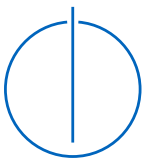
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Adjoint-Guided Mesh Refinement for Earthquake Simulations

Sven Hingst



DEPARTMENT OF INFORMATICS

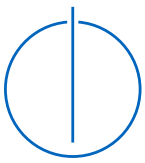
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Adjoint-Guided Mesh Refinement for Earthquake Simulations

Adjoint-gesteuerte Gitterverfeinerung für Erdbebensimulationen

Author:	Sven Hingst
Supervisor:	Prof. Dr. Michael Bader
Advisor:	M.Sc. Lukas Krenz
Submission Date:	15.06.2022



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.06.2022 Sven Hingst

Acknowledgments

I would like to thank my advisor Lukas Krenz, for his advice and regular meetings
I would like to thank my parents for their support.

Abstract

Seismic simulations are used for example for geothermal and mining exploration, and for studying earthquakes. Current seismic simulations usually manually define a refinement area around the source and the regions of interest. Recently an adaptive mesh-refinement technique is presented for tsunami simulations. It uses the adjoint equation to run the simulation in reverse from a point of interest. The result is combined with a forward simulation to construct the refinement. We adapt this method in this thesis for elastic wave equations. We primarily use statically adaptive mesh refinement. The generated refinements follow the paths which the waves take to the receivers. This results in lower error than manually created refinements for a test scenario which produces only P-Waves and a scenario which models the geology under Helsinki as a 1d-velocity model. However, the manual refinement is better in a simpler system which only consists of two different materials.

A prototype for creating dynamically adaptive grid with the same method was also implemented and tested with promising results.

Contents

Acknowledgments **iii**

Abstract **iv**

1 Introduction **1**

2 Theory **3**
2.1 Elastic Wave Equations . . . 3
2.2 Adjoint . . . 5
2.2.1 Derivation of the Adjoint . . . 5
2.2.2 Refinement Procedure . . . 7
2.3 Discontinuous Galerkin . . . 8
2.4 Peano Curve Traversal . . . 8

3 Implementation **10**
3.1 ExaHype . . . 11
3.2 PDEs . . . 11
3.3 Refinement Framework . . . 13
3.4 Three to one balancing . . . 14
3.5 Refinement in ExaHype . . . 15
3.6 Dynamically Adaptive Mesh Refinement . . . 16

4 Results **17**
4.1 Test Setup . . . 17
4.1.1 P-Waves only . . . 17
4.1.2 Adjoint initial condition . . . 19
4.1.3 Two Halfspaces . . . 21
4.1.4 Helsinki Model . . . 25
4.2 Problems . . . 29
4.2.1 Static Stress Region Near the Point Source . . . 29
4.2.2 Reflections on refinement boundary . . . 30
4.3 Dynamically Adaptive Mesh Refinement . . . 31

5 Conclusion **36**

List of Figures **37**

List of Tables **38**

Bibliography **39**

1 Introduction

Seismic simulation has many applications from modeling the impact of possible future earthquakes to exploration and feasibility checks for mining and geothermal operations. Seismic waves reflect and refract on material changes. The paths that a wave takes to areas of interest, are largely unpredictable without simulating the scenario. This means a larger domain is needed to simulate possible returning waves and this makes predicting areas needing refinement more difficult. Current simulations usually define a large coarse grid and refine the region around the epicenter and the receivers [Kre+21][Ulr+19]. [BH22] have developed a method which creates a statically adaptive grid based on velocity aware refinement.

In [DL16], a dynamically adaptive mesh refinement technique is introduced for tsunami simulations. It firstly runs the simulation on a coarse grid. It then simulates the scenario backwards from a region of interest by using an adjoint equation. They have shown that areas, where the inner product between both simulations is large, have a large impact on the solution at the region of interest. This is then used to refine the regions where the inner product exceeds a user defined threshold.

This thesis prototypes that method for elastic wave equations and evaluates its feasibility. Dynamically adaptive mesh refinement is rarely used for earthquake simulations. Therefore, the focus was set on statically adaptive mesh refinement, as it also is theoretically simpler to analyze and implement than dynamically adaptive mesh refinement. To better understand the statically adaptive refinement, a simple dynamically adaptive refinement method was later implemented and tested. The PDEs are solved with ExaHype [Rei+20], a Discontinuous Galerkin solver for C++. The combination of the results and the refinement is done with Python. Python is used to combine the forward and backward solutions and uses the results to create the refinement, which is then read by ExaHype.

Chapter 2 gives an introduction into elastic wave equations [LeV+02], the PDE discretisation Discontinuous Galerkin [HW07] and derives the adjoint method [DL16] for elastic wave equations. Chapter 3 gives an overview of ExaHype [Rei+20] and the structure and implementation of the adjoint-guided mesh refinement method. Chapter 4 presents three different scenarios, shows their refinements an details their results. The first scenario produces only P-Waves on a uniform material with a free surface boundary, the second scenario consists of two halfspaces with different materials and

the third one uses a 1D-velocity model for the geology below Helsinki. The quality of the refinements is tested by comparing them against manually constructed refinements. This chapter additionally compares a prototype of a dynamically adaptive variant to the static refinements of the Helsinki based scenario.

2 Theory

This chapter gives a short overview over elastic wave equations and the PDE solving method Discontinuous Galerkin. Additionally the adjoint-guided mesh refinement technique is explained and derived for the elastic wave equations.

2.1 Elastic Wave Equations

This section is based on [LeV+02]. Earthquakes can be modeled by a set of hyperbolic partial differential equations called elastic wave equations. The relevant quantities are stress σ, strain ϵ and the velocities $(u, v, w)^T$. Stress and strain are both $3x3$ symmetric tensors with six distinct elements. For small deformations, stress and strain are linearly dependent according to Hooke's law. Furthermore, assuming isotropic material properties allows substituting the strain with stress in these equations. We align the stress components to the planes normal to the coordinate axes. The stress is then composed of stresses $(\sigma^{11}, \sigma^{22}, \sigma^{33})$ normal to those planes and the shear stresses $\sigma^{12}, \sigma^{23}, \sigma^{13}$.

This results in the following system of equations:

$$\begin{aligned}
\sigma_t^{11} - (\lambda + 2\mu)u_x - \lambda v_y - \lambda w_z &= 0 \\
\sigma_t^{22} - \lambda u_x - (\lambda + 2\mu)v_y - \lambda w_z &= 0 \\
\sigma_t^{33} - \lambda u_x - \lambda v_y - (\lambda + 2\mu)w_z &= 0 \\
\sigma_t^{12} - \mu(v_x + u_y) &= 0 \\
\sigma_t^{23} - \mu(v_z + w_y) &= 0 \\
\sigma_t^{13} - \mu(u_z + w_x) &= 0 \\
\rho u_t - \sigma_x^{11} - \sigma_y^{12} - \sigma_z^{13} &= 0 \\
\rho v_t - \sigma_x^{12} - \sigma_y^{22} - \sigma_z^{23} &= 0 \\
\rho w_t - \sigma_x^{13} - \sigma_y^{23} - \sigma_z^{33} &= 0
\end{aligned} \tag{2.1}$$

Here ρ denotes the density, μ is the shear modulus and λ is a combination of several material parameters without having a direct physical interpretation itself. To allow for a faster numerical simulation, a simplified system with only two spatial dimensions

can be derived from that system. For this we set $q_z = 0$ by assuming that there are no variations in the third dimension. The resulting system looks like this:

$$\begin{aligned} \sigma_t^{11} - (\lambda + 2\mu)u_x - \lambda v_y &= 0 \\ \sigma_t^{22} - \lambda u_x - (\lambda + 2\mu)v_y &= 0 \\ \sigma_t^{12} - \mu(v_x + u_y) &= 0 \\ \rho u_t - \sigma_x^{11} - \sigma_y^{12} &= 0 \\ \rho v_t - \sigma_x^{12} - \sigma_y^{22} &= 0 \end{aligned} \tag{2.2}$$

These equations can be written in a compact form as a sum of matrices times the derivative vectors

$$q_t + Aq_x + Bq_y + Cq_z = 0, \tag{2.3}$$

and for the two-dimensional version

$$q_t + Aq_x + Bq_y = 0. \tag{2.4}$$

Due to the computational intensity of the three dimensional equation we will only use the two dimensional variant for this thesis. The eigenvalues s of the matrices are the shear wave speed c_s and the pressure wave speed c_p

$$s_1 = c_p, \qquad s_2 = -c_p, \qquad s_3 = c_s, \qquad s_4 = -c_s, \qquad s_5 = 0 \tag{2.5}$$

$$c_p = \sqrt{\frac{\lambda + 2\mu}{\rho}} \qquad\qquad c_s = \sqrt{\frac{\mu}{\rho}}. \tag{2.6}$$

The shear and the pressure wave are commonly known as S- and P-waves, with S-waves being slower and carrying more energy.

The source of the earthquake is modeled by a point dislocation with the seismic moment $M_0 = \mu DS$. D is the average offset of the fault and S is the area of the fault [Kan77]. The moment magnitude scale

$$M_w = \frac{log(M_0) - 9.05}{1.5} \tag{2.7}$$

(M_0 in Nm) is commonly used to measure the intensity of earthquakes [HK79]. For example a $M_0 = 10^{18} Nm$ corresponds to a 6.0 on that scale.

The surface interface is modeled by so-called free surface boundary conditions. The normal traction between air and the ground is negligible, so the stress has to be set to zero. For the solver we use, this can be achieved by having ghost cells which mirror the

neighboring stresses and extrapolate the velocities. For non-surface boundaries, outflowing conditions without reflections are wanted, as they are only domain boundaries and should not have any physical effects. Boundary conditions do not have control over stresses parallel to the boundary and therefore, outflow without reflections at the boundary can not be achieved [LeV+02]. Instead, the reflections are minimized by setting the values of the ghost cells at outflow boundaries to zero.

2.2 Adjoint

The idea of the adjoint is, having a method to compute a scalar product with an equation without having to compute the results of the equation directly. To better illustrate the adjoint method we apply it to systems of linear equations $Ax^{(i)} = b^{(i)}$ for $i \in 1 \dots n, n \in \mathbb{N}$. We also assume we are only interested in the result of $\phi^T x^{(i)}$ for a given vector ϕ. We can then define and solve the adjoint equation $A^T y = \phi$. The resulting vector y can be used to solve our initial problem as follows:

$$y^T b^{(i)} = (A^{-T}\phi)^T b^{(i)} = \phi^T A^{-1} b^{(i)} = \phi^T x^{(i)}. \tag{2.8}$$

This allows computing $\phi^T x^{(i)}$ by only solving one system and using the inner products afterwards instead of the traditional approach by first doing a LU decomposition and afterwards needing a quadratic amount of operations per equation for resolving the corresponding triangular system through forward and backward substitution. This principle can be applied to systems of PDEs even though we are using the adjoint differently than direct computational speedup.

2.2.1 Derivation of the Adjoint

This chapter derives the adjoint analogously to [DL16] For this we assume that we are interested in a small region or in the data of a single receiver. This region is described by a smoothed weight function φ.around the point or area of interest.

We further assume that the functional

$$J = \int_a^b \int_c^d \varphi(x,y) q(x,y,t_f) dy dx \tag{2.9}$$

approximates the values we are interested in. The Dirac-δ function is a function that fulfills $\delta(\alpha) = 0$ for $\alpha \neq 0$ and integrates to one over $\mathbb{R}$. As we are usually only interested in a single point, setting φ to a translated Dirac-δ function would be optimal. Directly implementing the δ function is neither possible, nor would it be numerically stable. Therefore a smoothed variant is used.

With $[a,b] \times [c,d]$ being the boundaries of our simulation domain and t_f final time of the simulation. We now derive the corresponding adjoint equation of the elastic wave equation (2.4). We multiply it by a test function p and integrate it over the domain and time

$$q_t + Aq_x + Bq_y = 0 \tag{2.10}$$

$$\int_a^b \int_c^d \int_{t_0}^{t_f} p^T (q_t + Aq_x + Bq_y) dt dy dx = 0 \tag{2.11}$$

We apply integration by parts for each summand. Each term is only integrated over the variable from which it contains a derivative.

$$\begin{aligned} \int_a^b \int_c^d p^T q \Big|_{t_0}^{t_f} dy dx + \int_c^d \int_{t_0}^{t_f} p^T A q \Big|_a^b dt dy + \int_a^b \int_{t_0}^{t_f} p^T B q \Big|_c^d dt dx - \\ \int_a^b \int_c^d \int_{t_0}^{t_f} q^T (p_t + (A^T p)_x + (B^T p)_y) dt dy dx = 0 \end{aligned} \tag{2.12}$$

We now define the adjoint equation as

$$p_t + (A^T p)_x + (B^T p)_y = 0. \tag{2.13}$$

We further set the boundary conditions of the adjoint equations to fulfill the following equations.

$$\begin{aligned} \int_a^b \int_{t_0}^{t_f} q^T B^T p \Big|_c^d dt dx = 0 \\ \int_c^d \int_{t_0}^{t_f} q^T A^T p \Big|_a^b dt dy = 0 \end{aligned} \tag{2.14}$$

Then, only the first term of (2.12) is not zero. Setting the initial condition of the adjoint equation to $p(x,y,t_f) = \phi(x,y)$ (the adjoint is solved backwards in time), results in the equation being:

$$\int_a^b \int_c^d p^T(x,y,t_0) q(x,y,t_0) dy dx = \int_a^b \int_c^d p^T(x,y,t_f) q(x,y,t_f) dy dx \tag{2.15}$$

$$= \int_a^b \int_c^d \varphi(x,y) q(x,y,t_f) dy dx = J \tag{2.16}$$

which in combination with the initial condition of the adjoint equation is our definition of J in (2.9).

