A numerical analysis of the seismic wave equation in different layers by the finite element method using fenics

by

DANIEL JAMES TARPLETT

THESIS

for the degree of

MASTER OF SCIENCE

(Master i anvendt matematikk og mekanikk)

Faculty of Mathematics and Natural Sciences
University of Oslo

December 2014

Det matematisk- naturvitenskapelige fakultet
Universitetet i Oslo

Abstract

In this thesis we will investigate the seismic wave equation in different layers by using the finite element method in space and the finite difference method in time. The performance of the programming will be done by comparisons with analytical solutions by using test-solution methods, and convergence tests will be used for error control.

Acknowledgements

Firstly, I would like to thank Geir K. Pedersen and Mikael Mortensen for all their advice and support in the process of writing, and for the help in both the mathematics and programming. I would like to thank Miroslav Kuchta for all the help he has given in the FEniCS Q&A forum and in person. I would also like to thank Finn Løvholt and Valerie Maupin for their role as supervisors. I would like to thank my fellow students for their friendship, and the faculty staff for all the help they have given. Lastly, I would like to thank my friends and family for their continuous support during the process of writing.

Contents

List of Figures

List of Tables

1 Introduction

Elastic waves in the earth are commonly described as seismic waves, and are produced by earthquakes, explosions and similar events. The study of these waves are important in their own right for warning and detection purposes, but the mathematical theory can also be used in other applications of science. It is common to use potential theory when studying seismic waves and seismology, but in this here we will concentrate on more direct solutions of the seismic wave equation. Numerical experiments will be done by using the finite difference method in time, and the finite element method in space. The finite element method is chosen because of it's ability to handle natural boundary conditions, but also because of it's ability to handle more complex geometries. The implementation is done in python using the FEniCS software, as it contains a scripting enviornment and syntax close to the mathematical formalism in the finite element method. In the numerical testing, we will also introduce a concept called test-solutions for simplifying analytic solutions. The overall goal of the thesis is to examine how FEniCS handles an implementation of the seismic wave equation with one and two layers of material. The work is divided into four seperate projects examining the different aspects of the method, and each with their own separate conclusions. We have also included a fifth section, where the mathematics for a further problem is discussed.

2 Theory

In this thesis, we will work with 2D functions in the x-z plane with the y axis pointing inward. We will use dyadic notation where boldface characters indicate vector quantities.

2.1 Governing equations

The scalar wave equation with a variable wave velocity and a damping term can be expressed by:

$$\frac{\partial^2 u}{\partial t^2} + b\frac{\partial u}{\partial t} = \nabla(c\nabla u) \tag{1}$$

where $u = u(x, z, t)$ is the displacement, $b(x, z)$ is the damping term, and $c(x, z)$ is the variable wave velocity. Under the continuum assumption as explained by Kundu and Cohen [2008, see pp. 4-5] the momentum equation for small particle displacements can be found from the momentum equation, as done by Stein and Wysession [2009], and is given by:

$$\rho\frac{\partial^2 \mathbf{u}}{\partial t^2} = \nabla \cdot \sigma + \mathbf{f} \tag{2}$$

where $\mathbf{u} = \mathbf{u}(x, z, t)$ is the velocity, $\rho = \rho(x, z)$ is the density, σ is the stress tensor and $\mathbf{f} = \mathbf{f}(x, z, t)$ denotes the body forces. Equation (2) can also be called the navieres primitive equation of motion. By studying the strain of a material in 3 dimensions as done by Stein and Wysession [2009, pp. 49-51], we can find the stress tensor

$$\sigma = \lambda(\nabla \cdot \mathbf{u})\mathbf{I} + \mu(\nabla\mathbf{u} + \nabla\mathbf{u}^T) \tag{3}$$

where we assume the material to be linear elastic, isotropic and that the stresses are symmetric. σ is the stress tensor, $\mathbf{u}$ is the displacement vector, $\mathbf{I}$ is the identity matrix, λ is lameés first constant, and μ is the shear modulus. Inserting equation (3) into equation (2), we get

$$\mathbf{u}_{tt} = \frac{(\lambda + \mu)}{\rho}\nabla(\nabla \cdot \mathbf{u}) + \frac{\mu}{\rho}\nabla^2\mathbf{u} + \mathbf{f} \tag{4}$$

which is the seismic wave equation.

2.2 The finite difference method

The classic definitions for discretizing derivatives can be found in multiple textbooks and multiple websites. Tveito and Winther [2005, pp. 46] gives a good derivation by using taylor series. We invoke the notation $u^n = u(x, y, z, t)$, $u^{n-1} = u(x, y, z, t - \Delta t)$ and $u^{n+1} = u(x, y, z, t + \Delta t)$. We approximate first derivatives by using the midpoint rule:

$$u_t \approx \frac{u^{n+1} - u^{n-1}}{2\Delta t} + O(\Delta t^2) \tag{5}$$

and second derivatives by the central difference formula:

$$u_{tt} \approx \frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} + O(\Delta t^2) \tag{6}$$

where we notice that both approximations have an second order error in time.

2.3 The finite element method

The finite element method is a vast collection of mathematical principals and ideas put together in a comprehensive framework for solving differential equations and boundary value problems. The full detail of the method is beyond the scope of this thesis, but we review the basic idea as given by Anders Logg [2012, pp. 77-94]. We divide the domain into triangles for two dimensional domains, and tetrahedrons for three dimensional domains and call these subdomains for elements. We then seek polynomial approximations to the unknown in each element and then assemble all the parts together to find the global system. We assume that our function can be approximated by the sum:

$$\mathbf{u}(\mathbf{x}) = \sum_{j=0}^{N} c_j \boldsymbol{\psi_j}(\mathbf{x}) \tag{7}$$

where c_j are unknown constants, $\mathbf{x}$ denotes the spatial coordinates and $\boldsymbol{\psi}_j$ are given functions of an arbitrary degree. The functions $\boldsymbol{\psi}_j$ are commonly refered to as basis functions or weight functions. Suppose our problem is to approximate our solution u with a function f. This gives the simple solution:

$$u(x, y) \approx f(x, y) \tag{8}$$

And the difference between these two give a residual:

$$R(x, y) = f(x, y) - u(x, y) \tag{9}$$

The point is now to minimize this residual as much as possible, and this can be done by methods including the interpolation, least squares or weighted residuals method as explained by Langtangen [1999, see pp. 142-144]. We will focus on the latter method, as this is used by the FEniCS software. We define a function space that is spanned by the basis functions:

$$\hat{V} = \text{ span}\{\boldsymbol{\psi_j}\}$$

And seek weight functions:

$$v \in \hat{V}$$

such that the inner product of the residual and the test function is zero:

$$\int_\Omega R(x,y) v d\Omega = 0 \qquad \forall v \in \hat{V} \tag{10}$$

Inserting the expression for R from equation (9) into the inner product in equation (10) we get the equation:

$$\int_\Omega u v d\Omega = \int_\Omega f v d\Omega \tag{11}$$

Equation (11) is the variational form of the problem, and constitutes a linear system of equations. The point of the finite element method is to solve this system using one of many integration methods, including LU solvers and krylov solvers. We end the review of the finite element method here, and interested readers can read the fenics book Anders Logg [2012] or many other good publications on the topic. The rest of this thesis will focus on the variational forms while FEniCS handles the rest.

2.4 Discretizing the wave equation

We first apply the finite difference scheme for time using equations (5) and (6) for the time derivatives in equation (1) and get the explicit formula in time:

$$\frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} + b\frac{u^{n+1} - u^{n-1}}{2\Delta t} = \nabla(c\nabla u^n) + f^n \tag{12}$$

By further introducing the help functions:

$$A = \frac{1}{1 + \frac{b\Delta t}{2}}$$
$$B = \frac{b\Delta t}{2} - 1$$

We get the explicit formula for the time stepping:

$$u^{n+1} = 2Au^n + ABu^{n-1} + A\nabla \cdot (c\nabla u^n) + A\Delta t^2 f^n \tag{13}$$

The space variables are then solved by using the finite element method. Using the chain rule for the laplace term:

$$\nabla \cdot (c\nabla u^n v) = \nabla \cdot (c\nabla u^n)v + c\nabla u^n \nabla v$$

and applying green's theorem, as done by Tveito and Winther [2005, see]:

$$\int_\Omega \nabla \cdot (c\nabla u^n v) d\Omega = \int_\Gamma \mathbf{n} \cdot c\nabla u^n v d\Omega$$

The variational form of equations (13) is:

$$\begin{aligned}\int_\Omega u^{n+1} v d\Omega = 2\int_\Omega A u^n v d\Omega + \int_\Omega A B u^{n-1} v d\Omega \\ - \int_\Omega cA\nabla u^n \nabla v d\Omega + \int_\Gamma A\mathbf{n} \cdot \nabla u^n v d\Gamma \\ + \Delta t^2 \int_\Omega A f^n v d\Omega\end{aligned} \tag{14}$$

2.5 Discretizing the momentum equation

The momentum equation is vector valued, and has components in the x,y, and z directions. The weight functions must therefore also have components in the x,y,z direction. In our two dimentional description, we get the velocity vector

$$\mathbf{u} = u\mathbf{i} + w\mathbf{k} \tag{15}$$

In all the projects, we will work with the same nodes for u and v. we use local form functions N_I where I is the global node number, and we use the local weight functions $\mathbf{w}_I = N_I$. the vector weight function has the form:

$$\mathbf{w} = a_x N_I \mathbf{i} + a_z N_I \mathbf{k} \tag{16}$$

where $a_x = 1$ and $a_z = 0$ gives the x-component of the variational form, and $a_x = 0$ and $a_z = 1$ gives the z-component. Using the chain rule on the stress tensor as we did for the wave equation, we get

$$\nabla \cdot (\sigma \cdot \mathbf{w}) = (\nabla \cdot \sigma) \cdot \mathbf{w} + \sigma : \nabla \mathbf{w}$$

And applying green's theorem

$$\int_\Omega \nabla \cdot (\sigma \cdot \mathbf{w}) d\Omega = \int_\Gamma \mathbf{n} \cdot \sigma \cdot \mathbf{w} d\Gamma$$

we get the variational form of equation (2)

$$\begin{aligned}\int_\Omega \rho \mathbf{u}^{n+1} \cdot \mathbf{w} d\Omega = 2\int_\Omega \rho \mathbf{u}^n \cdot \mathbf{w} d\Omega - \int_\Omega \rho \mathbf{u}^{n-1} \cdot \mathbf{w} d\Omega \\ + \Delta t^2 \int_\Gamma \mathbf{n} \cdot \sigma^n \cdot \mathbf{w} d\Gamma - \Delta t^2 \int_\Omega \sigma^n : \nabla \mathbf{w} d\Omega \\ + \Delta t^2 \int_\Omega \mathbf{f}^n \cdot \mathbf{w} d\Omega\end{aligned} \tag{17}$$

2.6 Boundary conditions

In this thesis, we will give 4 different boundary conditions that are valid for seismic waves and their interactions between solids, liquids and air.

Fixed boundary

At the fixed boundary, the velocity or displacement is known at the boundary node I. $\mathbf{w}_I$ is not used and the variational form in equation (17) is not solved. Instead, A value is directly inserted into the node points at the boundary:

$$\mathbf{u} = \mathbf{U}(x, z, t) \tag{18}$$

where $\mathbf{U}$ is a given boundary function.

Free boundary

The free boundary condition gives a known stress at the boundary, making the boundary integral term in (17) solvable.

$$\mathbf{n} \cdot \sigma = \boldsymbol{\sigma}_n \tag{19}$$

Where σ is the stress tensor, $\mathbf{n}$ is the normal vector and $\boldsymbol{\sigma}_n$ is a given function for the stress at the boundary. $\boldsymbol{\sigma}_n$ is often set to zero to model free surface boundary conditions..

Internal solid-solid boundary

The solid solid boundary condition describes a type of interaction between two solid media, like the Moho discontinuity discussed by Stein and Wysession [2009, see pp. 122] at the crust-mantle boundary. In the solid-solid interface, all velocity component and tractions must be continuous.

$$\begin{aligned} \boldsymbol{\sigma}^{(1)} &= \boldsymbol{\sigma}^{(2)} \\ \mathbf{u}^{(1)} &= \mathbf{u}^{(2)} \end{aligned} \tag{20}$$

where $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$ are the velocity vectors in layers 1 and 2, and $\boldsymbol{\sigma}^{(1)}$ and $\boldsymbol{\sigma}^{(2)}$ are the shear stresses in layers 1 and 2. In the finite element method, the solid-solid boundary gives duplicate nodes at the boundary, and are assembled into the global system.

Internal solid-liquid boundary

The solid-liquid boundary condition describes the interactions between solid and liquid media, like the sea floor and ocean. Due to the vanishing shear stress, the normal tractions and displacements need to be continuous. The shear stress in the solid vanishes at the boundary, and there is no restriction on the shear displacements.

$$\begin{aligned}
\boldsymbol{\sigma}_n^{(1)} &= \boldsymbol{\sigma}_n^{(2)} \\
\boldsymbol{\sigma}_s^{(1)} &= 0 \\
\mathbf{u}_n^{(1)} &= \mathbf{u}_n^{(2)} \\
\mathbf{u}_s^{(1)} &\neq \mathbf{u}_s^{(2)}
\end{aligned} \tag{21}$$

where $\boldsymbol{\sigma}_n$ denotes the normal stress, $\boldsymbol{\sigma}_s$ is the shear stress, $\mathbf{u}_n$ is the normal displacements, and $\mathbf{u}_s$ denotes the shear displacements. The solid-liquid boundary produces duplicate nodes at the boundary as for the solid-solid boundary, and are assembled into the global system.

2.7 Sponge layers

In the finite element method, boundaries are forced on the domain. If no boundary is specified as a essential boundary condition, the natural boundary conditions are applied. This gives difficulties if one wants the solution to flow out of the domain. One solution to this is by using sponge layers. The sponge layer is a type of damping layer often used to curb solutions to rest. We present two types of sponge layers: The damping function and the input method. The damping function can be implemented by inserting:

$$d = b\frac{\partial \mathbf{u}}{\partial t} \tag{22}$$

into the differential equation. This causes natural damping where
$b = b(x; \alpha_1, ..., alpha_N)$ is the damping function. the values $\alpha_1, ..., \alpha_N$ are constants that depend on the problem and domain. Large values of b cause a larger damp effect. The damping function is easily applied to simple geometries, but finding a function $b(x)$ in more complex boundaries can be difficult. In the input method we force the solution to be reduced by setting

$$\mathbf{u} = \mu\mathbf{u} \tag{23}$$

for every time step in the domain considered. $\mu \in (0, 1)$ gives the damping, where 0 is absolute damping and 1 is no damping effect. The input method is easily applied to more complex geometries, but the method itself can produce large discontinuities in the domain, giving total reflections instead of dampings.

2.8 Error control, stability and convergence

The combination of the finite difference and finite element method gives a explicit set of equations to be solved at each time step, and by this method we also impose stability conditions on the numerical scheme. Although important, the mathematics is involved, and left for further analysis, yet we will keep in mind the existence of stability in our programming. Another important property of the numerical scheme is the existense of numerical dispersion. For waves with an angular frequency ω, the numerical scheme produces a numerical frequency $\hat{\omega}$ where $\omega \neq \hat{\omega}$. Such an analysis is also quite involved in the finite element method, and is also left for further study, yet Langtangen [1999, see pp. 656] gives a nice review of the method for a finite difference scheme. In the numerical testing, we will have analytic solutions to compare our simulations with, and we put an emphasis on investigation of errors. The L2 norm error can be defined as

$$E_{L2} = \sqrt{\frac{\sum_{i=0}^{N}(u_e - u)^2}{N}} \tag{24}$$

where E_{L2} is the L2 norm error, u_e is the exact solution, u is the numerical solution and N is the number of nodes. For P1 elements we get a second order error in the spatial coordinates. Combined with the second order errors in the finite difference schemes for the time discretization, we get the error in the scheme

$$E_1 = A_x(\Delta x)^2 + A_z(\Delta z)^2 + A_t(\Delta t)^2$$

where we notice that halving this error gives

$$E_2 = A_x(\frac{\Delta x}{2})^2 + A_z(\frac{\Delta z}{2})^2 + A_t(\frac{\Delta t}{2})^2$$

and that the ratio between the errors are

$$\frac{E_2}{E_1} = 0.25$$

This shows that the error is reduced by a factor 4 when halving spatial and time steps. We will call the number 0.25 the error reduction rate. The spatial and time steps can be collected into a common parameter h, such that the error is given by

$$E = Ch^2 \tag{25}$$

where E is the error, C is some constant and $h = h(\Delta x, \Delta z, \Delta t)$ is a common parameter for the spatial and time steps. The exponent is commonly referred to as the convergence rate.

3 Waves on a sponge layer

In this first project, the performance of a sponge layer will be tested for a simple wave problem on a rectangular domain. Waves are sent into the sponge layer, and it's ability to damp out the motion will be analyzed. We assume a rectangular domain Ω with length L and height H. The domain is divided into two sub domains Ω_1 and Ω_2 divided by a vertical line at the point $x = x_S$, We give the first and second domain the lengths L_p and L_s respectively, and the height of both domains are H. the subscripts p and s are short for p-wave and sponge layer. The problem is shown in figure 1. Each domain is divided into $n_p \times m$ and $n_s \times m$ elements respectively.

