A multi-pole perfectly matched layer (PML) absorber for finite-difference time-domain (FDTD) seismic modeling in 2D

Master of Science Thesis

for the degree of Master of Science in Applied Geophysics

by

Eric Robert Eppenga

August 9, 2019

IDEA LEAGUE

JOINT MASTER’S IN APPLIED GEOPHYSICS

Delft University of Technology, The Netherlads

ETH Zürich, Switserland

RWTH Aachen, Germany

Date: August 9, 2019

Supervisors: Prof. Dr. Ir. E.C. Slob

Dr. Ir. A. Giannopoulos

Committee Member: Prof. Dr. Ir. E.C. Slob

Dr. Ir. A. Giannopoulos

Dr. Ir. C. Schmelzbach

Abstract

The multi-pole PML (MPML) is tested on models that simulate seismic waves traveling through the subsurface. Using a recursive integration technique a stretching function consisting of the sum of multiple stretching functions is implemented in the velocity-stress finite difference time domain wave equations. The MPML is implemented in both the rotated staggered grid (RSG) and the Virieux grid. The performance of the MPML is tested on a square model, rectangular model and a rectangular model with a free-surface and compared to other types of PML's implemented in these models. The main result is that the MPML can be implemented in the velocity stress wave equations giving stable results similar to other PML types.

Table of contents

Abstract i

List of figures iv

List of tables vii

1. Introduction 1

2. Basic theory of elastic wave propagation 2

2.1 Types of seismic waves 2

2.2 Seismic waves in the subsurface 4

2.3 The seismic wave propagation equations 4

2.4 The Finite Difference Time Domain Method 5

2.5 The Virieux grid 7

2.6 The rotated staggered grid 7

2.7 Stability 9

3. The perfectly matched layer 9

3.1 The absorbing boundary and the perfectly matched layer 9

3.2 Split field vs unsplit PML's 10

3.3 Different types of PML stretching functions 12

3.4 The multi-pole PML 14

3.5 Attenuation coefficients 18

3.6 Memory space 20

4. Results and discussion 21

4.1 A homogeneous square model 22

4.2 A homogeneous rectangular model 28

4.3 Optimal attenuation coefficients 35

4.4 Computing time 35

4.5 Evanescent waves 36

5. Conclusion 40

References

Appendix 43

Appendix A. Matlab codes 43

Appendix A.1 MPML on Virieux grid 43

Appendix A.2 RIPML on Virieux grid 50

Appendix A.3 ConvPML on Virieux grid 56

Appendix A.4 SOPML on Virieux grid 60

Appendix A.5 MPML on RSG 67

Appendix B. Figures 75

List of Figures

Figure 1: Different types of seismic waves. 3

Figure 2: A schematic view of a wave impinging on the interface between two layers with different elastic properties. 4

Figure 3: Schematic view of the collocated grid and staggered Virieux grid in 1D and the trouble of the collocated grid with small scale fluctuations. 6

Figure 4: A schematic view of the staggered grid (a) and Virieux grid (b) cell. 8

Figure 5: Illustration of the imaginary and real part of our domain and the attenuation happening in both. 11

Figure 6: Values of the attenuation coefficient in different regions of the PML zone. 14

Figure 7: A representation of the square model with its parameters. 22

Figure 8: Seismograms at receiver 3 in the square model for the different PML's. 23

Figure 9: Seismograms at receiver 3 in the square model for the different PML's. 23

Figure 10: Error in dB at receiver 3 at different times for the stdRIPML and the cfsRIPML. 24

Figure 11: Error in dB at receiver 3 at different times for the stdConvPML and the cfsConvPML. 24

Figure 12: Error in dB at receiver 3 at different times for the std-cfs-SOPML and the cfs-cfs-SOPML. 25

Figure 13: Error in dB at receiver 3 at different times for the std-cfs-MPML and the cfs-cfs-MPML. 25

Figure 14: Error in dB at receiver 3 at different times for the std-cfs-MPML and the cfs-cfs-MPML. 26

Figure 15: Total absolute error relative to the maximum amplitude at receiver 3. 27

Figure 16: Total absolute error relative to the maximum amplitude at receiver 3. 27

Figure 17: The rectangular shape model with its receiver and source location. 29

Figure 18: Seismograms at receiver 2 in the rectangular model for the different PML's. 29

Figure 19: Seismograms at receiver 2 in the rectangular model for the different PML's. 30

Figure 20: Error in dB at receiver 2 at different times for the stdRIMPML and the cfsRIMPML. 30

Figure 21: Error in dB at receiver 2 at different times for the stdConvMPML and the cfsConvMPML. 31

Figure 22: Error in dB at receiver 2 at different times for the std-cfs-MPML and the cfs-cfs-MPML. 31

Figure 23: Error in dB at receiver 2 at different times for the std-cfs-SOPML and the cfs-cfs-SOPML.

32

Figure 24: The error in dB at receiver 2 for the square model and the adjusted “square” model. 33

Figure 25: Error in dB at receiver 2 at different times for the std-cfs-MPML and the cfs-cfs-MPML.33

Figure 26: Total absolute error relative to the maximum amplitude at receiver 2. 34

Figure 27: Total absolute error relative to the maximum amplitude at receiver 2. 34

Figure 28: Schematic view of the model for testing how the MPML deals with evanescent waves. 37

Figure 29: Seismogram at the receiver location for the evanescent model for different PML types.37

Figure 30: Seismogram at the receiver location for the evanescent model for different PML types.38

Figure 31: The error at the receiver location in dB for the stdRIPML and the cfsRIPML. 38

Figure 32: The error at the receiver location in dB for the std-cfs-MPML, cfs-cfs-MPML and the std-cfs-SOPML. 39

Figure B.1: Error in dB for two different types of PML’s on the square model at receiver 1. 75

Figure B.2: Error in dB for two different types of PML’s on the square model at receiver 1. 75

Figure B.3: Error in dB for two different types of PML’s on the square model at receiver 1. 76

Figure B.4: Error in dB for two different types of PML’s on the square model at receiver 1. 76

Figure B.5: Total absolute error for different numbers of PML cells at receiver 1 for different PML types on the square model. 77

Figure B.6: Total absolute error for different numbers of PML cells at receiver 1 for different PML types on the square model. 77

Figure B.7: Seismogram at receiver 1 for different PML types on the square model. 78

Figure B.8: Seismogram at receiver 1 for different PML types on the square model. 78

Figure B.9: Error in dB for two different types of PML’s on the square model at receiver 2. 79

Figure B.10: Error in dB for two different types of PML’s on the square model at receiver 2. 79

Figure B.11: Error in dB for two different types of PML’s on the square model at receiver 2. 80

Figure B.12: Error in dB for two different types of PML’s on the square model at receiver 2. 80

Figure B.13: Total absolute error for different number of PML cells at receiver 2 on the square model. 81

Figure B.14: Total absolute error for different number of PML cells at receiver 2 on the square model. 81

Figure B.15: Seismogram for different PML types at receiver 2 on the square model. 82

Figure B.16: Seismogram for different PML types at receiver 2 on the square model. 82

Figure B.17: Error in dB for two different types of PML’s on the rectangular model at receiver 1. 83

Figure B.18: Error in dB for two different types of PML’s on the rectangular model at receiver 1. 83

Figure B.19: Error in dB for two different types of PML's on the rectangular model at receiver 1. 84

Figure B.20: Error in dB for two different types of PML's on the rectangular model at receiver 1. 84

Figure B.21: Total absolute error for different numbers of PML cells at receiver 1 for different PML types on the rectangular model. 85

Figure B.22: Total absolute error for different numbers of PML cells at receiver 1 for different PML types on the rectangular model. 85

Figure B.23: Seismogram at receiver 1 for different PML types on the rectangular model. 86

Figure B.24: Seismogram at receiver 1 for different PML types on the rectangular model. 86

Figure B.25: Error in dB for two different types of PML's on the rectangular model at receiver 3. 87

Figure B.26: Error in dB for two different types of PML's on the rectangular model at receiver 3. 87

Figure B.27: Error in dB for two different types of PML's on the rectangular model at receiver 3. 88

Figure B.28: Error in dB for two different types of PML's on the rectangular model at receiver 3. 88

Figure B.29: Total absolute error for different number of PML cells at receiver 3 on the rectangular model. 89

Figure B.30: Total absolute error for different number of PML cells at receiver 3 on the rectangular model. 89

Figure B.31: Seismogram for different PML types at receiver 3 on the rectangular model. 90

Figure B.32: Seismogram for different PML types at receiver 3 on the rectangular model. 90

List of tables

Table 1: Typical P- and S-wave velocities for different soil types. 2

Table 2: Different maximum attenuation coefficients used for each PML. 35

Table 3 : Comparison of the computation time difference between the cfsRIPML and the ConvPML and the calculation time difference between the cfsRIPML and the cfs-cfs-MPML. 36

Table 4: Comparison of the CPU time difference between the cfsRIPML and the ConvPML and the calculation time difference between the cfsRIPML and the cfs-cfs-MPML. 36

1. Introduction

After having acquired seismic data in the field this data has to be processed in order to get an image of the subsurface. In order to create this image we have to model how seismic waves travel through the subsurface. One of the methods used for this is the finite difference time domain modeling of the acoustic wave equations. The problem with this finite difference method is that unwanted reflections occur at the edge of the computational domain due to the truncation of the computer model. In order to prevent these unwanted reflections the wave has to be absorb at the edge of the computational domain. One way of absorbing the wave is by adding a perfectly matched layer (PML) at the edges of the domain. Inside this layer the wave would be slowly absorbed in order to have no reflections at the end of this perfectly matched layer. Different types of PML's have been proposed and tested, each with its own benefits and drawbacks. Recently Giannopoulos [18] showed that a new type PML, the multi-pole PML (or MPML), could be used as an effective PML in the modeling of electromagnetic waves.

In this thesis we will test if this MPML can also be used in the modeling of seismic waves using the stress-velocity wave equations. If this is indeed the case we will compare the performance of the MPML to other types of PML's.

2. Basic theory of elastic wave propagation

2.1 Types of seismic waves

When a sudden source is applied to the subsurface this results in two main types of waves: body waves and surface waves. This pressure source can be anything from an earthquake to something as small as someone walking on the surface.

The body waves are divided into pressure and shear waves. With pressure waves or P-waves the medium particles move in the same direction in which the wave traverses. For shear waves or S-waves the direction of the medium particles is perpendicular to the direction in which the wave traverses. The velocity of both types of waves is given by:

$$v_p=\sqrt{\frac{\lambda+2\mu}{\rho}}, \tag{1}$$

$$v_s=\sqrt{\frac{\mu}{\rho}}, \tag{2}$$

In which v_p is the P-wave velocity, v_s is the S-wave velocity, ρ is the density of the medium, λ is the bulk fluid incompressibility or Lamé's first parameter and μ is the shear modulus or Lamé's second parameter.

Since both λ and μ are always positive we note that the P-wave velocity is always larger than the S-wave velocity. We also see that S-waves do not propagate through fluids (and gasses) because of their shear strength. The S-wave velocity will become zero since μ will be zero. Typical body wave velocities are shown in Table 1. In Table 1 it can be seen that there is overlap in wave velocity for many different soil types.

Table 1: Typical P- and S-wave velocities for different soil types. Data is taken from [1].

Type of formation	P-wave velocity (m/s)	S-wave velocity (m/s)
Dry sand	400-1200	100-500
Wet sand	1500-2000	400-600
Saturated shales and clays	1100-2500	200-800
Salt	4500-5500	2500-3100
Granite	4500-6000	2500-3300
Coal	2200-2700	1000-1400
Water	1450-1500	-

The other main type of waves are called surface waves. These waves travel along the surface and their amplitude decreases drastically as they go deeper away from the surface. Surface waves are divided into Rayleigh waves, Love waves and Stoneley (or Scholte) waves. Rayleigh waves, also called ground roll,

travel along the interface between the air and the ground (so a gas-solid interface). With Rayleigh waves the particle motion is both perpendicular and parallel to direction in which the wave moves. The wave speed of Rayleigh waves is slower than that of P- and S-waves. Love waves can be seen as S-waves on the surface-air interface. They travel at approximately 90% of the speed of S-waves. Stonely waves move along a solid-solid interface in the subsurface. If a Stonely wave moves along a fluid-solid interface it is called a Scholte wave. As can be expected the speed of surface waves depends on the properties of the soil near interface. Each of the different types of waves is shown in Figure 1.