As t_0 is not bound directly by the equation it can be set to an arbitrary point between the start and end time. From this follows that points which have a large inner product between p and q have a large impact on our solution value.

For Eq (2.14) to hold we have to set appropriate boundary conditions. Zero boundaries trivially fulfill this requirement. For free surface boundaries we can expand the inner computation to

$$q^T B^T p = \lambda v_q \sigma_p^{11} + (\lambda + 2\mu) v_q \sigma_p^{22} + \mu u_q \sigma_p^{12} + \frac{\sigma_q^{12} u_p}{\rho} + \frac{\sigma_q^{22} v_p}{\rho}. \tag{2.17}$$

As free surface boundaries cause the stress to be zero at the boundary, setting both the forward and adjoint boundaries to free surface conditions causes the term to be zero as each summand contains at least one stress component.

2.2.2 Refinement Procedure

We adapt the refinement procedure of [DL16] in this thesis. Initially the scenario is run on a coarse grid with frequent snapshots of the domain. Independent of that, the adjoint, corresponding to the relevant region, is run with plots at the same timestamps as the forward one. The inner product can be computed from plots taken at the same simulation time. The result could then be directly used for adaptive mesh refinement if one was only interested in the state at the point of interest at exactly the final time.

In general data is needed for a longer time range. This also holds for earthquake simulations. Therefore, each forward checkpoint has to be multiplied by multiple adjoints. As the initial condition of the adjoint does not depend on time or on a forward solution at a specific time, any point in time can be defined as the initial time. Therefore, a single run of the adjoint is sufficient. To fully cover the time range of interest, interpolation in time would be required. Instead plots are taken at a sufficiently small time interval. The value that is taken into account for the refinement is the maximum value that the scalar product takes for a point:

$$J_{ref}(x, y, t) = \max_{0 \leq \tau \leq t_f - t} p^T(x, y, t_f - \tau) q(x, y, t). \tag{2.18}$$

As the initial condition of the adjoint is a smoothed δ-function, with no scaling based on physical properties, the absolute values of the scalar products do not have a direct physical relevance. Therefore the cutoff points for the refinement levels have to be chosen manually.

2.3 Discontinuous Galerkin

Discontinuous Galerkin [HW07] is a subclass of finite element methods. In our case the domain of the simulation is split into right angular cells. The quantities in each cell are represented by a polynomial basis function, usually a tensor product of Lagrange polynomials around Gauss-Legendre points. The cell values across bordering cells are not continuous.

The timestep length is governed by the CFL-condition [Dum+14]

$$\Delta t \leq \frac{1}{d} \frac{1}{2N+1} \frac{h}{|\lambda_{max}|}, \tag{2.19}$$

with h being the cell size, d the number of spatial dimensions and N the order of the cell polynomials. The largest eigenvalue of the Jacobian of the Flux $\lambda_{max} = c_p$ also contributes to the condition. It is the P-wave-speed for elastic wave equations.

For the update of a cell only the neighboring cells are needed. This allows for easy parallelization.

2.4 Peano Curve Traversal

Optimizing cache accesses plays a major role in optimizing a program. Space-filling curve based traversals are better at keeping neighboring cells in cache than iterating row or column wise.

One such method uses the Peano curve to construct the grids and create a refinement order [Bad13]. Figure 2.1 shows the construction of the Peano curve. The first iteration starts with the pattern containing a P. It is replaced by the 3×3 grid of patterns the arrow points to. In the following iterations all patterns are replaced by their production rule. For the grid traversal, the grid is constructed analogously during its initialization. It starts with a 3×3 grid (annotated with P). Each cell is then expanded iteratively according to the Peano curve definition until the desired size is reached. Each cell stores its first sub-cell (i.e. the sub-cell where the curve starts (e.g. the lower left corner for P)) and the following cell of the same level (if it exists). The traversal is done in the same order as depth-first-search would traverse. Figure 2.2 shows an example grid and the corresponding traversal. For adaptive mesh refinement the curve is only expanded for the cells which should be further refined.

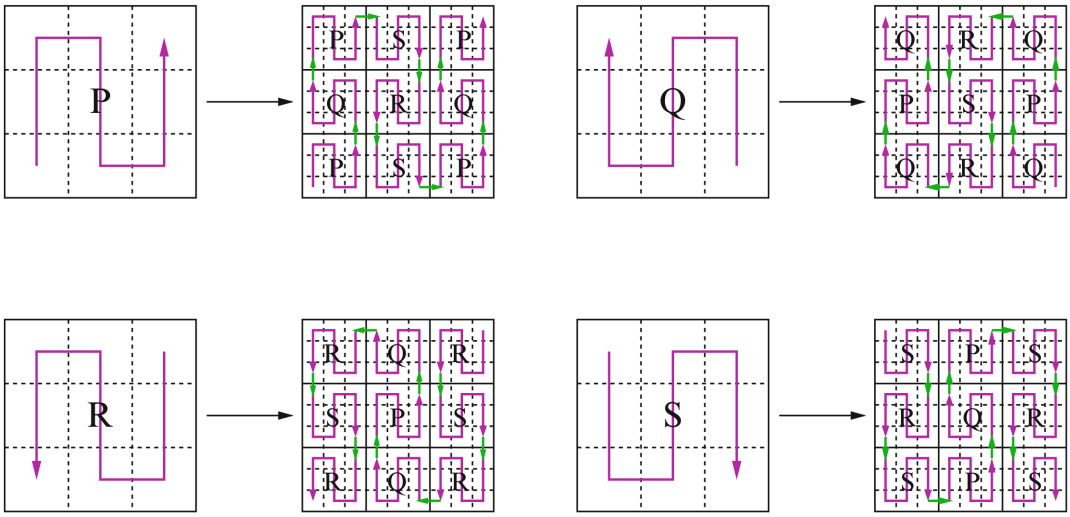


Figure 2.1: Replacement scheme for the Peano curve. Taken from [Bad13]

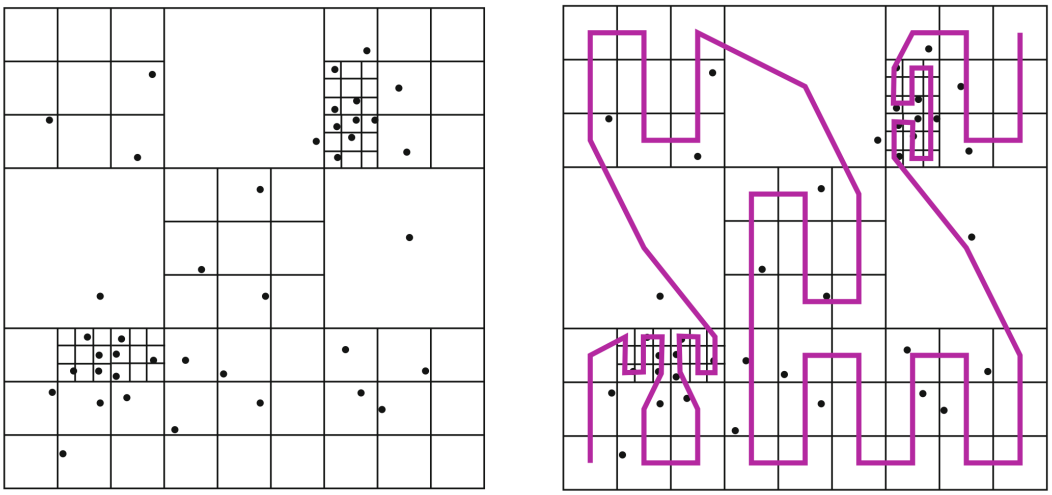


Figure 2.2: Example of a grid traversal which is constructed by the Peano curve. The refinement criterion for this example allows at most one black dot in a cell. Taken from [Bad13]

3 Implementation

This chapter describes the implementation progress and general architecture of the thesis. The PDEs are implemented in C++ with the ExaHype framework and the refinement procedure is created in Python. Figure 3.1 shows a general overview of the procedure. The C++ code code is available at [Hin22a], the Python code at [Hin22b] and a fork of the ExaHype engine at [EH22].

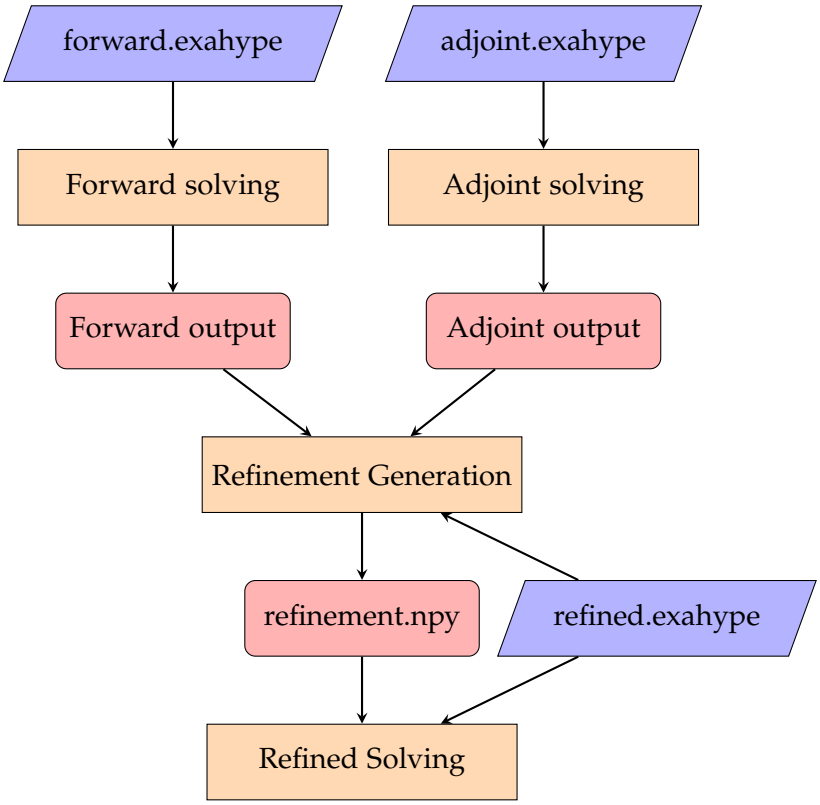


Figure 3.1: Flowchart of the adjoint-guided mesh refinement method. The purple trapezoids represent input files. The intermediate files are shown as red rectangles with rounded corners. The independent programs are shown as orange rectangles.

3.1 ExaHype

ExaHype [Rei+20] is an open-source PDE-framework for Discontinuous Galerkin and finite volume methods. Each ExaHype-project is based on a single configuration file. It defines the solver, PDE characteristics, optimizations and runtime parameters. A script uses the configuration file to generate C++ files accordingly. Most of those files are overridden when rerunning after configuration changes. A ".cpp/.h" file pair is generated for each solver. It is not regenerated in later passes of the toolkit to preserve user changes. It contains method stubs in which the PDE has to be implemented in (e.g. the initial condition, flux, eigenvalues, point sources, etc.) One specialty is that those function are called by ExaHype instead of the user having to manually assemble the solver by calling the functions of the framework.
The cells are iterated over in a Peano curve. Therefore the number of cell in the largest dimension is two less than a power of three (for the two rows of boundary cells). ExaHype only supports square cells, resulting in the domain being extended in directions that have a smaller domain size.

For adaptive mesh refinement the user defines a function which returns whether the refinement level of a cell should be either increased, kept or erased. The framework iterates over the cells and adapts the grid until all cells want to keep their current refinement level.

Build System Improvements: The build system of ExaHype consists of several Makefiles. When compiling a project, the `.o`-Files are put in the same folder as their `.cpp` source. This project needs the implementation of two different equations that are independent of each other and therefore should produce a different executable. As both equations have different parameters and may result in differently compiled files, the build has to be cleaned and rebuilt, every time one switches the equation that is compiled.
ExaHype contains multiple `.cpp`-Files with the same name, so copying all `.o`-Files into the same folder does not work. As a solution the Makefile creates a build folder at the primary Makefile and recreates the entire folder structure of the used `.cpp`-Files to place the `.o`-Files in them.