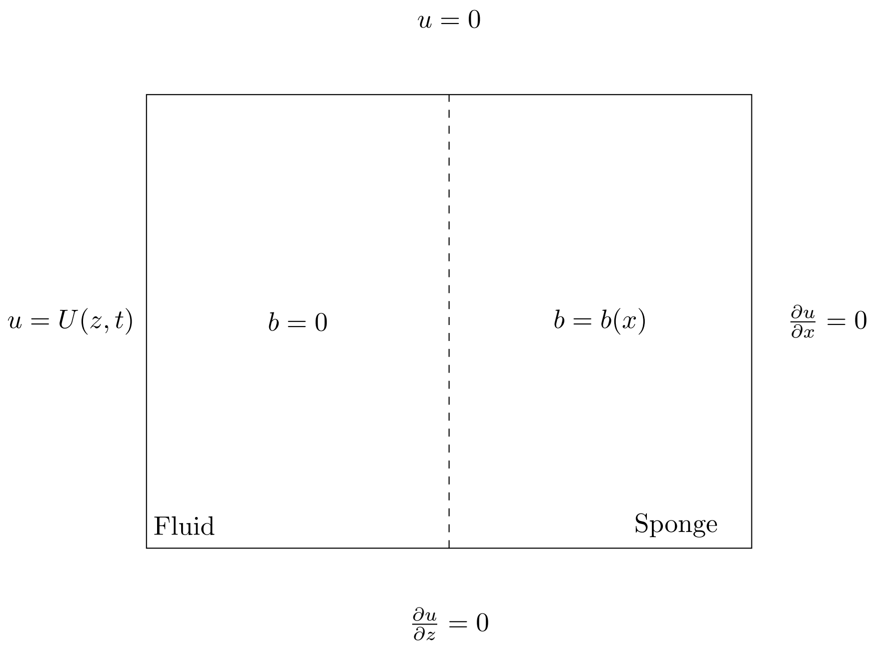


Figure 1: The problem where waves travel with horizontal incidence into a sponge layer

3.1 An analytic solution

In the fluid layer we have no damping and a constant wave velocity c_1. In the sponge layer we apply a damping coefficient only dependent on x and a constant wave velocity c_2. Equation (1) then reduces to:

$$\frac{\partial^2 u_1}{\partial t^2} = c_1^2 \nabla^2 u \qquad x \in (0, L_p) \tag{26}$$

$$\frac{\partial^2 u_2}{\partial t^2} + b(x)\frac{\partial u_2}{\partial t} = c_2^2 \nabla u \qquad x \in (L_p, L_s) \tag{27}$$

For the fluid and sponge respectivly. u_1 is the displacement in the fluid layer, and u_2 is the displacement in the sponge layer. The boundary value problem is subject to 4 boundary conditions in the domain. At the top $y = H$ we assume no displacements. At the bottom $y = 0$ and at the right $x = L_s$ we assume Neumann boundary conditions. At the left hand boundary $x = 0$ we have an inflow condition. All four boundary conditions are stated as

$$u_1(x,H,t) = 0 \tag{28}$$

$$\frac{\partial u_1(x,0,t)}{\partial z} = 0 \tag{29}$$

$$\frac{u_2(L,z,t)}{\partial x} = 0 \tag{30}$$

$$u_1(0,z,t) = U(z,t) \tag{31}$$

This boundary value problem has an analytical solution by solving equation (26) by separation of variables. The calculations are not done in this thesis, but the solution can be on the form

$$u_1(x,z,t) = A\sin(\omega t - kx)\cos(lz) \tag{32}$$

provided the dispersion relation is satisfied.

$$c^2 = \frac{\omega^2}{k^2 + l^2} \tag{33}$$

equation (31) needs to satisfy equations (26), (28) and (29), and a reasonable ansatz is a solution on the same form as equation (32). We assume

$$U(0,z,t) = A\sin(\omega t)\cos(l(z+B)) \tag{34}$$

where A is the amplitude of the incoming waves, and l and B are determined by the boundary conditions. By inserting equation (34) into equation (29), it is shown that $B = 0$ for non trivial solutions. By applying equation (34) into (28) the constants from equation (33) get the values:

$$l_k = \frac{\pi}{2h}(1+k)$$

where k takes the integer values 0,1,2,.. The resulting inflow condition is:

$$U(z,t) = A\sin(\omega t)\cos(\frac{\pi z}{2h}(1+k)) \tag{35}$$

3.2 Simulations and results

For the convergence tests, we run three simulations with a total simulation time of T=10s, and with equally spaced time and spatial resolutions. We use p1 elements, and the implementation is given in section 9.1. the time and spatial values specified as

- $\Delta t = 0.01$, $\Delta x = 1/24$, $\Delta z = 1/24$
- $\Delta t = 0.005$, $\Delta x = 1/48$, $\Delta z = 1/48$
- $\Delta t = 0.0025$, $\Delta x = 1/96$, $\Delta z = 1/96$

Run	L	x_s	b(x)	Δx	Δz	Δt	E_{max}	E_{l2n}	C_{max}	C_{l2n}
1	2	1	b_l	1/24	1/24	0.01	0.06645	0.02588	-	-
2	2	1	b_l	1/48	1/48	0.005	0.02225	0.00970	0.335	0.375
3	2	1	b_l	1/96	1/96	0.0025	0.01647	0.00662	0.740	0.682
1	3	1	b_l	1/24	1/24	0.01	0.06350	0.02390	-	-
2	3	1	b_l	1/48	1/48	0.005	0.01614	0.00594	0.254	0.250
3	3	1	b_l	1/96	1/96	0.0025	0.0093	0.00301	0.575	0.520
1	3	1	b_q	1/24	1/24	0.01	0.07274	0.02659	-	-
2	3	1	b_q	1/48	1/48	0.005	0.02215	0.00793	0.304	0.298
3	3	1	b_q	1/96	1/96	0.0025	0.0093	0.00354	0.419	0.447

Table 1: Table of numerical results for 3 different simulations. L is the total length of the domain, x_s is the x coordinate of the boundary between fluid and sponge. b_l and b_q denotes the linear and quadratic damping functions used. Δx and Δz are the element spacings in the x and z-directions, and Δt is the time step. E_{max} and E_{l2n} are the maximum and L2 norm errors in the simulations, and C_{max} and C_{l2n} are the error reduction rates for the maximum and L2 norm errors with the respect to the previous simulation

We test the sponge by using a linear and a quadratic function each given by

$$b_l(x; L_p) = 10(x - L_p) \tag{36}$$

$$b_q(x; L_p) = 10(x^2 - 2L_p x + L_p^2) \tag{37}$$

The linear function is continuous in the point L_p, and the quadratic function has the function value and the first derivative continous at L_p. The values $k = 0$ and $\omega = 10$ are chosen, so that the constants l_k and k_k get the forms:

$$l_0 = \frac{\pi}{2h}$$

$$k_0 \pm \sqrt{\frac{\omega^2}{c^2} - \frac{\pi^2}{4h^2}}$$

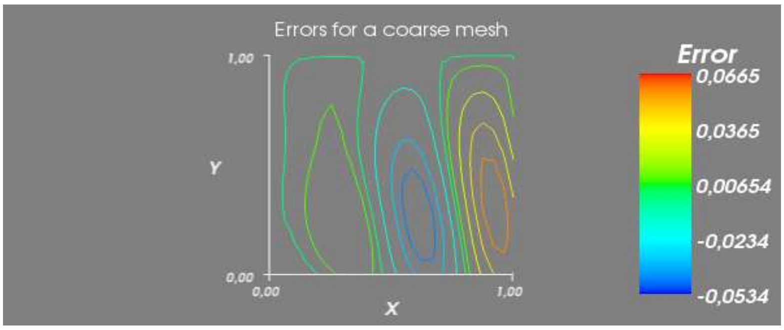
The three simulations are run with the following domains and damping functions.

- $L = 2$ and $L_p = 1$ with the damping coefficient in equation (36).
- $L = 3$ and $L_p = 1$ with the damping coefficient in equation (36).
- $L = 3$ and $L_p = 1$ with the damping coefficient in equation (37).

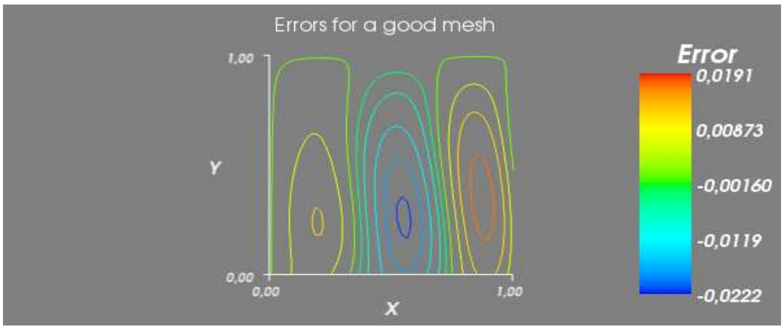
The results from the simulations are given in figure 2, 3, 4 and table 1.

3.3 Conclusion

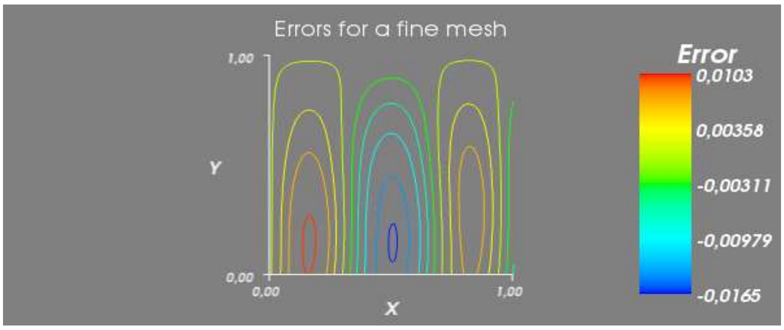
An analysis of the scheme shows that when halving the time steps and spatial steps, the maximum error and the L2 norm error from equation (25) should have an error reduction factor around 0.25. Table 1 shows a reduction of the L2 norm and maximum errors, but not with the correct factor. The second simulation with a larger sponge layer gives a slightly better result. The L2 norm and



(a)



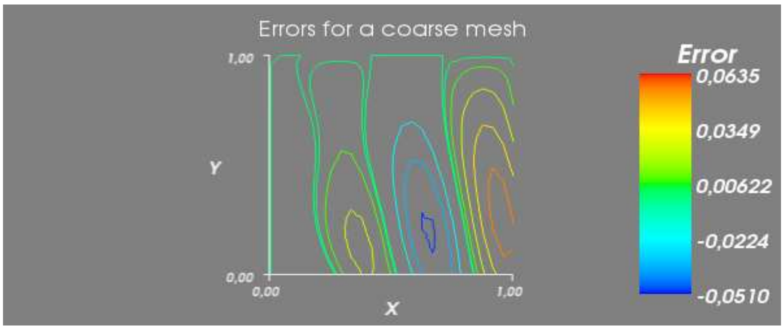
(b)



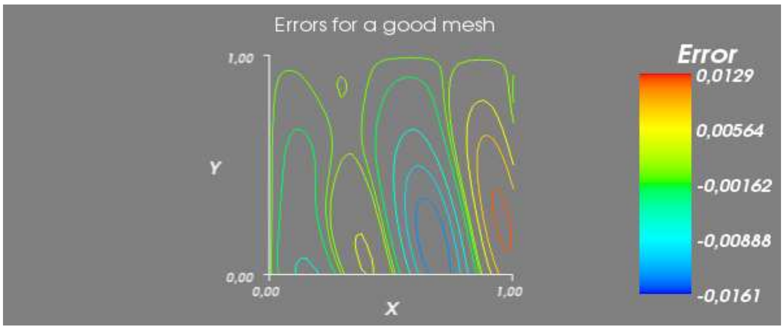
(c)

Figure 2: Figure of the errors in the fluid domain for the run with $L = 2$, $x_s = 1$ and a linear damping in the sponge layer. (a) shows the errors for the coarse mesh, (b) shows the errors for the finer mesh, and (c) shows the errors for the finest mesh

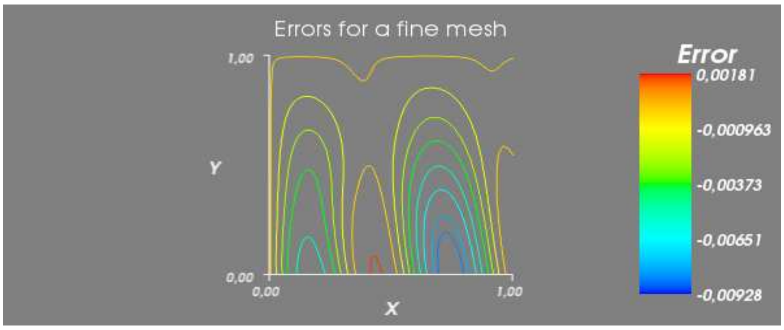
maximum error is reduced by almost a factor of 0.25 between simulation 1 and 2, but is only reduced by a factor 0.5 between simulations 2 and 3. The errors in with the quadratic damping function are worse than for the linear damping function for the same length of the sponge, The convergence is also worse between the first and second run, but is slightly better between the second and third run. In all cases, it seems that the errors from the sponge become more dominant for better resolutions. figures 2, 3 and 4 show a periodic behaviour



(a)



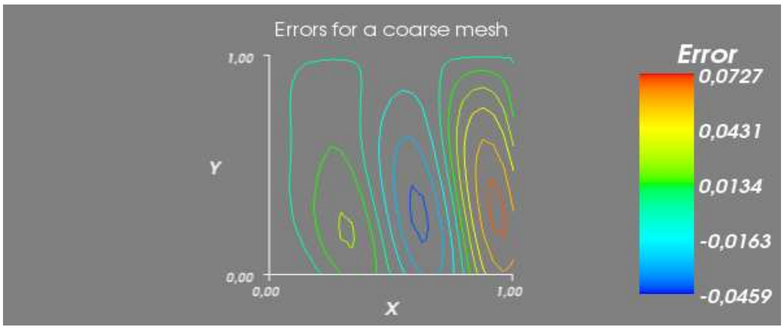
(b)



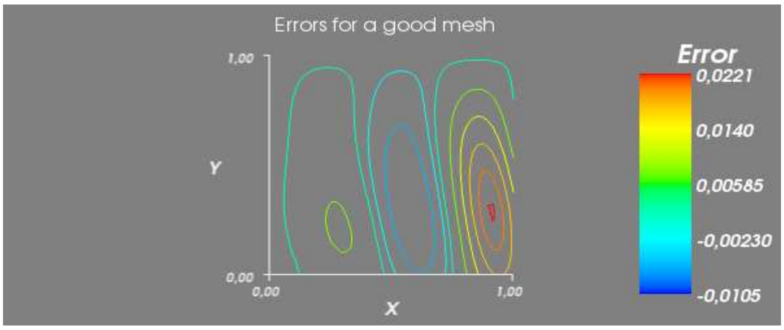
(c)

Figure 3: Figure of the errors in the fluid domain for $L = 3$, $x_s = 1$ and a linear damping in the sponge layer. (a) shows the errors for the coarse mesh, (b) shows the errors for the finer mesh, and (c) shows the errors for the finest mesh

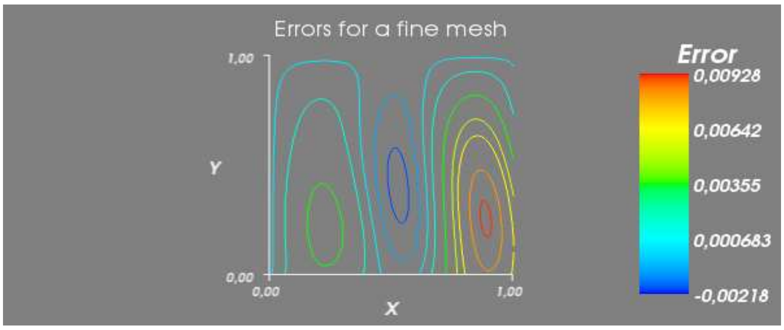
of the error, indicating that the sponge layer is producing reflected waves with a certain amplitude. In table 2 we have approximated values of the amplitudes from the reflected waves by subtracing the largest and smallest errors in figures and taking the square root2, 3 and 4. The amplitdes are large for poor resolutions, but are reduced with finer resolutions.



(a)



(b)



(c)

Figure 4: Figure of the errors in the fluid domain with $L = 3$, $x_s = 1$ and a quadratic damping function in the sponge layer. (a) shows the errors for the coarse mesh, (b) shows the errors for the finer mesh, and (c) shows the errors for the finest mesh

4 The Seismic Wave Equation with Test Solutions

In this project, an implementation of the momentum equation will be tested by simple analytic solutions, and the boundary value problem will be simplified by a technique we call test solutions. Assume a rectangular domain Ω of length L

Run	L	x_s	b(x)	A_r
1	2	1	b_l	0.245
2	2	1	b_l	0.144
3	2	1	b_l	0.116
1	3	1	b_l	0.239
2	3	1	b_l	0.120
3	3	1	b_l	0.074
1	3	1	b_q	0.244
2	3	1	b_q	0.128
3	3	1	b_q	0.075

Table 2: Table of the calculated amplitudes of the reflected waves from the sponge layer in all 3 simulations. L denotes the length of the domain, x_s the coordinate of the boundary between fluid and sponge, and b_l and b_q the linear and quadratic damping functions respectivley.

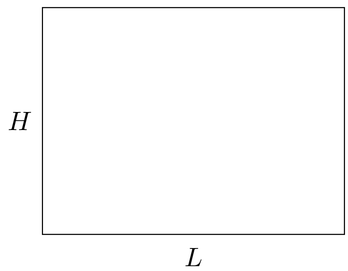


Figure 5: The rectanguar domain used in the problem

and height H, as given in figure 5. The domain is divided into $n \times m$ elements in the x and z directions respectively. We assume no body forces in this problem, so equations (2) and (3) reduce to

$$\rho \frac{\partial^2 \mathbf{u}}{\partial t^2} = \nabla \cdot \sigma \qquad \text{in } \Omega \tag{38}$$

$$\sigma = \lambda(\nabla \cdot \mathbf{u})\mathbf{I} + \mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T) \qquad \text{in } \Omega \tag{39}$$

in the domain. We consider the problem at the times $t = t_0, t_1, ..., t_n$, and assume that we have an analytic soluion $\mathbf{u}_e$ on the whole domain for all t. In the test solution method, u_e is applied as initial and boundary conditions. we then have

$$\mathbf{u}(x, z, t) = \mathbf{u}_e(x, z, t) \qquad \text{at } t = t_0 \tag{40}$$

$$\mathbf{u}(x, z, t) = \mathbf{u}_e(x, z, t) \qquad \text{at } t = t_1 \tag{41}$$

$$\mathbf{u}(x, z, t) = \mathbf{u}_e(x, z, t) \qquad \text{on } \Gamma \tag{42}$$

By using this method, the need to find more complex solutions by separation of variables or other teqniques are eliminated, and the programs ability to maintain an analytic solution for a given time is tested.

4.1 P and S wave analytic solutions

Known simple solutions of the seismic wave equation are compression and shear waves, denoted as P an S waves. P and S-waves can be divided into further categories as done in Stein and Wysession [2009], but we will concentrate on the coupled P-SV waves in our 2d analysis. A P-wave in the x-z plane can be defined as:

$$u_p = A\mathbf{n}e^{i(k\mathbf{n}\cdot\mathbf{r}-\omega t)} \tag{43}$$

where A is the amplitude of the wave, k is the wave number, ω is the angular frequency, t is the time, and **r** is the spatial coordinate vector,

$$\mathbf{r} = x\mathbf{i} + z\mathbf{k}$$

and **n** is the unit normal vector of the wave, given by:

$$\mathbf{n} = n_x\mathbf{i} + n_z\mathbf{k}$$

satisfying

$$|\mathbf{n}| = 1$$

An S-wave in the x-z plane can be defined by:

$$u_s = B(\mathbf{n}\times\mathbf{j})e^{i(k\mathbf{n}\cdot\mathbf{r}-\omega t)} \tag{44}$$

where **j** is the direction along the positive y-axis. The real part of equation (43) is on the form:

$$\mathbf{u} = A(n_x\mathbf{i} + n_z\mathbf{k})\cos(kn_x x + kn_z z - \omega t) \tag{45}$$

And this is a valid solution of equation 4 provided

$$\omega^2 = \frac{(\lambda + 2\mu)}{\rho}k^2 \tag{46}$$

is satisfied. The real part of the S wave from equation (44) is

$$\mathbf{u} = A(n_z\mathbf{i} - n_x\mathbf{k})\cos(kn_x x + kn_z z - \omega t) \tag{47}$$

and is a solution of equation (4) provided

$$\omega^2 = \frac{\mu}{\rho}k^2 \tag{48}$$

is satisfied.

