

Updates on computing the plasma dispersion/complex error function

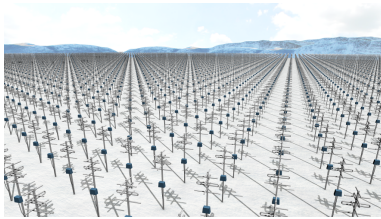
Stephan C. Buchert (scb@irfu.se),
Swedish Institute of Space Physics

18th EISCAT_3D User Meeting 2024-12-04, Kiruna

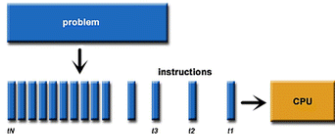
December 4, 2024



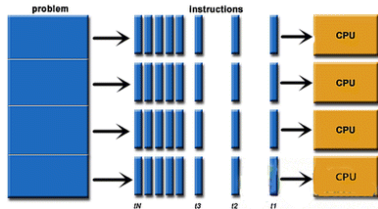
E3D data analysis



Serial operation schematic diagram



Parallel computing



- ▶ E3D: many beams, tristatic volume scattering, . . .
- ▶ $\sim 100 - 1000$ more data to analyse!
- ▶ way to go: parallel computing (?)

Special Function

Numerical calculation of the IS spectrum for Maxwellian distributions rests on either

- 1) the plasma dispersion function, code name `friedconte(z)`:
B.D. Fried, S.D. Conte, The plasma dispersion function. New York Academic Press, 1961.
- 2) complex error function `erfc(z)`:

$$\text{friedconte}(z) = i\sqrt{\pi} \exp(-z^2) (1 + \text{erfc}(iz)) \quad (1)$$

- 3) the Faddeeva function

$$\text{faddeeva}(z) = \exp(-z^2) \text{erfc}(-iz) \quad (2)$$

- 4) the Dawson integral

$$D(x) = \exp(-x^2) \int_0^x t^2 dt \quad (3)$$

Any one of the four functions is needed.

Fried-Conte

```
      SUBROUTINE PLASMA
C*THIS ROUTINE COMPUTES THE COMPLEX PLASMA DISPERSION FUNCTION
C GIVEN BY:
C           Z(S)=I*SQRT(PIE)*EXPC(-S**2)*(1.+ERFC(I*S)
C WHERE:
C           I=SQRT(-1.) ; S=X+I*Y=COMPLEX ARGUMENT
C FOR ABS(Y).GT.1.0, THE CONTINUED FRACTION EXPANSION GIVEN BY FRIED
C AND CONTE (1961) IS USED; WHILE FOR ABS(Y).LE.1.0, THE FOLLOWING
C DIFFERENTIAL EQUATION IS SOLVED:
C           D Z(S)
C           ----- = -2.*(1.+S*Z(S))
C           D S
C SUBJECT TO Z(0)=I*SQRT(PIE)
C
C           "F(K)"=TRUE FREQUENCY.
C           "X(K)"=NORMALIZED FREQUENCY.
C           "SCALEF"=FREQUENCY SCALING FACTOR FOR NORMALIZATION.
C -----
C BY WES SWARTZ
```

- ▶ Arecibo IS analysis written by Wes Swartz
- ▶ FORTRAN, IBM Mainframe
- ▶ Solving the diff. equation is not easy to parallelize :-)

The shortest code

Reference:

- [1] J.A.C. Weideman, "Computation of the Complex Error Function," SIAM J. Numerical Analysis, pp. 1497-1518, No. 5, Vol. 31, Oct., 1994
Available Online: <http://www.jstor.org/stable/2158232>

TABLE 1
Matlab program for computing $w(z) = e^{-z^2} \operatorname{erfc}(-iz)$ using the series (I).