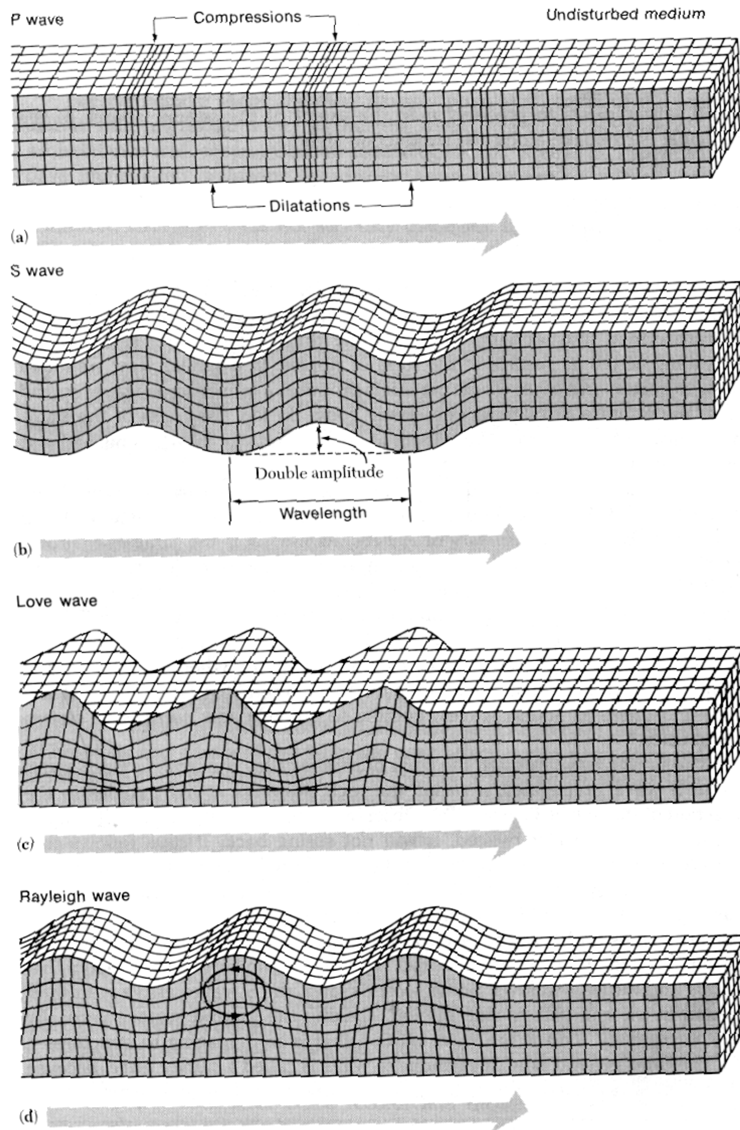


Figure 1: Different types of seismic waves [2]. (a) Shows the particle motion in case of a P-wave with respect to the wave direction indicated with the big arrow. (b) Shows the particle motion in case of a S-wave. (c) Shows the particle motion in case of a Love wave. (d) Shows the particle motion in case of a Rayleigh wave.

2.2 Seismic waves in the subsurface

If the subsurface would be homogeneous a seismic wave would travel away from the source and be slowly attenuation due to the geometrical spreading of the energy. The energy travels away from the sources in a spherical way and will therefore be attenuated at a rate of distance^{-2} in case of an omni-directional source. When however the subsurface is not homogeneous scattering will occur. As the wave hits the interface between the two layers with different elastic properties part of the energy will be reflected back and part of the energy will move through the interface. The angle at which the wave hits the interface on the reflection and transmission of the wave. When the wave hits the interface at an oblique angle, it will be scatter according the Snell's law. No matter if the incident wave was a P-wave or an S-wave, both P- and S- reflected and transmitted waves will occur. If the angle of the transmitted wave is 90 degrees (critical angle) then it will travel along the interface and will act as a source for new wave fronts according to Huygen's Theory of Wavelets. A schematic view waves impinging on the interface between two layers with different elastic properties is shown in Figure 2.

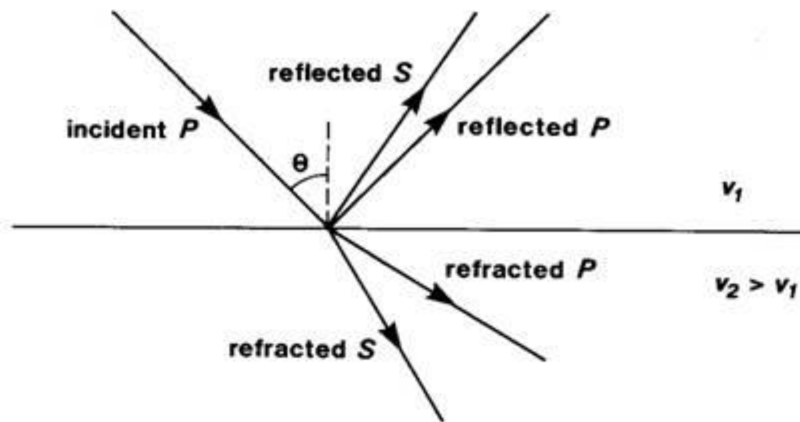


Figure 2: A schematic view of a wave impinging on the interface between two layers with different elastic properties [3]. The velocity in the bottom layer is higher than the velocity in the top layer. θ represents the angle of incident of the incident wave.

2.3 The seismic wave propagation equations

In order to compute how seismic waves travel through the subsurface we have to solve the seismic wave propagation equations. These equations are derived using Newton's second law and Hook's law and the full derivation can be found in [4]. For a two dimensional, linearly elastic and isotropic medium the coupled velocity-stress equations are given by:

$$\rho \frac{\partial v_x}{\partial t} = \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xz}}{\partial z}, \tag{3}$$

$$\rho \frac{\partial v_z}{\partial t} = \frac{\partial \sigma_{xz}}{\partial x} + \frac{\partial \sigma_{zz}}{\partial z} \tag{4}$$

$$\frac{\partial\sigma_{xx}}{\partial t}=(\lambda+2\mu)\frac{\partial v_x}{\partial x}+\lambda\frac{\partial v_z}{\partial z}, \tag{5}$$

$$\frac{\partial\sigma_{zz}}{\partial t}=(\lambda+2\mu)\frac{\partial v_z}{\partial z}+\lambda\frac{\partial v_x}{\partial x}, \tag{6}$$

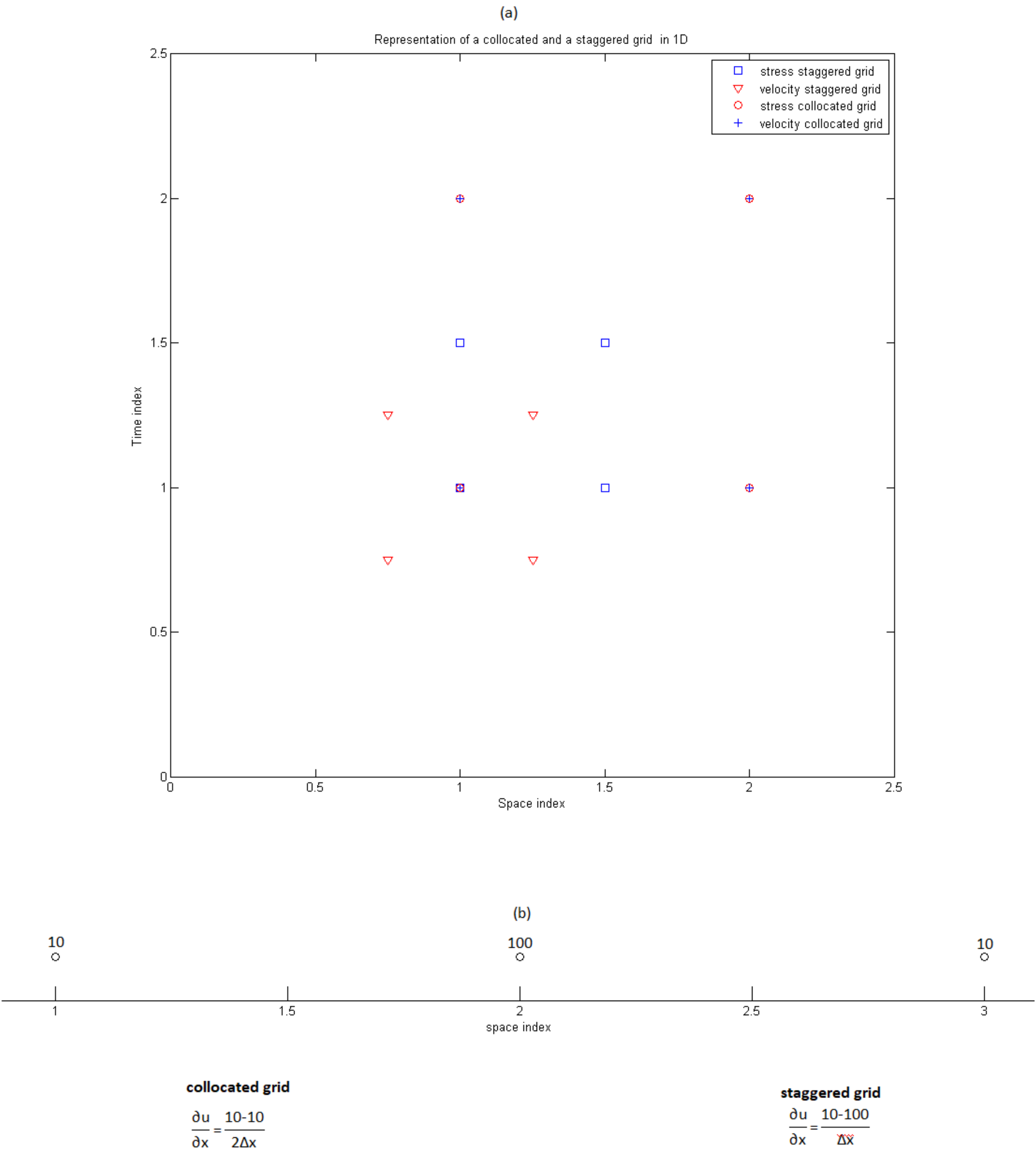
$$\frac{\partial\sigma_{xz}}{\partial t}=\mu\left(\frac{\partial v_x}{\partial z}+\frac{\partial v_z}{\partial x}\right), \tag{7}$$

In these equations v_x and v_z are the velocities in the x- and z-directions respectively, ρ is the density, σ_{xx}, σ_{xz} and σ_{zz} are the stresses in the xx-, xz- and zz-directions respectively and λ and μ are Lamé's first and second parameter respectively.

2.4 The Finite Difference Time Domain method

In order to compute the derivates in these equations finite difference time domain (FDTD) [22] methods are used. Using a forward finite difference methods proved to be not stable so therefore central difference methods are used. In order to apply finite difference the domain in which we are interested has to be discretized. This is done using a staggered grid [23]. The difference between a staggered and non staggered (collocated) grid is that the stress and pressure are both evaluated at grids shifted by half a step size in both the spatial and time domain. The staggered grid and collocated grid are shown in Figure 3a. The advantage over using a staggered grid compared to a collocated grid is that the stress and velocity are not decoupled so no checkerboard pattern will occur when there are large fluctuations between grid points located next to each other. Due to now taking half the step size of the collocated finite difference grid fluctuation between adjacent cells are taken into account. This is shown in Figure 3b. Another advantage of this halved step size in both time and space is that it gives an error 4 times smaller than with a non staggered grid.

In this thesis two different types of staggered grids are used. The first being the Virieux grid [5] an implication of the Yee grid [24] on the acoustic wave equations. The second grid is called the Rotated Staggered grid [6]. These grids were chosen in order to compare the results of this new PML with the results from other PML articles.



$$\frac{\partial u}{\partial x} = \frac{10-10}{2\Delta x}$$

$$\frac{\partial u}{\partial x} = \frac{10-100}{\Delta x}$$

Figure 3: Schematic view of the collocated grid and staggered Virieux grid in 1D and the trouble of the collocated grid with small scale fluctuations. (a) The locations in space and time where the field variables are taken from in the calculation of the derivatives. (b) The derivates in the space direction in 1D calculated for the collocated grid and the staggered grid. It can be seen that the collocated grid misses the small scale fluctuation in field variable u.

2.5 The Virieux grid

As said before with the Virieux grid the stress and pressure are both evaluated at grids shifted by half a step size in both the spatial and time domain. If we want to compute the Virieux grid the discretized stress-strain equations (3-7) become:

$$\frac{\rho_{i,j+\frac{1}{2}}}{\Delta t}\left(v_{x_{i,j+\frac{1}{2}}}^{k+1}-v_{x_{i,j+\frac{1}{2}}}^{k}\right)=\frac{1}{\Delta x}\left(\sigma_{xx_{i+\frac{1}{2},j+\frac{1}{2}}}^{k+\frac{1}{2}}-\sigma_{xx_{i-\frac{1}{2},j+\frac{1}{2}}}^{k+\frac{1}{2}}\right)+\frac{1}{\Delta z}\left(\sigma_{xz_{i,j+1}}^{k+\frac{1}{2}}-\sigma_{xz_{i,j}}^{k+\frac{1}{2}}\right), \tag{8}$$

$$\frac{\rho_{i+\frac{1}{2},j}}{\Delta t}\left(v_{z_{i+\frac{1}{2},j}}^{k+1}-v_{z_{i+\frac{1}{2},j}}^{k}\right)=\frac{1}{\Delta x}\left(\sigma_{xz_{i+1,j}}^{k+\frac{1}{2}}-\sigma_{xz_{i,j}}^{k+\frac{1}{2}}\right)+\frac{1}{\Delta z}\left(\sigma_{zz_{i+\frac{1}{2},j+\frac{1}{2}}}^{k+\frac{1}{2}}-\sigma_{zz_{i+\frac{1}{2},j-\frac{1}{2}}}^{k+\frac{1}{2}}\right), \tag{9}$$

$$\frac{1}{\Delta t}\left(\sigma_{xx_{i+\frac{1}{2},j+\frac{1}{2}}}^{k+\frac{1}{2}}-\sigma_{xx_{i+\frac{1}{2},j+\frac{1}{2}}}^{k-\frac{1}{2}}\right)=\frac{(\lambda+2\mu)_{i+\frac{1}{2},j+\frac{1}{2}}}{\Delta x}\left(v_{x_{i+1,j+\frac{1}{2}}}^{k}-v_{x_{i,j+\frac{1}{2}}}^{k}\right)+\frac{\lambda_{i+\frac{1}{2},j+\frac{1}{2}}}{\Delta z}\left(v_{z_{i+\frac{1}{2},j+1}}^{k}-v_{z_{i+1,j}}^{k}\right), \tag{10}$$

$$\frac{1}{\Delta t}\left(\sigma_{zz_{i+\frac{1}{2},j+\frac{1}{2}}}^{k+\frac{1}{2}}-\sigma_{zz_{i+\frac{1}{2},j+\frac{1}{2}}}^{k-\frac{1}{2}}\right)=\frac{\lambda_{i+\frac{1}{2},j+\frac{1}{2}}}{\Delta x}\left(v_{x_{i+1,j+\frac{1}{2}}}^{k}-v_{x_{i,j+\frac{1}{2}}}^{k}\right)+\frac{(\lambda+2\mu)_{i+\frac{1}{2},j+\frac{1}{2}}}{\Delta z}\left(v_{z_{i+\frac{1}{2},j+1}}^{k}-v_{z_{i+1,j}}^{k}\right), \tag{11}$$

$$\frac{1}{\Delta t}\left(\sigma_{xz_{i,j}}^{k+\frac{1}{2}}-\sigma_{xz_{i,j}}^{k-\frac{1}{2}}\right)=\frac{\mu_{i,j}}{\Delta x}\left(v_{z_{i+\frac{1}{2},j}}^{k}-v_{z_{i-\frac{1}{2},j}}^{k}\right)+\frac{\mu_{i+\frac{1}{2},j+\frac{1}{2}}}{\Delta z}\left(v_{x_{i,j+\frac{1}{2}}}^{k}-v_{x_{i,j-\frac{1}{2}}}^{k}\right), \tag{12}$$

All symbols in these equations are the same as in equations (3-7), Δx, Δz and Δt are the grid step size in the x, z and time direction respectively and i, j are the index of space and k is the index of time.

