WHAMP quick reference (v 1.4)

A. Vaivads

February 5, 2015

1 `WHAMP` introduction

`WHAMP` calculates the dispersion relation of Waves in Homogeneous Anisotropic Magnetized Plasma. Each plasma component consists of particles with charge q_j, mass m_j, density n_j and normalized distribution function $\tilde{f}^j(\mathbf{v}, \mathbf{r})$. The phase space density f^j can be obtained as $f^j = \tilde{f}^j n^j$. The general distribution function considered has the form of an anisotropic Maxwellian with the possibility to include a drift along the ambient magnetic field and loss cone:

$$
\begin{aligned}
f(v_\perp, v_\parallel) &= \sum_{j=1}^{m} \frac{n^j}{n} \tilde{f}^j = \sum_{j=1}^{m} \frac{n^j}{n} \frac{1}{\left(\sqrt{\pi} V_{th}^j\right)^3} \cdot e^{-\left(\frac{v_\parallel}{V_{th}^j} - V_{dr}^j\right)^2} \cdot \\
&\cdot \left\{ \frac{\Delta^j}{\alpha_1^j} e^{\left(-\frac{v_\perp^2}{\alpha_1^j V_{th}^{j\,2}}\right)} + \frac{1-\Delta^j}{\alpha_1^j - \alpha_2^j} \left[e^{\left(-\frac{v_\perp^2}{\alpha_1^j V_{th}^{j\,2}}\right)} - e^{\left(-\frac{v_\perp^2}{\alpha_2^j V_{th}^{j\,2}}\right)} \right] \right\}
\end{aligned}
$$

where

n^j	–	density of the j-th plasma component
n	–	total density
$\tilde{f}^j$	–	normalized distribution function of the j-th component
m	–	number of components (maximum ten)
V_{th}^j	–	thermal velocity of a component
V_{dr}^j	–	normalized drift velocity along magnetic field
Δ^j	–	depth of the loss-cone, values from 0 (max loss cone) to 1 (no loss cone)
α_1^j, α_2^j	–	size of the loss-cone and the temperature anisotropy

The sum of different Maxwellians allows to model distribution functions of very complex shape. Note that density can be also negative thus for example shell-like distribution functions can also be modeled.

`WHAMP` uses a local coordinate system ($\mathbf{e}_i$) defined by direction of magnetic field $\hat{\mathbf{b}} = (0, 0, 1)$ and wave vector $\mathbf{k} = (k_\perp, 0, k_\parallel)$. The velocity $\mathbf{v} = (v_\perp^x, v_\perp^y, v_\parallel)$ is divided in parallel and perpendicular components to the magnetic field. The electric field $\mathbf{E}(\omega, \mathbf{k})$ satisfies a wave equation

$$\mathbf{D}(\omega, \mathbf{k}) \cdot \mathbf{E}(\omega, \mathbf{k}) = 0$$

We can write dispersion tensor more explicitly as

$$\mathbf{D}(\omega, \mathbf{k}, \mathbf{r}) = \left(\mathbf{I}k^2 - \mathbf{k}\mathbf{k}\right) \frac{c^2}{\omega^2} - \varepsilon(\omega, \mathbf{k}, \mathbf{r})$$

The wave equation has non-trivial solutions if

$$D(\omega,\mathbf{k},\mathbf{r}) = \det \mathbf{D}(\omega,\mathbf{k},\mathbf{r}) = 0$$

The most difficult part in the solution is the dielectric tensor calculations. The standard derivation within linearized kinetic theory leads to the expression

$$\boldsymbol{\varepsilon} = \mathbf{I} - \sum_j \frac{\omega_{pl}^{j\mathbf{2}}}{\omega^2}(\mathbf{I} - \chi^j)$$

where

$$\chi^j = \sum_{n=-\infty}^{\infty} \int d\mathbf{v}\, \boldsymbol{\Pi}\, \frac{\frac{n\Omega^j}{v_\perp}\frac{\partial}{\partial v_\perp} + k_\parallel \frac{\partial}{\partial v_\parallel}}{\omega - k_\parallel v_\parallel - n\Omega_j} \tilde{f}^j$$