4.2 Simulations and results

The program is run with the P and S wave test solutions from equations (45) and (47). The variational form of the problem is given in (17) and we use p1 elements. The implementation is given in section 9.2. For both test solutions,

P	θ	Δx	Δz	Δt	E_{Max}	E_{L2}	C_{max}	C_{L2}	A_r
1	0	1/24	1/24	0.0075	1.71e-7	6.56e-8	-	-	0.0003
2	0	1/48	1/48	0.00375	4.07e-8	1.64e-8	0.238	0.250	0.0001
3	0	1/96	1/96	0.001875	9.94e-9	4.09e-9	0.244	0.249	6e-5
1	26.57	1/24	1/24	0.0075	5.41e-7	2.14e-7	-	-	0.0005
2	26.57	1/48	1/48	0.00375	1.30e-7	5.41e-8	0.240	0.252	0.0002
3	26.57	1/96	1/96	0.001875	3.19e-8	1.35e-8	0.246	0.250	0.0001
1	71.57	1/24	1/24	0.0075	7.65e-7	2.96e-7	-	-	0.0005
2	71.57	1/48	1/48	0.00375	1.83e-7	7.44e-8	0.239	0.251	0.0003
3	71.57	1/96	1/96	0.001875	4.48e-8	1.86e-8	0.245	0.250	0.0001
1	90	1/24	1/24	0.0075	1.71e-7	6.56e-8	-	-	0.0003
2	90	1/48	1/48	0.00375	4.07e-8	1.64e-8	0.238	0.250	0.0001
3	90	1/96	1/96	0.001875	9.94e-9	4.09e-9	0.244	0.249	6e-5

Table 3: Table containing the numerical results of the simulations of the seismic wave equation with a P wave test solution. The angle θ gives the angle of propagation with the x-axis, Δx and Δz give the element spacings in the x and y direction. Δt is the time step. E_{max} and E_{L2} denotes the maximum and L2 norm errors respectvely. C_{max} and C_{L2} are the error reduction rates for the maximum and L2 norm errors with respect to the previous simulation. A_r are the estimated amplitudes from the reflected waves

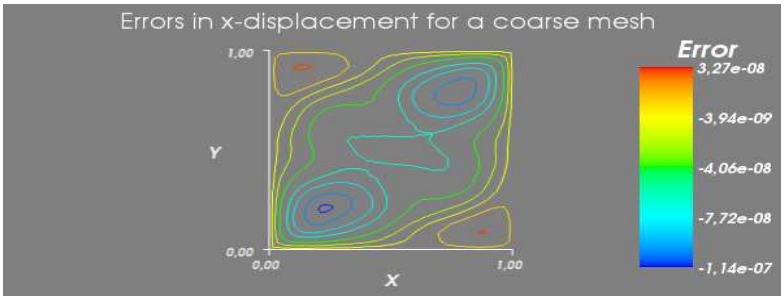
the length $L = 1$, height $H = 1$ and a total simulation time of $T = 5$ are chosen. For the material, the constants $\lambda = 1$, $\mu = 1$ and $\rho = 1$ are used. The wave parameters are $A = 1$ and $\omega = 0.5$. A convergence test is made by running 3 different simulations for both test solutions with the time and spatial steps evenly distributed

- $\Delta t = 0.0075$, $\Delta x = 1/24$, $\Delta z = 1/24$
- $\Delta t = 0.00375$, $\Delta x = 1/48$, $\Delta z = 1/48$
- $\Delta t = 0.001875$, $\Delta x = 1/96$, $\Delta z = 1/96$

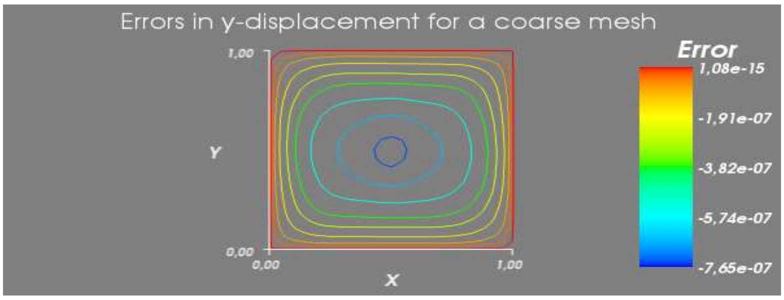
Some results of the simulations are given in tables 3 and 4. the component errors for the p-wave simulation with a propagation angle of $\theta = 71.57^0$ with the x-axis is given in figure 6. The component errors for an S-wave with a propagation angle of $\theta = 71.57^0$ with the x-axis is given in figure 7.

4.3 Conclusion

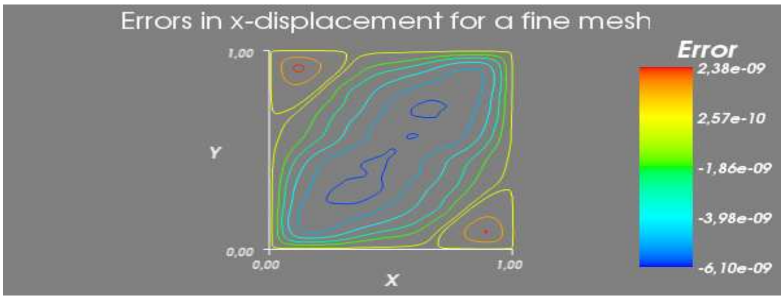
Tables 3 and 4 show the different simulations for different propagation angles for the P and S-wave test solutions. In all cases the error reduction rates are slightly better than 0.25 which we found in equation (25). From figure 6 we see that the errors in x-displacements are larger in the center of the mesh and close to the corner points, and kept to machine precision at the boundaries. The errors in z-displacements are largest at the center of the mesh, and decreases towards the boundaries, where the error is kept to machine precision. In figure 7, all displacements have their maximum error in the center of the mesh, and decrease towards the boundaries where the errors are kept to machine precision. In all cases, the errors are kept small, even for the coarsest time and element



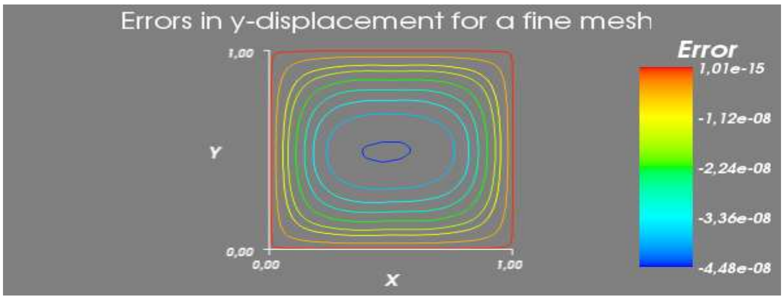
(a)



(b)

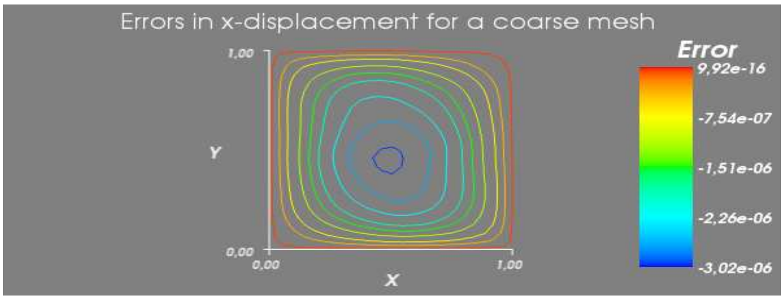


(c)

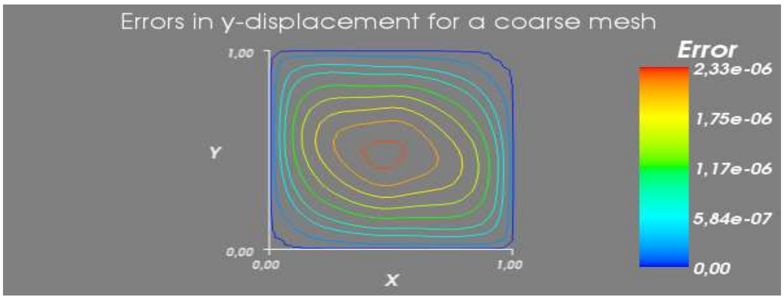


(d)

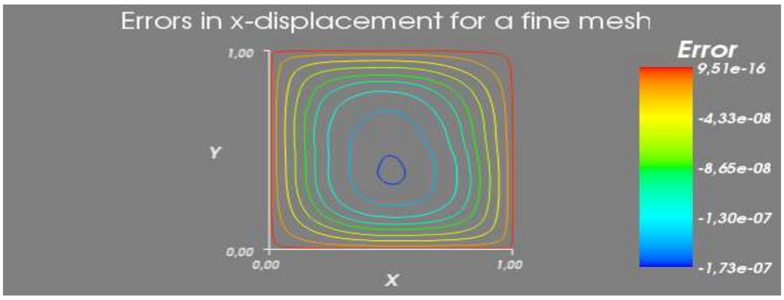
Figure 6: Errors for the x and z-components of displacement for a P-wave with an angle of 71.57^0 with the x-axis. (a) and (b) show the x and z-displacements for a 24x24 mesh respectively, and a time step of 0.0075. figures (c) and (d) show the x and z-displacements for a 96x96 mesh respectivley, and a time step of 0.001875.



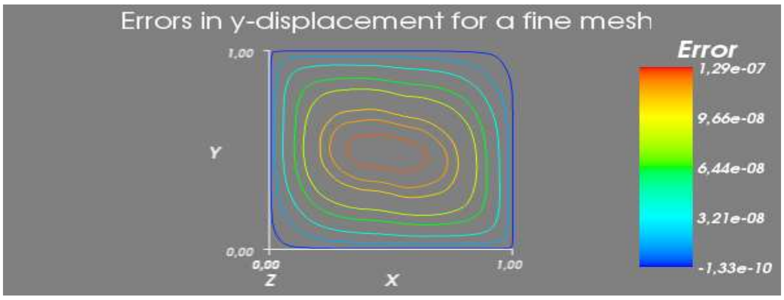
(a)



(b)



(c)



(d)

Figure 7: Errors for the x and z-components of displacement for an S-wave with an angle of 71.57^0 with the x-axis. (a) and (b) show the x and z-displacements for a 24x24 mesh respectively, and a time step of 0.0075. figures (c) and (d) show the x and z-displacements for a 96x96 mesh respectivley, and a time step of 0.001875.

S	θ	Δx	Δz	Δt	E_{Max}	E_{L2}	C_{max}	C_{L2}	A_r
1	0	1/24	1/24	0.0075	4.48e-7	1.71e-7	-	-	0.0004
2	0	1/48	1/48	0.00375	1.07e-7	4.28e-8	0.238	0.250	0.0002
3	0	1/96	1/96	0.001875	2.65e-8	1.07e-8	0.248	0.250	0.0001
1	26.57	1/24	1/24	0.0075	2.81e-6	1.43e-6	-	-	0.0012
2	26.57	1/48	1/48	0.00375	6.47e-7	3.49e-7	0.230	0.244	0.0006
3	26.57	1/96	1/96	0.001875	1.60e-7	8.67e-8	0.247	0.249	0.0003
1	71.57	1/24	1/24	0.0075	3.02e-6	1.44e-6	-	-	0.0012
2	71.57	1/48	1/48	0.00375	6.98e-7	3.53e-7	0.231	0.245	0.0006
3	71.57	1/96	1/96	0.001875	1.73e-7	8.77e-8	0.248	0.249	0.0003
1	90	1/24	1/24	0.0075	4.48e-7	1.71e-7	-	-	0.0004
2	90	1/48	1/48	0.00375	1.07e-7	4.28e-8	0.238	0.250	0.0002
3	90	1/96	1/96	0.001875	2.65e-8	1.07e-8	0.248	0.250	0.0001

Table 4: Table containing the numerical results of the simulations of the seismic wave equation with an S wave test solution. The angle θ gives the angle of propagation with the x-axis, Δx and Δz give the element spacings in the x and z-direction. Δt is the time step. E_{max} and E_{L2} denotes the maximum and L2 norm errors respectivley. C_{max} and C_{L2} are the error reduction rates for the maximum and L2 norm errors with respect to the previous simulation. A_r are the estimated amplitudes of the reflected waves

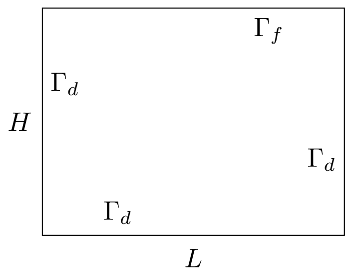


Figure 8: The problem with test solutions for dirichlet boundary conditions and a given surface stress

spacing. By looking at the tables equation 3, 4, the convergence formula (25) and our choices for Δx, Δz and Δt, we see that the constant C in equation (25) must be smaller than one for the simulations. We also keep in mind that a numerical dispersion analysis has not been made, implying that C could be even smaller. In our simulations, we see that the error has a periodic behaviour, implying that the boundaries are producing reflected waves into the domain. The amplitudes are estimated by taking the square of the L2 norm errors in tables 3 and 4, and we see that the amplitudes decrease for better resolutions of the mesh.

5 Seismic test solutions with a given stress

In this project, we aim at implementing the seismic wave equation with test solutions, as we did for the previous project, however in this project we apply a given stress to one of the boundaries instead of a given displacement. This gives insight as to how FEniCS handles boundary integrals and natural boundary conditions. We assume a rectangular domain, as given in figure 8, with the length L and height H. The domain is divided into $l \times m$ elements in the x and z-directions respectivley. As for the previous project, we neglect body forces for this implementation, giving the the equations of motion and stress:

$$\rho \frac{\partial^2 \mathbf{u}}{\partial t^2} = \nabla \cdot \sigma \qquad \text{in } \Omega \tag{49}$$

$$\sigma = \lambda(\nabla \cdot \mathbf{u})\mathbf{I} + \mu(\nabla \mathbf{u} + \nabla \mathbf{u}^T) \qquad \text{in } \Omega \tag{50}$$

Again, we assume an analytic solution u_e, and solve the problem for the times $t = t_0, t_1, ..., t_n$. We apply our analytic solution as boundary and initial conditions so that

$$\mathbf{u}(x, z, t_0) = \mathbf{u}_e(x, z, t_0) \qquad \text{on } \Omega \tag{51}$$

$$\mathbf{u}(x, z, t_1) = \mathbf{u}_e(x, z, t_0) \qquad \text{on } \Omega \tag{52}$$

$$\mathbf{u}(x, z, t) = \mathbf{u}_e(x, z, t) \qquad \text{on } \Gamma_d \tag{53}$$

$$\sigma(\mathbf{u}) = \sigma(\mathbf{u}_e) \qquad \text{on } \Gamma_f \tag{54}$$

5.1 P and S-wave analytic solutions

As for the previous project, the P and S-waves from equations (45) and (47) are solutions of the momentum equation provided the dispersion relations from equations (46) and (48) are satisfied respectivley. These solutions are applied as boundary conditions on Γ_d. On Γ_s, we apply the given surface stress.

$$\begin{aligned} \boldsymbol{\sigma}_n &= \mathbf{n} \cdot \sigma \\ &= \mathbf{k} \cdot (\sigma_{xx}\mathbf{ii} + \sigma_{xz}\mathbf{ik} + \sigma_{zx}\mathbf{ki} + \sigma_{zz}\mathbf{kk}) \\ &= \sigma_{zx}\mathbf{i} + \sigma_{zz}\mathbf{k} \end{aligned} \tag{55}$$

The components of stress are found from equation (3)

$$\begin{aligned} \sigma_{zx} &= \mu(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}) \\ \sigma_{zz} &= \lambda(\frac{\partial u}{\partial x} + \frac{\partial w}{\partial z}) + 2\mu \frac{\partial w}{\partial z} \end{aligned} \tag{56}$$

For the P-wave, the components of stress at Γ_f are:

$$\begin{aligned} \sigma_{zz} &= -\lambda A k(n_x^2 + n_z^2) \sin(kn_x x + kn_z z - \omega t) \\ &\quad - 2\mu A k n_y^2 \sin(kn_x x + kn_z z - \omega t) \\ \sigma_{zx} &= -2\mu A k n_x n_z \sin(kn_x x + kn_z z - \omega t) \end{aligned} \tag{57}$$

And for the S-wave, the components of stress at Γ_f are:

$$\begin{aligned} \sigma_{zx} &= \mu A k(n_x^2 - n_z^2) \sin(kn_x x + kn_z z - \omega t) \\ \sigma_{zz} &= -2\mu A k n_x n_z \sin(kn_x x + kn_z z - \omega t) \end{aligned} \tag{58}$$

P	θ	Δx	Δz	Δt	E_{Max}	E_{L2}	C_{max}	C_{L2}	A_r
1	0	1/24	1/24	0.0075	1.60e-6	2.77e-7	-	-	0.0005
2	0	1/48	1/48	0.00375	3.95e-7	6.61e-8	0.248	0.239	0.0003
3	0	1/96	1/96	0.001875	9.89e-8	1.62e-8	0.249	0.245	0.0001
1	26.57	1/24	1/24	0.0075	4.87e-6	8.49e-7	-	-	0.0009
2	26.57	1/48	1/48	0.00375	1.20e-6	2.01e-7	0.246	0.237	0.0004
3	26.43	1/96	1/96	0.001875	2.97e-7	4.91e-8	0.248	0.244	0.0002
1	71.57	1/24	1/24	0.0075	1.11e-6	2.25e-7	-	-	0.0005
2	71.57	1/48	1/48	0.00375	2.79e-7	5.66e-8	0.253	0.251	0.0002
3	71.57	1/96	1/96	0.001875	6.93e-8	1.41e-8	0.248	0.250	0.0001
1	90	1/24	1/24	0.0075	5.50e-7	1.81e-7	-	-	0.0004
2	90	1/48	1/48	0.00375	1.41e-7	4.68e-8	0.257	0.258	0.0002
3	90	1/96	1/96	0.001875	3.53e-8	1.18e-8	0.250	0.252	0.0001

Table 5: Table containing the numerical results of the simulations of the seismic wave equation with P-wave test solutions. The angle θ gives the angle of propagation with respect to the x-axis, Δx and Δz give the element spacings in the x and z direction. Δt is the time step. E_{max} and E_{L2} denotes the maximum and L2 norm errors. C_{max} and C_{L2} are the error reduction rates with respect to the previous simulation. A_r are the estimated amplitudes of the reflected waves

5.2 Simulations and results

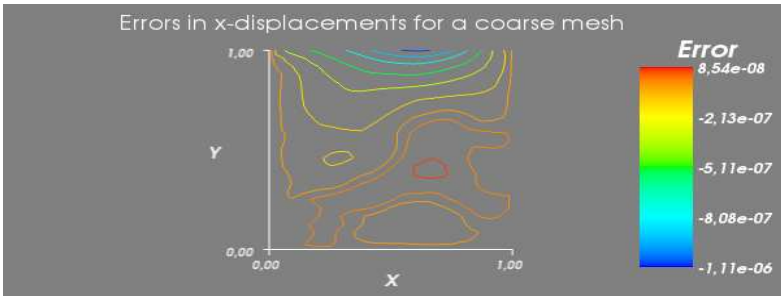
The variational form is given in equation (17) and we use p1 elements. The implementation is given in section 9.3. We run 3 simulations for both the P wave and the S wave test solutions with the length $L = 1$, height $h = 1$ and a total simulation time of $T = 5$. For the material, we choose the constants $\lambda = 1$, $\mu = 1$ and $\rho = 1$. We also choose the parameters $A = 1$ and $\omega = 0.5$. The convergence tests are made by varying the evenly distributed element and time spacings

- $\Delta t = 0.0075$, $\Delta x = 1/24$, $\Delta z = 1/24$
- $\Delta t = 0.00375$, $\Delta x = 1/48$, $\Delta z = 1/48$
- $\Delta t = 0.001875$, $\Delta x = 1/96$, $\Delta z = 1/96$

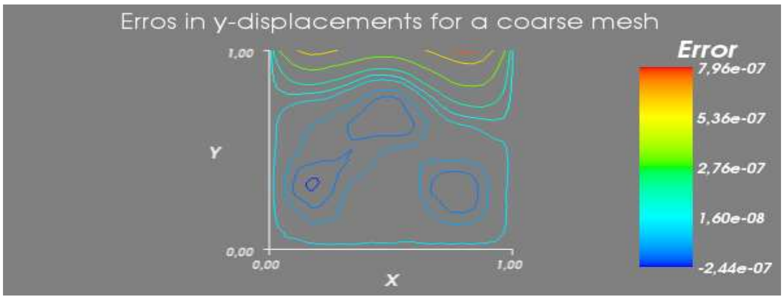
The results for the simulations are given in tables 5 and 6. The component errors for the simulations with an angle of $\theta = 71.57^0$ with the x-axis are given in figures 9 and 10.

