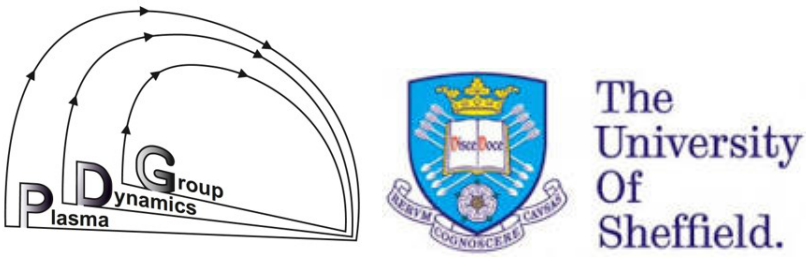
Sheffield Dispersion Diagram Code (SDDC)

version 2.0

User Manual

Developers: Samuel Skirvin (ACSE), Viktor Fedun (ACSE), Gary Verth (SoMaS)



Plasma Dynamics Group, The University of Sheffield

May 2020

Contents

1 Introduction

`Sheffield Dispersion Diagram Code (SDDC v2.0)`, written in Python, is designed to produce the dispersion diagram for magnetoacoustic magnetohydrodynamic (MHD) wave modes within a solar application. The version explained in this manual build on the previous v1.0 which was very much the spine of the code. This version is much more streamlined and contains additional features which improve the accuracy of the results and additional files for analysis of results. The improvements made in SDDC v2.0 include:

- Multiprocessing options - speeding up the elapsed time and optimisation of previous version(s).
- Analysis Files. These include files which import the output data from the main program to allow for use in plotting and visualisations.
- Inclusion of physics required to account for background plasma flows, uniform and non-uniform. This file can be provided upon request to the developers.

The bulk of the code is still the same as the previous version(s), however may be tweaked for improved optimisation. The same physics is still implemented as described in previous version(s) of the code, for more information on the main functions used in the code see the code manual for SDDC v1.0

2 Usage

This manual describes how to set up and run the code for different scenarios that can be encountered in a solar context, an example of a simple inhomogeneous density profile is provided. The manual is split into the following section:

(1) Software requirements.
(2) Initialising the plasma environment.
(3) Description of any plasma structure.

Examples of use.
(4) Manual_V2.py
(5) Manual_analysis_V2.py
(6) Manual_movie_V2.py

2.1 Software requirements

We recommend to use Python 3 or later versions to avoid any incompatible conflicts with software. The software package *ffmpeg* is also required if it is desired to create movies showing the method procedure by converting a series of images into a movie.

2.2 Initialise plasma environment

Traditionally, it is useful to investigate the propagation of waves under extreme conditions. On the Sun, the plasma in the corona is significantly different to the environment of the plasma in the photosphere. As a result, the dispersion diagrams are very different. The first step is to provide conditions for characteristic speeds internal and external to the waveguide, for example the sound speeds c_i, c_e and the Alfvén speeds v_{Ai}, v_{Ae} which provide the

context of the plasma environment.

The implementation of defining the profiles for characteristic speeds were incorrect in v1.0. This issue has been addressed in this version such that the initial equilibrium profiles are correct and equilibrium pressure balance is achieved. There is availability to plot these profiles and check for pressure balance.

In this version it is possible to supply the required profiles for either plasma density, temperature or magnetic field. However it is essential for validity of results that equilibrium total pressure balance is achieved across the whole waveguide interior and exterior so we urge the user to check this condition is met before running the code.

2.3 Description of any plasma structure

Previous analytical studies investigate wave propagation in a uniform medium, where the plasma properties are constant inside or outside the waveguide. The benefit of this method is that an inhomogeneous plasma can be investigated. Therefore, if desired, it is required to provide the code with an initial profile for specific plasma properties. These profiles can be provided for both internal or external plasma. The variables described in Section **2.2** are slightly modified to account for the introduction of an inhomogeneous medium.