There are two major drawbacks of a Virieux grid. First of all, it has trouble with modeling heterogeneous media with a large difference in parameters between adjacent cells. In order for the method to stay stable the material parameters have to be averaged or interpolated. For instance with fluid-solid interfaces. Secondly, when computing a medium with a free surface the free surface part of the medium has to be separately processed, again due to the high impedance contrast. The troubles a Virieux can be seen in the schematic view of the grid shown in Figure 4b. One can see that σ_{xz} is at a different location in the cell than σ_{xx} and σ_{zz}. In updating any of these stresses μ is needed. If there is a large difference in μ between the location σ_{xz} and σ_{xx}, σ_{zz} instabilities will occur and the method will become unstable. To overcome these difficulties the rotated staggered grid (RSG) was introduced [6].

2.6 The rotated staggered grid

The rotated staggered grid solves the problem the of the Virieux grid has with small scale heterogeneous media by placing all the elastic constants in the middle of the cell. Therefore averaging or interpolation off the elastic constants is no longer needed. Averaging of the density is still needed however. It can be seen as a rotated grid in the sense that both the diagonal and axial directions of a cell are used in calculating the derivates. A schematic view of the rotated staggered grid next to the Virieux grid is shown in Figure 4. This means that the coordinate and Jacobian transformation for the derivative are:

$$\tilde{z}=\frac{\Delta x}{\Delta r}+\frac{\Delta z}{\Delta r}z, \quad (13)$$

$$\tilde{x}=\frac{\Delta x}{\Delta r}-\frac{\Delta z}{\Delta r}z, \quad (14)$$

$$\frac{\partial}{\partial z}=\frac{\Delta r}{2\Delta z}\left(\frac{\partial}{\partial \tilde{z}}-\frac{\partial}{\partial \tilde{x}}\right), \quad (15)$$

$$\frac{\partial}{\partial x}=\frac{\Delta r}{2\Delta x}\left(\frac{\partial}{\partial \tilde{z}}+\frac{\partial}{\partial \tilde{x}}\right), \quad (16)$$

With $\tilde{x}$ and $\tilde{z}$ being the new spatial direction along which the derivates are evaluated, Δx and Δz being the cell size in the x and z direction and Δr being the length of the diagonal.

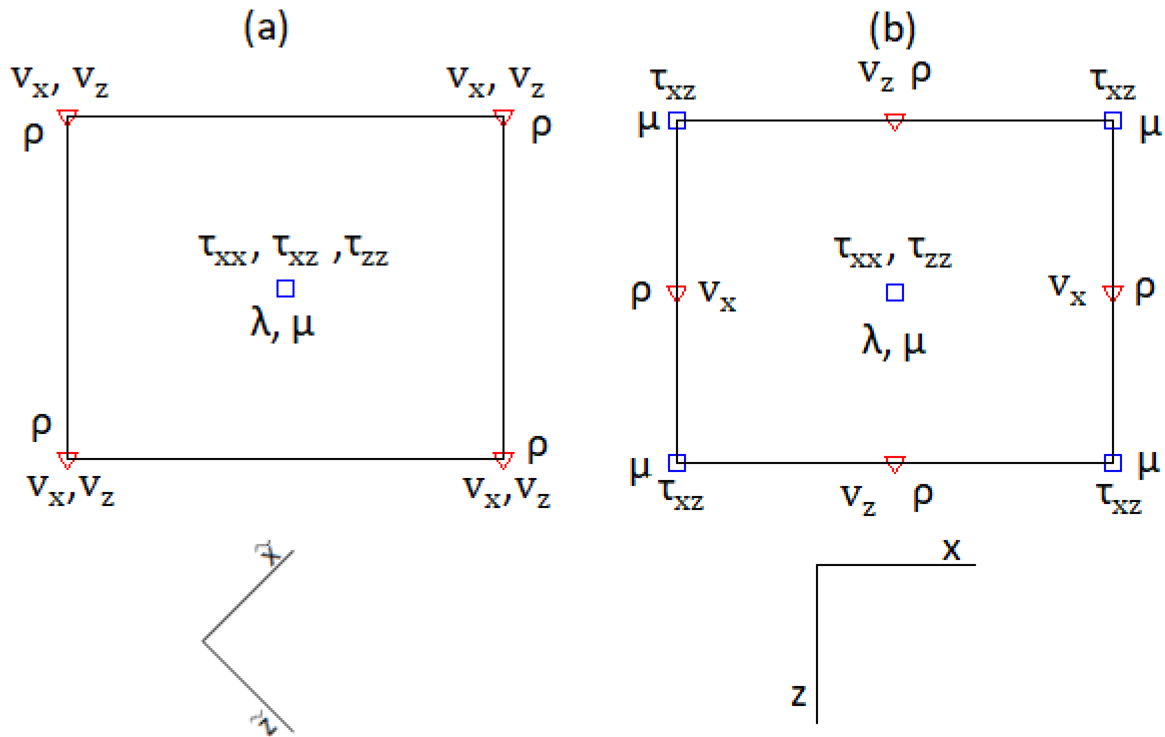


Figure 4: A schematic view of the staggered grid (a) and Virieux grid (b) cell. The symbols in these Figures represent the same field quantities as in equations (3-7), $\tilde{x}$ And $\tilde{z}$ represent the new directions along the diagonal lines along which the derivatives are calculated for the rotated staggered grid.

To show what this means for the spatial derivatives we show the spatial derivative for v_x in the x-direction for a 2D RSG grid in equation 17.

$$\left.\frac{\partial v_x}{\partial x}\right|_{i+\frac{1}{2},j+\frac{1}{2}}=\frac{v_{x_{i+1,j+1}}-v_{x_{i,j}}+v_{x_{i+1,j}}-v_{x_{i,j+1}}}{2\Delta x}, \quad (17)$$

2.7 Stability

Since we are using a finite difference approximation to compute our results we have to take stability criteria into account. There are two stability criteria for both grids we have to look at. The first one being numerical dispersion due to the step size in space. Because we are using staggered grids there should be no error in the amplitude only in the phase. Therefore we want to know how small Δx and Δz have to be for our approximation to be valid. Common sense suggest that our spatial step size should be related to the frequency and the velocity of our wavelet. This has been worked in [25] and it turns out this relation comes down to:

$$\Delta x \text{ (or } \Delta z) < \frac{v_s/f}{10}, \tag{18}$$

With v_s being S-wave velocity and f being the frequency of the wave and Δx and Δz being the step size in the x- or z-direction. This means that at least 10 grid points per wavelength are needed. For higher order finite difference schemes less than 10 grid points can be taken. For instance for 4th order finite difference schemes 5-6 grid points are already enough. For complete derivation of this relation one can look at books or papers like [7]. Since we have both P- and S-waves with different velocities one should always look at the slowest velocity when calculating the wavelength. This means that for the calculation of λ in the medium the S-wave velocity should be taken.

Next we look at what our step size in the time domain has to be. This is done using the Courant–Friedrichs–Lewy condition. It basically means that the time step has to be small enough that the approximation error at one grid point cannot propagate to the neighboring grid point. For a two dimensional case this comes down to:

$$\Delta t \leq \frac{1}{c\sqrt{\Delta x^2+\Delta z^2}}, \tag{19}$$

With Δt being the step size in time, Δx and Δz being the step size in the x- or z-direction and c being the maximum velocity of the wave. This means that in this case the maximum P-wave velocity has to be taken. Complete derivations of this relation can be found in papers and books like [7].

3. Perfectly matched layer

3.1 Absorbing boundary and perfectly matched layer

When simulating how a seismic wave travels through the subsurface there is one major problem with the finite difference approach. As soon as the wave hits the edge of the discretized domain a reflection on this edge will occur due to the truncation of the computer model. In the real medium this wave will continue past the edge of our medium without causing a reflection. To simulate the reality one could extend the computational domain far beyond the domain of interest to make sure no reflection from the edges reaches the domain of interest in the measured time window. This is however computationally very expensive. Therefore absorbing boundaries were created who would absorb the

wave at the edge in order to prevent reflections. The idea behind the absorbing boundaries was calculating what the field variables of an outgoing wave would be at the boundary. Therefore when solving wave equations using the FDTD method the computer would think the wave would continue in the outwards direction after reaching the boundary [26]. Many different types of boundaries were thought of like optimal boundary conditions, one wave equations and damping zones. However most of them experienced troubles when the wave would hit them at different angles. Having all frequencies of the wave absorbed proved to be difficult. In 1994 Bérenger found a solution to this problem [8]. He changed concept from an absorbing boundary to an absorbing layer. Now there was a small absorbing layer attached to the domain in which the wave would be absorbed. In this layer the wave would be attenuated and decayed exponentially. The only problem with this approach was that due to the contrast between the absorbing layer and the real medium there would be a reflection at the interface between the "real" domain and the absorbing layer. But Bérenger showed that there could be created a special sort of absorbing medium were there would be no reflection at the interface. This layer was called the perfectly matched layer, or PML in short. Only the orthogonal components of the wave would be attenuated in the PML region in order to create this absorbing layer.

3.2 Split field vs unsplit PML's

The first few PML's ever implemented in seismic were split-field PML's. This meant that the wave solutions were split into two artificial field components whose sum would be the original field component. For instance if we would take the pressure P this pressure would be split in P_x and P_z with the relation $P = P_x+P_z$. To arrive at the final solution these two fields had to be added together at the end of a simulation. This way it was possible to create an non-physical anisotropic medium inside the PML with the correct phase velocity and stress to attenuate the wave.

Un-split PML's were thought to be unpractical due to the computational costs they would require. In order to solve for the wave equation in the PML region a convolution had to be calculated and this seemed to be time consuming. However it was shown that it was computationally possible to do these convolution using different methods [9-13]. These methods included rewriting the convolution into integrals and solving this integrals using the Trapezoidal integration rule [10], solving the convolutions using recursive integration [12] or using the auxiliary differential equation technique proposed by [27].

The only problem with these early PML's was the need of having different wave equations in the "real" domain and in the PML zone. A more elegant way of deriving both the split field and un-split PML was found using complex coordinate stretching by Chew and Liu in 1996 [14]. The idea behind this was the stretching of the PML zone in the complex plane. This means that we change our spatial variable from a real to a complex variable:

$$\tilde{x}(x)=x+if(x), \tag{20}$$

With $\tilde{x}$ being the complex variable and x being the real variable and f(x) being some function deforming the contour along the imaginary axis. This function f(x) will be responsible for the attenuation in the PML region.

Now it was possible to create our non-physical anisotropic medium in the imaginary part of the complex plane leaving the real part unchanged. In other words: there is an analytic continuation of our wave equation in the complex plane. The waves will now be attenuated in the imaginary part of our plane. This is shown in Figure 5.

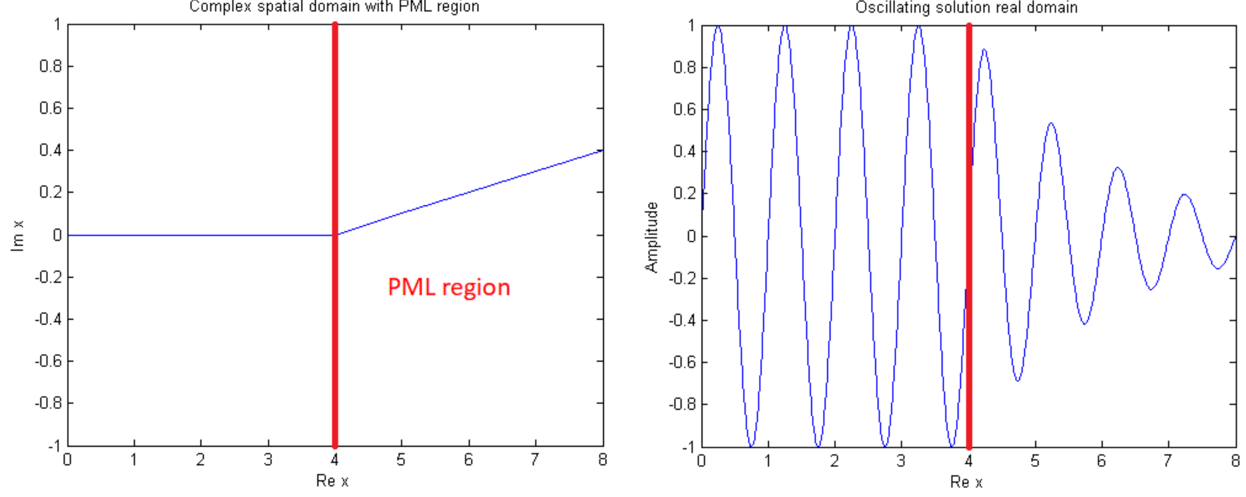


Figure 5: Illustration of the imaginary and real part of our domain and the attenuation happening in both.