5.3 Conclusion

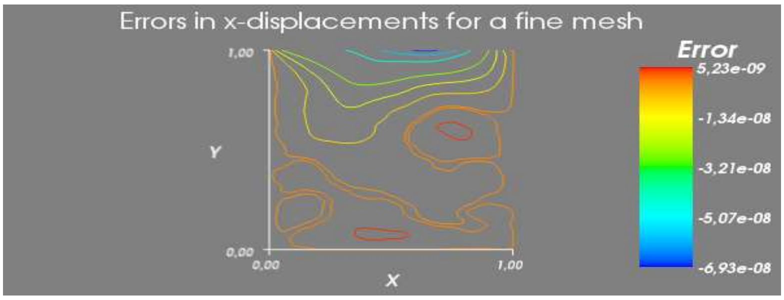
Tables 5 and 6 show that the error reduction rates for both the P and S-wave test solutions are close to the values estimated from equation (25), yet they are slightly worse than for the previous project for some of the simulations. Figure 9 shows the x and z-component errors for a wave propagating with an angle of $\theta = 71.57^0$ with the x-axis. from the figure, 4we see that the larger errors are found at Γ_f. Local error maximums are also found in parts of the inner domain, while the errors at Γ_d are kept to machine precision. Figure 10



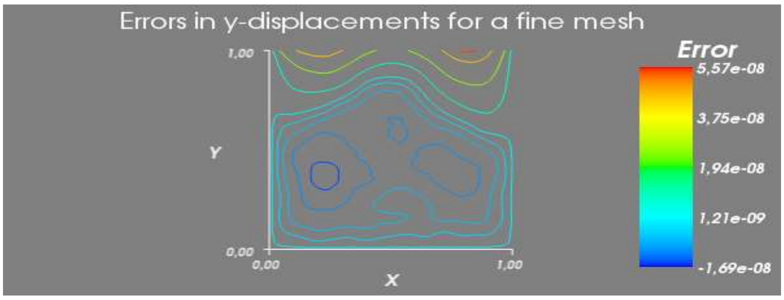
(a)



(b)

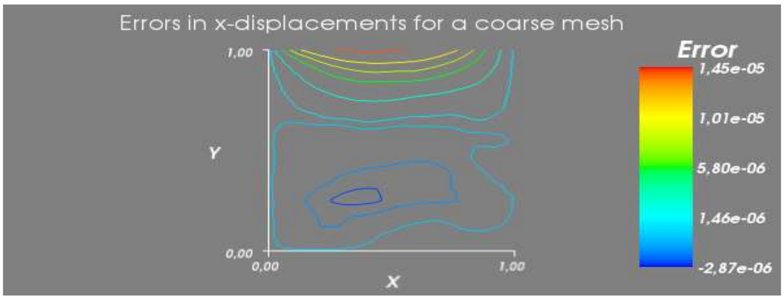


(c)

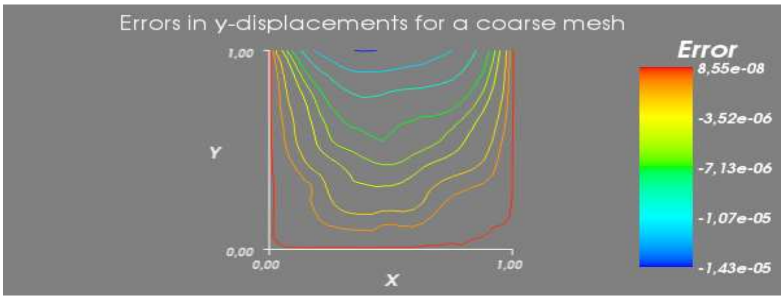


(d)

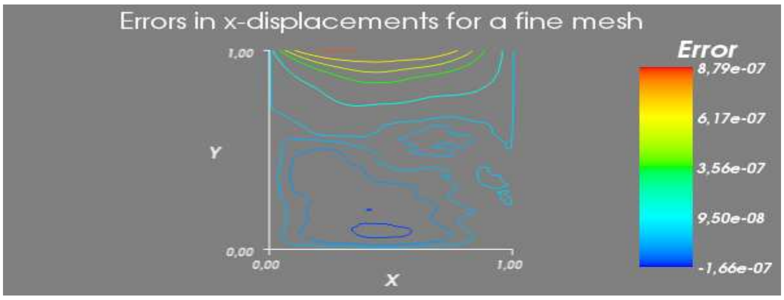
Figure 9: Figures of the displacement errors for a P-wave propagating with an angle of $\theta = 71.57^0$ with respect to the x-axis. (a) and (b) show the x and z-displacements for a 24x24 mesh with a time step of 0.0075. (c) and (d) show the x and z-displacement errors for a 96x96 mesh with time step 0.0001875



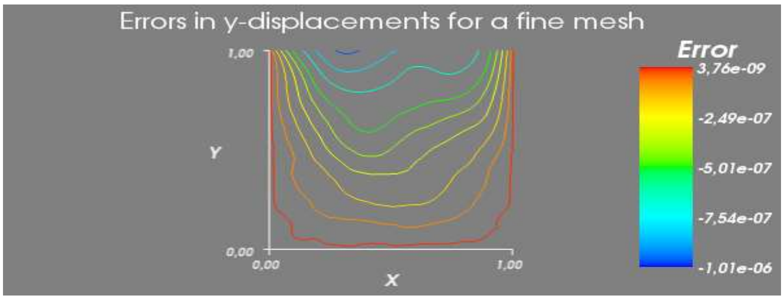
(a)



(b)



(c)



(d)

Figure 10: Figures of the displacement errors for an S-wave propagating with an angle of $\theta = 71.57^0$ with respect to the x-axis. (a) and (b) show the x and z-displacements for a 24x24 mesh with a time step of 0.0075. (c) and (d) show the x and z-displacement errors for a 96x96 mesh with time step 0.0001875

S	θ	Δx	Δz	Δt	E_{Max}	E_{L2}	C_{max}	C_{L2}	A_r
1	0	1/24	1/24	0.0075	2.12e-6	5.26e-7	-	-	0.0007
2	0	1/48	1/48	0.00375	5.32e-7	1.33e-7	0.250	0.253	0.0004
3	0	1/96	1/96	0.001875	1.35e-7	3.36e-8	0.254	0.252	0.0002
1	26.57	1/24	1/24	0.0075	3.36e-5	1.07e-5	-	-	0.0033
2	26.57	1/48	1/48	0.00375	8.53e-6	2.70e-6	0.254	0.253	0.0016
3	26.57	1/96	1/96	0.001875	2.17e-6	6.78e-7	0.255	0.251	0.0008
1	71.57	1/24	1/24	0.0075	1.45e-5	4.74e-6	-	-	0.0022
2	71.57	1/48	1/48	0.00375	3.89e-6	1.20e-6	0.269	0.252	0.0011
3	71.57	1/96	1/96	0.001875	1.01e-6	3.00e-7	0.259	0.251	0.0005
1	90	1/24	1/24	0.0075	1.83e-7	6.35e-8	-	-	0.0003
2	90	1/48	1/48	0.00375	4.81e-8	1.59e-8	0.263	0.251	0.0001
3	90	1/96	1/96	0.001875	1.15e-8	4.03e-9	0.239	0.253	6e-5

Table 6: Table containing the numerical results of the simulations of the seismic wave equation with S-wave test solutions. The angle θ gives the angle of propagation with respect to the x-axis, Δx and Δz give the element spacings in the x and z direction. Δt is the time step. E_{max} and E_{L2} denotes the maximum and L2 norm errors. C_{max} and C_{L2} are the error reduction rates with respect to the previous simulation. A_r are the estimated amplitudes of the reflected waves

shows the x and z-component errors for a wave propagating with an angle of $\theta = 71.57^0$ with the x-axis. The larger errors are in this case also found at Γ_f. For the x-displacements, local maxima of the errors are also found in parts of the interior domain, while the errors in z-displacement decrease towards the boundary Γ_d. For both the x and z-displacement, the errors at Γ_d are kept to machine precision. By looking at the errors in tables 5, 6, the convergence formula (25) and our choices for Δx, Δz, and Δt, we see that the constant C from equation (25) is smaller than 1 for our simulations as for the previous project. We also keep in mind that a numerical dispersion relation analysis is not made, and this implies that the constant C could be even better. In the simulations we see a periodic behaviour of the error that is larger at the free surface and smaller at the bottom. As for the previous project, this implies that the boundaries are producing reflected waves. The calculated amplitudes are given in tables 5 and 6, and in all cases, the amplitudes decrease for better resolutions.

6 A Two layer model with vertical incidence

In this project, the performance of the finite element method in two domains with different material properties will be tested by the test solution process. Assume a rectangular domain Ω divided into the two subdomains Ω_1 and Ω_2 as shown in figure 11. Ω_1 has a length L and a height h. Ω_2 has a length L and the height H. The domains are divided into $l \times m_1$ and $l \times m_2$ elements respectivley, and are separated by the horizontal line $z = 0$. In Ω_1, we have the physical parameters λ_1, μ_1 and ρ_1, and in Ω_2, we have λ_2, μ_2 and ρ_2. All waves are assumed to have the same angular frequencies ω. The stress tensors in the

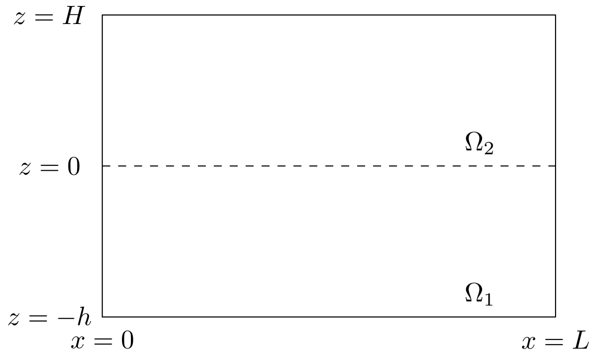


Figure 11: A two layer model for waves traveling at vertical incidence with the boundaries

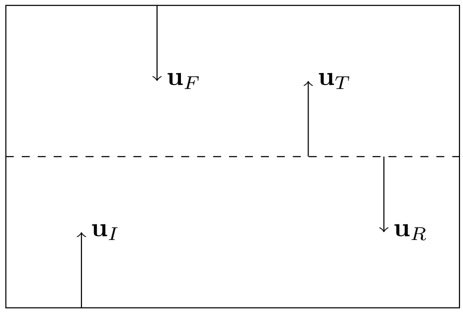


Figure 12: A two layer model for P-waves traveling at vertical incidence with an internal boundary and a free surface

two domains are then

$$\sigma_1 = \lambda_1(\nabla \cdot \mathbf{u}_1)\mathbf{I} + \mu_1(\nabla \mathbf{u}_1 + \nabla \mathbf{u}_1^T) \qquad \text{in } \Omega_1 \tag{59}$$

$$\sigma_2 = \lambda_2(\nabla \cdot \mathbf{u}_2)\mathbf{I} + \mu_2(\nabla \mathbf{u}_2 + \nabla \mathbf{u}_2^T) \qquad \text{in } \Omega_2 \tag{60}$$

and are inserted into equation (17) to get the variational forms for each layer respectivley.

6.1 P-wave analytic solutions

For the two layer problem from figure 12, an incoming wave from below produces a reflected and a transmitted wave. At the free boundary, the transmitted wave produces another reflected wave. The possible analytical wave solutions for the problem are

$$
\begin{aligned}
\mathbf{u}_I &= Ie^{i(\omega t - k_1 z)}\mathbf{k} \\
\mathbf{u}_R &= Re^{i(\omega t + k_1 z)}\mathbf{k} \\
\mathbf{u}_T &= Te^{i(\omega t - k_2 z)}\mathbf{k} \\
\mathbf{u}_F &= Fe^{i(\omega t + k_2 z)}\mathbf{k}
\end{aligned} \tag{61}
$$

where I denotes the incoming P-wave, R the reflected wave, T the transmitted wave and F the reflected wave from the free boundary. Theese waves are valid solutions of the seismic wave equation provided

$$
\begin{aligned}
\omega^2 &= (\frac{\lambda_1 + 2\mu_1}{\rho_1})k_1^2 \\
\omega^2 &= (\frac{\lambda_2 + 2\mu_2}{\rho_2})k_2^2
\end{aligned} \tag{62}
$$

for the two layers respectively. From the boundary condition (20) we must have continuity of displacements at $z = 0$. Inserting the wave solutions from equation (61) we get

$$Ie^{(i\omega t)} + Re^{(i\omega t)} = Te^{(i\omega t)} + Fe^{(i\omega t)} \tag{63}$$

Giving a relation between amplitudes:

$$I + R = T + F \tag{64}$$

From equation (20) we must have continuity of stress at $z = 0$, and inserting the wave solutions from equation (61) into the boundary condition we get

$$(\lambda_1 + 2\mu_1)k_1 ie^{i(\omega t)}(R - I) = (\lambda_2 + 2\mu_2)k_2 ie^{i(\omega t)}(F - T) \tag{65}$$

Giving:

$$\frac{k_1(\lambda_1 + 2\mu_1)}{k_2(\lambda_2 + 2\mu_2)}(R - I) = F - T \tag{66}$$

at $z = H$ we have a free boundary condition given from equation (19), and inserting the wave solutions from equation (61) into this condition gives:

$$-T(\lambda_2 + 2\mu_2)k_2 ie^{i(\omega t - k_2 H)} + F(\lambda_2 + 2\mu_2)k_2 ie^{i(\omega t + k_2 H)} = 0 \tag{67}$$

Giving the relation between the transmitted and reflected wave from the free surface as:

$$T = Fe^{2ik_2 H} \tag{68}$$

Equations (64), (66) and (68) give a system of equations that can be solved for R, T and F assuming I is known, and doing so produces the following amplitudes:

$$
\begin{aligned}
R &= -I\frac{(1 + C)}{(1 - C)} \\
T &= \frac{I}{(1 + r^{-1})}\Big(1 - \frac{(1 + C)}{(1 - C)}\Big) \\
F &= \frac{I}{(1 + r)}\Big(1 - \frac{(1 + C)}{(1 - C)}\Big)
\end{aligned} \tag{69}
$$

Where we have defined:

$$\begin{aligned}
\alpha &= \frac{k_1}{k_2}\frac{(\lambda_1 + 2\mu_1)}{(\lambda_2 + 2\mu_2)} \\
r &= e^{2ik_2H} \\
C &= \alpha\frac{(1+r)}{(1-r)}
\end{aligned} \tag{70}$$

for simplicity of notation. The two layer problem is a closed system, and this physically forces the incoming and reflected waves to have the same magnitude of amplitudes. Also, the transmitted and second reflected wave must also have the same amplitudes.

$$\begin{aligned}
|I| &= |R| \\
|T| &= |F|
\end{aligned}$$

To simplify our calculations a bit more, we show that C from equation 70 is a pure imaginary number $C = ci$. By using some complex theory we get:

$$\begin{aligned}
C &= \alpha\frac{1+e^{2ik_2H}}{1-e^{2ik_2H}} \\
&= \alpha\frac{(1+e^{2ik_2H})(1+e^{-2ik_2H})}{(1-e^{2ik_2H})(1+e^{-2ik_2H})} \\
&= \alpha\frac{2+e^{2ik_2H}+e^{-2ik_2H}}{e^{-2ik_2H}-e^{2ik_2H}} \\
&= \alpha\frac{2\cos(2k_2H)+2}{-2i\sin(2k_2H)} \\
&= \alpha i\frac{cos(2k_2H)+1}{sin(2k_2H)} \\
&= ci
\end{aligned}$$

Taking the absolute value of the amplitude of the reflected wave from equation (69) gives:

$$\begin{aligned}
|R| &= |-I\frac{(1+ci)}{(1-ci)}| \\
&= \sqrt{I^2\frac{(1+ci)(1-ci)}{(1-ci)(1+ci)}} \\
&= |I|
\end{aligned}$$

From equation (68), we get the relation:

$$\begin{aligned}
|T| &= |Fe^{2ik_2H}| \\
&= \sqrt{F^2(\cos(2ik_2H)+i\sin(2ik_2H))(\cos(2ik_2H)-i\sin(2ik_2H))} \\
&= \sqrt{F^2(\cos^2(2ik_2H)+\sin^2(2ik_2H))} \\
&= |F|
\end{aligned}$$

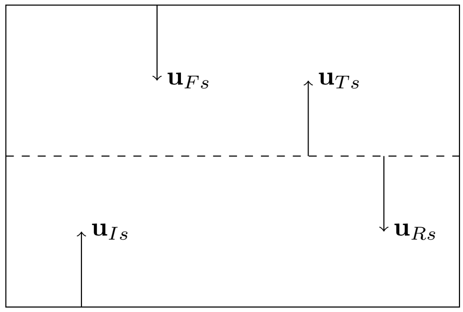


Figure 13: A two layer model for S-waves traveling at vertical incidence with an internal boundary and a free surface

We notice that the analytical solution provided is valid for general solid-solid and solid-fluid boundaries.

6.2 S-wave analytic solutions

For the S-waves, the solutions have a similar form as for the P-waves. The incoming S-wave produces a reflected and transmitted wave at the internal boundary for solid-solid boundaries, and the transmitted wave produces a new reflected wave at the free surface. The S-wave solutions are on the form

$$\mathbf{u}_{Is} = I_s e^{i(\omega t - k_1 z)}\mathbf{i} \tag{71}$$

$$\mathbf{u}_{Rs} = R_s e^{i(\omega t + k_1 z)}\mathbf{i} \tag{72}$$

$$\mathbf{u}_{Ts} = T_s e^{i(\omega t - k_2 z)}\mathbf{i} \tag{73}$$

$$\mathbf{u}_{Fs} = F_s e^{i(\omega t + k_2 z)}\mathbf{i} \tag{74}$$

where Is denotes the incoming wave, Rs the reflected wave, Ts the transmitted wave and Fs the reflected wave from the free surface. Theese equations are solutions of the seismic wave equation provided

$$\begin{aligned} \omega^2 &= (\frac{\mu_1}{\rho_1})k_1^2 \\ \omega^2 &= (\frac{\mu_2}{\rho_2})k_2^2 \end{aligned} \tag{75}$$

Are satisfied for layer 1 and 2 respectively. Continuity of displacement at $z = 0$ from equation (20) gives

$$I_s + R_s = T_s + F_s \tag{76}$$

Continuty of stress from equation (20) at $z = 0$ gives

$$\frac{k_1\mu_1}{k_2\mu_2}(R_s - I_s) = F_s - T_s \tag{77}$$

At the free boundary $z = H$, the free surface condition from equation (19) gives

$$T = Fe^{2ik_2H} \tag{78}$$

Equations (76), (77) and (78) gives a system of equations as for the P-wave solutions, and solving for the amplitudes gives:

$$R_s = -I_s\frac{(1 + C_s)}{(1 - C_s)} \tag{79}$$

$$T_s = I_s\frac{1}{(1 + r^{-1})}\Big(1 - \frac{(1 + C_s)}{(1 - C_s)}\Big) \tag{80}$$

$$F_s = I_s\frac{1}{(1 + r)}\Big(1 - \frac{(1 + C_s)}{(1 - C_s)}\Big) \tag{81}$$

where we have defined the help constants

$$\begin{aligned} \alpha_s &= \frac{k_1\mu_1}{k_2\mu_2} \\ r &= e^{2ik_2H} \\ C_s &= \alpha_s\frac{(1 + r)}{(1 - r)} \end{aligned} \tag{82}$$

We notice that all the constants are similar to what we had for the P-wave solutions, and following the same procedures as for the previous section, we see that energy is conserved. We notice that for $\mu_2 = 0$, $\sigma_2 = 0$, giving $\mathbf{u}_{Ts} = 0$ and $\mathbf{u}_{Fs} = 0$. We therefore need to apply the solid-liquid boundary condition from equation (21). In this case, the only remaining boundary condition is the vanishing stress at $z = 0$ from equation (21), giving $R = I$. So for the solid-liquid case, the amplitudes have the values

$$R = I \tag{83}$$

$$T = 0 \tag{84}$$

$$F = 0 \tag{85}$$

6.3 Simulations and results

The version of FEniCS used in this thesis does not handle complex numbers, so our analytic solutions are computed in python numpy arrays in scipy. Interested readers can read the scipy documentation by Jones et al.. This requires mesh information to be extracted from FEniCS, used in python numpy, and then ported back into FEniCS. This is done in the two layer code in section 9.4. We run 2 simulations for the P-wave test solutions, one with the solid-solid boundary, and another with the solid-liquid boundary. We do the same for the S-wave test solutions. We run the simulations on the domain with $L = 1$, $h = 1$ and $H = 1$. We choose the physical parameters $\rho_1 = 4$, $\rho_2 = 3$, $\mu_1 = 2$, $\lambda_1 = 3$, $\lambda_2 = 1$, and the wave parameters $\omega = 1$ and $I = 1$. We run the two simulations with $\mu_2 = 1$ and $\mu_2 = 0$ for the P and S-waves, and run convergence tests with equally spaced time and spatial steps

P	Δx	Δz	Δt	E_{Max}	E_{L2}	C_{max}	C_{L2}	A_r
1	1/12	1/12	0.005	0.00083	0.00023	-	-	0.015
2	1/24	1/24	0.0025	0.00023	6.63e-5	0.279	0.289	0.008
3	1/48	1/48	0.00125	5.90e-5	1.73e-5	0.255	0.261	0.004

Table 7: Results for P-waves vertically incident on a solid-solid boundary and a free surface.