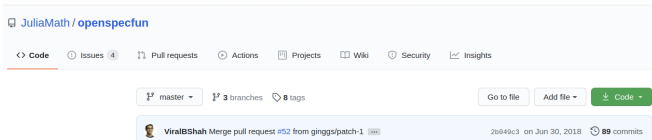
```
function w = cef(z,N)

% Computes the function w(z) = exp(-z^2) erfc(-iz) using a rational
% series with N terms. It is assumed that Im(z) > 0 or Im(z) = 0.

M = 2*N; M2 = 2*M; k = [-M+1:1:M-1]'; % M2 = no. of sampling points.
L = sqrt(N/sqrt(2)); % Optimal choice of L.
theta = k*pi/M; t = L*tan(theta/2); % Variables theta and t.
f = exp(-t.^2).*(L.^2+t.^2); f = [0; f]; % Function to be transformed.
a = real(fft(fftshift(f)))/M2; % Coefficients of transform.
a = flipud(a(2:N+1)); % Reorder coefficients.
Z = (L+i*z)./(L-i*z); p = polyval(a,Z); % Polynomial evaluation.
w = 2*p./(L-i*z).^2+(1/sqrt(pi))./(L-i*z); % Evaluate w(z).
```

- ▶ evaluates at each frequency (`polyval`) → can be done in parallel :-);
- ▶ has been judged to be not very accurate and fast?

The State of the Art



JuliaMath / openspecfun

<> Code Issues (4) Pull requests Actions Projects Wiki Security Insights

master 3 branches 8 tags Go to file Add file - Code -

ViralBShah Merge pull request #52 from gingsg/patch-1 2b049c3 on Jun 30, 2018 89 commits

`erfcx(x) = exp(x^2) erfc(x)` function, for real x , written by Steven G. Johnson, October 2012.

This function combines a few different ideas.

First, for $x > 50$, it uses a continued-fraction expansion (same as for the Faddeeva function, but with algebraic simplifications for $z=i*x$).

Second, for $0 \leq x \leq 50$, it uses Chebyshev polynomial approximations, but with two twists:

a) It maps x to $y = 4 / (4+x)$ in $[0,1]$. This simple transformation, inspired by a similar transformation in the octave-forge/specfun `erfcx` by Soren Hauberg, results in much faster Chebyshev convergence than other simple transformations I have examined.

b) Instead of using a single Chebyshev polynomial for the entire $[0,1]$ y interval, we break the interval up into 100 equal subintervals, with a switch/lookup table, and use much lower degree Chebyshev polynomials in each subinterval. This greatly improves performance in my tests.

- ▶ written by Steven G. Johnson, MIT, in C/C++ (2012)
- ▶ (co-authored FFTW, “photonic crystals”)
- ▶ today active in the Julia language for HPC

Rapid Computation of the Plasma Dispersion Function: Rational and Multi-pole Approximation, and Improved Accuracy

Huasheng Xie^{1,2,*}

¹Hebei Key Laboratory of Compact Fusion, Langfang 065001, China

²ENN Science and Technology Development Co., Ltd., Langfang 065001, China

(Dated: April 30, 2024)

The plasma dispersion function $Z(s)$ is a fundamental complex special integral function widely used in the field of plasma physics. The simplest and most rapid, yet accurate, approach to calculating it is through rational or equivalent multi-pole expansions. In this work, we summarize the numerical coefficients that are practically useful to the community. Besides the Padé approximation to obtain coefficients, which are accurate for both small and large arguments, we also employ optimization methods to enhance the accuracy of the approximation for the intermediate range. The best coefficients provided here for calculating $Z(s)$ can deliver twelve significant decimal digits. This work serves as a foundational database for the community for further applications.