The only difficulty is that solving differential equations along contours in the complex rather difficult is. This can be solved by performing a coordinate transformation back to the real domain:

Since, $\partial\tilde{x}=(1+i\frac{df}{dx})\partial x$ we get the coordinate transformation:

$$\frac{\partial}{\partial x} \rightarrow \frac{1}{1+i\frac{df}{dx}}\frac{\partial}{\partial \tilde{x}}, \tag{21}$$

If we now make function f(x) in a way that $\frac{df}{dx} > 0$ in the PML region and $\frac{df}{dx} = 0$ in the real domain we can write one wave equation that is valid in both regions.

This means that equations (3-7) can now be written in the frequency domain as:

$$i\omega\rho\tilde{v}_x = \frac{1}{s_x}\frac{\partial\tilde{\sigma}_{xx}}{\partial x} + \frac{1}{s_z}\frac{\partial\tilde{\sigma}_{xz}}{\partial z}, \tag{22}$$

$$i\omega\rho\tilde{v}_z = \frac{1}{s_x}\frac{\partial\tilde{\sigma}_{xz}}{\partial x} + \frac{1}{s_z}\frac{\partial\tilde{\sigma}_{zz}}{\partial z}, \tag{23}$$

$$i\omega\rho\tilde{\sigma}_{xx} = (\lambda+2\mu)\frac{1}{s_x}\frac{\partial\tilde{v}_x}{\partial x} + \lambda\frac{1}{s_z}\frac{\partial\tilde{v}_z}{\partial z}, \tag{24}$$

$$i\omega\rho\tilde{\sigma}_{zz} = \lambda\frac{1}{s_x}\frac{\partial\tilde{v}_x}{\partial x} + (\lambda+2\mu)\frac{1}{s_x}\frac{\partial\tilde{v}_z}{\partial z}, \tag{25}$$

$$i\omega\rho\tilde{\sigma}_{xz} = \mu\left(\frac{1}{s_x}\frac{\partial\tilde{v}_z}{\partial x} + \frac{1}{s_z}\frac{\partial\tilde{v}_x}{\partial z}\right), \tag{26}$$

Where s_x is $(1+i\frac{df}{dx})$, s_z is $(1+i\frac{df}{dx})$, ω is the circular frequency and i is $\sqrt{-1}$. The tilde on top of all the other variables is to show the we are now in the frequency domain but the variables themselves still represent the same field variable.

3.3 Different types of PML stretching functions

Chew and Lui [14] showed that a good choice for s turned out to be:

$$s_j=1+\frac{d_j}{1+i\omega}, \tag{27}$$

With the subscript j spatial direction and d_j being some function that determines the attenuation and $d_j \geq 0$. The division by ω is to make sure the attenuation rate in the PML is independent of frequency. We will call this the standard PML stretching function (std).

The only problem with this type of PML stretching function was that they had trouble with waves hitting the PML region at near grazing angles. This would happen for instance when the source was located close to the PML. Also elongated domains where near grazing angles are more common turned out the be difficult to compute correctly. Furthermore evanescent waves were still strongly being reflected at the PML boundary and the same held for low frequency waves. Therefore an improved PML function was created by Kuzuoglu and Mittra in 1996 [15] called the complex frequency shifted stretching function (cfs). The CFS PML stretching function was written as:

$$s_j=\kappa_j+\frac{d_j}{\alpha_j+i\omega}, \tag{28}$$

With κ_j and α_j being other attenuation functions and $\alpha_j \geq 0$ and $\kappa_j \geq 1$. It should be noted that if we take $\kappa_j = 1$ and $\alpha_j = 0$ we return to std. The α_j part of the cfs would deal with very low frequencies making sure the we would never divide by numbers close to zero. The κ_j part dealt with the waves impinging at near grazing angles.

The next step in the development of PML's was the higher order PML with the idea of enhancing the performance of the PML. The idea behind this was simply multiply different PML's with each other in order the reap the benefits from both. For instance the second order PML (SOPML) of a cfs with a std:

$$s_{so_j}=\left(\kappa_{1_j}+\frac{d_{1_j}}{\alpha_{1_j}+i\omega}\right)\left(\kappa_{2_j}+\frac{d_{2_j}}{\alpha_{2_j}+i\omega}\right), \tag{29}$$

With κ_{1_j}, d_{1_j}, α_{1_j}, κ_{2_j}, d_{2_j}, and α_{2_j} all being different attenuation functions. This could be extended to Nth order PML generalized as:

$$s_{No_j}=\prod_{m=1}^{N} k_{m_j}+\frac{d_{m_j}}{a_{m_j}+i\omega}, \tag{30}$$

Where each m now stands for a different attenuation function.

However when we take a closer look at for instance the second order PML we see that we have some undesired terms in our final PML. The imaginary part of a stretching function is most important in the attenuation in the PML zone. When looking at the imaginary parts of equations (26-28) we get:

$$\Im\left(std_j\right)=\frac{d_j}{\omega}, \tag{31}$$

$$\Im\left(cfs_j\right)=\frac{d_j\omega}{{\alpha_j}^2+\omega^2}, \tag{32}$$

$$\Im\left(s_{so_j}\right)=\frac{\kappa_{2_j}d_{1_j}}{\omega}+\frac{\kappa_{1_j}\omega d_{2_j}}{{\alpha_{2_j}}^2+\omega^2}+\frac{\alpha_{2_j}d_{1_j}d_{2_j}}{\omega({\alpha_{2_j}}^2+\omega^2)}, \tag{33}$$

In equation 32 we can now see that we get an extra terms due to the multiplication of two stretching functions. This will make the picking of stable attenuation functions a lot harder and may give undesirable effects when multiplying two PML stretching functions. Furthermore if we look at the real part of this second order stretching function:

$$\Re\left(s_{so_j}\right)=\kappa_{2_j}+\frac{d_{2_j}\alpha_{2_j}}{{\alpha_{2_j}}^2+\omega^2}-\frac{d_{1_j}d_{2_j}}{{\alpha_{2_j}}^2+\omega^2}, \tag{34}$$

If the real part of the stretching function get below 1 instabilities in our simulation will occur and our solution will blow up again making the picking of stable attenuation functions more difficult. With even higher order stretching function we get even more terms and this process becomes even more difficult.

However if we would simply add two different stretching functions to one another the imaginary part would look like. Again just as in the second order stretching function we use a cfs with a std:

$$\Im(s)=\frac{d_{1_j}}{\omega}+\frac{\omega d_{2_j}}{{\alpha_{2_j}}^2+\omega^2}, \tag{35}$$

This looks more like the idea behind the higher order stretching function of combining the benefits of two different stretching function to get an improved one. Also the picking of correct attenuation functions seems to be a lot easier. This new type of stretching function will be called a multi-pole stretching function resulting in a multi-pole PML or MPML.

In order to attenuate only the orthogonal components of the wave the attenuation functions in a PML will look like the ones shown in Figure 6.

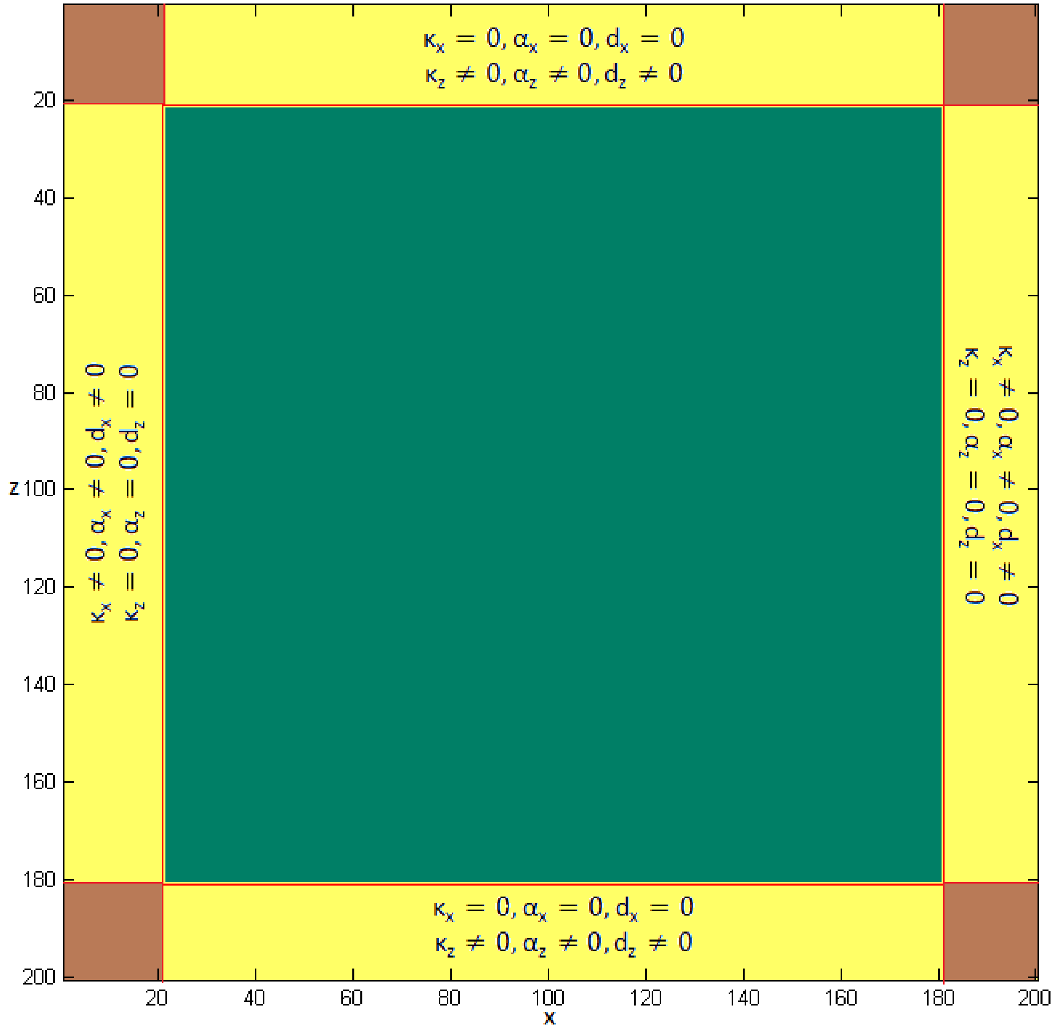


Figure 6: Values of the attenuation coefficient in different regions of the PML zone. The PML region is shown in yellow and the "real" domain in green. In the corner regions, indicated in brown, none of the attenuation functions is equal to zero.

3.4 The multi-pole PML

In general the multi-pole stretching function can be written as:

$$s_{mp}=\kappa_j+\sum_{m=1}^{N}\frac{d_{m_j}}{\alpha_{m_j}+i\omega}, \tag{36}$$

Where we have to remember that κ_j is now a function that is a sum of N different κ_j functions. The result however can be written as a single κ_j function.

The implementation of this PML in our wave equations and the calculation of the convolutions can be done using a similar approach as in [16]. and [17] using a recursive integration technique. Our solution follow closely the work of A. Giannopoulos in [18] but now done for seismic wave equations instead of

the electromagnetic wave equations. Where in the electromagnetic case we would have the electric field (E) and the magnetic field (H) we now have the velocity field (V) and the stress field (Σ). The E-, H and V field have two components in 2D while the Σ-field has three components. In electromagnetic we only have only transverse (shear) waves while with acoustics we have both S- (shear or transverse) and P-waves. This means that in the acoustic case we have to attenuate an extra different type of wave compared to the electromagnetic case. So in comparison to the electromagnetic wave equations, the acoustic wave equations have one extra field variable in which our attenuation function has to work.

