```
function [Zp,Z]=zetaph(z,Fn,F,N)
% Hua-sheng XIE, huashengxie@gmail.com, IFTS-ZJU, 2013-05-26 23:37
% Calculate GPDF, Generalized Plasma Dispersion Function, see [Xie2013]
%
% Modify the parameters in the code for your own usage
%
% Ref: [1] H. S. Xie, Generalized Plasma Dispersion Function:
% One-Solve-All Treatment, Visualizations and Application to
% Landau Damping, PoP, 2013. (Or http://arxiv.org/abs/1305.6476)
%
% Examples:
%  1. z=-1:0.01:1; [Zp,Z]=zetaph(z); plot(z,real(Zp),z,imag(Zp));
%  2. z=-1:0.01:1; F = '1./(1+v.^2)/pi'; [Zp,Z]=zetaph(z,0,F);
%     plot(z,real(Z),z,imag(Z));
%  3. [x,y]=meshgrid(-1:0.1:1,-1:0.1:1); z=x+1i*y; [Zp,Z]=zetaph(z,1);
%     subplot(121);surf(x,y,real(Zp)); subplot(122);surf(x,y,imag(Zp));
    if nargin<4, N = 128/4; end

    % Default for usual Plasma Dispersion Function
    if (nargin<2)
       Fn=0;
    end
    if (nargin<3)
       F = 'exp(-v.^2)/sqrt(pi)';
    end

    if(Fn==5)
       N=256*32;
    end

    if(Fn==1) % F=delta(v-vd)
       zd=0;
       [Zp,Z]=zdelta(z,zd);
    elseif(Fn==2) % flat top F=(H(v-va)-H(v-vb))/(vb-va)
       F='heaviside(v-0.5)-heaviside(v+0.5)';
       del=0;
       Z=calZ(z,F,N,del);
       za=-0.5; zb=0.5;
%         Z=(log(z-zb)-log(z-za))./(zb-za);
       Zp=(1./(z-zb)-1./(z-za))./(zb-za);
    elseif(Fn==3) % triangular distribution
%         F=['(heaviside(v+1)-heaviside(v)).*(v+1)-',...
%                '(heaviside(v)-heaviside(v-1)).*(v-1)'];
%         [Fp,F]=calFp(F);
       F='((v>-1)&(v<=0)).*(v+1)-((v>0)&(v<=1)).*(v-1)';
       Fp='((v>-1)&(v<=0))-((v>0)&(v<=1))';
       del=0;
       Z=calZ(z,F,N,del);
       Zp=calZ(z,Fp,N,del);
```

```
    elseif(Fn==4) % gamma distribution
       gn=2; vt=1;
       F=['(gamma(',num2str(gn),')/gamma(',num2str(gn),...
           '-0.5)/(sqrt(pi*',num2str(gn),')*',num2str(vt),...
           ')).*(1+v.^2/(',num2str(gn),'*',num2str(vt),...
           '^2)).^(-',num2str(gn),')'];
       [Fp,F]=calFp(F);
       Z=calZ(z,F,N);
       Zp=calZ(z,Fp,N);
    elseif(Fn==5) % incomplete Maxwellian distribution
       nu=-0.1;
       F=['heaviside(v+',num2str(nu),').*exp(-v.^2)/sqrt(pi)'];
       [Fp,F]=calFp(F);
       Z=calZ(z,F,N);
       Zp=calZ(z,Fp,N);
       Zp=Zp-exp(-nu.^2)/sqrt(pi)./(z-nu); % with correction
    elseif(Fn==6) % slowing down
       % using heaviside() instead of abs() to support complex v in F(v)
       vt=1.0; vc=4;
%         F=['(heaviside(v).*heaviside(',num2str(vc),'-v)./(v.^3+',...
%            num2str(vt),'^3)+(1-heaviside(v)).*heaviside(',...
%            num2str(vc),'+v)./((-v).^3+',num2str(vt),'^3))*3*sqrt(3)*',...
%            num2str(vt),'^2/(4*pi)'];
%         [Fp,F]=calFp(F);
       F=['(((v>=0)&(v<=',num2str(vc),'))./(v.^3+',...
          num2str(vt),'^3)+((v<0)&(v>=-',...
          num2str(vc),'))./((-v).^3+',num2str(vt),'^3))*3*sqrt(3)*',...
          num2str(vt),'^2/(4*pi)'];
       Fp=['(((v>=0)&(v<=',num2str(vc),')).*(-3.*v.^2)./(v.^3+',...
          num2str(vt),'^3).^2+((v<0)&(v>-',...
          num2str(vc),')).*(3.*v.^2)./((-v).^3+',num2str(vt),...