The matrix $\boldsymbol{\Pi}$ contains square terms of Bessel functions of order n and argument $k_\perp v_\perp/\Omega_j$.
Introducing the normalized values of all, frequency, wave vector and the speed of light gives:

$$\kappa_\perp^j = \frac{k_\perp \cdot V_{th}^j}{\Omega^j}\ ;\quad \kappa_\parallel^j = \frac{k_\parallel \cdot V_{th}^j}{\Omega^j}\ ;\quad \tilde{\omega}_{pl}^j = \frac{\omega_{pl}}{\Omega^j}\ ;\ \tilde{\omega}^j = \frac{\omega}{\Omega^j}\ ;\ \tilde{c} = \frac{c}{V_{th}^1} \tag{1}$$

where,	Ω^j	– gyrofrequency /the j-th plasma component/
	ω_{pl}^j	– plasma frequency
	V_{th}^j	– thermal velocity
	$\kappa_\perp^j, \kappa_\parallel^j, \tilde{\omega}_{pl}^j, \tilde{\omega}^j, \tilde{c}$	– dimensionless values
	c	– light velocity

we can rewrite the dispersion tensor

$$\begin{aligned}\mathbf{D}(\omega,\mathbf{k},\mathbf{r}) &= \left(\mathbf{I}{\kappa^1}^2 - \boldsymbol{\kappa}^\mathbf{1}\boldsymbol{\kappa}^\mathbf{1}\right)\tilde{c}^2 - \boldsymbol{\varepsilon}(\kappa_\perp^j, \kappa_\parallel^j, \tilde{\omega}^j) \\ &= \left(\mathbf{I}{\kappa^1}^2 - \boldsymbol{\kappa}^\mathbf{1}\boldsymbol{\kappa}^\mathbf{1}\right)\tilde{c}^2 - \left(\mathbf{I} - \sum_j \omega_{pl}^{j2}\left(\mathbf{I} - \chi^j(\kappa_\perp^j, \kappa_\parallel^j, \tilde{\omega}^j)\right)\right)\end{aligned}$$

All calculations in `WHAMP` are done with these new normalized values. To escape infinite sums containing Bessel functions the plasma dispersion function is evaluated using the modified Padé method of function approximation. This approximation is such that the largest error in `WHAMP` estimates of the dispersion function do not exceed 3% (the worst case). Its advantage is the high speed of calculations and validity in the whole range from small to large argument values.

Introducing the refractive index $\boldsymbol{\mu} = \mu_1\mathbf{e}_1 + \mu_3\mathbf{e}_3$, the determinant of the dispersion tensor becomes

$$D(\omega,\mathbf{k},\mathbf{r}) = A(\mu^2 - \varepsilon_{22}) - B + C, \tag{2}$$

where

$$\begin{aligned} A &= \mu_1^2\varepsilon_{11} + 2\mu_1\mu_3\varepsilon_{13} + \mu_3^2\varepsilon_{33}, \\ B &= (\mu_3\varepsilon_{23} - \mu_1\varepsilon_{12})^2 + \mu^2(\varepsilon_{11}\varepsilon_{33} - \varepsilon_{13}^2), \\ C &= (\varepsilon_{11}\varepsilon_{33} - \varepsilon_{13}^2)\varepsilon_{22} + (\varepsilon_{11}\varepsilon_{23} + \varepsilon_{12}\varepsilon_{13})\varepsilon_{23} + \\ &\quad + (\varepsilon_{33}\varepsilon_{12} + \varepsilon_{23}\varepsilon_{13})\varepsilon_{12}.\end{aligned}$$

2 WHAMP in MATLAB

It is possible (Mac,Linux) to run `WHAMP` directly in MATLAB. This requires that package '+whamp' is on your path. If you have 'irfu-matlab' package, '+whamp' is already part of it. The '+whamp' package is also part of `WHAMP` github. Most important routines in '+whamp':