2.4 `Manual_V2.py`

In this example it will be demonstrated how to run a basic task of the main program in SDDC. The specific case of a Gaussian profile in magnetic field inside a magnetic slab under coronal conditions will be investigated. The magnetic field increases from the boundary of the waveguide to the minimum value at the centre, the width of inhomogeneity can be varied and depends on the standard deviation chosen. For this example the standard deviation is chosen to be 1.25. The external medium is considered to be uniform, therefore the internal temperature, gas pressure, magnetic field (B_i), sound speed (c_i), Alfvén speed (v_{Ai}) and tube speed (c_{Ti}) are all inhomogeneous and depend on spatial coordinate x. These are shown in Figure 1

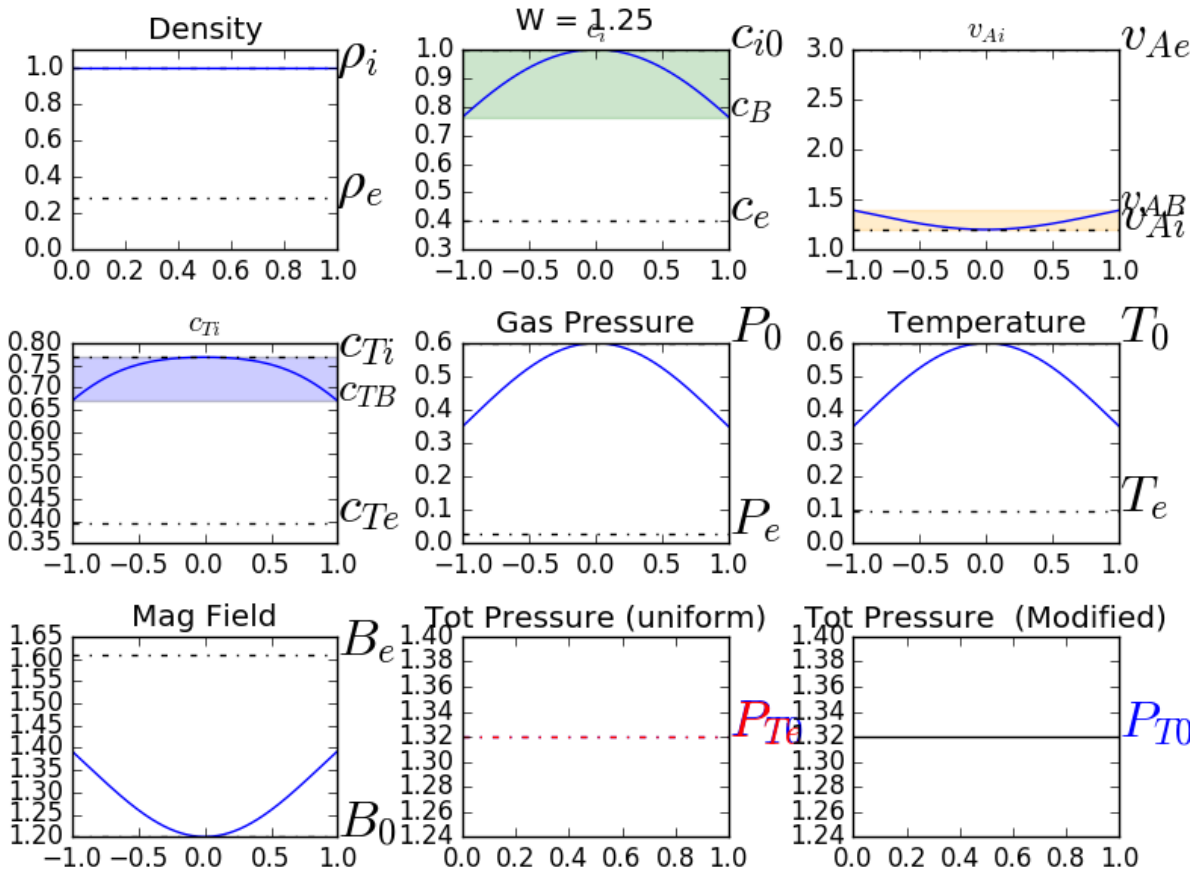


Figure 1: Initialising a spatial profile in magnetic field. This results in spatially dependant temperature, gas pressure and characteristic speeds $c_i, v_{Ai}\ c_{Ti}$.