3.2 PDEs

The elastic wave equation (2.2), in further context called the forward equation, is implemented directly. The only caveat is that because the matrices A, B of equation (2.4) are not part of derivatives the equation has to be implemented as a non-conservative prod-

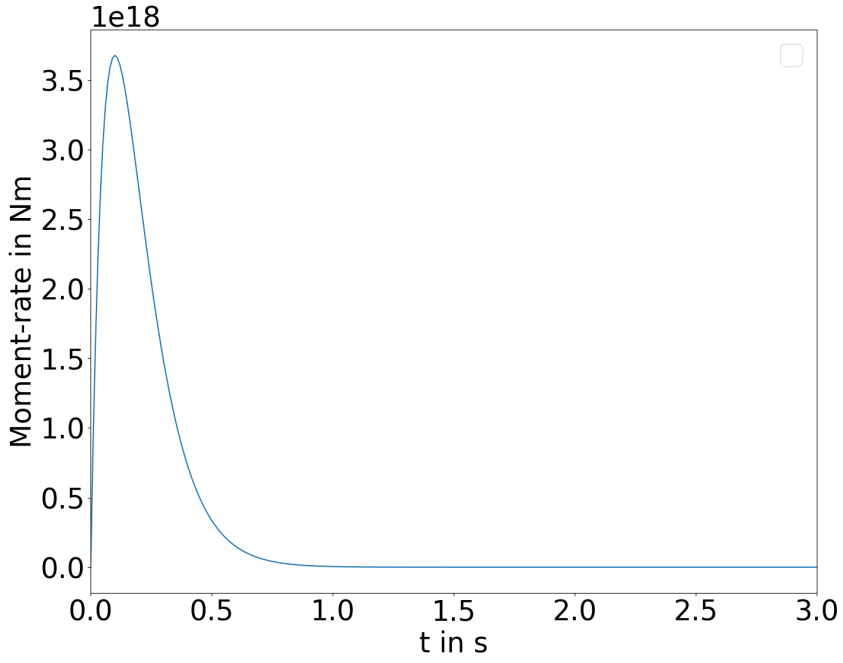


Figure 3.2: Plot of σ_{12} at the source of the earthquake. It is set to Equation (3.1) with $M_0 = 10^{18}$

uct instead of as a flux. The initial condition of the forward equation is implemented as a point source. The point source influences σ_{12} at its location by a force of

$$M_0 \cdot \frac{t}{0.1^2} \exp(-\frac{t}{0.1}). \tag{3.1}$$

This is the moment-rate time history as used by [06]. Figure 3.2 shows its value over time for the value of $M_0 = 10^{18}$ that is used for most experiments.

The adjoint is solved backwards in time but ExaHype can only solve equations forwards in time. Therefore, the equation has to be transformed by negating the time component

$$\frac{\partial p}{\partial t}\begin{pmatrix}\sigma^{11}\\ \sigma^{22}\\ \sigma^{12}\\ u\\ v\end{pmatrix} - \frac{\partial p}{\partial x}\begin{pmatrix}-\frac{1}{\rho}u\\ 0\\ -\frac{1}{\rho}v\\ -(\lambda+2\mu)\sigma^{11}-\lambda\sigma^{22}\\ -\mu\sigma^{12}\end{pmatrix} - \frac{\partial p}{\partial y}\begin{pmatrix}0\\ -\frac{1}{\rho}v\\ -\frac{1}{\rho}u\\ -\mu\sigma^{12}\\ -\lambda\sigma^{11}-(\lambda+2\mu)\sigma^{22}\end{pmatrix} = 0. \tag{3.2}$$

The initial condition of the adjoint sets some variables to

$$\frac{\exp(-4(x-x_s)^2-4(y-y_s)^2)}{4\pi}. \tag{3.3}$$

Which variables are chosen depends on the scenario and discussed in section 4.1.2.

Both equations create plots of the solution as vtk-unstructured-grids. The forward equation can also create probes that write the values at a single point over time.

3.3 Refinement Framework

As the goal of this thesis is to prototype and test the feasibility of adjoint-guided mesh refinement for elastic wave equations, we chose Python as the language to implement the remaining code in for faster implementation time and easier debuggability.

The python code requires the ExaHype configuration files for the forward equation (for both the coarse and the refined run) and templates for the output files of the forward and adjoint equation. The ExaHype configuration files are used for metadata like the domain size or the timestep of the output files.

The output files are read with the vtk-library. They contain unstructured grids with no apparent order of the points and cells contained, even if the plotted grid is Cartesian. The order of the points stays constant for all output files of a single run, meaning a forward and an adjoint run on the same grid will result in general in differently ordered points. The order of the points in both output files was checked for equality to possibly skip the interpolation step. As this never occurred, it was later removed. As a first step the adjoints are interpolated onto the forward grid. This allows usage of differently structured forward and adjoint grids. As interpolation kernel a nearest neighbor kernel is used, as it is assumed that the forward grid has approximately the same structure and linear interpolation is not possible due to limitations of the vtk-library (the linear interpolation kernel in the vtk-library simply averages all points in the vicinity [KIT22]). The result of the interpolation is stored on disk and reused if the same adjoint and forward combination is used. The results are identified by the hash of the second output file of both the adjoint and forward solution (the second plot is used as the first one contains only zeros for the forward equation).

After that the algorithm loops over all forward files and computes the inner product of it and each adjoint in the relevant time frame (e.g. for a forward plot at $t = 0.2$ all adjoints up to $t_{end} - 0.2$ are used). The maximum value of the inner product for each position is then mapped to the refinement grid (see section 3.4) with vtk interpolation. The level of refinement is determined by user-defined percentiles, e.g $\frac{1}{6}$ of all cells get refined once and $\frac{1}{18}$ of the area gets refined twice, meaning $\frac{1}{3}$ of the cells refined once get refined a second time.

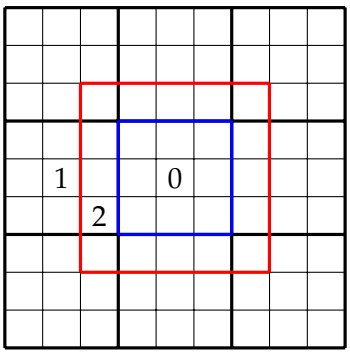


Figure 3.3: 3×3 refinement grid with max level $n = 2$. Empty cells mean zeros. The three-to-one balancing algorithm currently works on the blue level 0 cell. The explicitly stated 0 is the value responsible whether the cell should be refined. The red area is the region which is checked for level 2 refinements. As one is found, the cell has to be refined to level 1 and the 0 in the middle has to be set to 1.

The refinement is saved as a NumPy array in a `.npy` file. The `.npy`-format is a simple file format, starting with a short ASCII-header containing the dimensions, byte- and element-order of the array. After the header the array data is stored in binary in the same layout as in memory. This file is loaded during the initialization of ExaHype in the refined run. Examples of the refinement array can be seen in the Results Chapter 4 with Figure 4.1 having a more detailed description. The array has a value for every cell of the grid that can be further refined (i.e. not for cells of the highest level).

3.4 Three to one balancing

Too large differences in size between neighboring cells may lead to unwanted effects as the larger cells can not represent all frequencies that the smaller cells can. Section 4.2.2 shows that this can even be a problem with differences of only one level. Therefore we have to ensure that the level between neighboring cells differs at most by one, a process called two-to-one balancing (as levels are usually based on powers of two). ExaHype does not give access to the refinement status of neighboring cells. Therefore, we have to ensure this balance in the python code. This is the main reason we chose a direct mapping between the refinement grid and the cells in the solver.

Let $n + 1$ be the maximum refinement level, i.e. the number of times an unrefined cell can be refined. The refinement grid in python is a numpy array which has an entry for every cell of the grid that can be further refined (i.e. not for cells of the highest level)

An element of that array determines up to which level the corresponding cell should be refined. Algorithm 1 shows the procedure. It loops over all levels except the unrefined one (0) and the largest refinement level $(n+1)$. In this loop the algorithm loops over all cells of the next lower level to check whether they should be marked for refinement. A cell needs to be marked when either a subcell (of level lvl) or a neighboring cell of level lvl should be further refined (to at least $lvl+1$). Figure 3.3 shows an example with a maximum refinement level of 2.

```
1  for lvl : n → 1 do
2      foreach (x, y) ∈ PointsOfLvl(lvl-1) do
3          s ← StrideOfLevel(lvl) ;
4          for i : −2 → 2 do
5              for j : −2 → 2 do
6                  if grid_{x+s∗i,y+i∗s} > lvl then
7                      grid_{x,y} ← max(grid_{x,y}, lvl)
8                  end
9              end
10         end
11     end
12 end
```

Algorithm 1: Three to one balancing algorithm. The stride computation was simplified for visualization purposes and the bounds checking is omitted.

3.5 Refinement in ExaHype

For the refinement we introduce a new optional parameter for the solver attribute called "adg" in the `.exahype` file. If it is set to a file (the refinement NumPy array), the code enables refinement. During the initialization of the solver the NumPy array is loaded with the cnpy-library [Rog18] from the given file. NumPy treats 2d-arrays as matrices so the first index enumerates the rows but to make the python code more readable the first index was used for the x-direction and the second index for the y-direction. The array is transposed in the python code and therefore NumPy stores it in memory and in the files in FORTRAN-order, which allows us to treat the array as a C-order array during the refinement.

Listing 3.1 shows how the refinement level is read for a cell. `data` is the refinement array, `xsize` and `ysize` give its dimensions in x- and y- direction respectively.

Listing 3.1: Computation of the refinement level

```
auto xmap=(cellCentre[0]-offset[0])*xsize/domainsize[0];
auto xind= (int)std::floor(xmap);
auto ymap=(cellCentre[1]-offset[1])*ysize/domainsize[1];
auto yind= (int)std::floor(ymap);
return data[xind+(yind)*xsize];
```

If the requested level is higher for a cell than its current refinement level, the cell returns that it should be refined, otherwise it returns that it should keep its refinement level.

3.6 Dynamically Adaptive Mesh Refinement

We have also implemented a prototype for the dynamically adaptive mesh refinement. The duration of the simulation is split into 20 equal time slices. The refinement of each slice is constructed using the coarse forward outputs from the slice and the following one. For example, when setting the total duration to $3s$ then each slice is $0.15s$ long, so the first slice uses the coarse grids data from $0s$ to $0.3s$ and second from $0.15s$ to $0.45s$ this continues up to the last slice which only uses the data from $2.85s$ to the end.

This is implemented in the refinement construction in Python by using 20 accumulator arrays for the inner product. From those arrays 20 refinement grids are created by calling the same function, as for the statically adaptive refinement method, 20 times. The refinement arrays are written to disk as a single NumPy array of size $(20, x, y)$. The merging into a single array changes the array memory layout and requires the C++ code to treat the array as if it was using the FORTRAN-order. The only other changes for C++ code are calculating which grid to use, passing it to the function which reads the refinement level and the refinement function returning `ERASE` when the level given by the array is lower than the current level.

4 Results

In this chapter we test the adjoint-guided mesh refinement technique on multiple different scenarios and analyze the results. Additionally we briefly evaluate the dynamically adaptive variant of the technique.

4.1 Test Setup

ExaHype does not adapt the timestep length to plot probes at the specified times, instead it measures it after the first timestep that ends after the specified time. These time errors are usually under 1% of the plotting step, but are about nine times larger (due to the CFL-condition) for the coarse forward grids. Therefore all probes are resampled during the comparison step. They are linearly interpolated to the regular time points.

To compare probes against each other the Root Mean Square (RMS) error of [Kri+06] is used. It is defined as

$$RMS = \sqrt{\frac{\sum_t |q(t) - q_{REF}(t)|^2}{\sum_t |q_{REF}(t)|^2}}. \tag{4.1}$$

In all following examples the probe measuring interval is $20ms$. The coarse forward and adjoint equations are plotted every $25ms$, as fewer plots would result in forward and adjoint waves passing each other over certain spots in between plots. This would results in holes in the refinement where the waves passed each other outside of a plot. The base unit of the used coordinate systems is km.