- `whamp.run` - run `WHAMP`
- `whamp.plot_f` - plot distribution functions
- `whamp.m2xyz` - conversion of `WHAMP` output files when run from terminal

3 WHAMP distribution function examples

3.1 $\Delta = 1, \alpha_2 = 0, \alpha_1 = T_\perp/T_\parallel$

Anisotropic plasma

$$f(v_\perp, v_\parallel) = \frac{1}{(\sqrt{\pi}V_{th})^3}\, e^{-\left(\frac{v_\parallel}{V_{th}} - V_{dr}\right)^2} \frac{T_\parallel}{T_\perp} e^{-\frac{v_\perp^2}{(T_\perp/T_\parallel)V_{th}^{j\,2}}}$$

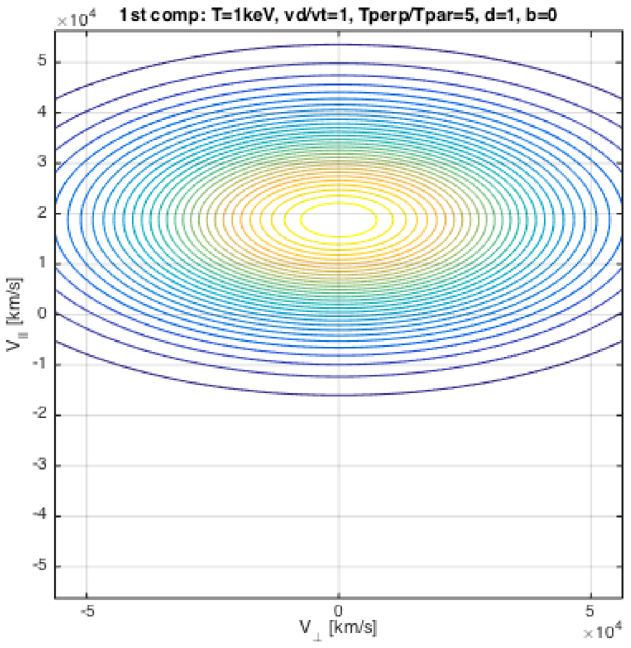


Figure 1: Example distribution function. Drift velocity $v_d = v_{th}$ and temperature anisotropy is such that $T_\perp/T_\parallel = 2$. The MATLAB commands below are used to create the plot.

```
Electrons = struct('m',0,'n',1,'t',1,'a',5,'vd',1,'d',1,'b',0);
whamp.plot_f(Electrons,'km/s')
```

4 WHAMP, running code

4.1 Running from terminal

Plasma model is specified with an input file. Note that $a = \alpha_1, b = \alpha_2$

```
============ Model files ===========
n(1)  n(2)  ...  n(10)   /per m3/
t(1)  t(2)  ...  t(10)   / keV, T_par /
d(1)  d(2)  ...  d(10)   / loss cone parameter, default 1.0 (no loss cone) /
a(1)  a(2)  ...  a(10)   / t_perp/t_par, default 1.0 /
b(1)  b(2)  ...  b(10)   / default 0, i.e. no loss cone/
ass(1)  ass(2)  ...  ass(10)    / 0-electrons, 1-protons, 16-oxygen /
vd(1)  vd(2)  ...  vd(10)   / v_drift/v_term /
fce / electron gyrofrequency in kHz /
pzl     / 1 - log scale, 0 - linear scale /

=====================================
Plasma with B=100nT, n=1cm-3, there are only
oxygen ions (Tperp=50eV, Tpar=10eV, vd=1*vt_par)
and 1eV electrons.
============ Example file with 9 lines ===========
1.e6 1.e6 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
.001 .001 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1.   1.   1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
5.   1.   1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
0.0  0.0  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
16.  0.0  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1.   0.0  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
2.8
0
```

Plasma parameters can be also changed later during the running of the code.
To start the WHAMP just execute:

```
>whamp -file <modelfile>
```