A new addition to SDDC v2.0 is a new sampling technique. In attempt to improve precision and accuracy of the technique the sampling domain has been modified. Now, the domain is split into each region separated between characteristic speeds and ordered (this will be different depending upon whether photospheric or coronal conditions are being investigated). The same number of samples are then generated between each smaller region. This is shown in Figure 2. The figure only contains 5 samples per region when in reality more (~ 50) are used in the calculation.

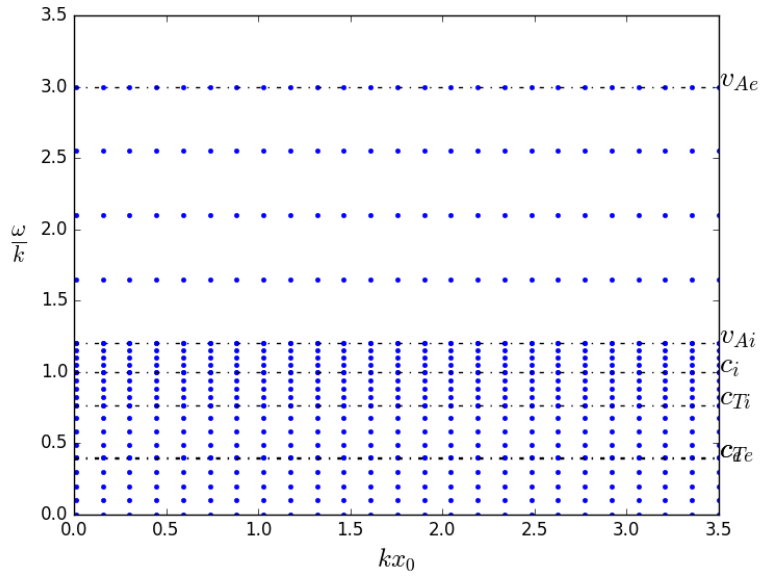


Figure 2: New sampling implemented in v2.0. Each region separated by characteristic speeds contains the same number of samples - in an attempt to retrieve more body modes.

```
if __name__ == '__main__':
    starttime = time.time()
    processes = []

    sausage_ws = multiprocessing.Queue()
    sausage_ks = multiprocessing.Queue()

    processes_kink = []

    kink_ws = multiprocessing.Queue()
    kink_ks = multiprocessing.Queue()


    for k in wavenumber:
      for i in range(len(speeds)-1):

        test_freq = np.linspace(speeds[i]*k, speeds[i+1]*k, 45.)   #was 60     #SET NO. OF SAMPLES BETWEEN EACH SPEED

        task = multiprocessing.Process(target=sausage, args=(k, sausage_ws, sausage_ks, test_freq))
        task_kink = multiprocessing.Process(target=kink, args=(k, kink_ws, kink_ks, test_freq))

        processes.append(task)
        processes_kink.append(task_kink)
        task.start()
        task_kink.start()

    for p in processes:
        p.join()

    for p in processes_kink:
        p.join()

    sol_ksl = [sausage_ks.get() for p in processes]
    sol_omegasl = [sausage_ws.get() for p in processes]

    sol_ksl = list(itertools.chain(*sol_ksl))   #flatten out the list of lists into one single list
    sol_omegasl = list(itertools.chain(*sol_omegasl))

    sol_ks_kinkl = [kink_ks.get() for p in processes_kink]
    sol_omegas_kinkl = [kink_ws.get() for p in processes_kink]

    sol_ks_kinkl = list(itertools.chain(*sol_ks_kinkl))
    sol_omegas_kinkl = list(itertools.chain(*sol_omegas_kinkl))
```

Figure 3: The section of code where multiprocessing is implemented. Multiprocessing is looped over wavenumber and each region of frequency.

Shown in Figure 3 is the section of code which handles the multiprocessing component of the algorithm. The multiprocessing package within python is utilised. The idea is that every sample of wavenumber is sent to a separate processor where the code can be run sampling through frequency. At the end all solutions are combined and saved in a data file (a 'pickle' file in python) and can be used for analysis purposes. The more processors available increases the speed at which the algorithm is completed. Lines 931 - 942 in Figure 3 are the bulk parts of the multiprocessing component here. The looping over wavenumber creates a separate task for each independent k value. The looping over frequency samples through each domain shown in Figure 2. The number of samples in each region is modified through line 934.

