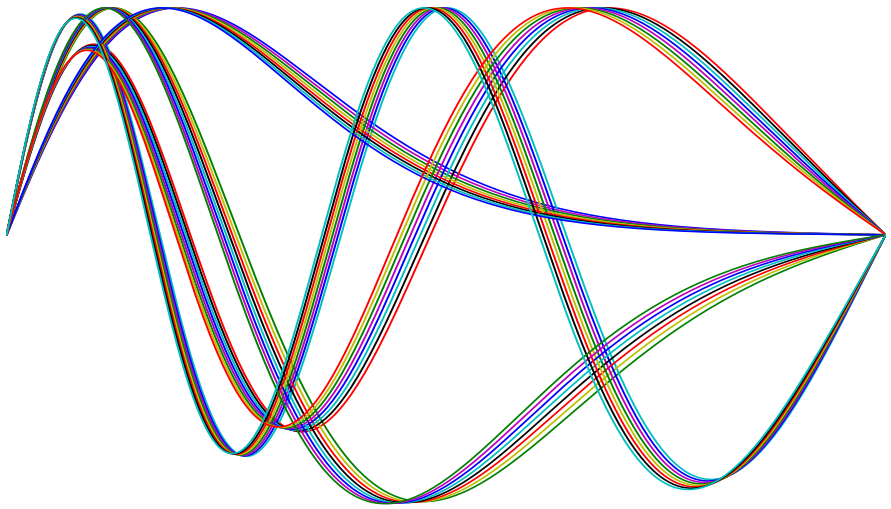
First Assignment Computational Methods for Astrophysical Applications

Vergauwen Bob

November 14, 2014



1 Introduction

In this paper we try to model the propagations of waves trough an incompressible slab of material that is in an equilibrium state. The incompressibility of the material will prevent the formation of pressure driven waves, so looking at waves in this material looks a bit like a lost job. But through the introduction of gravity there could form gravity driven waves who can propagate through the material. The equation describing these waves is given by

$$\frac{d}{dx}\left[\rho_0\omega^2\frac{\xi}{dx}\right] - K^2\left[\rho_0\omega^2 + \rho_0' g\right]\xi = 0. \tag{1}$$

Here ρ_0 is the density of the material, ω the frequency' K the wave number and g the graviton. Classically it is enough to provide one boundary condition to solve this differential equation, this will yield a solution for every value of σ. However we could also give a boundary condition on a latter time step, if this is the case the possible solutions to the problem reduce a lot and only typical values for ω will be allowed. These values for ω are now the eigenfrequencies of this problem and will depend on the typical parameters involved in the equations.

In this paper we will solve Equation 1 numerically using python and calculate the possible eigenfrequencies corresponding to the boundary conditions $\xi(0) = 0, \xi(1) = 0$. For the density we will only investigate a linear dependency, $\rho_0 = 1 + \sigma x$. We will not only spend some time explaining the results it self but we will cover the most important parts of the code implementation as well.

2 Code Implementation

The main goal of the project was to not only create a working code but the code had to be reusable as well. To match these design criteria we have chosen for an object orientated approach in python and made full use of the inheritance features provided by the language.

To speed up the computations a lot we used the python library `multiprocessing` to create multiple workers who can run in parallel, this is very easily created in python and it speeds your code up by a factor 8 (If you have 8 threads running on your computer).

2.1 Code structure

A good program always starts out with a good structure. For this we decided to split the code up in a number of packages , each of these packages has a typical task and several classes in it. As on example, there is a Worker package, this package includes everything to create new workers to parallellise the computations.

2.2 Solving the Differential Equation

For solving a differential equation of the form

$$\frac{d}{dt}Y(x,t) = f(x,t) \tag{2}$$

We need a method to integrate the function $f(x,t)$. In this paper we implemented the Runge Kutta method of order 4 to do this. The explicit formula for this method is given

by

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{3}$$

where k_i are given by

$$\begin{aligned} k_1 &= f(t_n, y_n), \\ k_2 &= f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}k_1), \\ k_3 &= f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}k_2), \\ k_4 &= f(t_n + h, y_n + hk_3). \end{aligned}$$

This method is of fourth order, this means that the error of the method drops as a fourth order function. The package Integrators was responsible for this, as will be clear from the code, we wrote the Runge Kutta solver as general as possible. When provided by a correct butcher matrix, the order of the method could be changed.

2.3 Finding the Eigenvalues

Solving the differential equation is the easy part of the assignment. Finding the eigenvalues was the real challenge and it was the task of the worker class.

Finding an eigenvalue comes basically down of finding a root of the function

$$F_{end}(\omega^2; \sigma, K, g), \tag{4}$$

which gives the function value of the differential equation at the position 1. We created a couple of workers each with its own way of finding these roots. These workers could than be run independently of each other and search for roots in it's own variable subspace.

Lets now look in more detail to the differed ways the workers try to find these roots.