4.1.1 P-Waves only

To validate the basic functionality of the method, a simple example without S-waves was run. The domain has an S-wave speed of zero, a P-wave speed of $6.0\frac{km}{s}$ and a density of $2700\frac{kg}{m^3}$. This results in $\mu = 0$. Therefore, the point source has to set σ_{11} and σ_{22} instead of σ_{12} as σ_{12} stays constant in time. The M_0 factor was set to 10^{18} which does not have all its described physical properties due to only simulating the

P-waves. We set a free surface boundary at $y = 0$, the source of the P-waves is set to $(5, 5)$ and the measuring probe to $(12, 5)$. Figure 4.1 shows the scenario, including the refinement created by the adjoint-guided mesh refinement method. The domain is $(-5, 25) \times (0, 15)$, the unrefined cell size is $\frac{30}{79}$ and the unrefined grid has $79 \times 40 = 3160$ cells. The scenario was run for $3s$, which is long enough for the wave that reflected at the surface to reach the probe, but short enough to avoid returning waves from other boundaries which should not be reflective. Due to simulating only P-waves, we set σ_{11} and σ_{22} to non-zero values in the adjoint initial condition. We chose to refine $\frac{1}{6}$ of the grid once and $\frac{1}{18}$ of the total area twice, i.e. $\frac{1}{3}$ of the refined cells get refined a second time. Figure 4.1 shows the refinement. For the adjoint-guided refinement the most refined region is the direct path between the source and the probe. The start, the finish and the area at the surface, where the wave reflects to the measuring probe, is also refined at the strongest level. The remaining part of the reflected path and the surrounding area of all paths is refined at a lower level.

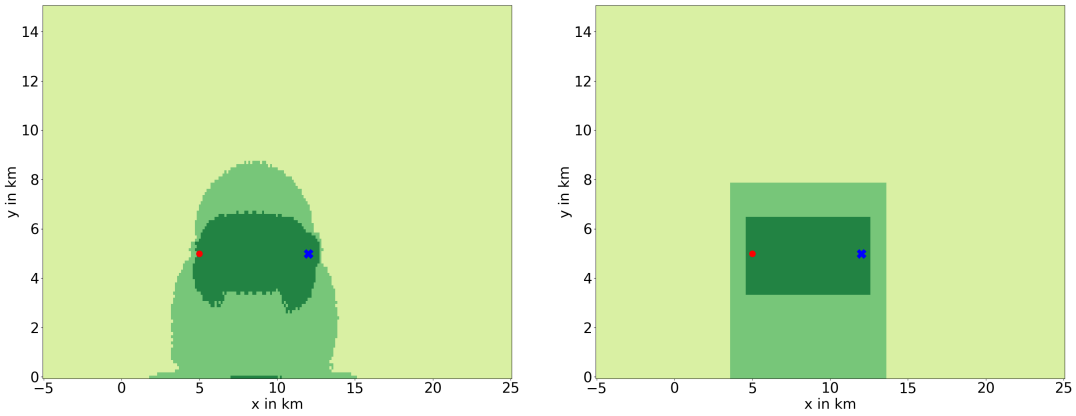


Figure 4.1: Refinement of the P-wave only scenario. The source is marked as a red dot, the probe as a blue X. On the left is the adjoint-guided refinement, on the right is a manually created refinement with the same number of cells. The lightest green means no refinement, the darkest means two levels of refinement and the middle one means one level of refinement.

To compare this refinement we manually constructed a refinement grid as there are no other widely used automatic mesh refinement methods for seismologic simulations. The number of refined cells of each level is the same for both the adjoint-guided refinement and the manual refinement. We create a box around the source for the level 2 refinement of the manual refinement. For the level 1 refinement we have enough cells to extend the box to the surface, where it can easily be seen that a reflection will occur

without looking at an adjoint or forward output.

A fully refined grid (2 layers more than coarse grid) was used to compute a reference result. Table 4.1 shows the RMS for all components of this scenario. It is zero for σ_{12} as σ_{12} is constantly zero and cannot change. For the variables with non-zero values the adjoint-guided refinement is slightly better than the manual refinement. Figure 4.2 shows how the velocity in x-direction u behaves over time and how close both refinement methods are to the reference solution. The arrival of the P-Wave (starting at $t = 1.2s$) and its reflection (at $t = 2.1$) are visible. The results show that the manual refinement is worse for the direct wave and almost as accurate as the adjoint guided refinement for the reflected wave.

Table 4.1: RMS for the point $(5, 12)$ in the P-wave scenario

	σ_{11}	σ_{22}	σ_{12}	u	v
adjoint-guided refinement	0.0220	0.0220	0	0.0146	0.0362
manual refinement	0.0237	0.0237	0	0.0168	0.0378

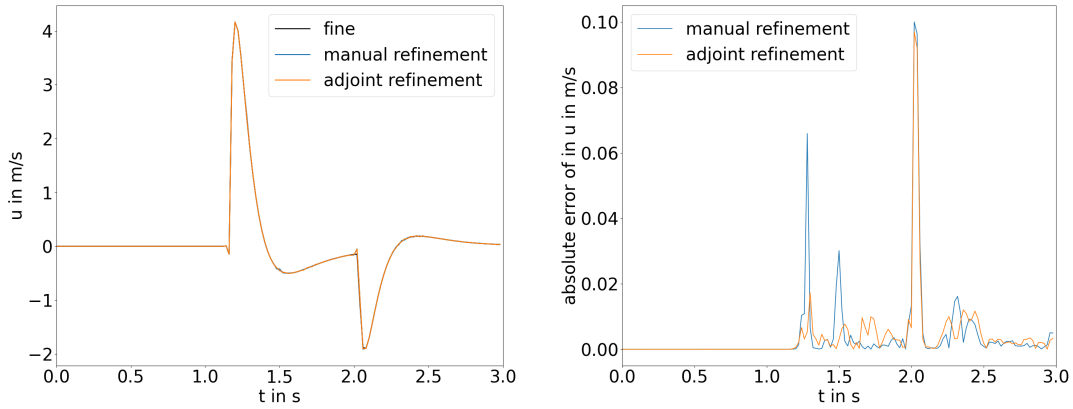


Figure 4.2: Plot of u in the P-wave scenario at $(12, 5)$ on the left and the absolute error of the same on the right.

4.1.2 Adjoint initial condition

For shallow water equations setting a pulse in the water height is chosen as the initial condition for the adjoint in [DL16]. There is no directly corresponding measure for elastic wave equations, so we looked at reasonable options. Setting u without setting v

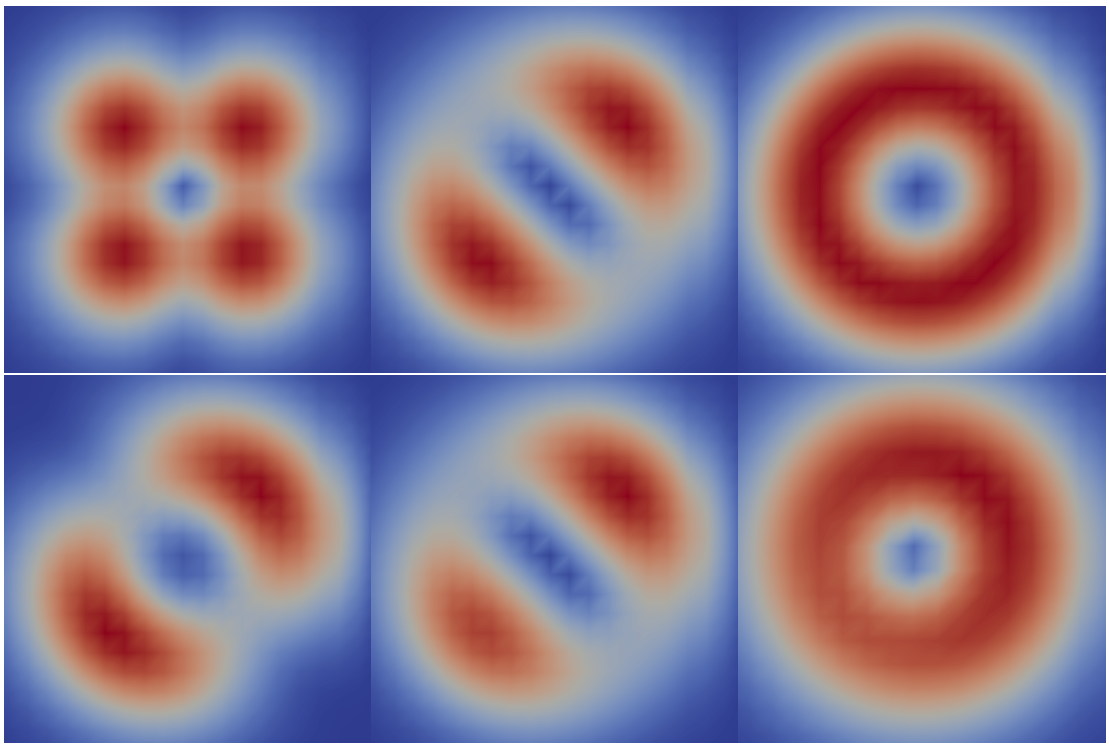


Figure 4.3: Adjoint velocity magnitude at $t = 0.08s$ for different adjoint initial conditions. Different linear color scales are used for each picture with red being the maximum and blue being zero. In the first row, from left to right: only σ_{12} is set, all three stress components are set, σ_{11} and σ_{22} are set. In the second row: both velocities are set, all five variables are set, σ_{11}, σ_{22},u and v are set.

is not useful as this is biased in one direction, the same holds for the other way round and for σ_{11} and σ_{12} respectively. Figure 4.3 shows the velocity magnitude of the tested initial conditions at $t = 80ms$ (the scenario has a uniform material). All plots use a different color scale as only relative differences in the adjoint have an impact on the refinement. In three of the scenarios the wave only spreads along the x-y-diagonal and they are therefore unfeasible as an initial condition.

The largest velocity magnitude for the initial condition that sets σ_{11} and σ_{22} is 0.54 but for the initial condition that sets the velocities it is only 0.039. This explains why the runs that sets σ_{11} and σ_{22} and the run that sets σ_{11}, σ_{22}, u and v produce a similar result but for the latter the upper right side of the circular wave has higher values than the lower left side which makes that one also unsuitable. The run that sets σ_{11} and

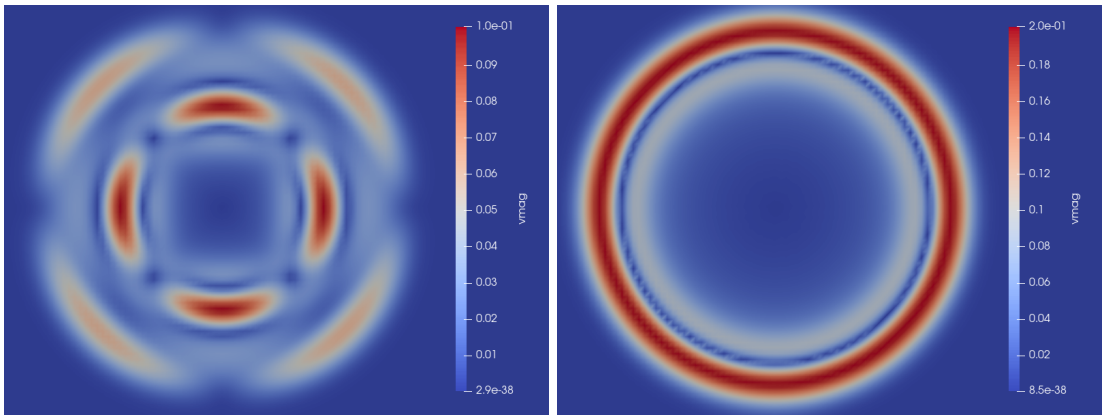


Figure 4.4: Adjoint velocity magnitude at $t = 0.5s$ for the adjoints which sets σ_{12} (left) as the initial condition and the one that sets σ_{11} and σ_{22} (right). As the absolute scale is not of relevance, both images contain a different scale for easier readability.

σ_{22} in the initial condition produces almost perfectly uniform velocities at the same distance to the source. The run that sets σ_{12} in the initial condition has no differences in the four cardinal directions, but achieves higher velocities along the diagonals. Figure 4.4 shows the last two candidates for the initial condition at $t = 500ms$. The run which sets σ_{12} in the initial condition shows the faster P-waves and the slower S-waves. Both waves are not circular, but have the same direction biases as the forward equation. The run that sets σ_{11} and σ_{22} as the initial condition produces only a single circular wave moving at P-wave speed. Both having the same waves as the forward equation and treating all angles the same can be useful as later results will show. To allow for better readability we will name both initial conditions. The adjoint that sets σ_{11} and σ_{22} in its initial condition will be called α**-adjoint** and the corresponding adjoint-guided mesh refinements are called α**-adjoint refinement**. The adjoint that sets σ_{12} in its initial condition will be called β**-adjoint** and the corresponding adjoint-guided mesh refinements are called β**-adjoint refinement**.