We start with a variable transform in equations (21-25) of:

$$\psi_j = \frac{1-s_j}{s_j} = \frac{1}{s_j}\left(\frac{1-s_j}{1}\right) = \frac{1}{s_j} - 1, \tag{37}$$

This means we can now write our equations as:

$$i\omega\rho\tilde{v}_x = (1+\psi_x)\frac{\partial\tilde{\sigma}_{xx}}{\partial x} + (1+\psi_z)\frac{\partial\tilde{\sigma}_{xz}}{\partial z}, \tag{38}$$

$$i\omega\rho\tilde{v}_z = (1+\psi_x)\frac{\partial\tilde{\sigma}_{xz}}{\partial x} + (1+\psi_z)\frac{\partial\tilde{\sigma}_{zz}}{\partial z}, \tag{39}$$

$$i\omega\rho\tilde{\sigma}_{xx} = (\lambda+2\mu)(1+\psi_x)\frac{\partial\tilde{v}_x}{\partial x} + \lambda(1+\psi_z)\frac{\partial\tilde{v}_z}{\partial z}, \tag{40}$$

$$i\omega\rho\tilde{\sigma}_{zz} = \lambda(1+\psi_x)\frac{\partial\tilde{v}_x}{\partial x} + (\lambda+2\mu)(1+\psi_z)\frac{\partial\tilde{v}_z}{\partial z}, \tag{41}$$

$$i\omega\rho\tilde{\sigma}_{xz} = \mu\left((1+\psi_x)\frac{\partial\tilde{v}_z}{\partial x} + (1+\psi_z)\frac{\partial\tilde{v}_x}{\partial z}\right), \tag{42}$$

If we now substitute:

$$\tilde{J}_{ju} = \psi_j\frac{\partial\tilde{\sigma}_{ju}}{\partial u}, \tag{43}$$

and

$$\tilde{M}_{ju} = \psi_j\frac{\partial\tilde{v}_{ju}}{\partial u}, \tag{44}$$

With j and u representing spatial directions x and z. We can finally write:

$$i\omega\rho\tilde{v}_x = \frac{\partial\tilde{\sigma}_{xx}}{\partial x} + \frac{\partial\tilde{\sigma}_{xz}}{\partial z} + \tilde{J}_{xx} + \tilde{J}_{xz}, \tag{45}$$

$$i\omega\rho\tilde{v}_z = \frac{\partial\tilde{\sigma}_{xz}}{\partial x} + \frac{\partial\tilde{\sigma}_{zz}}{\partial z} + \tilde{J}_{xz} + \tilde{J}_{zz}, \tag{46}$$

$$i\omega\rho\tilde{\sigma}_{xx} = (\lambda+2\mu)\frac{\partial\tilde{v}_x}{\partial x} + \lambda\frac{\partial\tilde{v}_z}{\partial z} + (\lambda+2\mu)\tilde{M}_{xx} + \lambda\tilde{M}_{xz}, \tag{47}$$

$$i\omega\rho\tilde{\sigma}_{zz} = \lambda\frac{\partial\tilde{v}_x}{\partial x} + (\lambda+2\mu)\frac{\partial\tilde{v}_z}{\partial z} + \lambda\tilde{M}_{zx} + (\lambda+2\mu)\tilde{M}_{zz}, \tag{48}$$

$$i\omega\rho\tilde{\sigma}_{xz} = \mu\left(\frac{\partial\tilde{v}_z}{\partial x} + \frac{\partial\tilde{v}_x}{\partial z}\right) + \mu(\tilde{M}_{zx} + \tilde{M}_{xz}), \tag{49}$$

We can now see that the attenuation is a simple correction to our original wave equation. The advantage of this being that the PML correction can simply be added to any existing code without the need for rewriting the original code. We are still looking at the frequency domain so for time domain data we would need the time domain equations. This can be done using recursive integration. We will do this for one of the equations but the process if similar for the other ones.

We start by substituting $\frac{1-s_j}{s_j}$ for ψ_j in equation 42 and get:

$$s_j\left(\tilde{J}_{ju}+\frac{\partial\tilde{\sigma}_{ju}}{\partial u}\right)=\frac{\partial\tilde{\sigma}_{ju}}{\partial u}, \tag{50}$$

If now substitute or multi-pole stretching function into equation 49:

$$\left(\kappa_j+\sum_{m=1}^{N}\frac{d_{m_j}}{\alpha_{m_j}+i\omega}\right)\left(\tilde{J}_{ju}+\frac{\partial\tilde{\sigma}_{ju}}{\partial u}\right)=\frac{\partial\tilde{\sigma}_{ju}}{\partial u}, \tag{51}$$

Next we create a set of functions for every pole of the multi-pole called $\tilde{\Lambda}_{uj_m}$ as:

$$\widetilde{\Lambda}_{uj_m}=\left(\frac{d_{m_j}}{\alpha_{m_j}+i\omega}\right)\left(\tilde{J}_{ju}+\frac{\partial\tilde{\sigma}_{ju}}{\partial u}\right), \tag{52}$$

Now we can write $\tilde{J}_{ju}$ as:

$$\tilde{J}_{ju}=\frac{1-\kappa_j}{\kappa_j}\frac{\partial\tilde{\sigma}_{ju}}{\partial u}-\frac{1}{\kappa_j}\sum_{m=1}^{N}\widetilde{\Lambda}_{uj_m}, \tag{53}$$

This may seem hard to solve since $\widetilde{\Lambda}_{uj_m}$ itself has the term $\tilde{J}_{ju}$ but using recursive integration it is possible to solve this equation. This will be shown below. We can write equation 51 as:

$$\alpha_{m_j}\widetilde{\Lambda}_{uj_m}+i\omega\widetilde{\Lambda}_{uj_m}=d_{m_j}\tilde{J}_{ju}+d_{m_j}\frac{\partial\tilde{\sigma}_{ju}}{\partial u}, \tag{54}$$

If we now divide by $i\omega$ we can write equation 53 as:

$$\widetilde{\Lambda}_{uj_m}=\frac{1}{i\omega}\left(d_{m_j}\tilde{J}_{ju}+d_{m_j}\frac{\partial\tilde{\sigma}_{ju}}{\partial u}-\alpha_{m_j}\widetilde{\Lambda}_{uj_m}\right), \tag{55}$$

Next we transform equation 54 into the time domain to get:

$$\Lambda_{uj_m}=\int_0^{\tau}\left(d_{m_j}J_{ju}+d_{m_j}\frac{\partial\sigma_{ju}}{\partial u}-\alpha_{m_j}\Lambda_{uj_m}\right)d\tau, \tag{56}$$

Notice the tilde on top of Λ missing since we are now looking at the time domain. Since $\tilde{J}_{ju}$ is evaluated at the same time instance as the velocities (which is at half time steps due to our staggered grids) and $\widetilde{M}_{ju}$ is evaluated at the same time instance as the stresses and we assume that both are zero at time $t \leq 0$ we can apply the extended trapezoidal rule to equation 55 to get:

$$\Lambda_{uj_m}^{k+\frac{1}{2}}=\sum_{p=0}^{n-1}\left[d_{m_j}\Delta t J_{uj}^{p+\frac{1}{2}}+d_{m_j}\Delta t\frac{\partial\sigma_{uj}^{p+\frac{1}{2}}}{\partial u}-\alpha_{m_j}\Delta t\Lambda_{uj_m}^{p+\frac{1}{2}}\right]+\frac{d_{m_j}\Delta t}{2}J_{uj}^{k+\frac{1}{2}}+\frac{d_{m_j}\Delta t}{2}\frac{\partial\sigma_{uj}^{k+\frac{1}{2}}}{\partial u}-\frac{\alpha_{m_j}\Delta t}{2}\Lambda_{uj_m}^{k+\frac{1}{2}}, \tag{57}$$

Which we can rewrite as:

$$\left(1+\frac{\alpha_{m_j}\Delta t}{2}\right)\Lambda_{uj_m}^{k+\frac{1}{2}}=\frac{d_{m_j}\Delta t}{2}J_{uj}^{k+\frac{1}{2}}+\frac{d_{m_j}\Delta t}{2}\frac{\partial\sigma_{uj}^{k+\frac{1}{2}}}{\partial u}+\Phi_{uj_m}^{k-\frac{1}{2}}, \tag{58}$$

With $\Phi_{uj_m}^{k-\frac{1}{2}}$ being a memory variable being:

$$\Phi_{uj_m}^{k-\frac{1}{2}}=\sum_{p=0}^{n-1}\left[d_{m_j}\Delta t J_{uj}^{p+\frac{1}{2}}+d_{m_j}\Delta t\frac{\partial\sigma_{uj}^{k+\frac{1}{2}}}{\partial u}-\alpha_{m_j}\Delta t\Lambda_{uj_m}^{p+\frac{1}{2}}\right], \tag{59}$$

This means that we can now compute $\Lambda_{uj_m}^{k+\frac{1}{2}}$ using:

$$\Lambda_{uj_m}^{k+\frac{1}{2}}=\frac{d_{m_j}\Delta t}{2+\alpha_{m_j}\Delta t}J_{uj}^{k+\frac{1}{2}}+\frac{d_{m_j}\Delta t}{2+\alpha_{m_j}\Delta t}\frac{\partial\sigma_{uj}^{k+\frac{1}{2}}}{\partial u}+\frac{2}{2+\alpha_{m_j}\Delta t}\Phi_{uj_m}^{k-\frac{1}{2}}, \tag{60}$$

Using this relation in equation the time domain equivalent 52 gives:

$$J_{uj}^{k+\frac{1}{2}}=\left(\frac{1}{\kappa_j+\sum_{m=1}^{N}\frac{d_{m_j}\Delta t}{2+\alpha_{m_j}\Delta t}}-1\right)\frac{\partial\sigma_{uj}^{k+\frac{1}{2}}}{\partial u}-\left(\frac{1}{\kappa_j+\sum_{m=1}^{N}\frac{d_{m_j}\Delta t}{2+\alpha_{m_j}\Delta t}}\right)\sum_{m=1}^{N}\frac{2\Phi_{uj_m}^{k-\frac{1}{2}}}{2+\alpha_{m_j}\Delta t}, \tag{61}$$

The update of the memory variable $\Phi_{uj_m}^{k+\frac{1}{2}}$ at each time step can be written as:

$$\Phi_{uj_m}^{k+\frac{1}{2}}=\Phi_{uj_m}^{k-\frac{1}{2}}+d_{m_j}\Delta t J_{uj}^{k+\frac{1}{2}}+d_{m_j}\Delta t\frac{\partial\sigma_{uj}^{k+\frac{1}{2}}}{\partial u}-\alpha_{m_j}\Delta t\Lambda_{uj_m}^{k+\frac{1}{2}}, \tag{62}$$

To make it easier we write $\Phi_{uj_m}^{k+\frac{1}{2}}$ as:

$$\Phi_{uj_m}^{k+\frac{1}{2}}=\frac{2-\alpha_{m_j}\Delta t}{2+\alpha_{m_j}\Delta t}\Phi_{uj_m}^{k-\frac{1}{2}}+\left(\frac{2d_{m_j}\Delta t}{2+\alpha_{m_j}\Delta t}\right)\times\left(\frac{1}{\kappa_j+\sum_{m=1}^{N}\frac{d_{m_j}\Delta t}{2+\alpha_{m_{j_l}}\Delta t}}\right)\frac{\partial\sigma_{uj}^{k+\frac{1}{2}}}{\partial u}-\left(\frac{2d_{m_j}\Delta t}{2+\alpha_{m_j}\Delta t}\right)\times\left(\frac{1}{\kappa_j+\sum_{m=1}^{N}\frac{d_{m_j}\Delta t}{2+\alpha_{m_j}\Delta t}}\right)\sum_{m=1}^{N}\frac{2\Phi_{uj_m}^{k-\frac{1}{2}}}{2+\alpha_{m_j}\Delta t}, \tag{63}$$

Since this is a very long equation we introduce the variables:

$$RA_j=\kappa_j+\sum_{m=1}^{N}\frac{d_{m_j}\Delta t}{2+\alpha_{m_j}\Delta t}, \tag{64}$$

$$RB_{m_j}=\frac{2}{2+\alpha_{m_j}\Delta t}, \tag{65}$$

$$RE_{m_j}=\frac{2-\alpha_{m_j}\Delta t}{2+\alpha_{m_j}\Delta t}, \tag{66}$$

$$RF_{m_j}=\frac{2d_{m_j}\Delta t}{2+\alpha_{m_j}\Delta t}, \tag{67}$$

This makes us able to write equations 60 and 62 in the compact form:

$$J_{uj}^{k+\frac{1}{2}}=\left(\frac{1}{RA_j}-1\right)\frac{\partial\sigma_{uj}^{k+\frac{1}{2}}}{\partial u}-\frac{1}{RA_j}\sum_{m=1}^{N} RB_{m_j}\,\Phi_{uj_m}^{k-\frac{1}{2}}, \tag{68}$$

$$\Phi_{uj_m}^{k+\frac{1}{2}}=RE_{m_j}\Phi_{uj_m}^{k-\frac{1}{2}}+\frac{RF_{m_j}}{RA_j}\left(\frac{\partial\sigma_{uj}^{k+\frac{1}{2}}}{\partial u}-\sum_{m=1}^{N} RB_{m_j}\,\Phi_{uj_m}^{k-\frac{1}{2}}\right), \tag{69}$$

As one might had already seen in the equations, each pole in the multi-pole stretching function needs its own memory variable. In our final equations 67 and 68 we see that it is no longer necessary to calculate $\Lambda_{uj_m}^{k+\frac{1}{2}}$ explicitly which save memory space.

The process of computing the MPML is now as follows in the PML zones. First $J_{uj}^{k+\frac{1}{2}}$ correction term is calculated at each time step after which the velocity field is updated using this PML correction term. Next the updated memory variable $\Phi_{uj_m}^{k+\frac{1}{2}}$ is calculated. After this is done the $\widetilde{M}_{ju}$ correction term is calculated after which the stress field is updated. Again the updated memory variable for the stress field is calculated after which the whole process is repeated for the next time step.

In order to test how this new PML performs against other PML's the convolution PML, second order PML and standard PML were also computed on the same data. For the convolution PML the convolution is solved using the recursive convolution technique from Roden and Gedney [11]. The second order PML uses a very similar recursive integration approach as the one used for the MPML. The standard PML can be created by using a single pole and setting $\kappa_j = 1$ and $\alpha_j = 0$ in our equations. The complete derivation of the convolution PML and the second order PML can be found in the papers [12] and [19] respectively.

3.5 Attenuation functions

Finding useful attenuation functions can be a very tricky and time consuming process since there is not a real "rule-of-thumb" for picking these functions. However from previous research we can get an idea of the best pick for these attenuation functions. It was found that a gradual increase of the attenuation was needed in order to prevent reflections at the interface between the PML region and "real" domain. In 2001 Collino and Toska [20] found that a polynomial increase in attenuation worked very well.