A real example that can be run from `src` directory:

```
> ./whamp -file ../Models/Ex1
# PLASMA FREQ.:      8.9786KHZ GYRO FREQ.:    2.8000KHZ   ELECTRON DENSITY:1.00000E+06M-3
# O+   DN= 1.00000E+06  T=  0.01000  D=1.00  A=5.00  B=0.00 VD= 1.00
# e-   DN= 1.00000E+06  T=  0.00100  D=1.00  A=1.00  B=0.00 VD= 0.00
#INPUT:
```

You can get help for input by entering "h". You get

```
 AN INPUT LINE MAY CONSIST OF UP TO 80 CHARACTERS.
 THE FORMAT IS:
 NAME1=V11,V12,V13,...NAME2=V21,V22,...NAME
 THE NAMES ARE CHOSEN FROM THE LIST:

 NAME              PARAMETER
 A(I)              THE ALPHA1 PARAMETER IN THE DISTRIBUTION.
                   (I) IS THE COMPONENT NUMBER, I=1 - 6.
```

```
B(I)              THE ALPHA2 PARAMETER IN THE DISTRIBUTION.
C                 THE ELECTRON CYCLOTRON FREQ. IN KHZ.
D(I)              THE DELTA PARAMETER IN THE DISTRIBUTION
F                 FREQUENCY, START VALUE FOR ITERATION.
L            L=1  THE P AND Z PARAMETERS ARE INTERPRETED
                  AS LOGARITHMS OF THE WAVE NUMBERS. THIS
                  OPTION ALLOWS FOR LOGARITHMIC STEPS.
             L=0  DEFAULT VALUE. LINEAR STEPS.
M(I)              MASS IN UNITS OF PROTON MASS.
N(I)              NUMBER DENSITY IN PART./CUBIC METER
P(I)              PERPENDICULAR WAVE VECTOR COMPONENTS.
                  P(1) IS THE SMALLEST VALUE, P(2) THE
                  LARGEST VALUE, AND P(3) THE INCREMENT.
S                 STOP! TERMINATES THE PROGRAM.
T(I)              TEMPERATURE IN KEV
V(I)              DRIFT VELOCITY / THERMAL VELOCITY.
Z(I)              Z-COMPONENT OF WAVE VECTOR. I HAS THE
                  SAME MEANING AS FOR P(I).
A NAME WITHOUT INDEX REFERS TO THE FIRST ELEMENT, "A" IS
THUS EQUIVALENT TO "A(1)". THE VALUES V11,V12,.. MAY BE
SPECIFIED IN I-, F-, OR E-FORMAT, SEPARATED BY COMMA(,).
THE "=" IS OPTIONAL, BUT MAKES THE INPUT MORE READABLE.
EXAMPLE: INPUT:A1.,2. B(3).5,P=.1,.2,1.E-2
THIS SETS A(1)=1., A(2)=2., B(3)=.5, P(1)=.1, P(2)=.2,
AND P(3)=.01. IF THE INCREMENT P(3)/Z(3) IS NEGATIVE, P/Z
WILL FIRST BE SET TO P(2)/Z(2) AND THEN STEPPED DOWN TO
P(1)/Z(1)
THE LAST SPECIFIED OF P AND Z WILL VARY FIRST.
IF THE LETTER "O" (WITHOUT VALUE) IS INCLUDED, YOU WILL
 BE ASKED TO SPECIFY A NEW OUTPUT FORMAT.
```