4.1.3 Two Halfspaces

As the first scenario actually using the elastic wave equations, we use two halfspaces next to each other with all edges having outflowing boundary conditions. Figure 4.5 shows the setup and defines the material parameters. We analyze two points, the first is a point at $(9, 13)$, close to the source in the same material as the source, and the second point $(13, 18)$ is further away in a different material than the source. Due to the

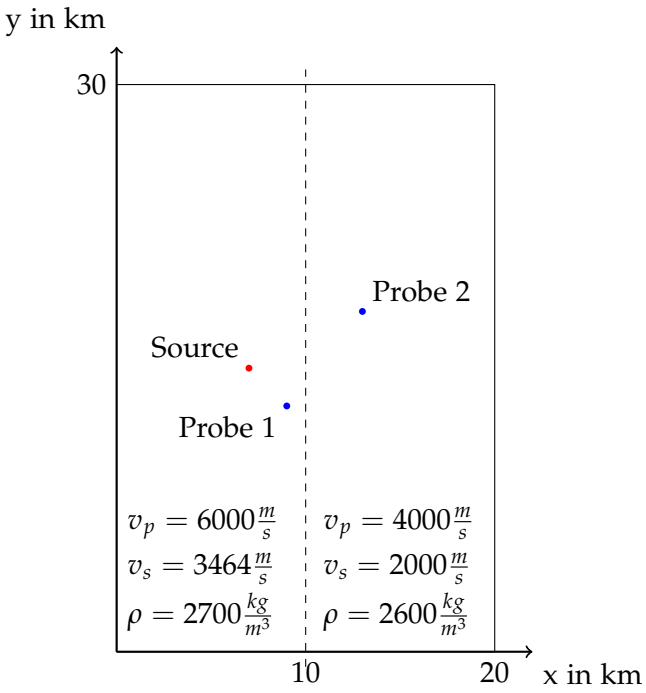


Figure 4.5: The setup of the scenario with two Halfspaces.

horizontal domain we now use a 53×79 grid (4187 cells) as the unrefined grid. As before we run the simulation up to $t = 3s$ to measure the passing S and P-waves at the probe points and to mostly avoid unwanted reflections from the artificial boundaries. The magnitude of the earthquake was set to 5.97 (a round value for $M_0 = 10^{18}$).

Table 4.2: RMS for the point $(9, 13)$ in the Two Halfspaces scenario. ($\frac{1}{9}$, $\frac{1}{18}$ cutoff, unless specified

	σ_{11}	σ_{22}	σ_{12}	u	v	q
manual refinement	0.00423	0.00425	0.00178	0.00469	0.00562	0.00310
α-adjoint refinement	0.00440	0.00521	0.00201	0.00426	0.00572	0.00356
β-adjoint refinement	0.00427	0.00427	0.00200	0.00620	0.00526	0.00320
α-adjoint refinement: $\frac{1}{18}$, $\frac{1}{36}$ cutoff	0.0154	0.0137	0.00888	0.0198	0.0209	0.0116

Figure 4.6 shows the refinements for both points. At first glance one might assume that the refinement area is too large for the $(9, 13)$ setup but Table 4.2 shows a significantly worse error for the smaller refinement. The table also shows that the manual

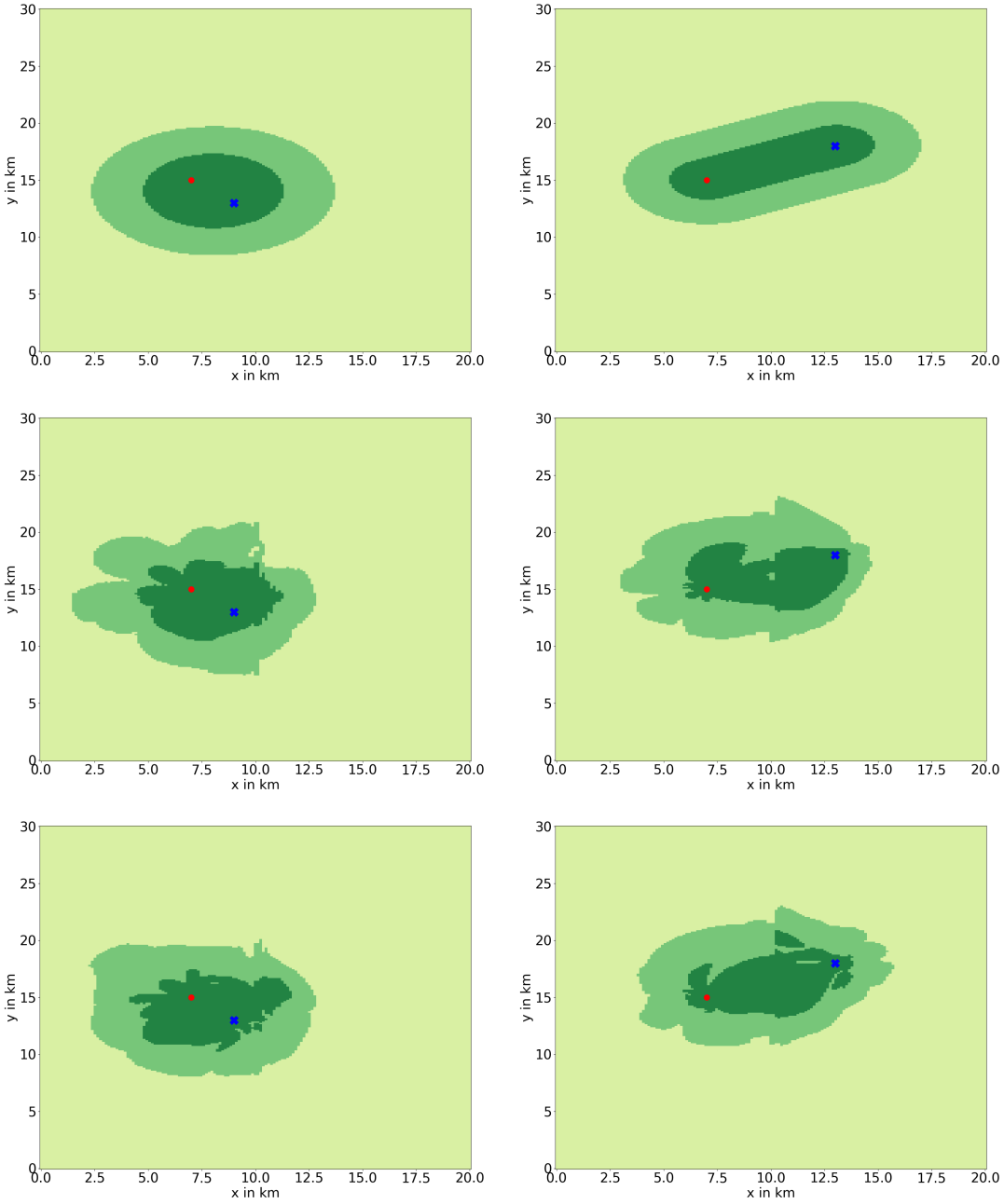


Figure 4.6: Refinements of the Two Halfspace scenario with the point of interest at $(9, 13)$ on the left and $(13, 18)$ on the right. The manual refinements are at the top, the middle ones are generated from adjoints which set σ_{11} and σ_{22} and the bottom ones are generated from adjoints that set σ_{12}. In all refinements $\frac{1}{6}$ of the cells are refined once and $\frac{1}{18}$ of the total area is refined twice.

refinement gets slightly better results than the β-adjoint and better results than the α-adjoint. However both methods had a better accuracy for one of the velocities than the manual approach . Figure 4.7 shows u at the probe location and how close the refinement are to the fully refined grid. The only visible deviation in the left graph are oscillations by the β-adjoint, starting from $t = 1.7s$, which has the highest RMS for u for this probe. The absolute errors correspond to the results one would expect from the RMS-table for u, although most of the error occurs after the passing of the waves.

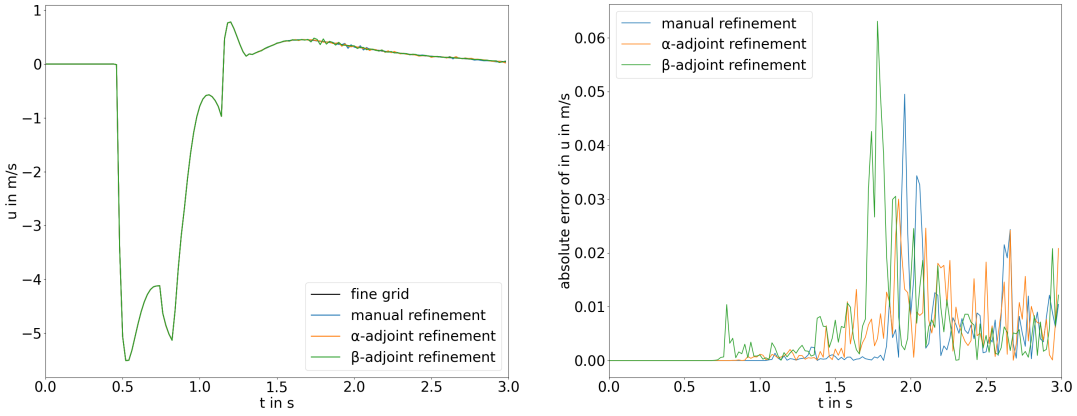


Figure 4.7: Plot of u in the Two Halfspaces scenario at $(9, 13)$ on the left and the absolute error of the same on the right.

Table 4.3: RMS for the point $(13, 18)$ in the Two Halfspaces scenario with $\frac{1}{9}$, $\frac{1}{18}$ cutoff.

	σ_{11}	σ_{22}	σ_{12}	u	v	q
manual refinement	0.00572	0.00818	0.00337	0.00524	0.00563	0.00625
α-adjoint refinement	0.0120	0.0135	0.0135	0.0120	0.0165	0.0127
β-adjoint refinement	0.00746	0.00763	0.0160	0.00719	0.0151	0.00956

Table 4.3 shows that for the second point the manual approach achieves a better accuracy for all variables, especially σ_{12}. Figure 4.8 shows the seismogram of the wave and the errors of the different refinements. Almost no differences between the refinements are visible in the plot of u. The error plot shows that the manual refinement is clearly better. This shows especially at the start, which is presumable due to not having holes in the refinement of the direct path from the source to the probe.

The manual refinements are created by circles of appropriate size for the point $(9, 13)$ and by including points up to certain distance around the line connecting the source

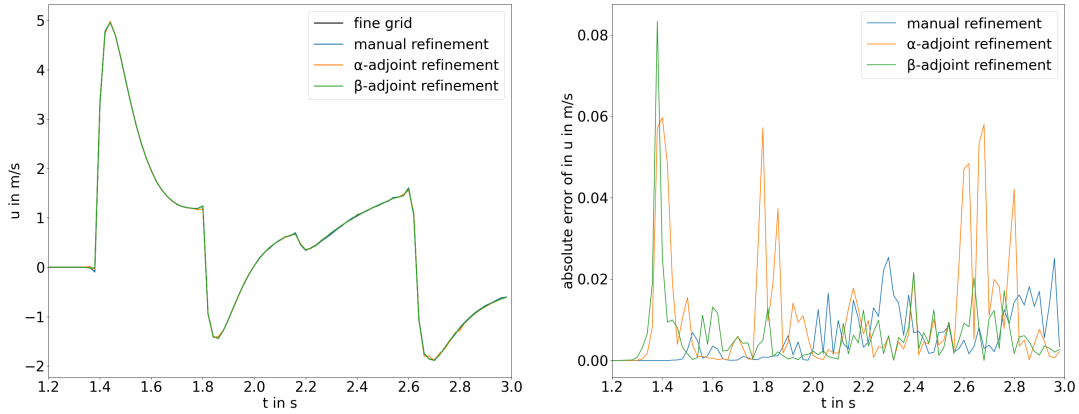


Figure 4.8: Plot of u in the Two Halfspaces scenario at $(13, 18)$ on the left and the absolute error of the same on the right. The x-axis starts at $t = 1.2$ as the first wave arrives at $t = 1.3$.

and the probe at $(13, 18)$, to refine the same number of cells as the adjoint refinements. These shapes are valid for comparisons to the adjoint refinements as they produce a lower error.

4.1.4 Helsinki Model

As a more sophisticated example the model from [Leo+21] is used. The model was created by observing seismic events at a geothermal borehole. Half of the sensors were placed in the borehole at different depths and the other half was placed in the vicinity. Figure 4.9 shows the resulting model of the P-wave speeds of the geology below Helsinki. The wave speeds only vary in depth, the S-wave speed is $\frac{1}{1.71}$ of the P-wave speed and the density was assumed to be $2700\frac{kg}{m^3}$.

The domain is set to $(-5, 25) \times (0, 15)$ with a free surface boundary at $y = 0$ and the y-axis denoting the depth below the surface. We reuse parameters from the previous scenarios. The end time is set to $t = 3.0s$, we have a $79 \times 40 = 3160$ cell configuration and set $M_0 = 10^{18}$ (5.97 magnitude).

We firstly focus on the point $(11, 2)$. Here we choose two different cutoff percentiles for the first refinement level ($\frac{1}{6}$ and $\frac{1}{9}$ and set the second level to $\frac{1}{12}$ for both refinements). Figure 4.10 shows the refinements. The manual refinements are created by the same method as the refinements for the point $(13, 18)$ of the previous experiment.