The first attenuation function we look at is d_j. Collino and Toska found that a good attenuation function for d_j was:

$$d_j = d_{j_0}\left(\frac{x}{L}\right)^2, \tag{70}$$

With d_{j_0} being the maximum value of d_j, x being the distance from the interface of the PML and the "real" domain and L being the thickness of the PML. In their paper Collino & Tsogka also found that a good value for d_{j_0} could be calculated with:

$$d_{j_0} = -\frac{3v_p}{2L}\ln(R), \tag{71}$$

With v_p being the P-wave velocity and R being the expected reflection coefficient for a normal incident P-wave. R can be calculated with:

$$\log_{10}(R) = -\frac{\log_{10}(npml)-1}{\log_{10}(2)} - 3, \tag{72}$$

With npml being the number of PML cells attached to the interface. As this might be theoretically a good value for d_{j_0} in some cases different values for d_{j_0} might give better results but in most cases this will give the best result.

The optimal functions for κ_j and α_j are harder to find since the model itself will have a strong influence on the best functions. However from previous research we found that for κ_j the function:

$$\kappa_j = 1 + (\kappa_{j_0} - 1)\left(\frac{x}{L}\right)^2, \tag{73}$$

From the research of Zhang and Shen 2010 [21] it was found that a good value for κ_{j_0} is given by:

$$\kappa_{j_0} = 2\frac{PPW_{fc}^S}{PPW_{FD}}, \tag{74}$$

With PPW_{FD} being the minimum number of discrete points per wavelength for the numerical scheme to be stable and PPW_{fc}^S being given by:

$$PPW_{fc}^S = \frac{v_s}{\Delta h f_c} \tag{75}$$

With v_s the S-wave velocity, Δh the grid spacing and f_c the central frequency of the wave. However this relation was only valid for a narrowband wavelet.

A good choice for the last attenuation function α_j was found to be:

$$\alpha_j = \alpha_{j_0}\left(1 - \left(\frac{x}{L}\right)\right), \tag{76}$$

With a good value for α_{j_0} being πf_c. The reverse scaling of α_{j_0} is due to fact that singularity problems will occur at the boundary if α_j is not large there.

For the second order PML the attenuation functions for both poles were kept the same as in the formulas above. In order to maintain stability the following criteria have to be fulfilled with the second order PML: $\alpha_{1_j}\alpha_{2_j}>\omega^2$ and $\kappa_{2_j}\geq 1$.

However as mentioned before these are optimal values found for particular cases and these will not be optimal in every setting. During this thesis it was found that actually a smaller maximum attenuation coefficient than the recommended gave better results. Especially at longer measuring times where the solution would sometimes exponentially increase if the equations for calculating the maximum value attenuation coefficient shown in this chapter above were used. Therefore the maximum values were all rescaled by dividing by a certain constant. The optimal division constant was found through trial and error. For instance for the standard PML the optimal value for d_{j_0} was found to be $d_{j_0}/1.5$ if we used equation 71 for the calculation of d_{j_0}.

3.6 Memory space

An important concept in the computation of seismic waves is the memory space and computing time needed to perform the simulations. In general more memory space is needed for a PML than for other absorbing boundaries. However, the performance of the PML is much better compared to the others and as soon as our "real" domain becomes very large in comparison to our PML zone this extra memory becomes negligible. However for smaller domains the memory space needed for PML's is still an important factor and therefore a comparison for the memory space needed for the different types of PML's is discussed in this chapter. Memory requirement reduce with increasing code efficiency.

There are different ways of programming the different types of PML's. For instance one can create vectors to store the attenuation functions where the attenuation coefficient is only calculated in the PML zones and therefore the vector would have a length of 1xnpml. It is also possible to create a vector over the entire domain length containing the attenuation function in the PML region and making sure there is no attenuation in the "real" domain. This can be done by setting the value of κ_{j_0} to 1, d_{j_0} to zero and α_{j_0} to zero. Or one can even create whole matrices over the entire domain ("real" + PML) with values of the attenuation function at each position. This last option would require the most memory space of course. At certain positions in the PML where the attenuation function would be zero storage wouldn't even be needed.

Since we are not looking at split field PML's we do not go into much detail about the advantages in memory space when compared to the un-split PML's. One can however easily see that in 1D the memory space needed is already halved by the fact that it is now only necessary to compute one field variable for every two before.

When comparing a double pole MPML (or second order PML) to a single pole PML we see that we now have double the amount of memory needed since we now have twice the amount of attenuation functions. For smaller domains this may be a significant influence on the overall storage space needed.

For larger domains however this becomes almost negligible since this storage is only needed for the PML regions which are very small in comparison to the entire domain.

4. Results & Discussion

Three different types of models are tested with different PML's applied to each of them. A homogeneous square model was tested, a homogeneous rectangular model was tested and a model was created to test how well the PML could handle evanescent waves. For each of the models a reference model was made by extended the domain far enough that no reflections would reach the receivers in the measured time period. The error at each receiver was then calculated using the following two formula's:

$$\text{Error}_{db}\big|_{u,j}^{k} = 20\log_{10}\frac{\left\|E_{PML}\big|_{u,j}^{k} - E_{ref}\big|_{u,j}^{k}\right\|}{\left\|E_{ref_{max}}\right\|}, \tag{77}$$

$$\text{Error}_{u,j} = \sum_{k=1}^{nk}\sqrt{\frac{\left(E_{PML}\big|_{u,j}^{k} - E_{ref}\big|_{u,j}^{k}\right)^2}{\left|E_{ref_{max}}\right|}}, \tag{78}$$

With $E_{PML}|_{u,j}^{k}$, the value of the field variable in the model with PML, $E_{ref}|_{u,j}^{k}$ the value of the field variable in the reference model, $E_{ref_{max}}$ the maximum absolute amplitude in the reference model, $\text{Error}_{db}|_{u,j}^{k}$ local error in dB and $\text{Error}_{u,j}$ the total local error scaled to the maximum amplitude of the wave. The subscripts u and j denote the spatial location, the superscript k denotes the time instance and nk is number of time instances.

One can note that by taking the $20\log_{10}$ of the $\text{Error}_{u,j}$ one gets the total local error in decibel.

The following different PML types were tested and compared to one another for different numbers of PML cells:

- The standard and complex frequency shifted stretching function with the recursive integration technique from chapter 2.10, from now in known as stdRIPML and cfsRIPML respectively
- The standard and complex frequency shifted stretching function with the convolution technique from [12], from now in known as stdConvPML and cfsConvPML respectively
- Two second order PML's with the recursive integration technique from [19], one a combination of a standard PML and a complex frequency shifted stretching function and the other a combination of two complex frequency stretching function from now on known as std-cfs-SOPML and cfs-cfs-SOPML respectively.
- Two double pole multi-pole PML's with the recursive integration technique from chapter 2.10, one a combination of a standard PML and a complex frequency shifted stretching function and the other a combination of two complex frequency stretching function from now on known as std-cfs-MPML and cfs-cfs-MPML respectively.

The codes used in the testing of the different types of PML's are shown in Appendix A.

4.1 A homogeneous square model

The first model to be tested was a homogeneous square model on a Virieux grid. This is the easiest model to make and implement and it has been shown that PML work really well on this type of model. The model parameters are shown in Figure 7. A Ricker wavelet with a central frequency of 10 Hz was used. The source was located at position (135,135) in cell numbers. Three receivers were placed named receiver 1 at location (135,140), receiver 2 at location (135,250) and receiver 3 at location (185,185). To test the performance of the PML the number of PML cells was changed between 5 and 20 cells.

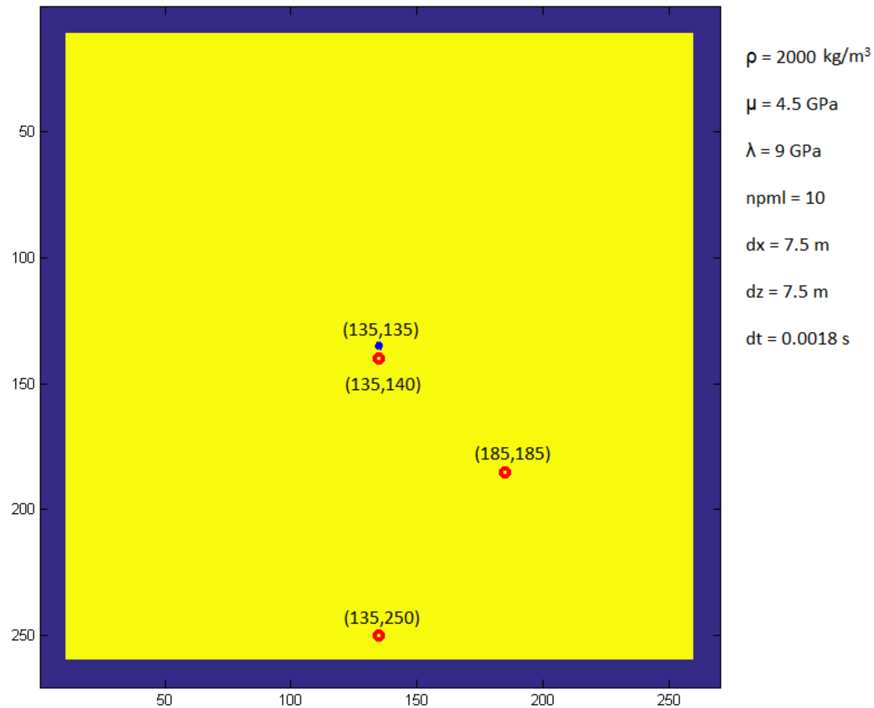


Figure 7: A representation of the square model with its parameters.

With the coordinates being given in cell numbers. The resulting vertical component of particle velocity recorded at receiver 3 for the different PML's is shown in Figures 8 and 9.

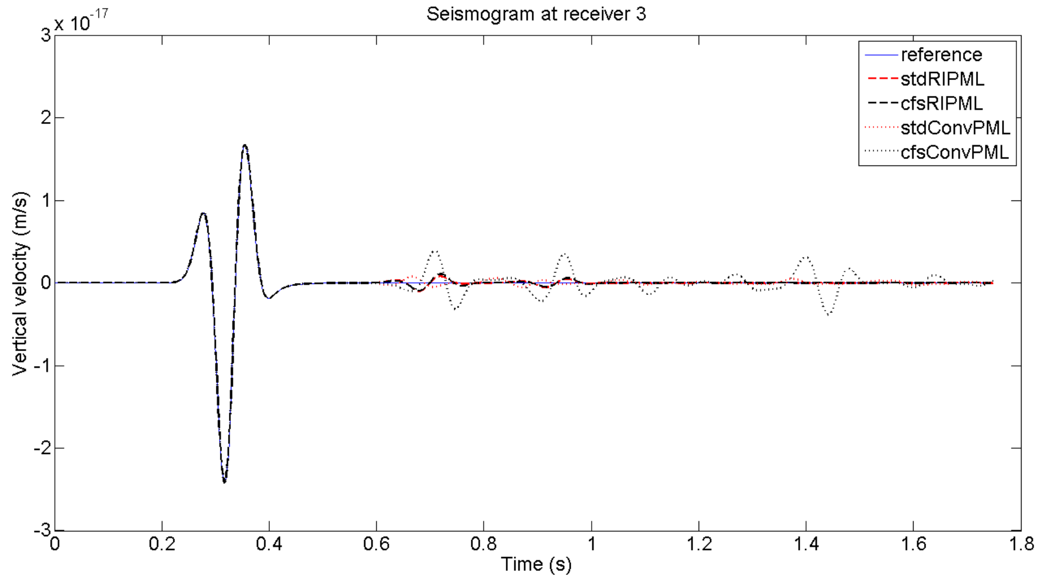


Figure 8: Seismograms at receiver 3 in the square model for the different PML's.

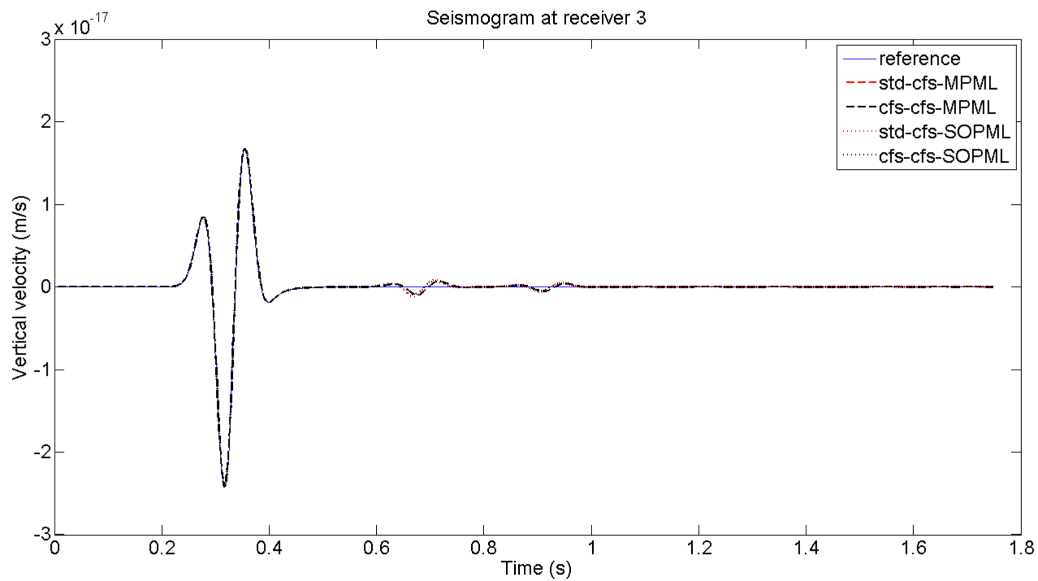


Figure 9: Seismograms at receiver 3 in the square model for the different PML's.