2.3.1 The Simple Stepping Worker

The first and easiest to implement worker is the `workerSimple`. A simplified version of the algorithm it uses to find the roots is described in Algorithm 1. The basic idea is to start with a large enough value for ω and decrease it step by step. If you step over a root (If the sign of the end point flips) go back and decrease the step size. repeat this process till you are close enough to the zero. For finding the next root just step away till you are out of the tolerance boundaries and repeat the process till you find a next point using the same convention. A huge downsides of this algorithm that it is slow and it relies on the initial conditions of the differential equations. A main advantage of the method is that it is guaranteed to converge toward a root and it is easy to understand.

2.3.2 The Non Linear Worker

An other approach we tried was to use a non linear solver already implemented in python. The worker doing so is the workerNLS. I tried out some predefined non linear solvers but found that the Anderson mixing algorithm worked best for this problem. The big problem when using this solver was to guess the starting value ω for the iterative solver. One of our ideas was to calculate some roots randomly, based on these than calculate the interpolated polynomial trough these points and based on this do a guess for the next

```
initialization;
guess = 1;
shrinksize = 0.9;
overcount = 1;
endP = endPoint(guess);
while Pend ≤0 do
    guess = guess*10;
    endP = endPoint(guess)
end
while Pend ≥MAXTOL do
    previous = guess;
    guess = previous * shrinksize;
    endP = endPoint(guess);
    if abs(endP)≤MAXTOL then
        break;
    end
    if endP≤0 then
        overCount = overCount +1;
        shrinksize = shrinksize + 9/(10**overCount);
    end
end
```

Algorithm 1: The simplified version of the simple stepping algorithm, the step size is far from ideal and the convergens is sometimes very slow but it is always guaranteed to converges.

root. This however was hard to implement and did not work all of the time, so we moved away from this approach. However this method works good if we only need to find the first root and start out by a large enough guess for ω. The advantage of this method was it's speed. The disadvantage was finding higher order roots, and sometimes the algorithm did not converges at all.

2.3.3 The Worker2 worker

This was one of the early attempts to find the eigenvalues of the equation. The worker tries to estimate the derivative of the function so he doesn't need to calculate it. It turned out that this was not a stable way to find good eigenvalues for the wave equation.

3 Results

In this section we briefly give some results and explain our findings. We used the standard functions for P, Q and ρ. If we used a positive sign for g no roots existed, this is the case there is a heavier fluid on top of a lighter one and was not stable. The function calculated for the positive values for g blew up exponentially.

3.1 Dispersion relations

3.1.1 Dispersion of K^2

As a first experiment conducted we held all parameters fixed, ($\sigma = 1, g = -1$) except for the value of K^2. For this parameters we than searched for the eigenvalues for several modes. This yields dispersion curves of the allowed values for ω as a function of K^2 as given in Figure 1. The result in this plot looks very similar to what is given in the assignment.

3.1.2 Dispersion of σ^2

Next we calculated the dispersion relation for σ^2. All the other parameters where again held constant ($g = -1, K^2 = 1$). The result of this is given in Figure 1. The shape of the curves look very similar as the dispersion for K, but the value for omega drops rapidly if we search for higher order solutions.

3.1.3 Dispersion of g

The final dispersion relation we investigated was the one of g and ω^2. This is given in Figure 1. Surprisingly there looks like a linear relation between g and ω^2, this is completely different from the previous two dispersion relations.

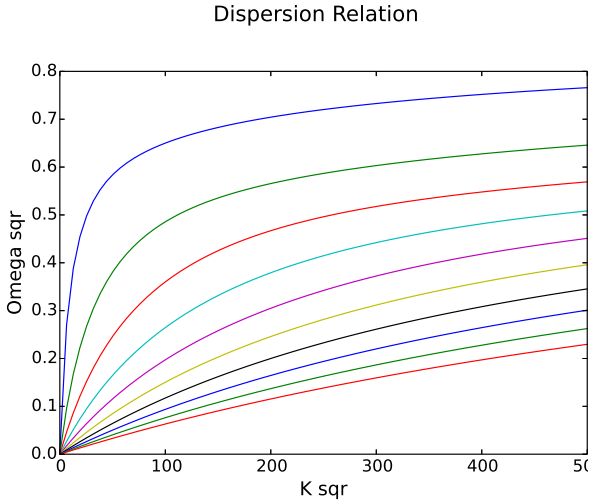
4 Conclusion

From all these dispersion relations some information can be concluded. In this section we will give an overview about our findings and formulate some conclusions.