The code can then be run in the same way as done in v1.0. If the user only has one processor available to them then this is no problem, as the code will loop through the same processor - however expect computation time to be longer. In the example file attached with the manual, the output data, the solutions for frequencies and wavenumber for trapped modes are saved in a 'pickle' file.

2.5 `Manual_analysis_V2.py`

A further addition in this version is the availability of additional python scripts for analysis. The files have the capabilities of loading in the data from the main algorithm and plotting the solutions on a dispersion diagram. Visualisations of the plasma profiles can also be done with this script. The attached analysis script also includes the option to plot the solutions with a polynomial fit through them, such that not only dots are plotted and the separate branches can be observed more clearly. Separate analysis files are available for cases under photospheric and coronal cases. These files are very similar however undergo analysis slightly differently due to the differing dispersion diagrams produced and types of wave modes found.

For the case attached we still consider a coronal slab with a magnetic field modelled as a Gaussian with a width of 1.25. The data is loaded in from the file '`B_width125_coronal.pickle`' and the arrays are then arranged in a sorted order. Due to any multiprocessing which has been undertaken, the solutions are returned in whatever order the multiprocessing was carried out in, such that the resulting arrays may not be sorted in value order. Therefore to initially have all the data in a neat form we sort all solutions in order of increasing wavenumber. The solutions are then separated into different branches and stored in new arrays. Each branch will related to a specific type of wave mode, for both sausage and kink solutions, for example fast body or slow body (fast/slow surface or body for photospheric conditions). This is done to allow for curves to be fitted to each individual branch using the function `polyfit` as part of the numpy package. Any solutions which do not lie close enough to the main branch are stored in a separate array and plotted separately. Curves are not fitted to any solutions which lie in any continuum, the area of inhomogeneity between the boundary value and absolute maximum of each characteristic speed. The resulting dispersion diagram for this scenario is shown in Figure 4. It is possible to not plot any solutions which are not exact trapped modes, for a clearer more understandable dispersion diagram. The one shown here contains all solutions for comparison reasons.