Matlab code for Z fun with optimized J=8 pole	Python code for Z fun with optimized J=8 pole
<pre>function Zeta = zfunJ8(z) Zeta=0 * z; bj(1)=-0.00383968430671409 - 0.0119854387180615i; bj(2)=-0.321597857664957 - 0.218883985607935i; bj(3)=-2.55515264319988 + 0.613958600684469i; bj(4)=-2.7379446984183 + 5.69007914897806i; cj(1)=2.51506776338386 - 1.60713668042405i; cj(2)=-1.68985621846204 - 1.66471695485661i; cj(3)=0.981465428659098 - 1.70017951305004i; cj(4)=0.322078795578047 - 1.71891780447016i; bj(5:8)=conj(bj(1:4)); cj(5:8)=conj(cj(1:4)); idx=(imag(z)>=0); Zeta(~idx)=2i*sqrt(pi)*exp(-z(~idx).^2); for j=1:length(bj) Zeta[idx]=Zeta[idx]+bj(j)/(z(idx)-cj(j)); Zeta(~idx)=Zeta(~idx)+conj(bj(j))/(conj(z(~idx))-cj(j)); end end</pre>	<pre>import numpy as np def zfunJ8(z): Zeta = np.zeros_like(z, dtype=complex) bj = np.array([0.00383968430671409 - 0.0119854387180615i, -0.321597857664957 - 0.218883985607935i, 2.55515264319988 + 0.613958600684469i, -2.7379446984183 + 5.69007914897806i]) cj = np.array([2.51506776338386 - 1.60713668042405i, -1.68985621846204 - 1.66471695485661i, 0.981465428659098 - 1.70017951305004i, -0.322078795578047 - 1.71891780447016i]) bj = np.concatenate((bj, np.conj(bj[:-1]))) cj = np.concatenate((cj, -np.conj(cj[:-1]))) idx = np.imag(z) >= 0 Zeta[~idx] = 2j * np.sqrt(np.pi) * np.exp(-z[~idx]**2) for j in range(len(bj)): Zeta[idx] += bj[j] / (z[idx] - cj[j]) Zeta[~idx] += np.conj(bj[j]) / (np.conj(z[~idx]) - cj[j]) return Zeta</pre>

FIG. 10. Sample code for calculate Z function with optimized $J = 8$ pole for all range of argument z , with max errors of 10^{-6} . One who needs higher accuracy, can use the larger J coefficients, such as $J = 10, 12, 16, 20, 24$.

▶ Matlab and Python not compiled, i.e. not fast?

References

- [9] J. A. C. Weideman. 1994. Computation of the complex error function. *SIAM J. Numer. Anal.* 31, 5, 1497–1518.
- [10] R. N. Franklin, The computation of the plasma dispersion function, *Plasma Phys.* 10, 805 (1968).
- [11] G. Németh, A. Ág, and G. Páris, Two-sided Padé approximations for the plasma dispersion function, *J. Math. Phys.* 22, 1192 (1981).
- [12] B. D. Fried, Burton, C. L. Hedrick and J. McCune, Two-Pole Approximation for the Plasma Dispersion Function, *Physics of Fluids*, 11, 1, 249-252 (1968).
- [13] B. S. Newberger, Efficient numerical computation of the plasma dispersion function, *Comput. Phys. Commun.* 42, 305 (1986).
- [14] A. Tjulín, A. I. Eriksson, and M. André, Physical interpretation of the Padé approximation of the plasma dispersion function, *J. Plasma Phys.* 64, 287 (2000).
- <https://github.com/hsxie/pdrk/>.
- [16] H.S. Xie, BO: A unified tool for plasma waves and instabilities analysis, *Comput. Phys. Comm.* 244 (2019) 343-371. Xie, H. S., Denton, R., Zhao, J. S. and Liu, W, BO 2.0: Plasma Wave and Instability Analysis with Enhanced Polarization Calculations [arXiv:2103.16014](https://arxiv.org/abs/2103.16014) 2021. <https://github.com/hsxie/bo/>.
- [17] Ronnmark, K. 1982. WHAMP - Waves in Homogeneous Anisotropic Multicomponent Plasmas, *Tech. Rep. 179*, Kiruna Geophysical Institute, Kiruna, Sweden.
- [18] P. Hunana, A. Tenerani, G. P. Zank, M. L. Goldstein, G. M. Webb, M. Velli and L. Adhikari, A brief guide to fluid models with anisotropic temperatures Part 2 - Kinetic theory, Padé approximants and Landau fluid closures, 2019.
- [19] G. W. Hammett and W. F. Perkins, *Phys. Rev. Lett.*, 64, 3019 (1990).

Conclusions:

- ▶ Keep track of developments for computing IS spectra *fast*
- ▶ Coordinate efforts by users and EISCAT for E3D data analysis!