S	Δx	Δz	Δt	E_{Max}	E_{L2}	C_{max}	C_{L2}	A_r
1	1/12	1/12	0.005	8.50e-5	3.04e-5	-	-	0.006
2	1/24	1/24	0.0025	2.4e-5	7.63e-6	0.283	0.251	0.003
3	1/48	1/48	0.00125	5.5e-6	1.95e-6	0.229	0.256	0.001

Table 8: Results for S-waves vertically incident on a solid-solid boundary and a free surface.

- $\Delta t = 0.005$, $\Delta x = 1/12$, $\Delta z = 1/12$
- $\Delta t = 0.0025$, $\Delta x = 1/24$, $\Delta z = 1/24$
- $\Delta t = 0.00125$, $\Delta x = 1/48$, $\Delta z = 1/48$

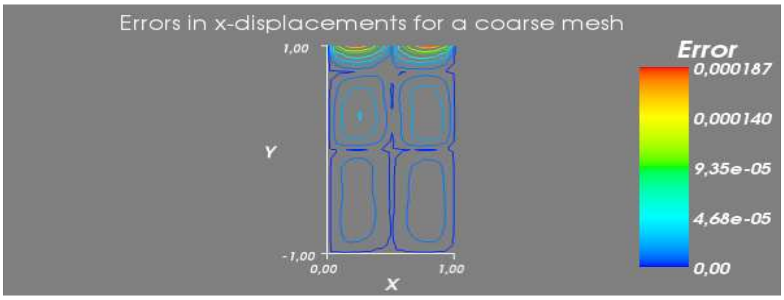
The results of the simulations are given in tables 7, 8, 9 and 10. The x and z-displacement errors are given in figures 15, 14, 17, and 16.

6.4 Conclusion

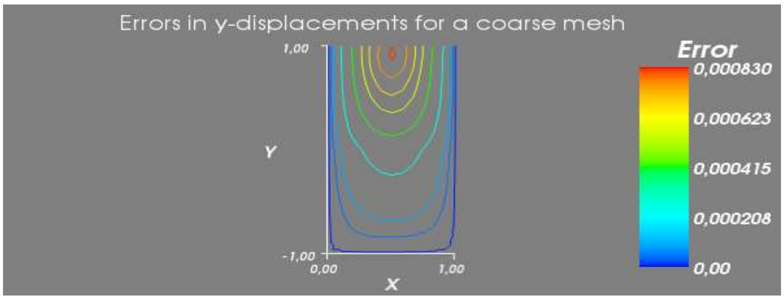
tables 7 and 9 show the results of the simulations for a P-wave on a solid-solid and solid-liquid boundary respectively. The tabes show a clear convergence of the error, yet the error in the solid-liquid case is much worse than for the solid-solid case. The component errors for the P-wave simulations are given in figures 14 and 15. We notice that though the model only has displacements in the z-direction for P-waves, some x-displacements are produced by the numerical scheme. For the solid-solid case, the larger errors for the x and z-componets are found at the free surface, and the smallest errors are found at the boundaries. In the simulations, the x and z errors have a periodic behaviour, showing that the scheme is producing standing waves at the boundaries. In the solid-liquid case, the errors in x and z-components are smaller in the solid layer, and larger in the fluid layer. In the simulations, the errors in the x-components are chaotic, starting at the internal boundary and spreading into the rest of the domain. the z-component error has a semi periodic behaviour spreading from the free surface and into the whole domain. In the fluid domain, large errors are found just inside the boundaries at the two sides of the domain.

P	Δx	Δz	Δt	E_{Max}	E_{L2}	C_{max}	C_{L2}	A_r
1	1/12	1/12	0.005	0.05694	0.01306	-	-	0.114
2	1/24	1/24	0.0025	0.01500	0.00331	0.263	0.253	0.058
3	1/48	1/48	0.00125	0.00387	0.00083	0.258	0.252	0.029

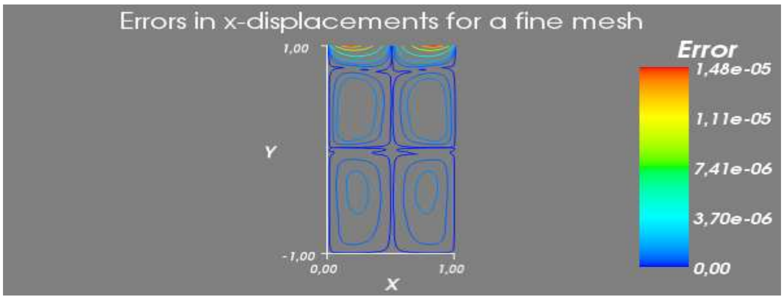
Table 9: Results for P-waves vertically incident on a solid-liqid boundary and a free surface.



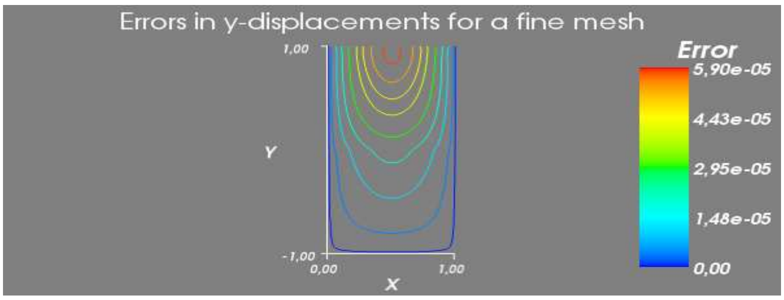
(a)



(b)

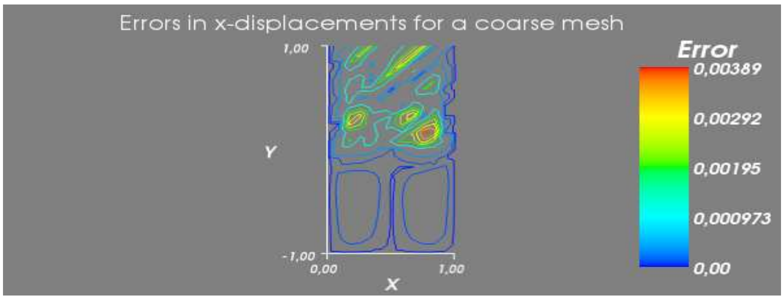


(c)

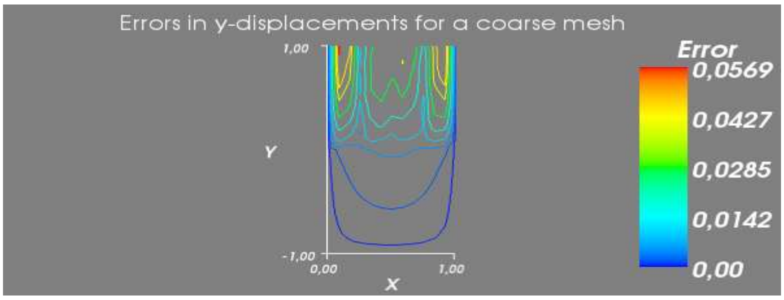


(d)

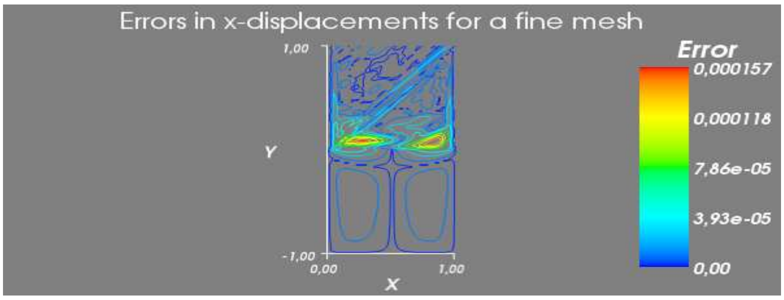
Figure 14: Errors in the x and z components for P-waves hitting a solid-solid boundary. Figure (a) and (b) shows the x and z-component errors for a 12x24 mesh respectively. Figures (c) and (d) shows the x and z-component errors for a 48x96 mesh respectively



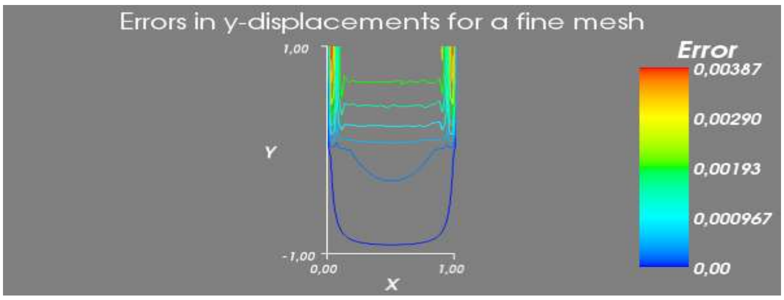
(a)



(b)

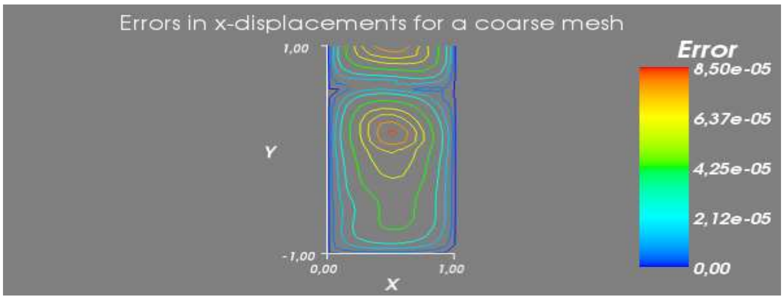


(c)

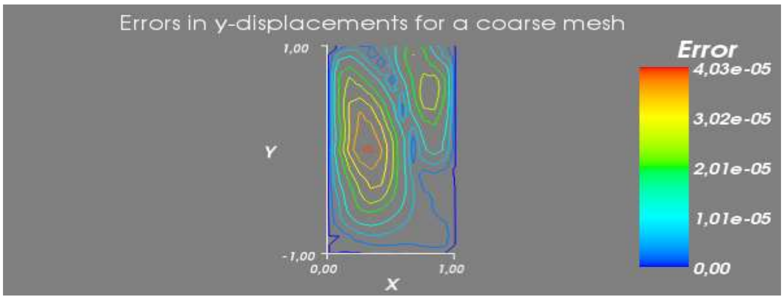


(d)

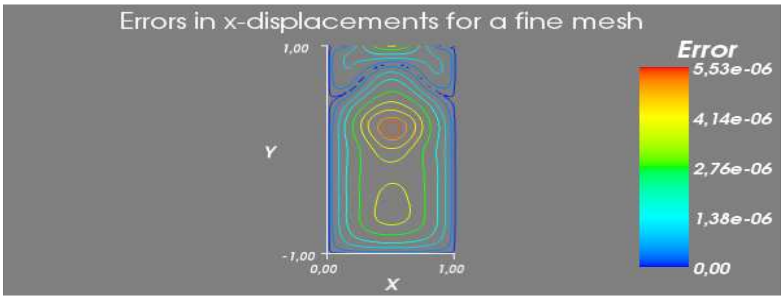
Figure 15: Errors in the x and z components for P-waves hitting a solid-liquid boundary. Figure (a) and (b) shows the x and y-component errors for a 12x24 mesh respectively. Figures (c) and (d) shows the x and z-component errors for a 48x96 mesh respectively



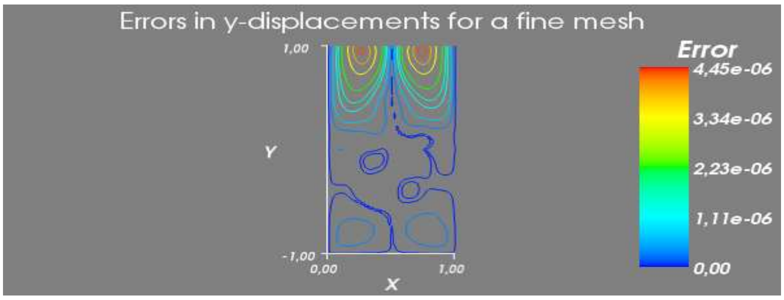
(a)



(b)

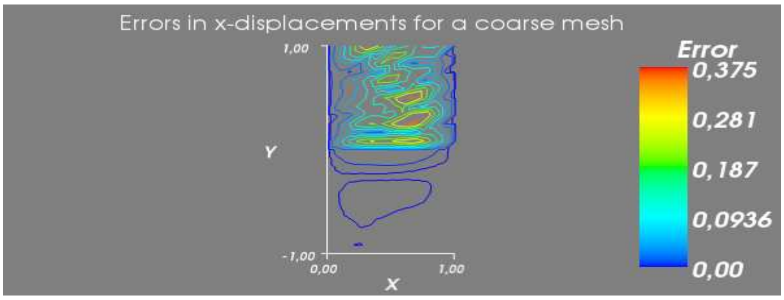


(c)

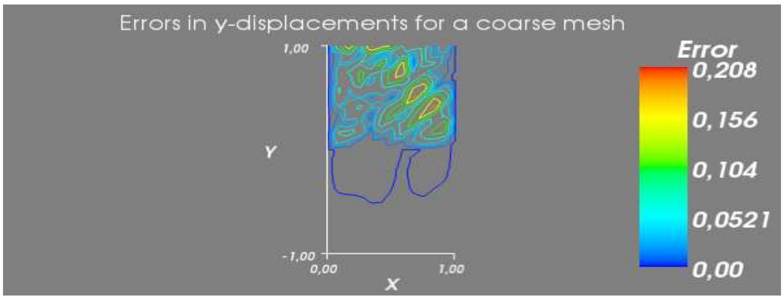


(d)

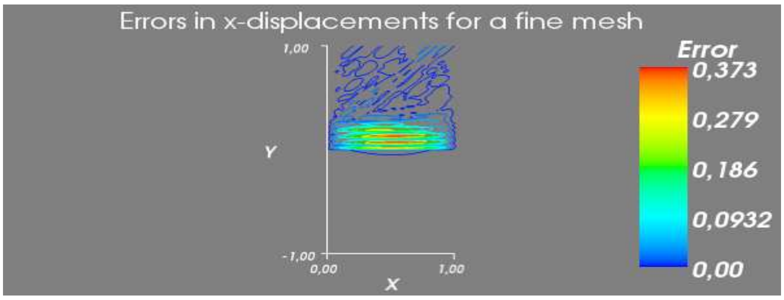
Figure 16: Errors in the x and z components for S-waves hitting a solid-solid boundary. Figure (a) and (b) shows the x and z-component errors for a 12x24 mesh respectively. Figures (c) and (d) shows the x and z-component errors for a 48x96 mesh respectively



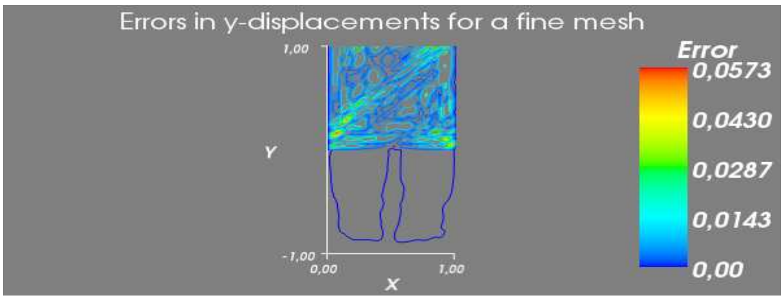
(a)



(b)



(c)



(d)

Figure 17: Errors in the x and z components for S-waves hitting a solid-liquid boundary. Figure (a) and (b) shows the x and z-component errors for a 12x24 mesh respectively. Figures (c) and (d) shows the x and z-component errors for a 48x96 mesh respectively

S	Δx	Δz	Δt	E_{Max}	E_{L2}	C_{max}	C_{L2}	A_r
1	1/12	1/12	0.005	0.37459	0.07808	-	-	0.279
2	1/24	1/24	0.0025	0.36459	0.05934	0.973	0.760	0.244
3	1/48	1/48	0.00125	0.37266	0.04476	1.022	0.754	0.212

Table 10: Results for S-waves vertically incident on a solid-liqid boundary and a free surface.

Tables 8 and 10 show the results of the simulations for an S-wave on a solid-solid and solid-liquid boundary respectively. In the solid-solid case, we see error reduction rates close to 0.25. For the solid-liquid case, the error reduction rates for the maximum error are irregular, and the L2 norm has an error reduction rate close to 0.75. From figures 16 and 17 we see that the numerical scheme produces z-displacements, even though the S-waves only have x-displacements. At the solid-solid boundary, the errors are kept to machine precision at the test solution boundaries, and are larger in the interior domain. The errors in x-displacements are periodic, and largest at the free surface and fluid layer. The errors in z-displacements are periodic in the whole boundary. the For the solid-liquid boundary, we see that the errors in the solid are small, but figure 17 shows that displacements propagate into the fluid layer. Although displacements are expected to propagate into the fluid domain as a result of numerical dispersion, we see no clear convergence or periodicity of the displacement errors in the fluid layer. In the solid layer, we have a periodic behaviour of both the x and z-errors displacements of the error.

In almost all cases, it seems that the interactions with the boundaries are producing additional reflected and transmited waves. These waves have an amplitude that can be approximated by taking the square root of the L2 norm errors in each simulation. This is done in tables 7, 9, 8 and 10. For the case of the S-wave on a solid-liquid boundary, the errors need to be investigated and the programming reviewed.

