOP*.

- set_rk.f90 (Fortran module)

- rk_erfcx_Cody.f90 (Fortran module)
- Faddeyeva_v2_mod_rk.f90 (Fortran module)

  The code for TOMS algorithm 916, downloaded from the CALGO. This code has been edited as per the author's instructions (see comments in Faddeyeva_v2_mod_rk.f90) to disable computation of derivatives and increase performance. To include this algorithm, the testing programs erfs_comparison_test.f90 and erfs_grid_test.f90 must be compiled with the preprocessing flag *-DZAG*.
- Faddeeva.c (C code)
- Faddeeva.h (C header file)