Figure 4.11 shows that the reflected P-wave and the S-wave, on its direct path, overlap at our point of interest. It also shows that the S-waves and the P-waves split into two

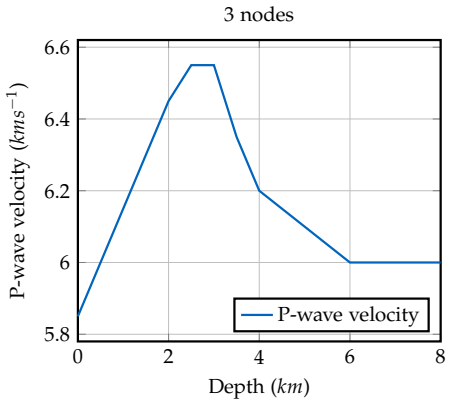


Figure 4.9: A model of the P-Wave speed under Helsinki developed by [Leo+21] .

separate waves when they reflect at the boundary. The region with the highest velocity is around the point source. This is caused by the stress differences remaining in the material after an earthquake.

Table 4.4 shows the results of the run with fewer refined cells. The β-adjoint refinement produces the best results after the α-adjoint refinement. The manual refinement achieves better errors than the α-adjoint refinement for σ_{12} and v, but in total is significantly worse than the other two refinements. Figure 4.12 shows u and its absolute error over time. This is the first scenario where the errors are large enough to be easily visible in the primary plot. In the first part the manual refinement is better, but it gets worse results for the start and end of the reflected S-wave ($t = 2.2$ and $t = 2.5$).

Table 4.4: RMS for the point $(11, 2)$ in the Helsinki scenario with $\frac{1}{9}$, $\frac{1}{12}$ cutoff

	σ_{11}	σ_{22}	σ_{12}	u	v	q
manual refinement	0.0971	0.0724	0.0798	0.105	0.0517	0.0865
α-adjoint refinement	0.0592	0.0629	0.0967	0.0958	0.0527	0.0653
β-adjoint refinement	0.0623	0.0555	0.0766	0.0845	0.0455	0.0614

Table 4.5 shows the results when increasing the number of cells that are refined once by 50% (to $\frac{1}{6}$). All three runs produce better results than the less refined variant (Table 4.4). The manual refinement has a higher RMS error for σ_{11}, σ_{22} and u than both adjoint refinements. It lands in between the errors for σ_{12} and achieves the best result for v. In total the α-adjoint achieves the best result even though the β-adjoint has a lower error for three of the five variables. Figure 4.13 shows u and its absolute error over time. In the plot of u very few errors are visible, mostly just that the run with the manual

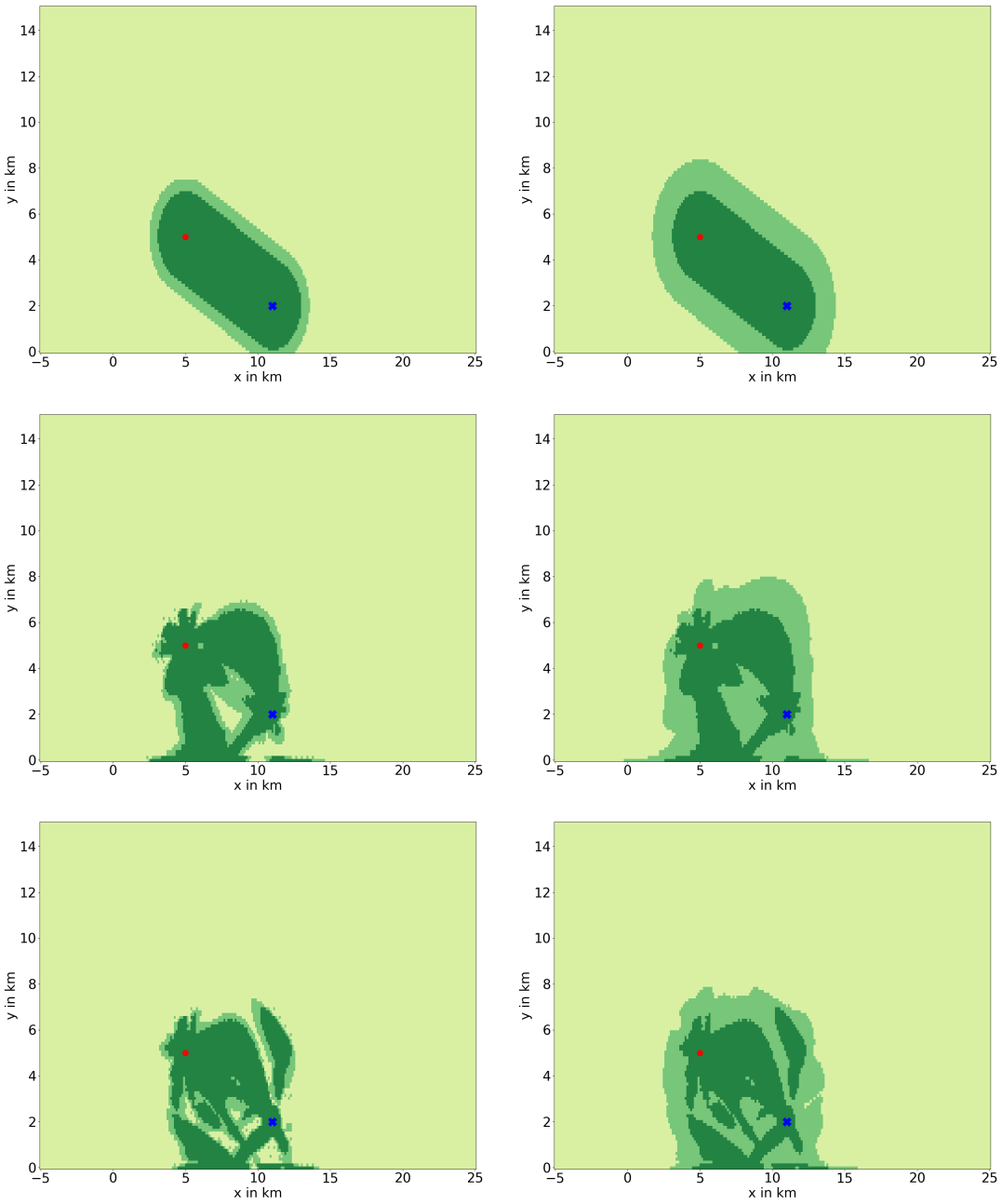


Figure 4.10: Refinements of the Helsinki scenario with the point of interest at $(11, 2)$. The manual refinements are at the top, the α-adjoint refinements are in the middle and the β-adjoint refinements are at the bottom On the left $\frac{1}{9}$ of the cells are refined once and $\frac{1}{12}$ of the total area is refined twice on the right $\frac{1}{6}$ of the cells are refined once and $\frac{1}{12}$ are refined twice.

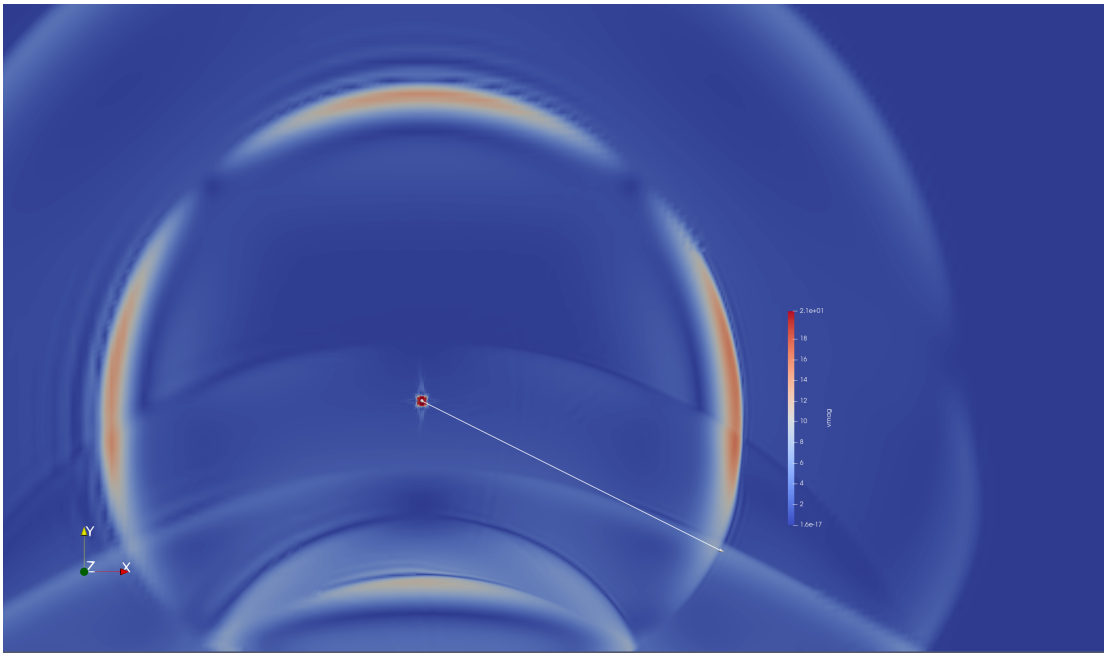


Figure 4.11: Velocity magnitude of the Helsinki scenario at $t = 1.8s$, α-adjoint refinement with $\frac{1}{6}, \frac{1}{12}$ cutoff. The straight white line connects the source (on the left) and the probe location. The waves are more visibly clearer defined in the refined part on the right than on the left.

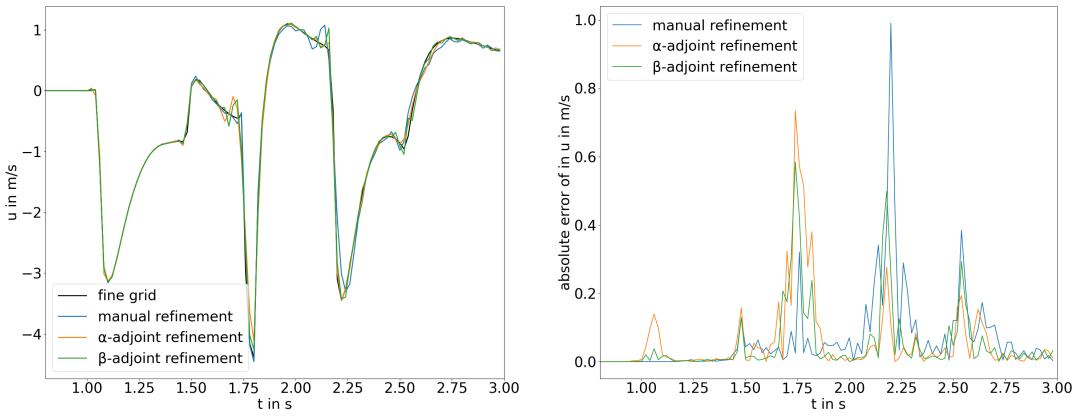


Figure 4.12: Plot of u in the Helsinki scenario with $\frac{1}{9}, \frac{1}{12}$ cutoff at $(11, 2)$ on the left and the absolute error of the same on the right. The plot starts at $t = 0.8$ as the first wave arrives at $t = 1.0$.

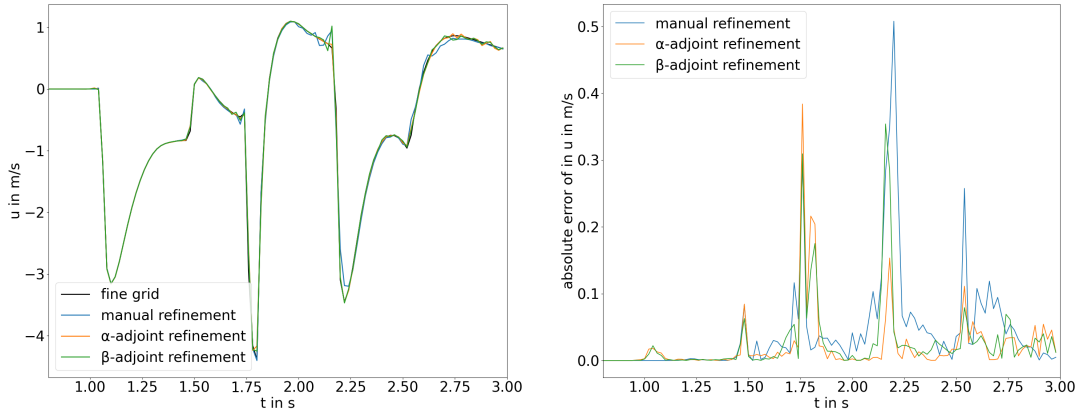


Figure 4.13: Plot of u in the Helsinki scenario with $\frac{1}{6}, \frac{1}{12}$ cutoff at $(11, 2)$ on the left and the absolute error of the same on the right. The plot starts at $t = 0.8$ as the first wave arrives at $t = 1.0$.

refinement is deviating at the end. The error plot shows almost the same result as the errors for the less refined runs at half the scale.