4.2 WHAMP running from matlab

See help where are given examples:

```
help whamp.run
```

4.3 WHAMP input

Plasma parameters can be specified in the model file. To start the calculations the range of $k_\perp$ and $k_\parallel$, the step length in $k_\perp$ and $k_\parallel$, and a start value for the frequency must be specified. When running WHAMP, this means specifying p, z and f, where $p = \kappa_\perp^1 = k_\perp V_{th}^1/\Omega^1, z = \kappa_\parallel^1 = k_\parallel V_{th}^1/\Omega^1, f = \omega/\Omega^1$, where the thermal velocity of plasma with temperature T^{eV} (expressed in eV) can be calculated as $V_{th} = \sqrt{2eT^{eV}/m}$. **The normalization is always with respect to the first component of the plasma model**. In an anisotropic plasma it is the parallel thermal speed that is used for thermal velocity estimates. Using frequency in Hz instead of rad/s we have

$$\mathrm{p} = \frac{V_{th}^1}{\lambda_\perp f_{c1}}, \quad \mathrm{z} = \frac{V_{th}^1}{\lambda_\parallel f_{c1}}, \quad \mathrm{f} = \frac{\omega}{f_{c1}}. \tag{3}$$

In our example file O^+ is the first component. The gyrofrequency of O^+ is 0.095 Hz, the thermal speed of 10eV oxygen ions is 10.9 km/s. The Alfvén velocity is 545 km/s. Thus if we with WHAMP want to find Alfvén wave below oxygen gyrofrequency, let's say at 0.01 Hz, we would put $p = 0$ (parallel propagation), $f =$

$f^{Hz}/f_{cO^+} = 0.01/.095 \approx 0.105$, and parallel wavelength $\lambda = v_A/f = 0.545\ 10^6/0.0105 = 51.9\ 10^6$ km and $z = 10.9\ 10^3/(51.9\ 10^6 * 0.095) \approx 0.0022$. Running WHAMP from terminal we would enter:

```
p0z.0022f.1
```

4.4 WHAMP output when running from terminal

WHAMP can print different output parameters. The simplest is to print wave vectors and frequency, for this enter "pzf". The output has the same normalization as input. The example above would be:

```
> ./whamp -file ../Models/Ex1
# PLASMA FREQ.:     8.9786KHZ GYRO FREQ.:    2.8000KHZ   ELECTRON DENSITY:1.00000E+06M-3
# O+   DN= 1.00000E+06  T=  0.01000  D=1.00  A=5.00  B=0.00 VD= 1.00
# e-   DN= 1.00000E+06  T=  0.00100  D=1.00  A=1.00  B=0.00 VD= 0.00
#INPUT:

p0z.0022f.1
#OUTPUT:

pzf
    0.0000000      0.0022000    1.0640799E-01  3.34E-11

#INPUT:
```

NOTE! For the frequency both, the real and imaginary part is printed.
The following options are available for output

```
####
============ output ===========

    e   (ex,ey,ez)
    b   (bx,by,bz)
    f   frequency <real,imaginery>
    g   group velocity
    h   |e|/|b| [(mV/m)/nT]
    p   k perpendicular
    s   (sp, sz)   spatial growth
    u   energy ration between the total wave energy and energy in electric field
    v   Poynting flux (in uW/m2 for <E^2>=0.5(mV/m)^2)
    z   k paralel
    y   energy density and flux of each component
```

Thus if you would like to print k-vector components and frequency, and in the next line all electric field components, you would enter "pzf/e". Example from session:

```
#INPUT:

o
#OUTPUT:

pzf/e
    0.0000000      0.0022000    1.0640799E-01  3.34E-11
```

```
 EX= 0.7071  0.0000  EY=-0.0000  0.7071  EZ= 0.0000  0.0000

#INPUT:
```

For fields the first is real and the second is imaginary part. Note that in this example the wave is right hand polarized $E_y = iE_x$.

4.5 Dispersion surfaces

One can specify input of wave vectors and steps in $\log_{10}$ space executing "l1" in terminal. This is convenient when estimating dispersion surfaces over large wave vector interval. Let's look at the interval $p = 10^{-3} \ldots 10^{-1}, z = 10^{-3} \ldots 10^{-1}$, where we for every p value go through all z values (z changes first). First we find the starting point, then we check z interval for only one p value and finally we run p intervals for different z values.