7 A two layer model with an oblique angle

In the previous project, P and S waves were sent with a vertical incidence towards a the boundary between two layers, and the interactions were examined. In that project, we found a numerical problem in the solid-liquid boundary for S-waves. Due to that problem, it is unwise to continue with a numerical analysis of waves sent with an oblique angle. However, in this project we set up the mathematical model for the solid-liquid boundary problem for future references. Assume the rectangular domain Ω divided into the two subdomains Ω_1 and Ω_2 for the solid and fluid layer respectivly, as given in figure 18. Ω_1 is divided into $l \times m_s$ elements, and Ω_2 is divided into $l \times m_f$ elements. The stress tensor from equation (3) for each layer is given as:

$$\sigma_1 = \lambda(\nabla \cdot \mathbf{u}_1)\mathbf{I} + \mu(\nabla \mathbf{u}_1 + \nabla \mathbf{u}_1^T) \tag{86}$$

$$\sigma_2 = \kappa(\nabla \cdot \mathbf{u}_1) \tag{87}$$

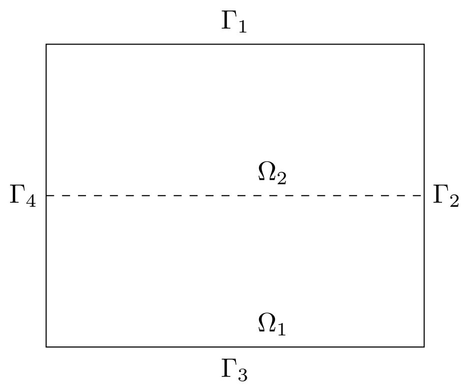


Figure 18: The two layer domain for waves sent with an oblique angle

and inserted into the momentum equation. The variational form from equation (17) is then solved in each sub domain.

7.1 An Analytic solution with an incoming P-wave

The different waves and their directions are found from simple geometric considerations. The closed system consists of 5 waves interacting with each other given in equation (88), and the problem is given in figure 19. In the figure we have made the assumption that the P-wave velocity in the solid is larger than the S-wave velocity in the solid, and that the S-wave velocity in the solid is larger than the P-wave velocity in the fluid. Stein and Wysession [2009, see pp. 203] gives a table showing that this is correct for the ocean-crust model. An incoming P-wave always produces a reflected P-wave, and a reflected S-wave. The fluid layer does not support S-wave motion, so only a P-wave is transmitted through the fluid. The free surface then produces a reflected P-wave.

$$
\begin{aligned}
\mathbf{u}_I &= I(\sin(\theta_I)\mathbf{i} + \cos(\theta_I)\mathbf{k})e^{i(k_1 x \sin(\theta_I) + k_1 z \cos(\theta_I) - \omega t)} \\
\mathbf{u}_R &= R(\sin(\theta_R)\mathbf{i} - \cos(\theta_R)\mathbf{k})e^{i(k_1 x \sin(\theta_R) - k_1 z \cos(\theta_R) - \omega t)} \\
\mathbf{u}_S &= S(\cos(\theta_S)\mathbf{i} + \sin(\theta_S)\mathbf{k})e^{i(k_s x \sin(\theta_s) - k_s z \cos(\theta_s) - \omega t)} \\
\mathbf{u}_T &= T(\sin(\theta_T)\mathbf{i} + \cos(\theta_T)\mathbf{k})e^{i(k_2 x \sin(\theta_T) + k_2 z \cos(\theta_T) - \omega t)} \\
\mathbf{u}_F &= F(\sin(\theta_F)\mathbf{i} - \cos(\theta_F)\mathbf{k})e^{i(k_2 x \sin(\theta_F) - k_2 z \cos(\theta_F) - \omega t)}
\end{aligned}
\tag{88}
$$

We set the boundary between media at $z = 0$ and the free surface at $z = H$. We make the physical observation, also mathematically explained by Stein and Wysession [2009, pp. 71-72] that the angles:

$$
\begin{aligned}
\theta_R &= \theta_I \\
\theta_T &= \theta_F
\end{aligned}
\tag{89}
$$

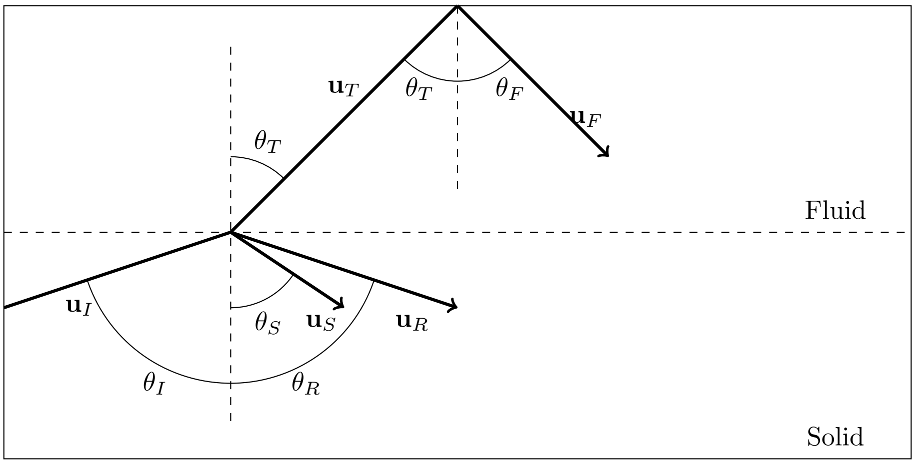


Figure 19: The problem for a P-wave hitting the boundary between solid and fluid

We set $\mathbf{u}^{(1)} = \mathbf{u}_I + \mathbf{u}_R + \mathbf{u}_S$ and $\mathbf{u}^{(2)} = \mathbf{u}_T + \mathbf{u}_R$. The free surface boundary condition (19) states that traction on the surface should be zero as in the previous project, and because the fluid does not support shear motion, only the normal traction needs to be considered. This gives the relation

$$u_x^{(2)}(x, H, t) = -w_z^{(2)}(x, H, t) \tag{90}$$

Inserting equation (88) into (90) and doing some mathematics gives the relation

$$T = -Fe^{-2ik_2 \cos\theta_T} \tag{91}$$

At the internal solid-liquid boundary we have three boundary conditions. The normal displacement and normal traction must be continuous, and that the tangential tractions in the solid vanish. This is after some simplifications stated as:

$$w^{(1)}(x, 0, t) = w^{(2)}(x, 0, t) \tag{92}$$

$$u_z^{(1)}(x, 0, t) = -w_x^{(1)}(x, 0, t) \tag{93}$$

$$\begin{aligned}\kappa(u_x^{(2)}(x, 0, t) + w_z^{(2)}(x, 0, t)) &= \lambda(u_x^{(1)}(x, 0, t) + w_z^{(1)}(x, 0, t)) \\ &+ 2\mu w_z^{(1)}(x, 0, t)\end{aligned} \tag{94}$$

From equation (92) we have:

$$\begin{aligned}T\cos\theta_T e^{i(k_2 \sin\theta_T x)} &= F\cos\theta_T e^{i(k_2 \sin\theta_T x)} + I\cos\theta_I e^{i(k_1 \sin\theta_I x)} \\ &- R\cos\theta_I e^{i(k_1 \sin\theta_I x)} + S\sin\theta_S e^{i(k_s \sin\theta_s x)}\end{aligned} \tag{95}$$

From this equation we make an important physical observation. Because both sides of the equation have to be constant and equal for all x, we must demand that

$$k_1 \sin\theta_I = k_s \sin\theta_s = k_2 \sin\theta_T \tag{96}$$

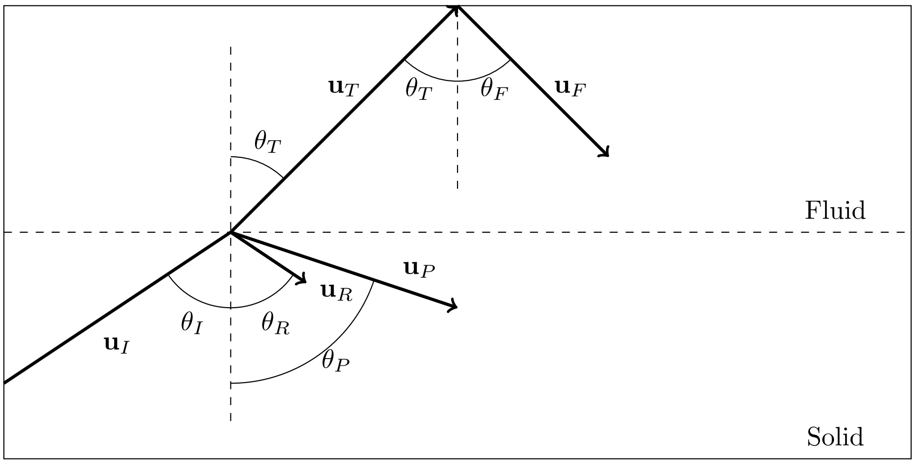


Figure 20: The problem for an S-wave hitting the boundary between solid and fluid

which is a form of snellś law. Inserted into the rest of the boundary conditions, the system of equations determining the amplitudes are found, and given in equation (97):

$$\begin{aligned} F &= -Te^{2ik_2 H\cos(\theta_T)} \\ \cos(\theta_I)(I-R)+\sin(\theta_s)S &= \cos(\theta_T)(T-F) \\ k_1\sin(2\theta_I)(I-R) &= k_s\cos(2\theta_s)S \\ S\mu k_s\sin(2\theta_s) &= k_1(\lambda+2\mu\cos^2(\theta_I))(I+R)-\kappa k_2(T+F) \end{aligned} \tag{97}$$

Notice that when $\theta_I = 0$, the system of equations reduces to the results in equations (64), (66) and (68) from the previous project. The system (97) is complicated, and the hand calculations are not done in this thesis. However, numerical methods can be used to solve for the amplitudes by using the complex linear system solver in the scipy module for python, explained by the documentation by Jones et al.. To verify the results for the closed system, conservation of energy can be examined by

$$|E_1| = |E_1| \tag{98}$$

where 1 denotes layer 1 and 2 deotes layer 2. In the numerical solver, this equality must be correct to machine precision.

7.2 An analytic solution from an incoming S-wave

The problem with an incoming S-wave is almost equal to the case with the incoming P-wave. The S-wave produces a reflected S-wave, a reflected P-wave and a transmitted P-wave. The transmited P-wave then produces a reflected P-wave at the free surface. The 5 interacting waves are given as. Again we have assumed that $c_p > c_s > c_f$ where c_p is the P-wave velocity in the solid, c_s is the S-wave velocity in the solid, and c_f is the P-wave velocity in the fluid.

$$\begin{aligned}
\mathbf{u}_{Is} &= I_s(-\cos(\theta_{Is})\mathbf{i} + \sin(\theta_{Is})\mathbf{k})e^{i(k_1 x \sin(\theta_{Is}) + k_1 z \cos(\theta_{Is}) - \omega t)} \\
\mathbf{u}_{Rs} &= R_s(\cos(\theta_{Rs})\mathbf{i} + \sin(\theta_{Rs})\mathbf{k})e^{i(k_1 x \sin(\theta_{Rs}) - k_1 z \cos(\theta_{Rs}) - \omega t)} \\
\mathbf{u}_{Ps} &= P_s(\sin(\theta_{Ps})\mathbf{i} - \cos(\theta_{Ps})\mathbf{k})e^{i(k_p x \sin(\theta_{Ps}) - k_p z \cos(\theta_{Ps}) - \omega t)} \\
\mathbf{u}_{Ts} &= T_s(\sin(\theta_{Ts})\mathbf{i} + \cos(\theta_{Ts})\mathbf{k})e^{i(k_2 x \sin(\theta_{Ts}) + k_2 z \cos(\theta_{Ts}) - \omega t)} \\
\mathbf{u}_{Fs} &= F_s(\sin(\theta_{Fs})\mathbf{i} - \cos(\theta_{Fs})\mathbf{k})e^{i(k_2 x \sin(\theta_{Fs}) - k_2 z \cos(\theta_{Fs}) - \omega t)}
\end{aligned} \tag{99}$$

Again, by physical observations it is known that $\theta_I = \theta_R$ and $\theta_T = \theta_F$. The free surface boundary condition is in this problem also equal to the case with an incoming P-wave, and given from equation (91). Continuity of vertical displacement at the internal boundary again forces the angles to follow the type of snellś law:

$$k_1 \sin(\theta_{Is}) = k_{Ps} \sin(\theta_{Ps}) = k_2 \sin(\theta_{Ts}) \tag{100}$$

The set of equations determining the amplitude ratios are found from the boundary conditions (92), (93) and (94) and gives the system of equations determining the amplitude ratios provided I is known in equation (101).

$$\begin{aligned}
F_s &= -T_s e^{2ik_2 H \cos(\theta_T)} \\
\sin(\theta_{Is})(I_s + R_s) &= (T_s - F_s)\cos(\theta_{Ts}) + P_s \cos(\theta_{Ps}) \\
k_1 \cos(2\theta_{Is})(I_s + R_s) &= -P_s k_p \sin(2\theta_{Ps}) \\
k_2 \kappa (T_s + F_s) &= P_s k_P(\lambda + 2\mu\cos^2(\theta_{Ps})) + (I_s - R_s)k_1 \mu \sin(2\theta_{Is})
\end{aligned} \tag{101}$$

Notice that for $\theta_I = 0$, the system is unsolvable because no waves are transmitted into the fluid, and instead we use the results from the previous project with $T_s = 0$, $F_s = 0$, $P_s = 0$ and $R_s = I_s$. We also notice that in this case we have a critical angle at

$$\theta_1 = \frac{k_p}{k_1}$$

where no reflected P-wave is produced at the internal boundary. The equations in (101) are then solved with $P = 0$. The system is solved in the same manner as for the P-wave solution. Again, the closed system is verified by conservation of energy, giving

$$|E_1| = |E_2| \tag{102}$$

for layer 1 and 2 respectively, and these need to be correct to machine precision when solved numerically.

8 Discussion

At the beginning of this thesis, the goal was to build a model to to solve an earthquake problem and the following P-SV wave propagations in the sea floor,

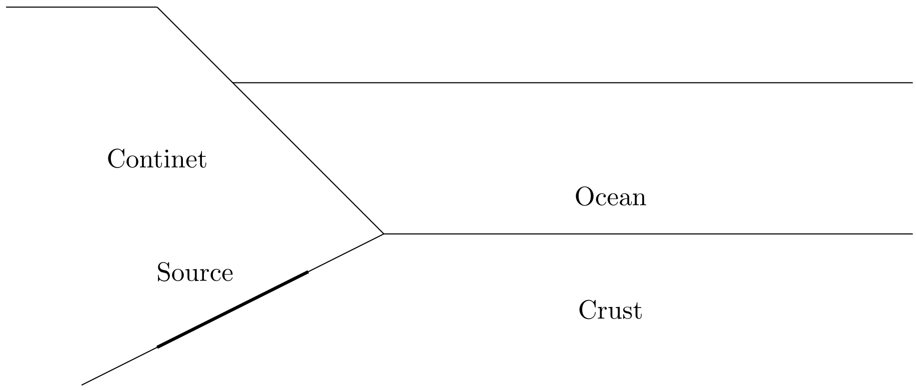


Figure 21: The earthquake model for future study. The model includes the ocean, crust and continent, where the earthquake has itś source between the crust and continent.

continent and sea. The projects in this thesis where originally intended to be exercises to test the different parts of the software before a final implementation was attempted. However problems occured in the two-layer model. We have seen that FEniCS handles single domains in a sufficient way by the test solution process. The free surface imposes more errors, but convergence is still maintained. More difficulties are seen with multiple layers. The sponge layer model has a nice convergence at more coarse resolutions, but this is lost as the resolutions improve as the errors from the reflected waves become more dominant. In The two layer model with vertical incidence, we have nice convergence rates for the P-waves on the solid-solid and solid-liquid problems, but larger errors are found in the solid-liquid boundary. The S-waves have a nice convergence in the solid-solid problem, but we lose convergence for incoming S-waves in a solid-liquid boundary, as large chaotic displacement errors are found in the the fluid domain. In all cases, except the latter, we see a periodic behaviour of the errors, and this shows that the single and multiple layer test-solution process produces small reflected waves at the boundaries. In future researh, a numerical dispersion analysis of the model should be performed to better understand the behaviour of the different simulations. The sponge layer we have used is easily implemented for simple geometries and boundaries found in this thesis, but finding a function b for more complex domains can be very difficult. We discussed another way of implementing the sponge that should be attempted in the future. The two layer model with an incoming S-wave also needs attention, as this does not work with the current implementation. A finite element analysis should be made in FEniCS to better understand the behaviour of the discontinuities in the solid-liquid boundary, so the problems can be handled. After such an analysis is made, the problem in section 7 should be implemented and tested. Further research can be made by inverstigating the P-SV wave system in more complex domains. A reasonable goal is then the earthquake model in figure 21, examining the propagation of seismic waves in a realistic problem, and investigating the full tsunami story that follows. In the end, we would like to remark that although the methods in this work are directed toward seismology, the general theory of the multilayer approach can also be implemented in other aspects of science.

9 Appendix

Below are listings of the codes used in the thesis. The codes are written in python version 2.7.6, and the FEniCS version 1.3. In total, 4 codes have been used. The wave project, the two seismic test solution projects and the project with two layers.