Table 4.5: RMS for the point $(11, 2)$ in the Helsinki scenario with $\frac{1}{6}, \frac{1}{12}$ cutoff

	σ_{11}	σ_{22}	σ_{12}	u	v	q
manual refinement	0.0590	0.0433	0.0497	0.0650	0.0305	0.0525
α-adjoint refinement	0.0319	0.0354	0.0560	0.0397	0.0420	0.0364
β-adjoint refinement	0.0396	0.0343	0.0489	0.0454	0.0357	0.0387

4.2 Problems

4.2.1 Static Stress Region Near the Point Source

One of the problems that causes suboptimal refinements is the static stress region around the point source. Figure 4.11 shows that this is the region with the largest velocity. Figure 4.14 shows the magnitude of the stress at the same time. We had to use a logarithmic scale to show both the point source and the waves. The magnitude of the stress at the point source is 100 to 1000 times larger than at the wavefront of the S-waves. This has the effect that a position, in which the angle of the variables to the adjoint has an error of 0.01 (radians), can move from an scalar product of zero to a scalar product

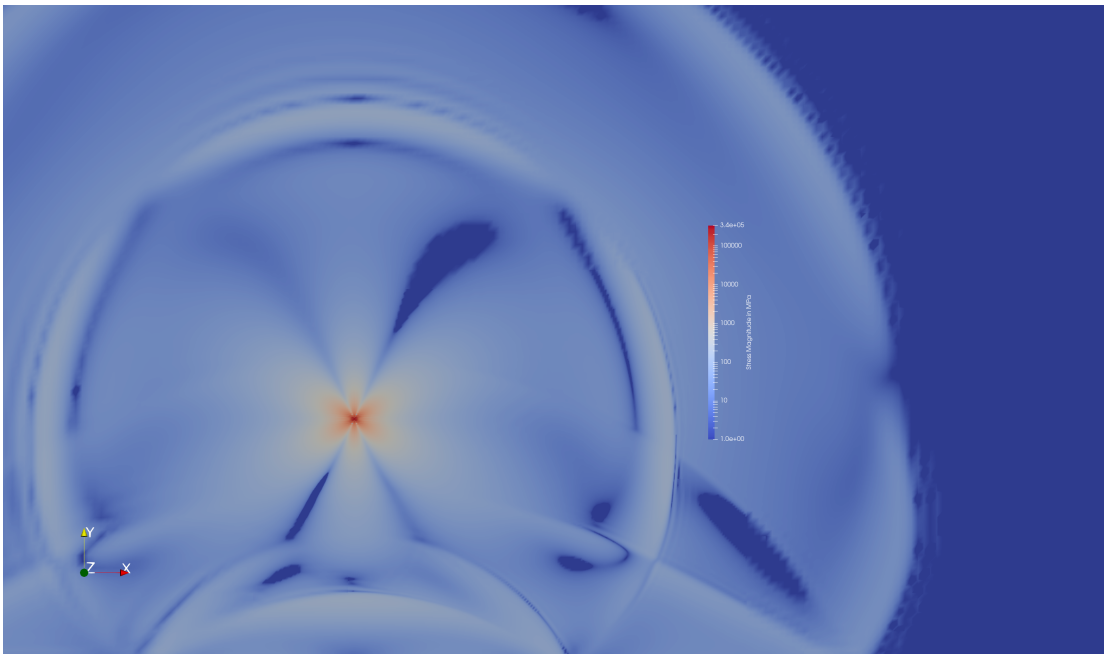


Figure 4.14: The magnitude of the stress of the Helsinki scenario (α-adjoint refinement) at $t = 1.8$. A logarithmic scale is used to show more than just the area around the point source.

that is larger than the maximum scalar product at the wavefront. This effect can be seen in the six-armed star pattern of the stress at the point source, reappearing in many refinements (e.g. the α-adjoint refinements in Figure 4.6 and Figure 4.10).

4.2.2 Reflections on refinement boundary

Another reason why some adjoint-guided results are worse than the manually refined ones is due to reflections occurring on refinement borders. One of the easiest ways to see this, are the error spikes on the manual refinement (Figure 4.1 and 4.2) of the only P-waves scenario. The reflected waves from the refinement boundary arrive at 1.26s for the fine refinement (1.45km to the refinement boundary in y-direction) and at 1.49s. Figure 4.15 shows the section where the oscillations are visible.

An older version of the refinement algorithm produced the refinement grid in Figure 4.16 for the only P-Waves scenario (section 4.1.1) Even though the refinement covers the direct path from the source to the point of interest the refinement is worse than the manual refinement, due to chaotic reflections at the refinement boundary.

The reflections might be less of an issue in PDE-solvers which do grid refinement by

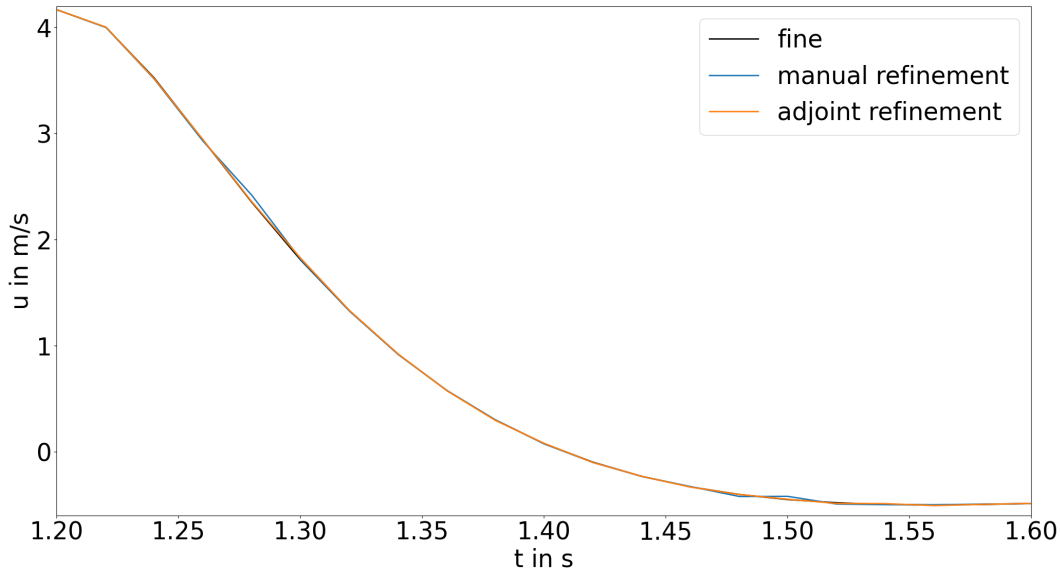


Figure 4.15: Partial plot of u in the P-waves only scenario. In the manual refinement are visible oscillations at $t = 1.26s$ and $t = 1.49s$ which are caused by the reflection at the boundaries.

halving the cells in each dimensions, instead of splitting them into thirds.

4.3 Dynamically Adaptive Mesh Refinement

To test whether some of those problems occured from using statically adaptive mesh refinement, we implemented a prototype of a dynamically adaptive mesh refinement variant.

This was originally not tested extensively, due to ExaHype failing to construct at least one of the 20 generated refinement grids and therefore, all runs crashed before finishing. The run which managed simulate the most (75%), had a significantly worse result than the statically adaptive refined grids and took longer (even though simulating less grid cells at the same time). The following example was run later to reuse one of the experiment setups of this thesis and experienced no such problems.

We continue using the scenario from section 4.1.4 with the point of interest at $(11, 2)$. For the dynamic adaptivity we have set the refinement parameters to only refine $\frac{1}{12}$ once and $\frac{1}{24}$ twice to account for the additional time needed for grid changes and to not allow the dynamic refinement to refine everything at once. The actual time needed

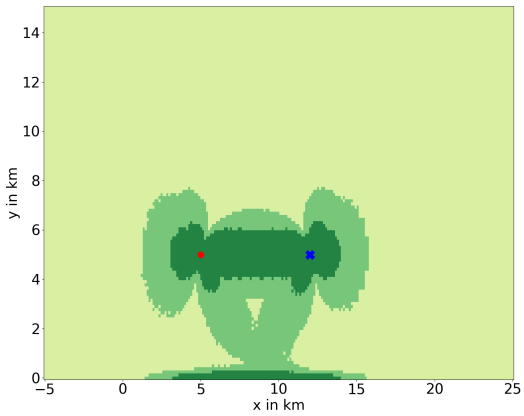


Figure 4.16: Refinement for the P-wave only scenario, produced by an outdated version of the refinement method.

is 26 and 27 minutes for the dynamic refinements and 38 and 40 minutes for the static refinements. The run durations were reconstructed from the timestamps of the output files. They were run single threaded and in single process (due to frequent crashes and limited speedup of ExaHype's parallelization) on a zen2 processor at 4.2 GHz. Figure 4.17 shows both the refinement and the the waves at the same time. It shows that the algorithm refines almost 180° of the S-wave wavefront, while it only minimally refines the path for the P-waves. Only the direct path to the point of interest, the path which reflects at the surface and a stretch at the boundary, preparing for surface waves, are refined. It uses more cells than for that to refine the region around the source even though no new waves will emerge from it, giving another example for the problem with the static stress region, discussed in section 4.2.1). Figure 4.18 shows the six refinements that follow after the previous example, which continue to provide the minimal refinement for paths of the P-waves and a broad refinement for the S-waves.

Table 4.6 shows that both dynamically adaptive refinements achieve a lower error than the statically adaptive refinements.

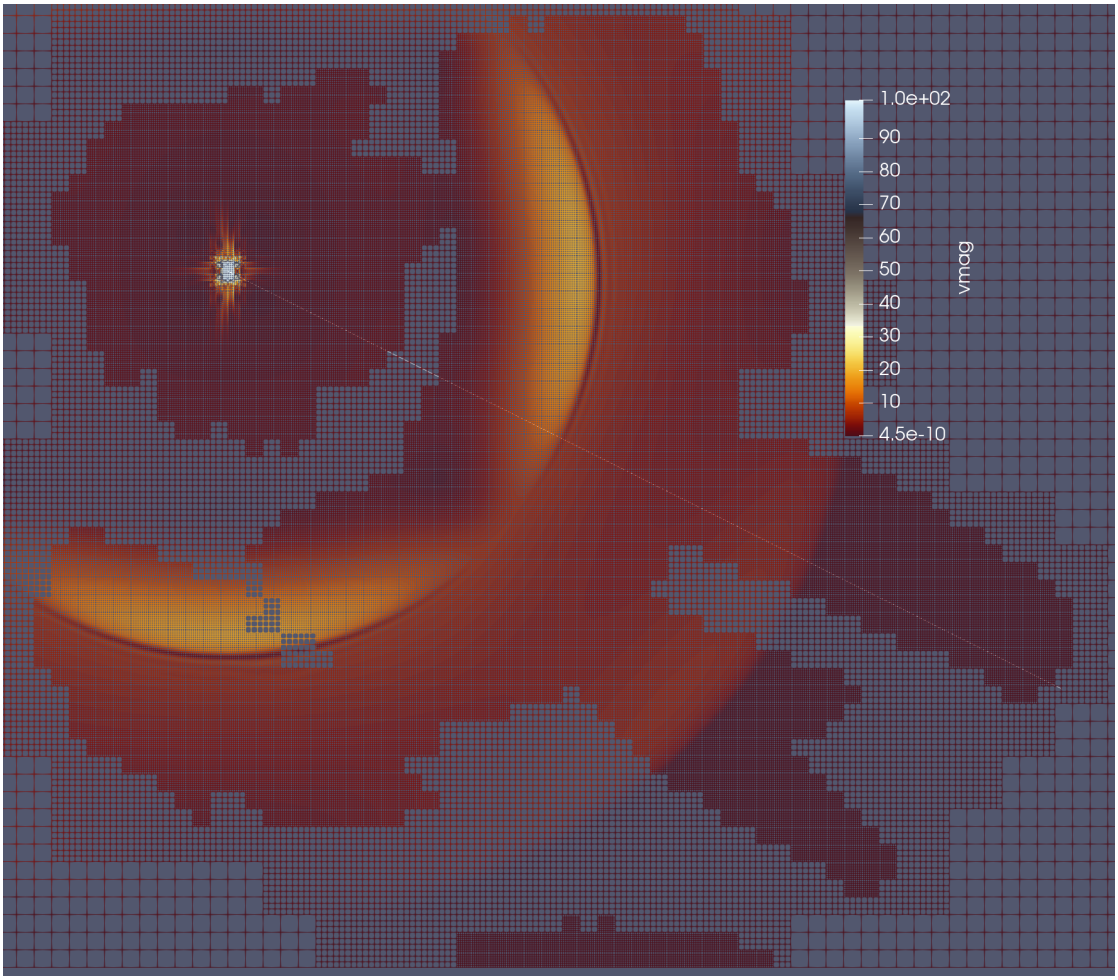


Figure 4.17: Wireframe of the refinement and the velocity magnitude of the dynamic β-adjoint refinement at $t = 0.75s$, directly after switching to the fifth refinement grid. The white diagonal line shows the direct path between the source and the point of interest. Each cell is shown as a 3×3-grid (Exahype does this to interpolate the Legendre-polynomials of a cell to the output). All doubly refined cells and most cells that are refined once, are included in the picture. The edges of the grid are colored according to the local velocity magnitude.