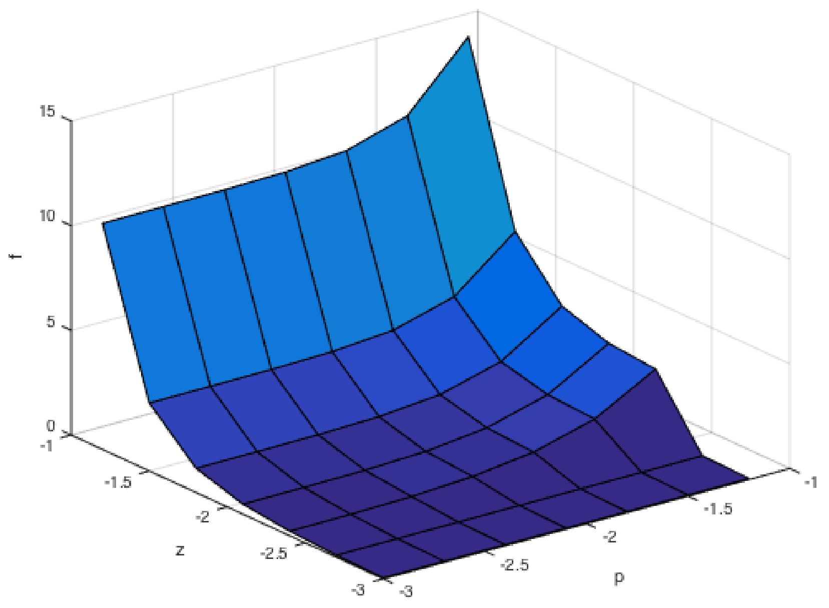
```
#INPUT:

l1
    0.0010000      0.0010000    4.7821169E-02 -3.90E-08

#INPUT:

z-3p-3f.04
    0.0010000      0.0010000    4.7821169E-02 -3.90E-08

#INPUT:

z-3,-1,.3
    0.0010000      0.0010000    4.7821169E-02 -3.90E-08
    0.0010000      0.0019953    9.9003895E-02 -2.69E-08
    0.0010000      0.0039811    2.1508729E-01 -6.39E-08
    0.0010000      0.0079433    4.7770868E-01 -3.44E-08
    0.0010000      0.0158489    1.1588537E+00 -2.81E-08
    0.0010000      0.0316228    3.2487234E+00 -2.59E-08
    0.0010000      0.0630957    1.0804231E+01 -1.43E-09

#INPUT:

z-3,-1,.3p-3,-1,.3
```

After the last command one gets the screen full with the result. These results one can copy into some file "test" which one can load in from matlab and plot as a dispersion surface. The command lines in matlab are

```
load test
[p,z,f,fim]=whamp.m2xyz(test);
surf(log10(p),log10(z),f)
xlabel('p');ylabel('z');zlabel('f');
```

`whamp.m2xyz` reorders the `WHAMP` result file into p, z vectors and f, γ matrices.
The result is OK for the first `WHAMP` try but one can note a few things

- One needs smaller steps to get nice dispersion relation,



- One can jump from one dispersion surface to another, `WHAMP` tries to find the closest one from some given initial conditions which often does not mean the same as to stay on the same dispersion surface.
- One can expect a lot of "trial and error" before one gets a satisfactory result

Constructing the same dispersion surface from MATLAB would be:

```
Oxygen              = struct('m',16,'n',1,'t',10,'a',5,'vd',1);
Electrons           = struct('m',0,'n',1,'t',1);
PlasmaModel         = struct('B',100);
PlasmaModel.Species = {Oxygen,Electrons};
InputParameters     = struct('fstart',0.04,'kperp',[-3 .3 -1],...
                      'kpar',[-3 .3 -1],'useLog',1);
Output              = whamp.run(PlasmaModel,InputParameters)

surf(log10(Output.kperp),log10(Output.kpar),real(Output.f))
xlabel('p');
ylabel('z');
zlabel('f');
```

4.6 References

- WHAMP - latest version on https://github.com/irfu/whamp

Good luck!