In Figure 8 we see that all PML's seem to be working approximately equally well with the exception of the cfsConvPML. However, on the naked eye the stdRIPML seems to give the best result. The reflections after 1 second show us that ConvPML's produce some undesired artifact. This means that either our code for the ConvPML's had a mistake in it or this shifting in the position of the later arriving waves is due to the different maximum attenuation coefficient used in each PML which means some waves are slowed down more than others.

A quick look at Figure 9 tells us that indeed the MPML and SOPML perform better than the first order PML's but the difference is minimal. The reflections after 1 second measuring time that were visible in the ConvPML's are now not visible anymore. On first glance both the SOPML and the MPML seem to perform equally well.

The error in dB at receiver 3 for each PML calculated with equation 76 is shown in the Figures 10-13. We limit our error axis to -120dB since no machine will measure the signal below -120dB.

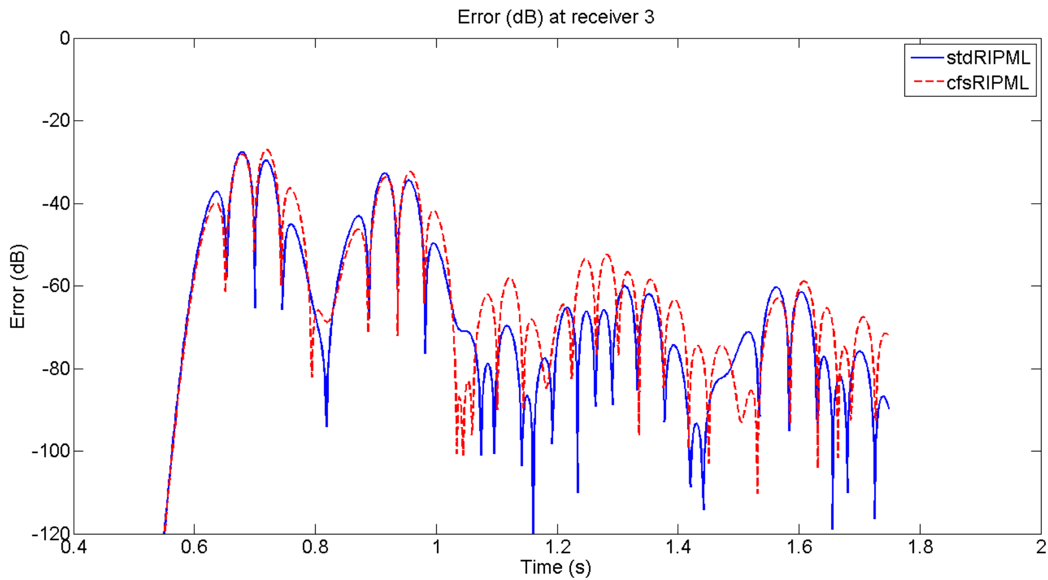


Figure 10: Error in dB at receiver 3 at different times for the stdRIPML and the cfsRIPML.

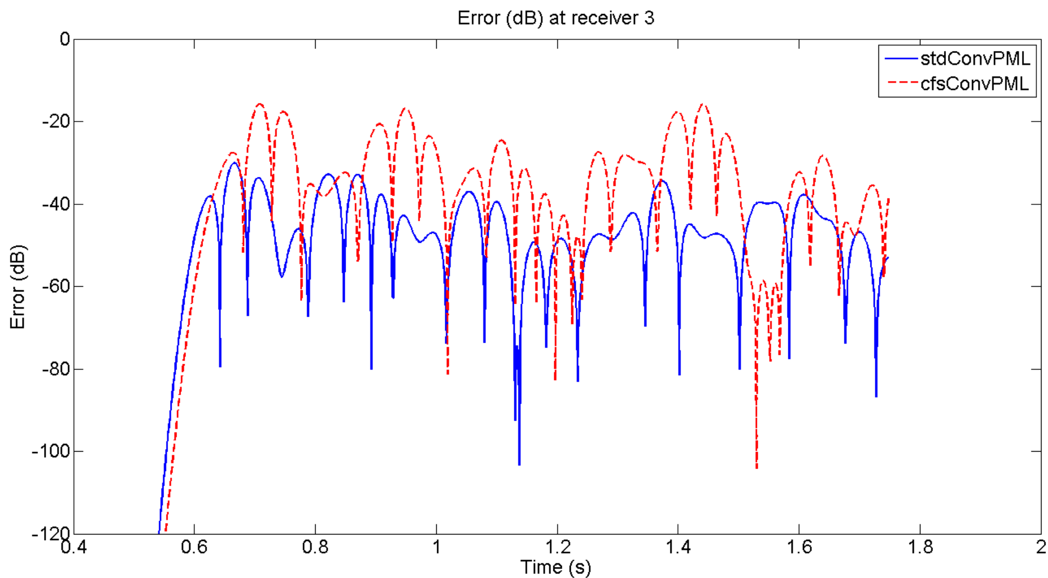


Figure 11: Error in dB at receiver 3 at different times for the stdConvPML and the cfsConvPML.

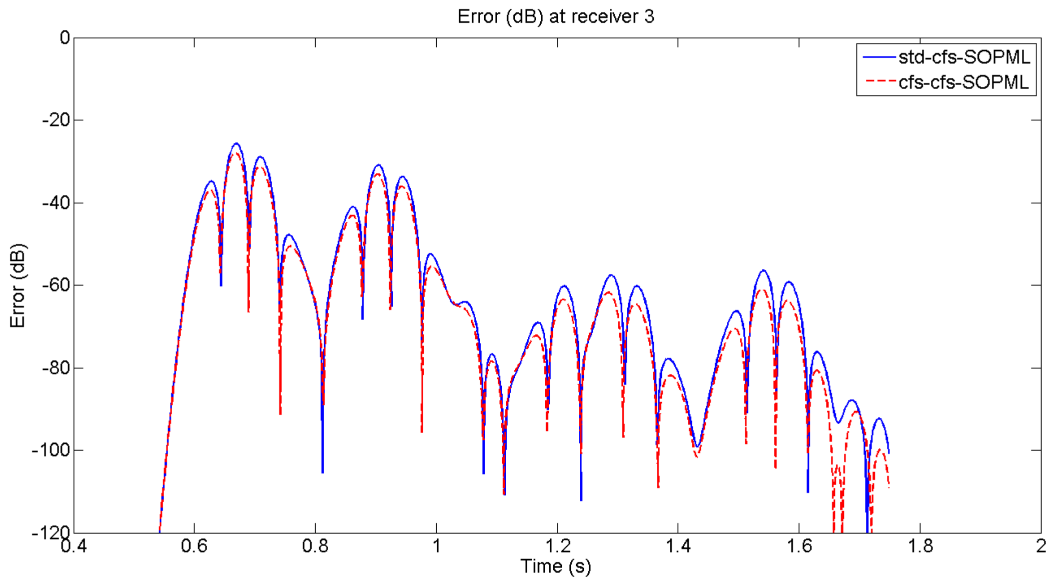


Figure 12: Error in dB at receiver 3 at different times for the std-cfs-SOPML and the cfs-cfs-SOPML.

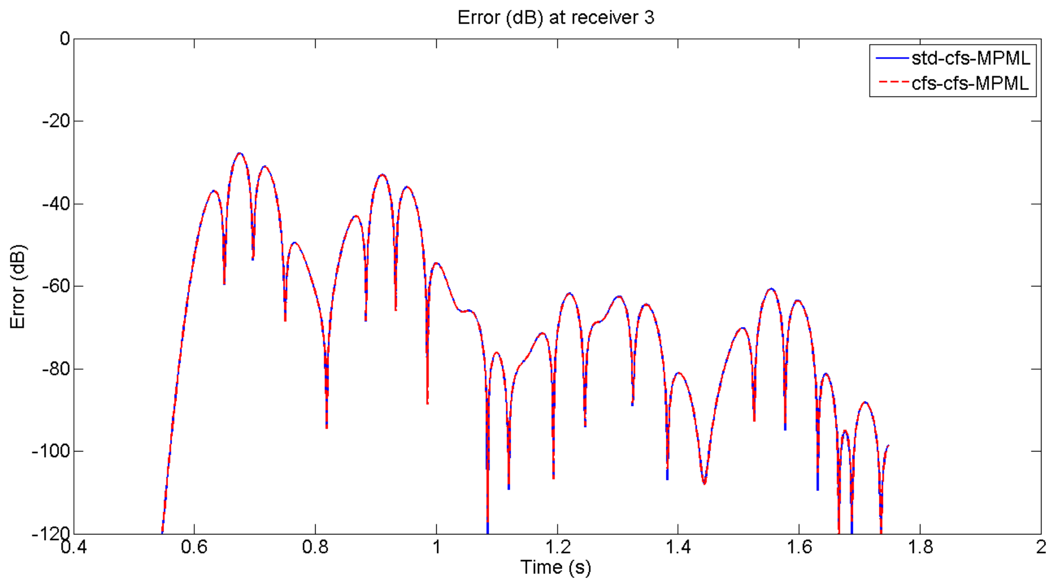


Figure 13: Error in dB at receiver 3 at different times for the std-cfs-MPML and the cfs-cfs-MPML.

From Figures 10 to 13 we observe that all the PML's perform approximately equally well. The ConvPML's have a slightly bigger error that averages just above the -50dB level while the others average just below the -50dB level. We can see that both the std-cfs-MPML and the cfs-cfs-MPML perform almost equally well. We do see a general decrease in the mean error at later times showing that if we would measure for a very long time this error would again go towards $-\infty$ (no difference between the

reference and PML seismogram). A zoomed in comparison for the for the cfsRIPML, the cfs-cfs-SOPML and the cfs-cfs-MPML is shown in Figure 14.

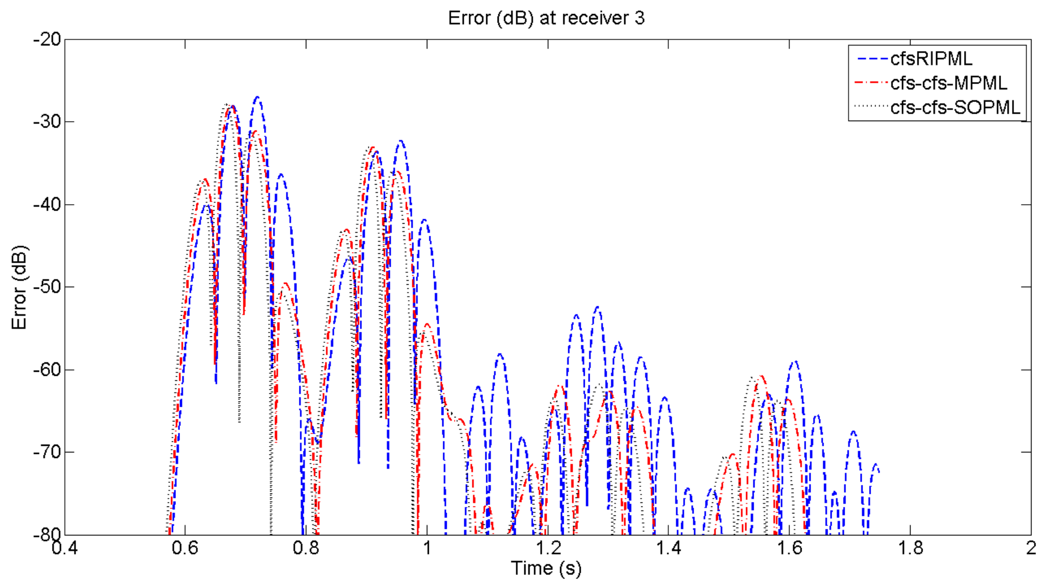


Figure 14: Error in dB at receiver 3 at different times for the std-cfs-MPML and the cfs-cfs-MPML.

In Figure 14 we see that the cfs-cfs-MPML and the cfs-cfs-SOPML perform better than the cfsRIPML with the differences ranging between a 1-10dB. It should be noted that an error reduction of -6dB means a factor 2 better performance. We see that especially in the 1 to 1.4 seconds time window the error is reduced drastically.

The results of the number of PML cells on total absolute error calculated with equation 78 are shown in Figures 15 and 16.

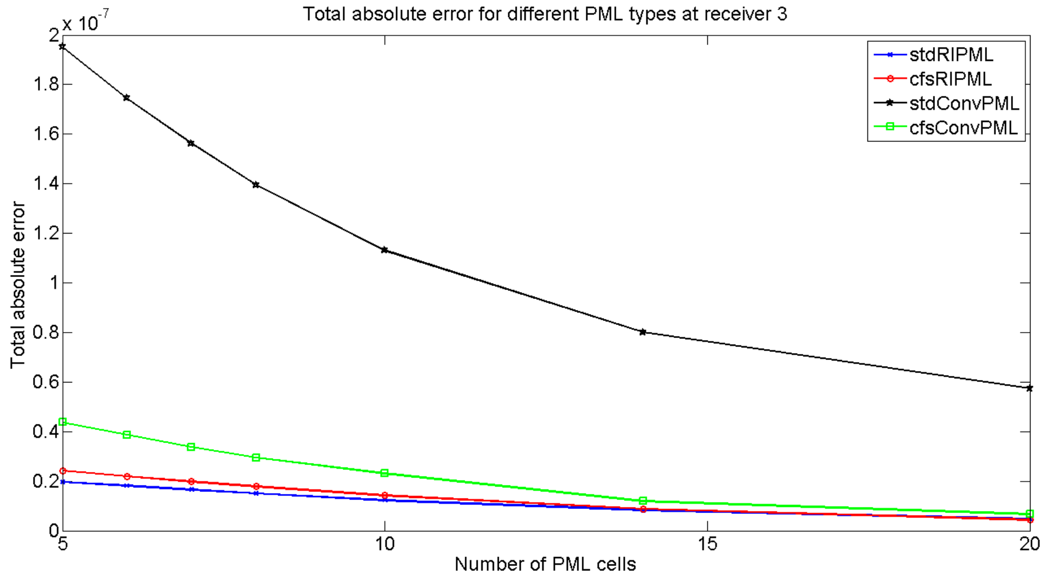


Figure 15: Total absolute error relative to the maximum amplitude at receiver 3.