          '^3).^2)*3*sqrt(3)*',num2str(vt),'^2/(4*pi)'];
       Z=calZ(z,F,N);
       Zp=calZ(z,Fp,N);
       Zp=Zp+3*sqrt(3)*vt^2/(4*pi)/(vc^3+vt^3).*(1./(z-vc)-...
           1./(z+vc)); % with correction
    elseif(Fn==0) % for arbitrary analytical input function F
       [Fp,F]=calFp(F);
       Z=calZ(z,F,N);
       Zp=calZ(z,Fp,N);
    else % default Maxwellian
       F = 'exp(-v.^2)/sqrt(pi)';
       [Fp,F]=calFp(F);
       Z=calZ(z,F,N);
       Zp=calZ(z,Fp,N);
    end

end
```

```
function [Fp,F]=calFp(F)
% input as F = '1./(1+v.^2)', calculate the derivative
    syms v;
    Fp=char(diff(eval(F),'v'));
    Fp=strrep(Fp,'*','.*');
    Fp=strrep(Fp,'/','./');
    Fp=strrep(Fp,'^','.^');
end

function [Zp,Z]=zdelta(z,zd)
% for delta distribution function
    if nargin<2, zd = 0; end
    Z=-1./(z-zd);
    Zp=1./(z-zd).^2;
end

function Z=calZ(z,F,N,del)
   if nargin<4, del = 1; end
   Z=hilb(z,F,N,del);
   ind1=find(isnan(Z));
   z1=z(ind1)+1e-10;          % avoid NaN of higher order singular point
   Z(ind1)=hilb(z1,F,N,del);% e.g., z=ia for Lorentzian F = '1./(a^2+v.^2)'
end

function Z=hilb(z,F,N,del,L)
% note: 1. in fact, a_n need calcualte only once for a fixed F, so you can
%   rewrite the code to speed up.
%       2. f(z) in analytic continuation can also be replaced by
%   sum{a_n*rho_n}
%       3. usually, del=1, but for flat-top and triangular distributions,
%   it seems we should set del=0 for correct analytic continuation

    if nargin<5, L = sqrt(N/sqrt(2)); end  % optimal choice of L
%     L=10;
    if nargin<4, del = 1; end

    % 1. Define initial parameters
    Z = zeros(size(z)); % initialize output

    % 2. Calculate
    idx=find(imag(z) == 0);
    idx1=find(imag(z) ~= 0);
    % 2.1 real line
    for id=idx
        n  = [-N:N-1]';                   % Compute the collocation points
        v  = L*tan(pi*(n+1/2)/(2*N));
        FF = eval(F);               % sample the function
        FF(isnan(FF))=0;
        Fz = eval(['@(v)',F]);     % define F(z), for analytic continuation
```

```
        % Weideman95 method to calculate the real line Hilbert transform
        a  = fft(fftshift(FF.*(L-1i*v)));      % These three lines compute
        a  = exp(-1i*n*pi/(2*N)).*fftshift(a);   % expansion coefficients
        a  = flipud(1i*(sign(n+1/2).*a))/(2*N);
        t  = (L+1i*z(id))./(L-1i*z(id)); % The evaluation of the transform
        h  = polyval(a,t)./(t.^N.*(L-1i*z(id)));    % reduces to polynomial
                                        % evaluation in the variable t
        Z(id) = h + 1i.*Fz(z(id));
    end
    % 2.2. upper and lower half plane
    for id=idx1
        M = 2*N;  M2 = 2*M;
        k = [-M+1:1:M-1]';% M2 = no. of sampling points
        theta = k*pi/M; v = L*tan(theta/2);      % define theta & v
        FF = eval(F);              % sample the function
        FF(isnan(FF))=0;
        Fz = eval(['@(v)',F]);     % define F(z), for analytic continuation

        % Weideman94 method to calculate upper half plane Hilbert transform
        W = (L^2+v.^2);                               % default weight function
        FF = FF.*W; FF = [0; FF];                   % function to be transformed
        a = (fft(fftshift(FF)))/M2;                 % coefficients of transform
        a0 = a(1); a = flipud(a(2:N+1));              % reorder coefficients
        z1 = (imag(z(id))>0).*z(id)+(imag(z(id))<0).*conj(z(id));
        t = (L+1i*z1)./(L-1i*z1); p = polyval(a,t); % polynomial evaluation
        h = 1i*(2*p./(L-1i*z1).^2+(a0/L)./(L-1i*z1));   % Evaluate h(f(v))
        % convert upper half-plane results to lower half-plane if necesary
%         Z(id) = h.*(imag(z(id))>0)+conj(h-2i.*Fz(z1)).*(imag(z(id))<0);
        Z(id) = h.*(imag(z(id))>0)+(conj(h)+...
            del*2i.*Fz(z(id))).*(imag(z(id))<0);
    end
    Z=Z.*pi;
end
```