Faddeeva Package

From AbInitio

Steven G. Johnson (http://math.mit.edu/~stevenj) has written free/open-source C++ code (with wrappers for C, Matlab, GNU Octave, Python, Scilab, and Julia) to compute the various error functions of arbitrary complex arguments. In particular, we provide:

- **w**, the Faddeeva function $w(z) = e^{-z^2}\mathrm{erfc}(-iz)$, where erfc is the complementary error function.
- **erf**, the error function $\mathrm{erf}(z)$
- **erfc**, the complementary error function $\mathrm{erfc}(z) = 1 - \mathrm{erf}(z)$
- **erfcx**, the scaled complementary error function $\mathrm{erfcx}(z) = e^{z^2}\mathrm{erfc}(z) = w(iz)$
- **erfi**, the imaginary error function $\mathrm{erfi}(z) = -i\mathrm{erf}(iz)$
- **Dawson**, the Dawson function $\mathrm{Dawson}(z) = \frac{\sqrt{\pi}}{2}e^{-z^2}\mathrm{erfi}(z)$

Given the Faddeeva function $w(z)$ and the other complex error functions, one can also easily compute Voigt functions, Fresnel integrals, and similar related functions as well. In benchmarks of our code, we find that it is comparable to or faster than most competing software for these functions in the complex plane (but we also have special-case optimizations for purely real or imaginary arguments), and we find that the accuracy is typically at at least 13 significant digits in both the real and imaginary parts.

Because all of the algorithms are based on algorithms for the Faddeeva function, we call this the **Faddeeva Package**.

Contents

Download

Download the source code from:

- http://ab-initio.mit.edu/Faddeeva.cc and http://ab-initio.mit.edu/Faddeeva.hh (updated 18 December 2012)

See also below for wrappers to call the Faddeeva package from other languages.

Usage

To use the code, include the `Faddeeva.hh` header file:

```
#include "Faddeeva.hh"
```

and compile and link the `Faddeeva.cc` source code. You can then call various functions. For example:

```
extern std::complex<double> Faddeeva::w(std::complex<double> z, double rele
```

This function `Faddeeva::w(z, relerr)` computes $w(z)$ to a desired relative error `relerr`.

Omitting the `relerr` argument, or passing `relerr=0` (or any `relerr` less than machine precision $\varepsilon \approx 10^{-16}$), corresponds to requesting machine precision, and in practice a relative error $< 10^{-13}$ is usually achieved. Specifying a larger value of `relerr` may improve performance for some z (at the expense of accuracy).

Similarly, the erf, erfc, erfcx, erfi, and Dawson functions are computed by calling:

```
extern std::complex<double> Faddeeva::erf(std::complex<double> z, double re
extern std::complex<double> Faddeeva::erfc(std::complex<double> z, double r
extern std::complex<double> Faddeeva::erfcx(std::complex<double> z, double
extern std::complex<double> Faddeeva::erfi(std::complex<double> z, double r
extern std::complex<double> Faddeeva::Dawson(std::complex<double> z, double
```

Since these functions are purely real for real arguments $z=x$, we provide the following specialized interfaces for convenience (and a slight performance gain, although the complex functions above automatically execute specialized code for purely real arguments):

```
extern double Faddeeva::erf(double x);
extern double Faddeeva::erfc(double x);
extern double Faddeeva::erfcx(double x);
extern double Faddeeva::erfi(double x);
extern double Faddeeva::Dawson(double x);
```

(These functions always compute to maximum accuracy, usually near machine precision.)

It is also sometimes useful to compute Im[$w(x)$] for real x, since $\mathrm{Im}[w(x)] = e^{-x^2}\mathrm{erfi}(x)$ in that case (like the Dawson function but without the $\sqrt{\pi}/2$ factor). [Note that Re[$w(x)$] is simply $\exp(-x^2)$ for real x.] Im[$w(x)$] can be computed efficiently to nearly machine precision by calling:

```
extern double Faddeeva::w_im(double x);
```

Wrappers: C, Matlab, GNU Octave, Python, Scilab, Julia

Wrappers are available for this function in other languages.