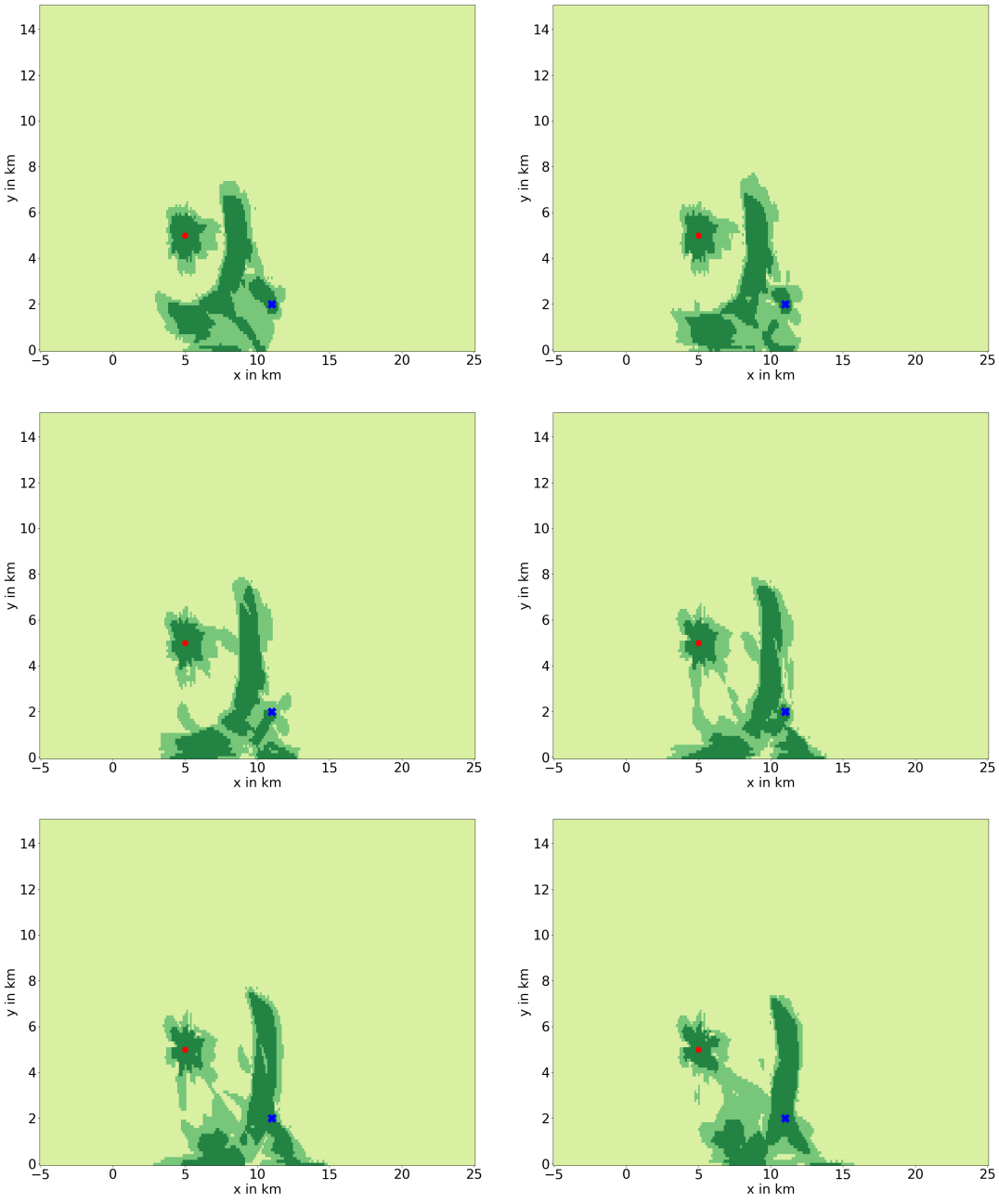


Figure 4.18: Refinement grids six to eleven for the dynamic β-adjoint refinement (from left to right, top to bottom). These grids are active from $t = 0.9s$ to $t = 1.8s$.

Table 4.6: RMS for the point $(11, 2)$ in the Helsinki scenario with $\frac{1}{6}$, $\frac{1}{12}$ cutoff for the static refinement and $\frac{1}{12}$,$\frac{1}{24}$ refinement cutoffs for the dynamically adaptive mesh refinements.

	σ_{11}	σ_{22}	σ_{12}	u	v	q
static α-adjoint refinement	0.0319	0.0354	0.0560	0.0397	0.0420	0.0364
staticβ-adjoint refinement	0.0396	0.0343	0.0489	0.0454	0.0357	0.0387
dynamic α-adjoint refinement	0.0275	0.0322	0.0364	0.0363	0.0196	0.0304
dynamic β-adjoint refinement	0.0265	0.0202	0.0264	0.0328	0.0141	0.0242

5 Conclusion

We have derived the adjoint-guided mesh refinement technique for elastic wave equations in chapter 2. This works perfectly analogously to the derivation for shallow water equations by [DL16]. In chapter 3, we have implemented the adjoint and forward equations in ExaHype, developed a script to create the refinement and a method to read and apply it in a ExaHype simulation. We have tested the adjoint-guided mesh refinement method on three scenarios. The first scenario produces only P-Waves on a uniform material with a free surface boundary, the second scenario consists of two halfspaces with different materials and third one uses a 1D-velocity model for the geology below Helsinki. We have found that the adjoint-guided mesh refinement technique produces refinements that visibly follow the path from the source of the seismic event to the point of interest. We have manually created refinements with an equal number of refined cells to test our method. The adjoint-guided refinements have performed better for reflected waves than the manual refinements and worse for the waves on a direct path. This caused the adjoint-guided method to perform better for the Helsinki and only P-Waves scenario and worse in the two halfspace scenario which does not simulate a surface.

One of the issues is that the region around the source contains a larger magnitude of a static stress that does not move than the stress in the waves, which causes regions near the source to be refined even though they have essentially no impact at the point of interest. Another issue is that the waves are reflecting on refinement boundaries. This might be less of an issue for future projects as most PDE-solvers do grid refinement by halving the cells in each dimensions, instead of splitting them into thirds.

We have compared the method that sets the normal stresses for the initial condition of the adjoint (α-adjoint) and the method that sets the shear stress for the adjoint initial condition (β-adjoint). The afore mentioned problems make it difficult to determine whether the α- or the β-adjoint produces the better refinement.

We also prototyped a dynamically adaptive mesh refinement variant. The preliminary number of tests we ran showed better results than both the statically adaptive mesh refinement and the manually constructed one.

Furthermore, the dynamically adaptive method could be improved by tuning parameters like reducing the time a refinement is used or reducing the overlap between two refinements that are sequential in time. Major reworks could try to normalize S- and P-waves to account for the difference in magnitude in them.

List of Figures

2.1 Grammar of a Peano curve 9
2.2 Grammar of a Peano curve 9

3.1 Flowchart of The Process 10
3.2 Plot of σ_{12} at the source of the earthquake 12
3.3 three to one balancing example 14

4.1 Refinement of the P-wave only scenario 18
4.2 P-Wave plot of u 19
4.3 adjoint initial condition 20
4.4 adjoint initial condition 21
4.5 Two Halfspace setup 22
4.6 Refinement for Two Halfspaces 23
4.7 plot of u at (9,13) for Two Halfspaces 24
4.8 plot of u at (13,18) for Two Halfspaces 25
4.9 Helsinki Model 26
4.10 Refinement for Helsinki 27
4.11 Wavefield of the Helsinki scenario 28
4.12 plot of u at (11,2) for the Helsinki scenario 28
4.13 plot of u at (11,2) for the Helsinki scenario 29
4.14 Wavefield showing stress at point source 30
4.15 Reflection error 31
4.16 Reflection error 32
4.17 Wireframe of of the dynamic β-adjoint 33
4.18 Refinement grids of of the dynamic β-adjoint 34

List of Tables

4.1 RMS for P-Waves 19
4.2 RMS for Two Halfspaces (9,13) 22
4.3 RMS for Halfspaces (13,18) 24
4.4 RMS for Helsinki (11,2) 26
4.5 RMS for Helsinki (11,2) 29
4.6 RMS for Helsinki (11,2) 35

Bibliography

[06] *SISMOWINE*. 2006. URL: http://www.sismowine.org/index.html?page=model&mset=WP.

[Bad13] M. Bader. *Space-Filling Curves - An Introduction with Applications in Scientific Computing*. Texts in Computational Science and Engineering 9. URL: http://link.springer.com/book/10.1007/978-3-642-31046-1. Springer-Verlag, 2013.

[BH22] A. Breuer and A. Heinecke. "Next-Generation Local Time Stepping for the ADER-DG Finite Element Method." In: *arXiv preprint arXiv:2202.10313* (2022).

[DL16] B. N. Davis and R. J. LeVeque. "Adjoint methods for guiding adaptive mesh refinement in tsunami modeling." In: *Global Tsunami Science: Past and Future, Volume I*. Springer, 2016, pp. 4055–4074.

[Dum+14] M. Dumbser, O. Zanotti, R. Loubère, and S. Diot. "A posteriori subcell limiting of the discontinuous Galerkin finite element method for hyperbolic conservation laws." In: *Journal of Computational Physics* 278 (2014), pp. 47–75.

[EH22] ExaHype-Contributors and S. Hingst. *Fork of the ExaHyPE-Engine*. 2022. URL: https://github.com/shingst/ExaHyPE-Engine.

[Hin22a] S. Hingst. *C++ code of this thesis*. 2022. URL: https://github.com/shingst/adjoint.

[Hin22b] S. Hingst. *Python code of this thesis*. 2022. URL: https://github.com/shingst/adjoint-python.

[HK79] T. C. Hanks and H. Kanamori. "A moment magnitude scale." In: *Journal of Geophysical Research: Solid Earth* 84.B5 (1979), pp. 2348–2350.

[HW07] J. S. Hesthaven and T. Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, 2007.

[Kan77] H. Kanamori. "The energy release in great earthquakes." In: *Journal of geophysical research* 82.20 (1977), pp. 2981–2987.

[KIT22] KITWARE. *latest commit for vtkLinearKernel.cxx at time of writing*. 2022. URL: https://github.com/Kitware/VTK/blob/bbe1da823dee1cf94fee6b18813b492861d22368/Filters/Points/vtkLinearKernel.cxx.

[Kre+21] L. Krenz, C. Uphoff, T. Ulrich, A.-A. Gabriel, L. S. Abrahams, E. M. Dunham, and M. Bader. "3D acoustic-elastic coupling with gravity: the dynamics of the 2018 Palu, Sulawesi earthquake and tsunami." In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2021, pp. 1–14.

[Kri+06] M. Kristeková, J. Kristek, P. Moczo, and S. M. Day. "Misfit criteria for quantitative comparison of seismograms." In: *Bulletin of the seismological Society of America* 96.5 (2006), pp. 1836–1850.

[Leo+21] M. Leonhardt, G. Kwiatek, P. Martínez-Garzón, M. Bohnhoff, T. Saarno, P. Heikkinen, and G. Dresen. "Seismicity during and after stimulation of a 6.1 km deep enhanced geothermal system in Helsinki, Finland." In: *Solid Earth* 12.3 (2021), pp. 581–594.

[LeV+02] R. J. LeVeque et al. *Finite volume methods for hyperbolic problems*. Vol. 31. Cambridge university press, 2002.

[Rei+20] A. Reinarz, D. Charrier, M. Bader, L. Bovard, M. Dumbser, K. Duru, F. Fambri, A.-A. Gabriel, J.-M. Gallard, S. K. Koeppel, L. Krenz, L. Rannabauer, L. Rezzolla, P. Samfass, M. Tavelli, and T. Weinzierl. "ExaHyPE: An Engine for Parallel Dynamically Adaptive Simulations of Wave Problems." en. In: *Computer Physics Communications* 254 (Sept. 2020), p. 107251. DOI: 10.1016/j.cpc.2020.107251.

[Rog18] C. Rogers. *cnpy*. 2018. URL: https://github.com/rogersce/cnpy.

[Ulr+19] T. Ulrich, S. Vater, E. H. Madden, J. Behrens, Y. van Dinther, I. Van Zelst, E. J. Fielding, C. Liang, and A.-A. Gabriel. "Coupled, physics-based modeling reveals earthquake displacements are critical to the 2018 Palu, Sulawesi tsunami." In: *Pure and Applied Geophysics* 176.10 (2019), pp. 4069–4109.