9.1 Code for the sponge layer project

```
from dolfin import *
import math as mt

def solver(L,h,xel,yel,xs,dt,T,omega,vel,k,damp,viz,save):
    """
    Function for solving the scalar wave equation in a
    rectangular domain with given boundary and initial
    conditions
    ---------------------------------------------------
    INPUT:
    L:     Length of domain
    h:     Height of domain
    xel:   Number of elements per unit length in the x-direction
    yel:   Number of elements per unit length in the y-direction
    xs:    Coordinate of the vertical line seperating fluid and sponge
    dt:    Time step
    T:     Total simulation time
    omega: Angular frequency
    vel:   Velocity of waves
    k:     Constant determinging the
    damp:  lin for linear, and quad for quadratic damping
    viz:   True for showing simulation plot
    save:  True for saving errors at time T
    ----------------------------------------------------
    OUTPUT:
    Returns the error between analytic and exact
    solution in the fluid layer
    Saves plots of the component errors if save=True
    """
    # Starting time
    t = 0

    # elements per length
    l = L*xel
    m = h*yel

    # Define functionspace
    mesh = RectangleMesh(0,0,L,h,l,m)
    V = FunctionSpace(mesh, "CG", 1)
    u = TrialFunction(V)
    v = TestFunction(V)

    # Define subdomains
    class Fluid(SubDomain):
        def inside(self,x,on_boundary):
            return (between(x[0], (0,xs)))

    class Sponge(SubDomain):
        def inside(self,x,on_boundary):
            return (between(x[0], (xs,L)))

    fluid = Fluid()
    sponge = Sponge()
    domains = CellFunction("size_t", mesh)
    domains.set_all(0)
    fluid.mark(domains, 0)
    sponge.mark(domains, 1)

    # Create submesh from fluid domain
    submesh = SubMesh(mesh, fluid)
    Vf = FunctionSpace(submesh, "CG", 1)
```

```
    # Variable expressions
    ce = Constant(vel)

    # Set the damping to lin or quad
    if damp=="lin":
        be = Expression("x[0] < xs ? 0 : 10*(x[0]-xs)", xs=xs)
    elif damp=="quad":
        be = Expression("x[0] < xs ? 0 : 10*(pow(x[0],2)-2*xs*x[0]+pow(xs,2))",
                        xs=xs)
    else:
        print "Insert lin or quad"
        exit()

    # Define important constants
    step2 = Constant(1/dt**2)
    step3 = Constant(1/(2*dt))

    # Initial conditions
    Ixy = Constant(0)
    Vxy = Constant(0)

    # Essential boundary conditions
    inflow = Expression("sin(omega*t)*cos(pi*x[1]/(2*h)*(1 + k))",
                        omega=omega,h=h,k=k,t=t)
    free = Constant(0)
    def surface(x, ob): return ob and abs(x[1]-h) < DOLFIN_EPS
    def leftfun(x, ob): return ob and abs(x[0]) < DOLFIN_EPS

    left = DirichletBC(V, inflow, leftfun)
    topp = DirichletBC(V, free, surface)
    bcs = [left, topp]

    # Set all functions into domain
    c = interpolate(ce, V)
    b = interpolate(be, V)
    u1 = interpolate(Ixy, V)
    u2 = interpolate(Vxy, V)

    # Variational forms
    F = step2*inner(u,v)*dx - 2*step2*inner(u1,v)*dx + step2*inner(u2,v)*dx +\
        b*step3*inner(u,v)*dx - b*step3*inner(u2,v)*dx +\
        c*inner(nabla_grad(u1), nabla_grad(v))*dx

    A = assemble(lhs(F))
    u = Function(V)
    t = 2*dt

    while t <= T + 2*dt + DOLFIN_EPS:
        # Plot if viz=True
        if viz==True:
            plot(u2, range_max=1.0, range_min=-1.0, title="Numerical solution")
        inflow.t = t
        begin("Computing at time level t = %g" %t)
        LL = assemble(rhs(F))
        [bc.apply(A,LL) for bc in bcs]
        solve(A, u.vector(), LL)
        end()

        u2.assign(u1)
        u1.assign(u)

        t += dt

    # Exact solution
    lk = mt.pi*(1 + k)/(2.*h)
    kk = mt.sqrt(omega**2/vel**2 - lk**2)
    ue = Expression("sin(omega*t - kk*x[0])*cos(lk*x[1])",
                    omega=omega,kk=kk,lk=lk,t=t-2*dt)

    # Interpolate into fluid domain
    u2e = interpolate(ue, Vf)
    u2s = interpolate(u2, Vf)

    diff = TrialFunction(Vf)
    vf = TestFunction(Vf)
    left = inner(diff, vf)*dx
    righ = inner(u2e, vf)*dx - inner(u2s, vf)*dx
    lass = assemble(left)
    rass = assemble(righ)
    d = Function(Vf)
    solve(lass, d.vector(), rass)

    # Save error to file
    if save==True:
        file1 = File("e-d-%s-L-%s-h-%s-xel-%s-yel-%s-xs-%s-dt-%s-T-%s.pvd" \
                     % (damp,L,h,xel,yel,xs,dt,T))

        file1 << d

    # Return the absolute value of the error
```

```
    error = abs(d.vector().array())
    return error


def run_simulation():
    """
    Test program for running an experiment showing the
    plot on screen with given values and returning the
    error. The maximum and L2 norm errors
    are printed at terminal
    """
    L = 3
    h = 1
    xel = 24
    yel = 24
    xs = 1
    dt = 0.01
    T = 10
    omega = 10.
    vel = 1.
    k = 0
    damp="lin"
    viz = True
    save = False
    error = solver(L,h,xel,yel,xs,dt,T,omega,vel,k,damp,viz,save)
    error_max = error.max()
    error_l2n = mt.sqrt(sum(error**2/len(error)))

    print "Maximum error: ", error_max
    print "L2 norm error: ", error_l2n

def test_convergence():
    """
    Program for running a convergence test with given physical
    values. The time and spatial steps are halved to test that
    convergence is reached. Component errors are then saved to VTK
    files.
    """
    L = 3
    h = 1
    xs = 1
    T = 10
    vel = 1.
    omega = 10.
    k = 0
    damp = "quad"
    viz=False
    save=True

    # Lists to store error values
    E_max = []
    E_l2n = []

    # Lists with dt, dx and dy values
    timestep = [0.01, 0.005, 0.0025]
    xelement = [24, 48, 96]
    yelement = [24, 48, 96]

    for i in range(len(timestep)):
        dt = timestep[i]
        xel = xelement[i]
        yel = yelement[i]
        error = solver(L,h,xel,yel,xs,dt,T,omega,vel,k,damp,viz,save)

        error_max = error.max()
        error_l2n = mt.sqrt(sum(error**2/len(error)))

        E_max.append(error_max)
        E_l2n.append(error_l2n)


    # Check convergence
    C_max = []
    C_l2n = []
    for i in range(len(E_max)-1):
        C_max.append(E_max[i+1]/E_max[i])
        C_l2n.append(E_l2n[i+1]/E_l2n[i])

    print 40*'-'
    print 'MAXIMUM ERROR'
    print E_max
    print 40*'-'
    print 'L2 NORM'
    print E_l2n
    print 40*'-'
    print 'CONVERGENCE MAXIMUM ERROR'
    print C_max
    print 40*'-'
    print 'CONVERGENCE L2 NORM'
    print C_l2n
    print 40*'-'
```

```
def main():
    run_simulation()
    #test_convergence()

if __name__=='__main__':
    main()
```

9.2 Code for the seismic test solution with dirichlet conditions

```
from dolfin import *
import math as mt

def solver(L,h,xel,yel,dt,T,lamda,mu,rho,A,omega,nx,ny,wavetype,viz,savefile):
    """
    Function for solving the seismic wave equation on a rectangular
    domain with with inhomogeneous dirichlet bcs on all sides. The solver
    is tested with either a p wave or s-wave solution.
    INPUT:
    L        : Length of domain
    h        : Height of domain
    xel      : Number of elements per unit length in x direction
    yel      : Number of elements per unit length in y direction
    dt       : Time step
    T        : Total simulation time
    lamda    : lamees first parameter
    mu       : shear modulus
    rho      : density of material
    A        : Amplitude of test solution
    omega    : angular frequency of test solution
    nx       : component of normal vector of test solution in x direction
    ny       : component of normal vector of test solution in y direction
    wavetype: Choose the wave type "P" or "S"
    viz      : Vizualize results if true
    savefile: Save plotfiles if true

    OUTPUT:
    Returns the absolute value of the error in all node points
    """
    # Set solver and plotter
    solver = LUSolver("mumps")

    # Compute number of elements in x and y direction
    l = L*xel
    m = h*yel

    # Function space and functions
    mesh = RectangleMesh(0,0,L,h,l,m)
    V = VectorFunctionSpace(mesh, "CG", 1)
    u = TrialFunction(V)
    v = TestFunction(V)

    # Constants
    stepr = Constant(dt**2/rho)

    # Test Wave type
    if wavetype == "P": # Pressure wave
        Au = A*nx
        Av = A*ny
        vel = mt.sqrt((lamda + 2*mu)/rho*(nx**2+ny**2)) # Wave velocity
        k = omega/vel # Dispersion relation

    elif wavetype == "S": # Shear wave
        Au = A*ny
        Av = -A*nx
        vel = mt.sqrt(mu/rho*(nx**2 + ny**2)) # Wave velocity
        k = omega/vel # Dispersion relation


    t = 0
    # Initial conditions
    Ixy = Expression(("Au*cos(k*nx*x[0] + k*ny*x[1] - omega*t)",
                      "Av*cos(k*nx*x[0] + k*ny*x[1] - omega*t)"),
                     Au=Au,Av=Av,nx=nx,ny=ny,k=k,omega=omega,t=t)


    Vxy = Expression(("Au*cos(k*nx*x[0] + k*ny*x[1] - omega*t)",
                      "Av*cos(k*nx*x[0] + k*ny*x[1] - omega*t)"),
                     Au=Au,Av=Av,nx=nx,ny=ny,k=k,omega=omega,t=t+dt)
```

```
    u2 = interpolate(Ixy, V)
    u1 = interpolate(Vxy, V)

    # Boundary condition
    def boundary(x, on_boundary): return on_boundary
    bc = DirichletBC(V, Ixy, boundary)

    # Stress tensor
    def sigma(u, lamda, mu):
        return lamda*div(u)*Identity(2) + mu*(grad(u) + grad(u).T)

    # Variational form
    F = inner(u, v)*dx - 2*inner(u1, v)*dx + inner(u2, v)*dx +\
        stepr*inner(sigma(u1, lamda, mu), grad(v))*dx

    A = assemble(lhs(F))  # Assemble left hand side
    u = Function(V)
    t = 2*dt
    dxy = Function(V)
    dxx = dxy.sub(0)
    dyy = dxy.sub(1)
    ue = Function(V)
    while t <= T + DOLFIN_EPS:
        # Update time dependent bc functions
        Ixy.t = t
        ue.assign(interpolate(Ixy, V))

        # Solve
        begin("Solving at time step t=%g" % t)
        b = assemble(rhs(F))
        bc.apply(A, b)
        solver.solve(A, u.vector(), b)
        end()

        # Plot solution
        if viz==True:
            plot(u, range_max = 1.0, range_min = -1.0,
                    title="Numerical solution")

        elif viz == 'xerror':
            dxy.vector()[:] = ue.vector().array() - u.vector().array()
            plot(dxx, range_max=1e-6, range_min=-1e-6, mode='color')

        elif viz == 'yerror':
            dxy.vector()[:] = ue.vector().array() - u.vector().array()
            plot(dyy, range_max=1e-6, range_min=-1e-6, mode='color')

        u2.assign(u1)
        u1.assign(u)

        t += dt

    # Exact solution
    Ixy.t = t-dt
    uexact = interpolate(Ixy, V)

    # Compute component differences
    dxy[:] = uexact.vector().array() - u.vector().array()

    if savefile == True:
        # Save component errors in simulation
        file1 = File("dbc_x_%s_wave_xel_%s_yel_%s_dt_%s_nx_%s_ny_%s.pvd" \
                     % (wavetype, xel, yel, dt, nx, ny))
        file1 << dxx

        file2 = File("dbc_y_%s_wave_xel_%s_yel_%s_dt_%s_nx_%s_ny_%s.pvd" \
                     % (wavetype, xel, yel, dt, nx, ny))
        file2 << dyy

        file3 = File("dbc_u_%s_wave_xel_%s_yel_%s_dt_%s_nx_%s_ny_%s.pvd" \
                     % (wavetype, xel, yel, dt, nx, ny))
        file3 << u


    # return the error
    error = abs(uexact.vector().array() - u.vector().array())
    return error


def test_convergence():
    L = 1
    h = 1
    T = 5
    lamda = 1.
    mu = 1.
    rho = 1.
    A = 1.
    omega = 0.5
    nx = 0
    ny = 1
```

```
    wavetype = "S"

    dtlist = [0.0075, 0.00375, 0.001875]
    xelist = [24, 48, 96]
    yelist = [24, 48, 96]

    # Compute errors
    errorlist = []
    normlist = []
    for k in range(len(dtlist)):
        dt = dtlist[k]
        xel = xelist[k]
        yel = yelist[k]
        error = solver(L,h,xel,yel,dt,T,
                       lamda,mu,rho,A,omega,nx,ny,wavetype,viz=False,
                       savefile = True)

        # Compute l2 norm
        norm = mt.sqrt(sum((error)**2/(len(error))))
        normlist.append(norm)
        errorlist.append(error.max())


    # Check convergence
    cmax = []
    cl2n = []
    for i in range(len(errorlist)-1):
        cmax.append(errorlist[i+1]/errorlist[i])
        cl2n.append(normlist[i+1]/normlist[i])

    print 40*'--'
    print 'MAXIMUM ERROR'
    print errorlist
    print 40*'--'
    print 'L2 NORM'
    print normlist
    print 40*'--'
    print 'CONVERGENCE MAXIMUM ERROR'
    print cmax
    print 40*'--'
    print 'CONVERGENCE L2 NORM'
    print cl2n
    print 40*'--'

def run_simulation():
    L = 1
    h = 1
    xel = 24
    yel = 24
    dt = 0.001
    T = 5.0
    lamda = 1.
    mu = 1.
    rho = 1.
    A = 1.
    omega = 0.5
    nx = 2
    ny = 1
    wavetype = "S"
    viz = 'xerror'
    savefile = False
    error = solver(L,h,xel,yel,dt,T,
                   lamda,mu,rho,A,omega,nx,ny,wavetype,viz,
                   savefile)

    norm = mt.sqrt(sum(error**2/len(error)))
    print 20*'--'
    print 'MAXIMUM ERROR'
    print error.max()
    print 20*'--'
    print 'L2 NORM'
    print norm
    print 20*'--'



def main():
    run_simulation()
    #test_convergence()

if __name__ == '__main__':
    main()
```

9.3 Code for the seismic test solutions with given surface stress

```python
from dolfin import *
import mayavi as ma
import math as mt

def solver(L,h,xel,yel,dt,T,lamda,mu,rho,A,omega,nx,ny,wavetype,viz,savefile):
    """
    Function for solving the seismic wave equation on a rectangular
    domain with with inhomogeneous dirichlet bcs on 3 sides, and with
    a given stress on the top. The solver
    is tested with either a p wave or s-wave solution.
    INPUT:
    L        : Length of domain
    h        : Height of domain
    xel      : Number of elements per unit length in x direction
    yel      : Number of elements per unit length in y direction
    dt       : Time step
    T        : Total simulation time
    lamda    : lamees first parameter
    mu       : shear modulus
    rho      : density of material
    A        : Amplitude of test solution
    omega    : angular frequency of test solution
    nx       : component of normal vector of test solution in x direction
    ny       : component of normal vector of test solution in y direction
    wavetype: Choose the wave type "P" or "S"
    viz      : Vizualize results if true
    savefile: Save plotfiles if true

    OUTPUT:
    Returns the absolute value of the error in all node points
    """
    # Set solver parameters
    solver = LUSolver("mumps")

    # Compute number of elements in x and y direction
    l = L*xel
    m = h*yel
    t = 0

    # Function space and functions
    mesh = RectangleMesh(0,0,L,h,l,m)
    V = VectorFunctionSpace(mesh, "CG", 1)
    Vf = FunctionSpace(mesh, "CG", 1)
    u = TrialFunction(V)
    v = TestFunction(V)

    # Constants
    stepr = Constant(dt**2/rho)

    # Wave type
    if wavetype == "P":
        Au = A*nx
        Av = A*ny
        vel = (lamda + 2*mu)/rho*(nx**2+ny**2)
        k = omega/mt.sqrt(vel)
        g = Expression(("-2*mu*A*k*nx*ny*sin(k*nx*x[0]+k*ny*x[1]-omega*t)",
                        """-lamda*A*k*nx*nx*sin(k*nx*x[0]+k*ny*x[1]-omega*t)
                        -lamda*A*k*ny*ny*sin(k*nx*x[0]+k*ny*x[1]-omega*t)
                        -2*mu*A*k*ny*ny*sin(k*nx*x[0]+k*ny*x[1]-omega*t)"""),
                       mu=mu,A=A,k=k,nx=nx,ny=ny,omega=omega,lamda=lamda,
                       t=t)


    elif wavetype == "S":
        Au = A*ny
        Av = -A*nx
        vel = mu/rho*(nx**2+ny**2)
        k = omega/mt.sqrt(vel)
        g = Expression(("""mu*A*k*nx*nx*sin(k*(nx*x[0]+ny*x[1])-omega*t)
                        -mu*A*k*ny*ny*sin(k*(nx*x[0]+ny*x[1])-omega*t)""",
                        "2*mu*A*k*nx*ny*sin(k*(nx*x[0]+ny*x[1])-omega*t)"),
                       mu=mu,A=A,k=k,nx=nx,ny=ny,omega=omega,lamda=lamda,
                       t=t)

    # Initial conditions
    Ixy = Expression(("Au*cos(k*nx*x[0] + k*ny*x[1] - omega*t)",
                      "Av*cos(k*nx*x[0] + k*ny*x[1] - omega*t)"),
                     Au=Au,Av=Av,nx=nx,ny=ny,k=k,omega=omega,t=t)

    Vxy = Expression(("Au*cos(k*nx*x[0] + k*ny*x[1] - omega*t)",
                      "Av*cos(k*nx*x[0] + k*ny*x[1] - omega*t)"),
                     Au=Au,Av=Av,nx=nx,ny=ny,k=k,omega=omega,t=t+dt)

    u2 = interpolate(Ixy, V)
    u1 = interpolate(Vxy, V)

    # Set Dirichlet boundary conditions
    def left(x, on_b): return on_b and abs(x[0]) < DOLFIN_EPS
    def bott(x, on_b): return on_b and abs(x[1]) < DOLFIN_EPS
    def righ(x, on_b): return on_b and abs(x[0] - L) < DOLFIN_EPS
```

```python

    # Set dirichlet values
    lbc = DirichletBC(V, Ixy, left)
    bbc = DirichletBC(V, Ixy, bott)
    rbc = DirichletBC(V, Ixy, righ)

    # List of dirichlet conditions
    bcs = [rbc, bbc, lbc]

    # Stress tensor
    def sigma(v):
        return lamda*div(v)*Identity(2) + \
            mu*(grad(v) + grad(v).T)

    # Variational forms
    F = inner(u, v)*dx - 2*inner(u1, v)*dx + inner(u2, v)*dx +\
        stepr*inner(sigma(u1), grad(v))*dx - stepr*dot(g, v)*ds

    A = assemble(lhs(F))
    u = Function(V)
    t = 2*dt
    ue = Function(V)
    d = Function(V)
    dxx = d.sub(0)
    dyy = d.sub(1)

    # Main loop
    while t <= T + DOLFIN_EPS:
        # Update time dependent bc functions
        Ixy.t = t
        g.t = t-dt
        ue.assign(interpolate(Ixy, V))

        # Solve
        begin("Solving at time t=%g" %t)
        b = assemble(rhs(F))
        [bc.apply(A, b) for bc in bcs]
        solver.solve(A, u.vector(), b)
        end()

        # Plot solution
        if viz==True:
            plot(u, range_max = 1.0, range_min = -1.0,
                 title="Numerical solution")

        if viz == 'xerror':
            d.vector()[:] = ue.vector().array() - u.vector().array()
            plot(dxx, range_min=-1e-6, range_max=1e-6, mode='color')

        if viz == 'yerror':
            d.vector()[:] = ue.vector().array() - u.vector().array()
            plot(dyy, range_min=-1e-6, range_max=1e-6, mode='color')


        u2.assign(u1)
        u1.assign(u)

        t += dt

    # Exact solution
    Ixy.t = t-dt
    uexact = interpolate(Ixy, V)

    # Error at time T
    d.vector()[:] = uexact.vector().array() - u.vector().array()


    if savefile == True:
        file1 = File("str_u_%s_wave_xel_%s_yel_%s_dt_%s_nx_%s_ny_%s.pvd"\
                     % (wavetype, xel, yel, dt, nx, ny))
        file1 << uexact

        file2 = File("str_x_%s_wave_xel_%s_yel_%s_dt_%s_nx_%s_ny_%s.pvd"\
                     % (wavetype, xel, yel, dt, nx, ny))
        file2 << dxx

        file3 = File("str_y_%s_wave_xel_%s_yel_%s_dt_%s_nx_%s_ny_%s.pvd"\
                     % (wavetype, xel, yel, dt, nx, ny))
        file3 << dyy


    error = abs(uexact.vector().array() - u.vector().array())
    return error


def test_convergence():
    L = 1
    h = 1
    T = 5
    lamda = 1.
    mu = 1.
```

```
    rho = 1.
    A = 1.
    omega = 0.5
    nx = 0
    ny = 1
    wavetype = "S"

    dtlist = [0.0075, 0.00375, 0.001875]
    xelist = [24, 48, 96]
    yelist = [24, 48, 96]

    # Compute errors
    errorlist = []
    l2normlist = []
    for k in range(len(dtlist)):
        dt = dtlist[k]
        xel = xelist[k]
        yel = yelist[k]
        error = solver(L,h,xel,yel,dt,T,\
                            lamda,mu,rho,A,omega,nx,ny,wavetype,viz=False,\
                            savefile=True)

        # Compute l2 norm
        l2 = mt.sqrt(sum(error**2/len(error)))
        l2normlist.append(l2)
        errorlist.append(error.max())


    # Check convergence
    cmax = []
    cl2n = []
    for i in range(len(errorlist)-1):
        cmax.append(errorlist[i+1]/errorlist[i])
        cl2n.append(l2normlist[i+1]/l2normlist[i])

    print 40*'--'
    print 'MAXIMUM ERROR'
    print errorlist
    print 40*'--'
    print 'L2 NORM'
    print l2normlist
    print 40*'--'
    print 'CONVERGENCE MAXIMUM ERROR'
    print cmax
    print 40*'--'
    print 'CONVERGENCE L2 NORM'
    print cl2n
    print 40*'--'

def run_simulation():
    L = 1
    h = 1
    xel = 24
    yel = 24
    dt = 0.0075
    T = 5
    lamda = 1.
    mu = 1.
    rho = 1.
    A = 1.
    omega = 0.5
    nx = 1
    ny = 0
    wavetype = "P"
    viz = 'xerror'
    savefile = False
    error = solver(L,h,xel,yel,dt,T,\
                        lamda,mu,rho,A,omega,nx,ny,wavetype,viz,\
                        savefile)
    norm = mt.sqrt(sum(error**2/len(error)))
    print 20*'--'
    print 'MAXIMUM ERROR'
    print error.max()
    print 20*'--'
    print 'L2 NORM'
    print norm
    print 20*'--'



def main():
    run_simulation()
    #test_convergence()

if __name__ == '__main__':
    main()
```