- C: Download the files http://ab-initio.mit.edu/Faddeeva.c and http://ab-initio.mit.edu/Faddeeva.h (in *addition* to Faddeeva.cc (http://ab-initio.mit.edu/Faddeeva.cc) from above) to obtain a pure C version (you do *not* need a C++ compiler), using C99 complex numbers. The complex functions are `Faddeeva_erf(double complex z, double relerr)` etc. instead of `Faddeeva::erf`, and the real-argument versions are `Faddeeva_erf_re(double x)` etc. (Note that in gcc you may need to compile with the `-std=c99` flag to enable C99 support.)
- Matlab (also available here (http://www.mathworks.com/matlabcentral/fileexchange/38787)): We provide source code for compiled Matlab plugins (MEX files) to interface all of the error functions above from Matlab.
  - Download the code and documentation from: http://ab-initio.mit.edu/Faddeeva-MATLAB.zip (a zip file)
  - The provided functions are called `Faddeeva_w`, `Faddeeva_erf`, `Faddeeva_erfc`, `Faddeeva_erfi`, `Faddeeva_erfcx`, and `Faddeeva_Dawson`, equivalent to the C++ functions above. All have usage of the form `w = Faddeeva_w(z)` [or `w = Faddeeva_w(z, relerr)` to pass the optional relative error], to compute the function value from an array or matrix `z` of complex (or real) inputs.
  - For convenience, a script to compile all of the plugins using the mex command (http://www.mathworks.com/help/matlab/ref/mex.html) in Matlab is included. Assuming you have a C++ compiler installed (and have run `mex -setup` to tell Matlab to use it), you can simply run the `Faddeeva_build.m` script in Matlab to compile all of the Faddeeva functions.
  - Install the resulting `*.mex*` files, along with the `*.m` help files, into your Matlab path (http://www.mathworks.com/help/matlab/ref/path.html)

- GNU Octave: Similar to Matlab, above, we provide source code for compiled GNU Octave plugins (.oct files (http://www.gnu.org/software/octave/doc/interpreter/Oct_002dFiles.html)) for all of the error functions above. (*Note:* our code for complex-argument erf, erfc, erfcx, erfi, and dawson functions has been merged into Octave (http://hg.savannah.gnu.org/hgweb/octave/graph/9811b32b645e) and should be included in a future release.)
  - Download the code and documentation from: http://ab-initio.mit.edu/Faddeeva-octave.tgz (a gzipped tar file)
  - The provided functions are called `Faddeeva_w`, `Faddeeva_erf`, `Faddeeva_erfc`, `Faddeeva_erfi`, `Faddeeva_erfcx`, and `Faddeeva_Dawson`,

with usage identical to the Matlab plugins above.
    - A Makefile is included. Assuming you have a C++ compiler and the `mkoctfile` command installed (`mkoctfile` comes with Octave, possibly in an `octave-devel` or similarly named package in GNU/Linux distributions), you can simply run `make` to compile the plugins, and `sudo make install` to install them system-wide (assuming you have system administrator privileges); otherwise put the compiled `.oct` files somewhere in your octave path (http://www.gnu.org/software/octave/doc/interpreter/Manipulating-the-load-path.html) .

- Python: Our code is used to provide `scipy.special.erf`, `scipy.special.wofz`, and the other error functions in SciPy starting in version 0.12.0 (see here (https://github.com/scipy/scipy/commit/ed14bf0)).

- Scilab has a patch (http://bugzilla.scilab.org/show_bug.cgi?id=6092) to call the Faddeeva Package which should hopefully be incorporated into the next Scilab release.

- Julia (http://julialang.org/) uses the Faddeeva Package to provide its complex `erf`, `erfc`, `erfcx`, `erfi`, and `dawson` functions.

Algorithms

Our implementation uses a combination of different algorithms, mostly centering around computing the Faddeeva function $w(z)$.

To compute the Faddeeva function for sufficiently large $|z|$, we use a continued-fraction expansion for $w(z)$ similar to those described in

- Walter Gautschi, "Efficient computation of the complex error function (http://dx.doi.org/10.1137/0707012) ," *SIAM J. Numer. Anal.* **7** (1), pp. 187–198 (1970). G. P. M. Poppe and C. M. J. Wijers, "More efficient computation of the complex error function (http://dx.doi.org/10.1145/77626.77629) ," *ACM Trans. Math. Soft.* **16** (1), pp. 38–46 (1990); this is TOMS Algorithm 680 (http://www.netlib.org/toms/680) .

Unlike those papers, however, we switch to a completely different algorithm for smaller $|z|$ or for z close to the real axis:

- Mofreh R. Zaghloul and Ahmed N. Ali, "Algorithm 916: Computing the Faddeyeva and Voigt Functions (http://dx.doi.org/10.1145/2049673.2049679) ," *ACM Trans. Math. Soft.* **38** (2), 15 (2011). Preprint available at arXiv:1106.0151 (http://arxiv.org/abs/1106.0151) .

(I initially used this algorithm for all z, but the continued-fraction expansion turned out to be faster for larger $|z|$. On the other hand, Algorithm 916 is competitive or faster for smaller $|z|$, and appears to be significantly more accurate than the Poppe & Wijers code (http://www.netlib.org/toms/680) in some regions, e.g. in the vicinity of $|z|=1$ [although comparison with other compilers suggests that this may be a problem specific to gfortran]. Algorithm 916 also has better relative accuracy in Re[z] for some regions near the real-z axis. You can switch back to using Algorithm 916 for all z by changing

`USE_CONTINUED_FRACTION` to `0` in the code.)