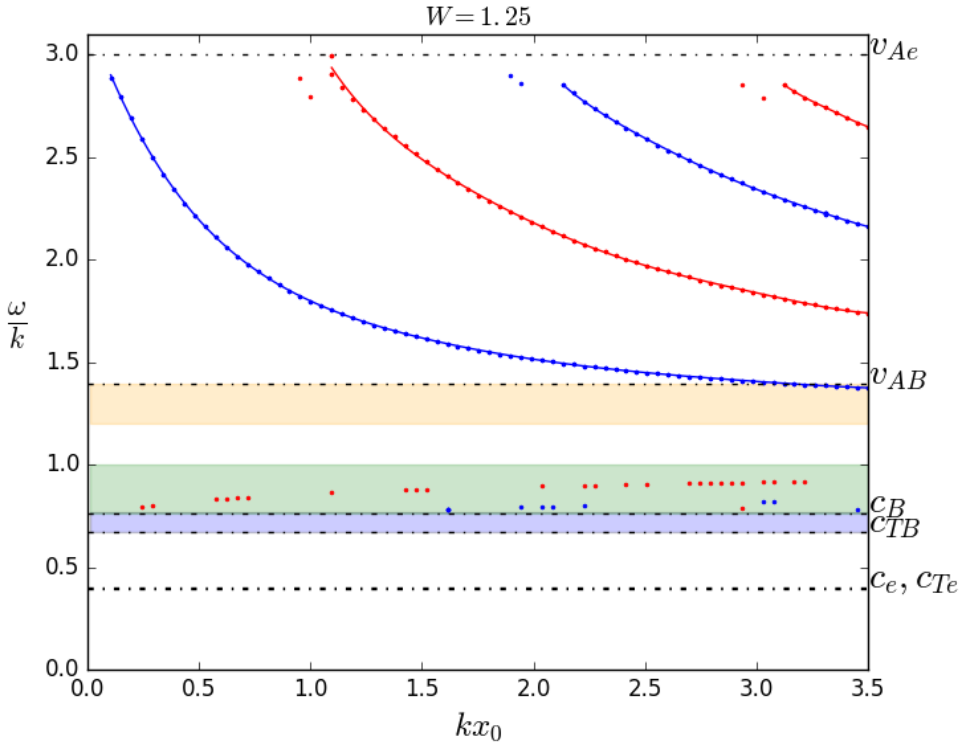


Figure 4: The resulting dispersion diagram of a coronal slab modelled with an internal structuring of magnetic field with properties as given in Figure 1.

2.6 `Manual_movie_V2.py`

The purpose of this analysis tool is the visualise the behaviour of plasma properties in the whole domain using the obtained solutions. The script contains the same procedure as the main code, implementing the shooting method, however rather than searching for solutions, uses the previously obtained values of frequency and wavenumber pairs. The idea is that the user can investigate further the physical behaviour of the solutions by analysing the total pressure, horizontal velocity and any coefficients used in the equations.

This tool allows the user to investigate in more detail how the method has obtained the solution in question. They can observe whether or not a resonant solution has been found by checking the nature of the displacement (if it tends to large values or not) and confirm solutions as trapped modes. Similarly to the analysis file, the required '.pickle' file is read in. The same script is repeated as the analysis file such to produce the corresponding dispersion diagram. After the diagram has been reproduced, the shooting method script (same as the one used in the main code) is implemented, however rather than scanning for ω and k solutions, uses the obtained solutions instead. This script will work best if the user chooses a specific branch to analyse on the dispersion diagram, and loop through the solutions on that branch. The branch should be indicated in Lines 964 and 965. The user should also ensure that the variables they wish to plot are indicated in lines $1054 - 1172$ by commenting out (#) the parameters that they do not wish to see (e.g. cusp frequency, coefficients of equation etc). It should also be noted that for best results the exact same parameters should be used in this analysis tool as used in the main algorithm which obtained the solutions (i.e. characteristic speed choices, normalisation of wavenumber, initial shooting conditions etc.). The output will be an animation showing the chosen parameters for each solution on the dispersion diagram.

3 Future Versions

The version presented here is a further extension of the version 1.0 and is still being improved and developed rapidly. Work is already underway to advance the code to be more applicable to various types of solar data. Below are outlined a brief overview of what can be expected in future versions:

- Wider choices of geometry - In this version a Cartesian geometry is considered however work is already underway to further extend this option to include cylindrical and elliptical geometries. These geometries would provide a better model of some features observed in the solar atmosphere and would compliment observations more reliably. The cylinder model will be able to handle internal and external flows, inhomogeneous plasma distributions, magnetic twist and flow in the ϕ direction.
- Code supportive of asymmetric profiles - This version of the code takes advantage of fundamental physical properties of the kink and sausage MHD wave modes to obtain solutions on the dispersion diagram. This physics is based off symmetrical homogeneous models. When these models become asymmetric, in whatever way, the perturbation at each boundary is not supported by previous basic models. New physics will be included in the model such that asymmetric profiles about the waveguide centre can be considered.

4 Release notes

`SDDC version 2.0 (2020)` can produce dispersion diagrams in different inhomogeneous plasma regimes found in the solar atmosphere in planar geometry. The update from v1.0 contains additional python scripts that serve as an analysis tool to compliment the main program.

5 Queries

Samuel J. Skirvin (ACSE), `sjskirvin1@sheffield.ac.uk`
Viktor Fedun (ACSE), `v.fedun@sheffield.ac.uk`
Gary Verth (SoMaS), `g.verth@sheffield.ac.uk`

6 Citation in the literature

We acknowledge the Plasma Dynamics Group at the University of Sheffield and for support provided in the creation of `Sheffield Dispersion Diagram Code, SDDC v2.0, 2020`. The development of the code was also supported by the European Union's Horizon 2020 research and innovation program under grant agreement No.824135 (SOLARNET). SJS also thanks STFC for the PhD studentship under which the code was produced and developed.