9.4 Code for the seismic waves on multiple layers

```python
from dolfin import *
import scitools.std as sc
import os

def solver(l,h,L,H,xel,yel,dt,endt,ys,rho1,\
           rho2,mu1,mu2,lamda1,lamda2,wtype,part,w,I,viz,saveerror,animate):
    """
    Function for solving the elastic wave equation in a two-layer system
    consisting of rectangular domains by using known boundary conditions at
    the sides and bottom and a free boundary at the surface. The implementation
    is verified by a known analytic solution.

    INPUT:
    ----------------------------------------------------------------------
    l         : Start of domain in x-direction
    h         : Start of domain in y-direction
    L         : End of domain in x-direction
    H         : End of domain in y-direction
    xel       : Number of elements in x-direction per unit length
    yel       : Number of elements in y-direction per unit length
    dt        : Time step
    endt      : End time
    ys        : Horisontal line that separates media
    rho1      : Mass density in layer 1
    rho 2     : Mass density in layer 2
    mu1       : Shear modulus in layer 1
    mu2       : Shear modulus in layer 2
    lamda1    : Lames constant in layer 1
    lamda2    : Lames constant in layer 2
    wtype     : "P" for P wave, "S" for shear wave
    part      : Runs simulation with imaginary or real parts of waves
    w         : Angular velocity of waves
    I         : Amplitude of incoming waves
    viz       : Choose vizualization 'solution', 'error' or 'none'
    saveerror: Save the component errors at time T if true
    animate   : Save the solution in VTK files

    OUTPUT:
    ------------------------------------------------------------------------
    Plots numerical solution
    """
    # Create new directory for save files
    # Create animation file in directory
    if animate == True:
        sol = File("%s-%s-%s-%s-%s-%s-%s-%s-%s-%s-%s-%s.pvd" % \
                   (viz,wtype,yel,dt,endt,rho1,rho2,mu1,mu2,lamda1,lamda2,w))

    if saveerror == True:
        xer = File("xer-%s-%s-%s-%s-%s-%s-%s-%s-%s-%s-%s-%s.pvd" % \
                   (part,wtype,yel,dt,endt,rho1,rho2,mu1,mu2,lamda1,lamda2,w))
        yer = File("yer-%s-%s-%s-%s-%s-%s-%s-%s-%s-%s-%s-%s.pvd" % \
                   (part,wtype,yel,dt,endt,rho1,rho2,mu1,mu2,lamda1,lamda2,w))

    # Define the solver method
    solver = LUSolver("mumps")
    n = xel*(L - l)
    m = yel*(H - h)

    # Dispersion relations and amplitudes
    # depending on the incoming wave
    if wtype == 'P':
        vel1 = sc.sqrt((lamda1 + 2*mu1)/rho1)
        vel2 = sc.sqrt((lamda2 + 2*mu2)/rho2)
        k1 = w/vel1
        k2 = w/vel2

        # Useful expressions
        al = k1/k2*(lamda1 + 2*mu1)/(lamda2 + 2*mu2)
        r = sc.cos(2*k2*H) + 1j*sc.sin(2*k2*H)
        C = al*(sc.cos(2*k2*H) + 1)/sc.sin(2*k2*H)*1j

        # Amplitudes
        R = -I*(1. + C)/(1. - C)
        F = I/(1. + r)*(1. - (1. + C)/(1. - C))
        T = I/(1. + 1./r)*(1. - (1. + C)/(1. - C))

    elif wtype == 'S':
        vel1 = sc.sqrt(mu1/rho1)
        vel2 = sc.sqrt(mu2/rho2)
        k1 = w/vel1
        if mu2 == 0:
            k2 = 0
            R = I
            F = 0
            T = 0
        else:
            k2 = w/vel2
```

```python
            al = k1*mu1/(k2*mu2)
            r = sc.cos(2*k2*H) + 1j*sc.sin(2*k2*H)
            C = al*(sc.cos(2*k2*H) + 1)/sc.sin(2*k2*H)*1j

            # Amplitudes
            R = -I*(1. + C)/(1. - C)
            F = I/(1. + r)*(1. - (1. + C)/(1. - C))
            T = I/(1. + 1./r)*(1. - (1. + C)/(1. - C))

    # Domain and sub domains
    mesh = RectangleMesh(l,h,L,H,n,m)
    solidmesh = AutoSubDomain(lambda x: x[1] < 0 + DOLFIN_EPS)
    fluidmesh = AutoSubDomain(lambda x: x[1] > 0 - DOLFIN_EPS)
    cf = CellFunction("size_t", mesh, 0)
    fluidmesh.mark(cf,1)
    solid = SubMesh(mesh, cf, 0)
    fluid = SubMesh(mesh, cf, 1)

    # Functionspace and functions
    V = VectorFunctionSpace(mesh, "CG", 1)
    D = FunctionSpace(mesh, "DG", 0)
    u = TrialFunction(V)
    v = TestFunction(V)
    u2 = Function(V) # First initial condition u(0)
    u1 = Function(V) # Second initial condition u(dt)
    us = Function(V) # Solution Function u(t)
    ue = Function(V) # Exact solution u_e(t)
    ud = Function(V) # Error function u_e(t) - u(t)
    udx = ud.sub(0)  # x-component of the error
    udy = ud.sub(1)  # y-component of the error

    # Extract dofs from sub meshes
    sdofx, sdofy = submesh_dofs(mesh, solid, V)
    fdofx, fdofy = submesh_dofs(mesh, fluid, V)

    # Convert coordinates to python syntax
    gdim = mesh.geometry().dim()
    X = V.dofmap().tabulate_all_coordinates(mesh).reshape((-1, gdim))
    x = X[:,0]
    y = X[:,1]

    # Vector coordinates in solid layer
    xxs, xys = x[sdofx], y[sdofx]
    yxs, yys = x[sdofy], y[sdofy]

    # Vector coordinates in fluid layer
    xxf, xyf = x[fdofx], y[fdofx]
    yxf, yyf = x[fdofy], y[fdofy]

    # Define subfunctions
    rhof = Expression("x[1] > ys ? rho2 : rho1",
                      ys=ys, rho1=rho1, rho2=rho2)
    muf = Expression("x[1] > ys ? mu2 : mu1",
                     ys=ys, mu1=mu1, mu2=mu2)
    lamdaf = Expression("x[1] > ys ? lamda2 : lamda1",
                        ys=ys, lamda1=lamda1, lamda2=lamda2)

    rho = interpolate(rhof, D)
    mu = interpolate(muf, D)
    lamda = interpolate(lamdaf, D)

    # Stress tensor
    def sigma(u, lamda, mu):
        return lamda*div(u)*Identity(2) + mu*(grad(u) + grad(u).T)

    # First Initial condition
    t = 0
    fxs, fys = u_solid(xxs, xys, yxs, yys, part, wtype, w, t, k1, I, R)
    fxf, fyf = u_fluid(xxf, xyf, yxf, yyf, part, wtype, w, t, k2, T, F)
    u2.vector()[fdofx] = fxf
    u2.vector()[fdofy] = fyf
    u2.vector()[sdofx] = fxs
    u2.vector()[sdofy] = fys

    # Second initial condition
    t = dt
    fxs, fys = u_solid(xxs, xys, yxs, yys, part, wtype, w, t, k1, I, R)
    fxf, fyf = u_fluid(xxf, xyf, yxf, yyf, part, wtype, w, t, k2, T, F)
    u1.vector()[fdofx] = fxf
    u1.vector()[fdofy] = fyf
    u1.vector()[sdofx] = fxs
    u1.vector()[sdofy] = fys


    # Essential boundary conditions
    def bottom(x, on_b): return on_b and abs(x[1]-h) < DOLFIN_EPS
    def left(x, on_b): return on_b and abs(x[0]) < DOLFIN_EPS
    def right(x, on_b): return on_b and abs(x[0]-L) < DOLFIN_EPS
    leftbc = DirichletBC(V, ue, left)
    righbc = DirichletBC(V, ue, right)
    bottbc = DirichletBC(V, ue, bottom)
```

```
    bcs = [leftbc, righbc, bottbc]

    # Variational form
    Form = inner(rho*u,v)*dx - 2*inner(rho*u1,v)*dx + inner(rho*u2,v)*dx + \
           dt**2*inner(sigma(u1,lamda,mu),grad(v))*dx

    t = 2*dt
    leftside = assemble(lhs(Form))
    while t <= endt + DOLFIN_EPS:
        # Update exact solution and boundary conditions
        fxs, fys = u_solid(xxs, xys, yxs, yys, part, wtype, w, t, k1, I, R)
        fxf, fyf = u_fluid(xxf, xyf, yxf, yyf, part, wtype, w, t, k2, T, F)
        ue.vector()[fdofx] = fxf
        ue.vector()[fdofy] = fyf
        ue.vector()[sdofx] = fxs
        ue.vector()[sdofy] = fys

        # Solve for u
        begin("Solving at time step t=%g" % t)
        rightside = assemble(rhs(Form))
        [bc.apply(leftside, rightside) for bc in bcs]
        solver.solve(leftside, us.vector(), rightside)

        ud.vector()[:] = abs(ue.vector().array() - us.vector().array())

        # Plot solution
        if viz == 'solution':
            plot(us, range_min=-1.5, range_max=1.5,
                 title='Numerical solution')
            if animate == True:
                sol << us

        elif viz == 'error':
            plot(ud, range_min = -1.0, range_max = 1.0, mode='color',
                 title='Error at time t=%g' % t)
            if animate == True:
                sol << ud

        elif viz == 'xerror':
            plot(udx, range_min = -0.01, range_max = 0.01, mode='color',
                 title='Error in x-component at time t=%g' % t)
            if animate == True:
                sol << udx

        elif viz == 'yerror':
            plot(udy, range_min = -0.01, range_max = 0.01,mode='color',
                 title='Error in y-component at time t=%g' % t)
            if animate == True:
                sol << udy


        elif viz == 'exact':
            plot(ue, range_min = -2.5, range_max = 2.5,
                 title='Exact solution')
        end()

        # Update for next time step
        u2.assign(u1)
        u1.assign(us)
        t += dt

    # Compte component differences at time T
    ud.vector()[:] = abs(ue.vector().array() - us.vector().array())
    udx, udy = ud.split(deepcopy=True)
    if saveerror == True:
        xer << udx
        yer << udy

    # Find error
    return abs(ue.vector().array() - us.vector().array())


def submesh_dofs(mesh, submesh, V):
    """
    Function for extracting dofs from subdomains, and
    returning two lists of x and y components of
    the dofs in the subdomain
    """
    tdim = mesh.topology().dim()
    dofmap = V.dofmap()
    xdof = V.sub(0).dofmap()
    ydof = V.sub(1).dofmap()

    submesh_dofx = set()
    submesh_dofy = set()

    parent_cell_indices = submesh.data().array('parent_cell_indices',tdim)
    for i in range(submesh.num_cells()):
        cell = parent_cell_indices[i]
        [submesh_dofx.add(dof) for dof in xdof.cell_dofs(cell)]
        [submesh_dofy.add(dof) for dof in ydof.cell_dofs(cell)]
```

```python

    dofx = sc.array(list(submesh_dofx))
    dofy = sc.array(list(submesh_dofy))

    return dofx, dofy


def u_solid(xx, xy, yx, yy, part, wtype, w, t, k, I, R):
    """
    Function for evaluating the analytic
    solution in the solid layer by either a
    P or S wave test solution
    """
    if wtype == 'P':
        usx = (0 + 0j)*sc.cos(xy)
        usy = I*(sc.cos(w*t-k*yy) + 1j*sc.sin(w*t-k*yy)) +\
              R*(sc.cos(w*t+k*yy) + 1j*sc.sin(w*t+k*yy))

    elif  wtype == 'S':
        usx = I*(sc.cos(w*t-k*xy) + 1j*sc.sin(w*t-k*xy)) +\
              R*(sc.cos(w*t+k*xy) + 1j*sc.sin(w*t+k*xy))
        usy = (0 + 0j)*sc.cos(yy)

    if part == 'real':
        usx = sc.ascontiguousarray(sc.real(usx))
        usy = sc.ascontiguousarray(sc.real(usy))

    if part == 'imag':
        usx = sc.ascontiguousarray(sc.imag(usx))
        usy = sc.ascontiguousarray(sc.imag(usy))

    return usx, usy

def u_fluid(xx, xy, yx, yy, part, wtype, w, t, k, T, F):
    """
    Function for evaluating the analytic
    solution in the fluid layer by either a
    P or S wave test solution
    """
    if wtype == 'P':
        ufx = (0 + 0j)*sc.cos(xy)
        ufy = T*(sc.cos(w*t-k*yy) + 1j*sc.sin(w*t-k*yy)) +\
              F*(sc.cos(w*t+k*yy) + 1j*sc.sin(w*t+k*yy))

    elif wtype =='S':
        ufx = T*(sc.cos(w*t-k*xy) + 1j*sc.sin(w*t-k*xy)) +\
              F*(sc.cos(w*t+k*xy) + 1j*sc.sin(w*t+k*xy))
        ufy = (0 + 0j)*sc.cos(yy)


    if part == 'real':
        ufx = sc.ascontiguousarray(sc.real(ufx))
        ufy = sc.ascontiguousarray(sc.real(ufy))

    if part == 'imag':
        ufx = sc.ascontiguousarray(sc.imag(ufx))
        ufy = sc.ascontiguousarray(sc.imag(ufy))

    return ufx, ufy


def run_simulation():
    """
    Function for running a single simulation with given values
    INPUT:
    Nothing, values are changed directly in the function

    OUTPUT:
    Prints the maximum error and the l2 norm error in the terminal
    """
    # Constants
    l =  0
    h = -1
    L = 1
    H = 1
    xel = 24
    yel = 24
    dt = 0.01
    endt = 10
    ys = 0
    rho1 = 4.
    rho2 = 3.
    mu1 = 2.
    mu2 = 0.
    lamda1 = 3.
    lamda2 = 1.
    wtype = 'S'
    part  = 'real'
    w = 1.
    I = 1.
    viz = 'yerror'
```

```
    saveerror = False
    animate = False

    error = solver(l,h,L,H,xel,yel,dt,endt,\
                   ys,rho1,rho2,mu1,mu2,lamda1,lamda2,wtype,part,
                   w,I,viz,saveerror,animate)

    # Find max and norm errors
    errormax = error.max()
    errornor = sc.sqrt(sum(error**2/len(error)))

    # Print errors on screen
    print 30*'--'
    print 'MAXIMUM ERROR'
    print errormax
    print 30*'--'
    print 'L2 NORM ERROR'
    print errornor
    print 30*'--'

def test_convergence():
    """
    Function for running 3 simulations with a finer time and
    spatial spacing and testing that the error converges.

    INPUT:
    Values are changed directly in function

    OUTPUT:
    returns:
    - Prints maximum error in each simulation
    - prints convergence rates for maximum error
    - prints the L2 norm error in each simulation
    - prints the convergence rates for the L2 norm errors
    """
    # Constants
    l =  0
    h = -1
    L = 1
    H = 1
    endt = 10
    ys = 0
    rho1 = 4.
    rho2 = 3.
    mu1 = 2.
    mu2 = 0.
    lamda1 = 3.
    lamda2 = 1.
    wtype = 'P'
    part = 'real'
    w = 1.
    I = 1.
    viz = 'none'
    saveerror = True
    animate = False

    # Convergence values
    dtlist = [0.01, 0.005, 0.0025]
    xelist = [24, 48, 96]
    yelist = [24, 48, 96]

    # Errors
    errorlist = []
    normlist = []
    for k in range(len(dtlist)):
        dt = dtlist[k]
        xel = xelist[k]
        yel = yelist[k]
        error = solver(l,h,L,H,xel,yel,dt,endt,ys,rho1,\
                       rho2,mu1,mu2,lamda1,lamda2,wtype,part,
                       w,I,viz,saveerror,animate)

        norm = sc.sqrt(sum((error)**2/(len(error))))
        normlist.append(norm)
        errorlist.append(error.max())

    # Check convergence
    cmax = []
    cl2n = []
    for i in range(len(errorlist)-1):
        cmax.append(errorlist[i+1]/errorlist[i])
        cl2n.append(normlist[i+1]/normlist[i])

    print 40*'--'
    print 'MAXIMUM ERROR'
    print errorlist
    print 40*'--'
    print 'L2 NORM'
    print normlist
    print 40*'--'
    print 'CONVERGENCE MAXIMUM ERROR'
```

```
        print cmax
        print 40*'--'
        print 'CONVERGENCE L2 NORM'
        print cl2n
        print 40*'--'

def main():
    run_simulation()
    #test_convergence()

if __name__=='__main__':
    main()
```

References

Garth N. Wells Anders Logg, Kent-Andre Mardal. *Automated solutions to differential equations by the finite element method*. Springer, Berlin, Germany, 2012.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. Scipy documentation. URL `http://docs.scipy.org/doc/scipy/reference/`. Last accessed in october 2014.

Spencer Kimball, Peter Mattis, Michael Natterer, Sven Neumann, et al. Gimp: Gnu image manipulation program. URL `URL www.gimp.org`. Last accessed in may 2014.

Pijush K Kundu and Ira M Cohen. Fluid mechanics. 4th, 2008.

Hans Petter Langtangen. *Computational partial differential equations: numerical methods and diffpack programming, 2nd*. Springer Verlag, 1999.

Kitware personell. Paraview:. URL `www.paraview.org`. Last accessed in october 2014.

Prabhu Ramachandran and Gaël Varoquaux. The mayavi data visualizer. URL `URL http://mayavi. sourceforge. net`. Last accessed in june 2014.

Seth Stein and Michael Wysession. *An introduction to seismology, earthquakes, and earth structure*. John Wiley & Sons, 2009.

Aslak Tveito and Ragnar Winther. *Introduction to partial differential equations: a computational approach*, volume 29. Springer, 2005.