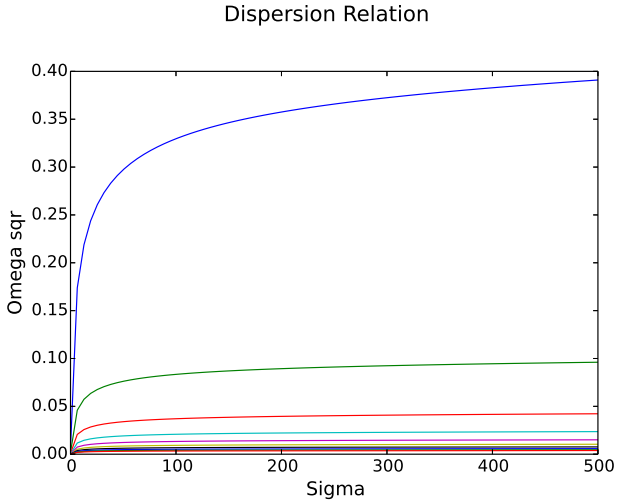
Let us first start by looking at the Dispersion diagram of g and ω as seen in in the lower plot in Figure 1. The shape of these curves look complicity differed from the two other graphs. In fact the dispersion relations are just straight line. This behaviour is exactly as we would expect, it tells us that the equation are scale invariant, as the parameter g told us something about the scale of the problem, it gave a dimension to the equations. The linear relation between the eigenfrequencies and g means that when g gets rescaled than the eigenfrequencies are rescaled by the same amount.

The two other dispersion relations in Figure 1 look very similar is shape, although there is a much greater difference between two neighbour eigenvalues in the σ dispersion relation than for the K dispersion relation. Recall that σ was the gradient of the density. If this gradient becomes larger we would expect the waves to more and more diverge from there sinusoidal shape, this is exactly what we could see when we plot some solutions for increasing σ, an example plot can be seen in Figure 2.

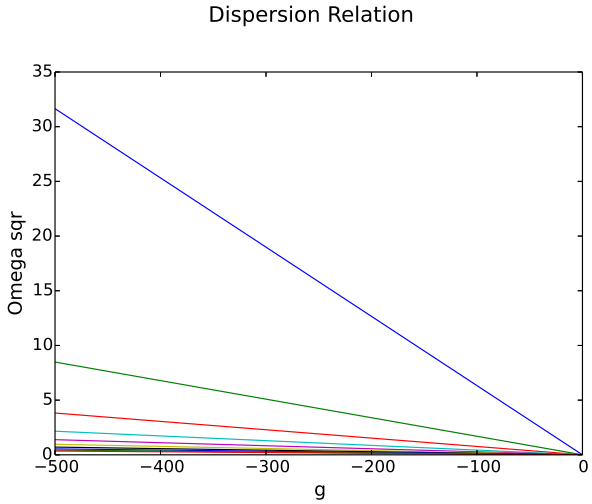
The influence of K was not entirely clear to me, when we changed the K value the eigenvalues changed according to the dispersion relation as given in right plot in Figure 1 but the shape of the curve remained the same, so if we changed the value of K over a curtain range and normalized the solutions all the curves perfectly overlapped each other. The physical interpretation of this was that K did not influence the medium in which the wave was propagating and thereby did not influence the shape of the wave. K is only a internal parameter to the wave itself and thereby will only influence the eigenfrequencies



(a) The dispersion relation of K and ω. The line at the top is for the firs mode, the green line underneath is for the second mode and so on.



(b) The dispersion relation of σ and ω. The line at the top is for the firs mode, the green line underneath is for the second mode and so on.



(c) The dispersion relation of g and ω. The line at the top is for the firs mode, the green line underneath is for the second mode and so on.

Figure 1: Several dispersion relations for the various variables and modes of the equation.

and the amplitude of the wave, K is probably directly linked to the energy transported by the wave.

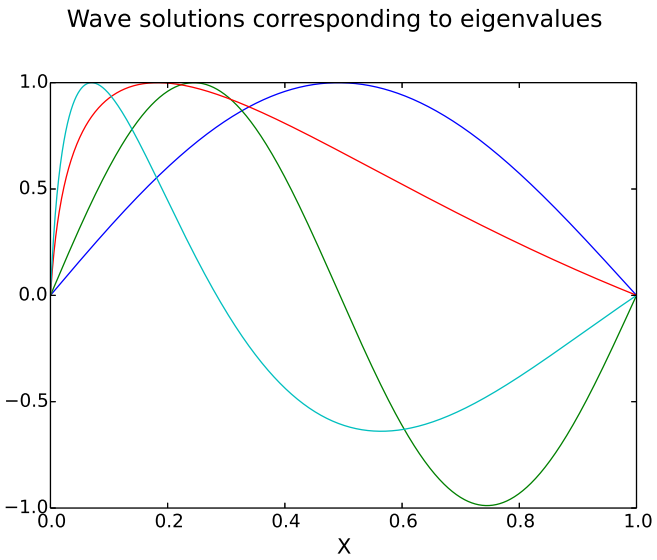


Figure 2: Solutions for the wave equation matching the boundary conditions for increasing σ. The dark blue and green curve are for values of σ very close to 0. The other two curves have a larger value for σ and start to diverge more from an ideal sine wave solution.