Note that this is SGJ's **independent re-implementation** of these algorithms, based on the descriptions in the papers *only*. In particular, we did not refer to the authors' Fortran or Matlab implementations (respectively), which are under restrictive "semifree (http://www.gnu.org/philosophy/categories.html) " ACM copyright terms (http://www.acm.org/publications/policies/softwarecrnotice) and are therefore unusable in free/open-source software.

Algorithm 916 requires an external complementary error function erfc(x) function for *real* arguments x to be supplied as a subroutine. More precisely, it requires the scaled function erfcx(x) = e^{x^2}erfc(x). Here, we use an erfcx routine written by SGJ that uses a combination of two algorithms: a continued-fraction expansion for large x and a lookup table of Chebyshev polynomials for small x. (I initially used an erfcx function derived from the DERFC routine in SLATEC, modified by SGJ to compute erfcx instead of erfc, but the new erfcx routine is much faster, and also seems to be faster than the calerf (http://www.netlib.org/specfun/erf) rational-Chebyshev code by W. J. Cody.)

Similarly, we also implement special-case code for real z, where the imaginary part of w is Dawson's integral. Similar to erfcx, this is also computed by a continued-fraction expansion for large $|x|$, a lookup table of Chebyshev polynomials for smaller $|x|$, and finally a Taylor expansion for very small $|x|$. (This seems to be faster than the dawsn function (http://www.netlib.org/cephes/doubldoc.html#dawsn) in the Cephes library, and is substantially faster than the gsl_sf_dawson (http://www.gnu.org/software/gsl/manual/html_node/Dawson-Function.html) function in the GNU Scientific Library.)

The other error functions can be computed in terms of $w(z)$. The basic equations are:

$$\operatorname{erfcx}(z) = e^{z^2}\operatorname{erfc}(z) = w(iz)$$ (scaled complementary error function)

$$\operatorname{erfc}(z) = e^{-z^2}w(iz) = \begin{cases} e^{-z^2}w(iz) & \operatorname{Re} z \geq 0 \\ 2 - e^{-z^2}w(-iz)) & \operatorname{Re} z < 0 \end{cases}$$ (complementary error function)

$$\operatorname{erf}(z) = 1 - \operatorname{erfc}(z) = \begin{cases} 1 - e^{-z^2}w(iz) & \operatorname{Re} z \geq 0 \\ e^{-z^2}w(-iz) - 1 & \operatorname{Re} z < 0 \end{cases}$$ (error function)

$\operatorname{erfi}(z) = -i\operatorname{erf}(iz)$; for **real** x, $\operatorname{erfi}(x) = e^{x^2}\operatorname{Im}[w(x)] = \dfrac{\operatorname{Im}[w(x)]}{\operatorname{Re}[w(x)]}$ (imaginary error function)

$$\operatorname{Dawson}(z) = \frac{\sqrt{\pi}}{2}e^{-z^2}\operatorname{erfi}(z) = \frac{i\sqrt{\pi}}{2}\begin{cases} e^{-z^2} - w(z) & \operatorname{Re} z \geq 0 \\ w(-z) - e^{-z^2} & \operatorname{Re} z < 0 \end{cases}$$; for **real** x,

$$\operatorname{Dawson}(x) = \frac{\sqrt{\pi}}{2}\operatorname{Im}[w(x)]$$ (Dawson function)

Note that we sometimes employ different equations for positive and negative Re(z) in order to avoid numerical problems arising from multiplying exponentially large and small quantities. For erfi and the Dawson function, there are simplifications that occur

for real x as noted. In some cases, however, there are additional complications that require our implementation to go beyond these simple formulas. For erf, large cancellation errors occur in these formulas near $|z|=0$ where $w(z)$ is nearly 1, as well as near the imaginary axis for Re[erf], and in these regimes we switch to a Taylor expansion. Similarly, for the Dawson function we switch to a Taylor expansion near the origin or near the real axis. (Similar problems occur for erfi, but our erfi implementation simply calls our erf code.)

Test program

To test the code, a small test program is included at the end of `Faddeeva.cc` which tests $w(z)$ against several known results (from Wolfram Alpha) and prints the relative errors obtained. To compile the test program, `#define TEST_FADDEEVA` in the file (or compile with `-DTEST_FADDEEVA` on Unix) and compile `Faddeeva.cc`. The resulting program prints `SUCCESS` at the end of its output if the errors were acceptable.

License

The software is distributed under the "MIT License" (also called the *Expat License*), a simple permissive free/open-source license (which is compatible with the GPL and most other licenses (http://www.gnu.org/licenses/license-list.html#Expat)):

> *Copyright © 2012 Massachusetts Institute of Technology*
> *Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:*
> *The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.*
> *THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.*