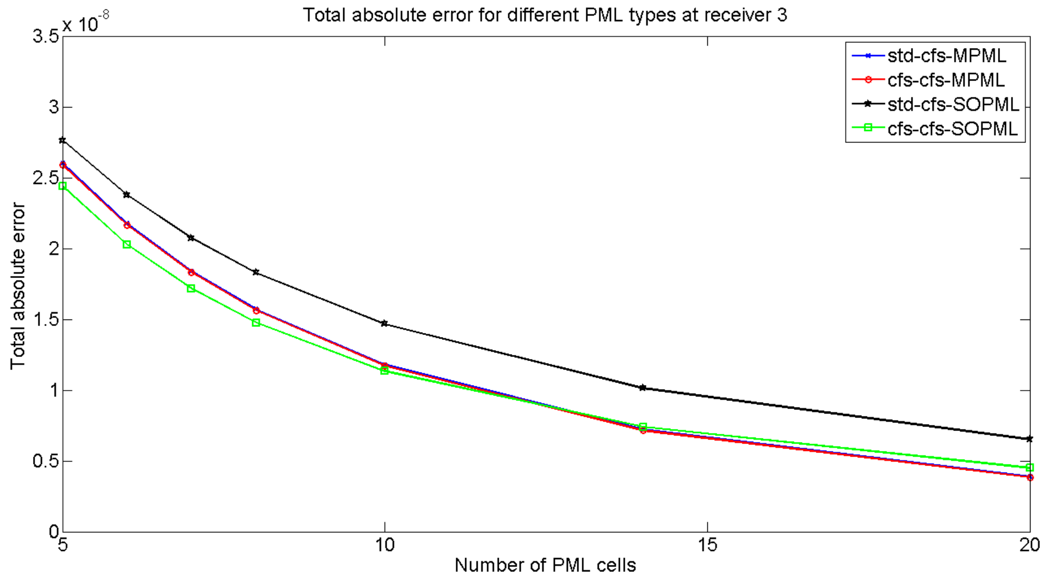


Figure 16: Total absolute error relative to the maximum amplitude at receiver 3.

From Figures 15 and 16 we note that indeed the performance of the PML increases with an increasing number of cells. They all seem to reach a certain asymptote which means that for each PML there is a number of PML cells after which the attenuation does not improve anymore by adding more cells. Again it can be seen that the ConvPML performs worse overall. By taking the $20\log_{10}$ of these values we get total absolute error in dB. For the std-cfs-MPML this meant that by increasing the number of PML cells from 5 to 20 the total error decreases by 16 dB. By increasing the number PML cells by a factor 4 our

total error has decreased by a factor of 5.33. It shows that the increase of number of PML cell is indeed worthwhile if one wants to get an optimal result.

The other receiver locations gave similar results, and the results are shown in Appendix B. It may seem that the results from receiver 1 are in all cases far better than the results from receiver 2 and 3. However this difference is due to the position of the receivers. Receiver 1 is located very close to the source on a straight line in the z-direction. This means that there will be no overlap between reflections from the PML and the “original wave”. Receiver 2 is located close to the PML boundary and in a straight vertical line from the source location relative to the outer boundary. Now we see that the error manifest itself already in the tail of the outgoing wave making the overall error larger due to overlap with the first reflections from the PML. An important difference between receiver 3 and receivers 1 and 2 is that receiver 3 is located in a diagonal line from the source instead of a vertical line. This means that reflections from both the boundary parallel to the x-direction and the boundary parallel to the z-direction arrive at the same time at receiver 3. For receiver 1 and 2 there is a difference in arrival times of reflections from the boundary parallel to the x-direction and the boundary parallel to the z-direction. Due to interference of this two waves arriving at the same time the error can become larger.

4.2 A homogeneous rectangular model

In previous research it was found that PML’s had trouble with rectangular shaped domains. In an rectangular domain waves that impinge the interface at very low angles can cause strong reflections. This due to the directionality of the attenuation functions in the PML. As was shown in Figure 6 only the orthogonal components of the wave are absorbed in the PML. If a wave impinges on the PML at a near-grazing angle that wave is travelling almost parallel to the PML. One can see that this would create difficulties for the PML. Therefore we also tested a rectangular model on a Virieux grid. This model is shown in Figure 17. A Ricker wavelet with a central frequency of 10 Hz was used as the source time signature. The number of PML cells was varied between 5 and 20 cells. The source was located at position (135,35) in cell numbers. Receiver 1 was located at position (145,35), receiver 2 at position (155,35) and receiver 3 at position (145,45) as indicated in Figure 16.

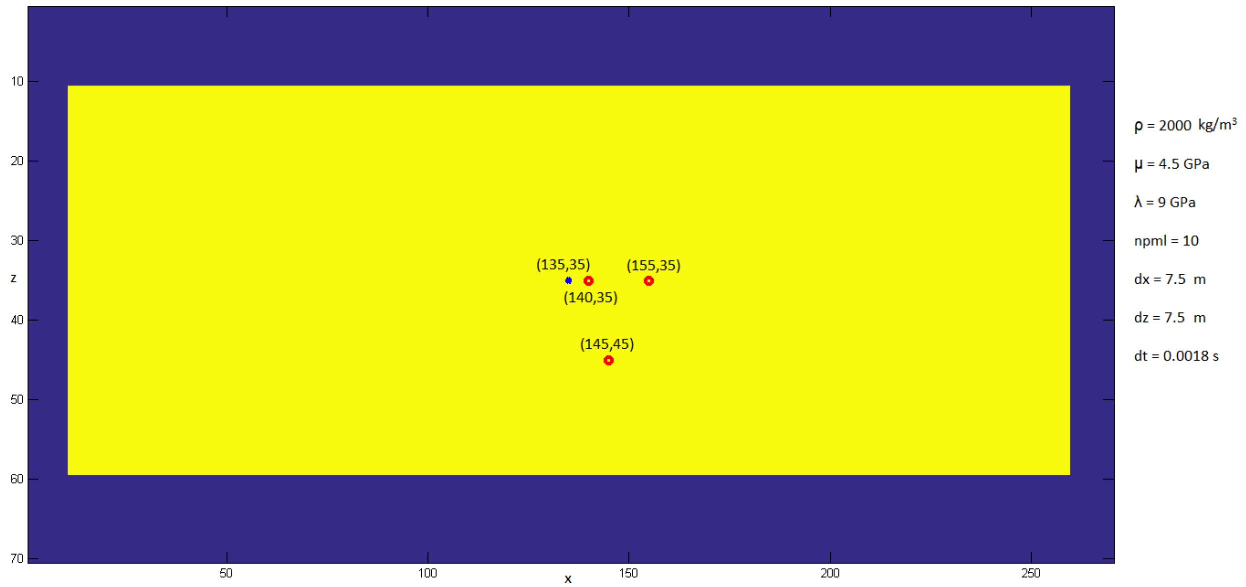


Figure 17: The rectangular shape model with its receiver and source location.

The resulting vertical component particle velocity at the location of receiver 2 is shown in the Figures below.

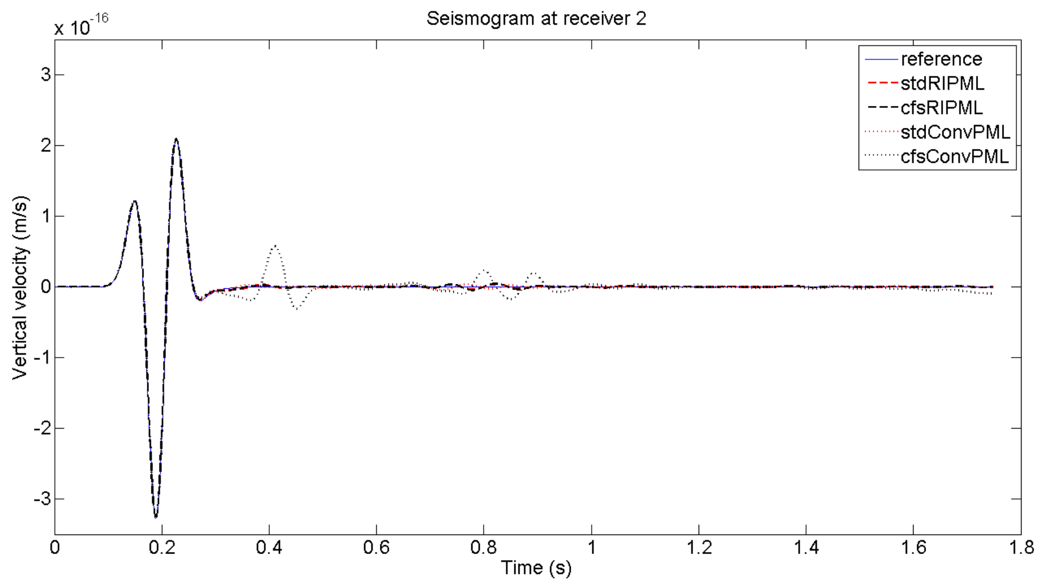


Figure 18: Seismograms at receiver 2 in the rectangular model for the different PML's.

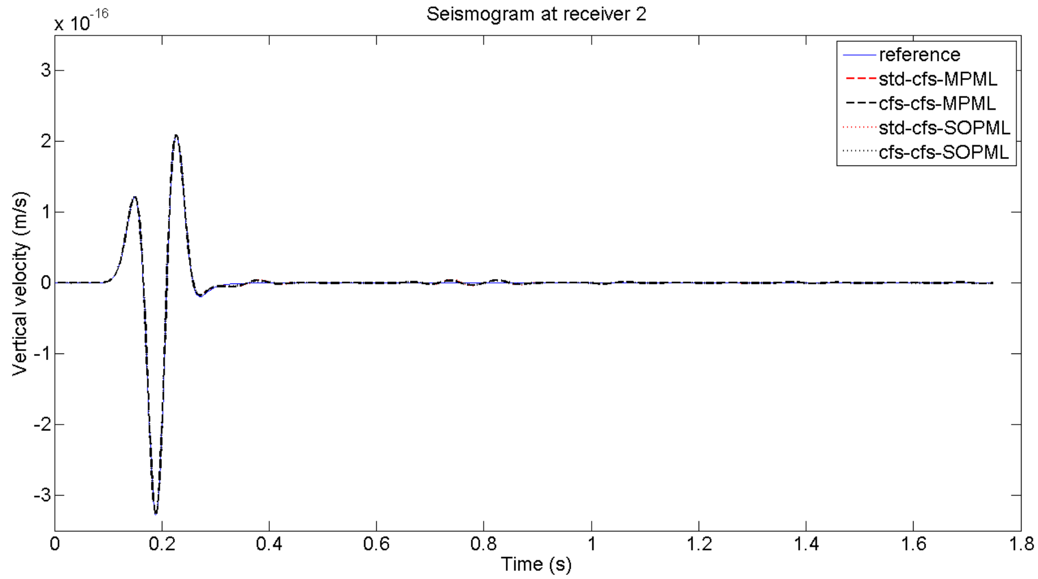


Figure 19: Seismograms at receiver 2 in the rectangular model for the different PML's.

A quick look at Figures 18 and 19 shows that all the different PML types seem to perform fairly similar. The only difference being the cfsConvPML which has us believe that the attenuation coefficients picked were not close to the optimal attenuation coefficients yet.

Next we show the error in dB at each time step for the rectangular model in Figures 20 to 23.

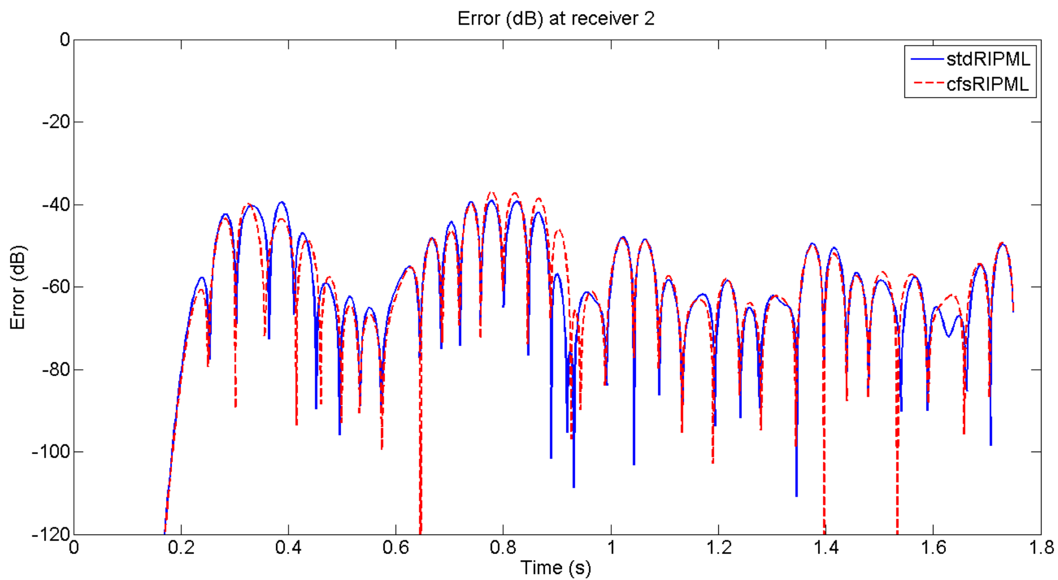


Figure 20: Error in dB at receiver 2 at different times for the stdRIMPML and the cfsRIMPML.

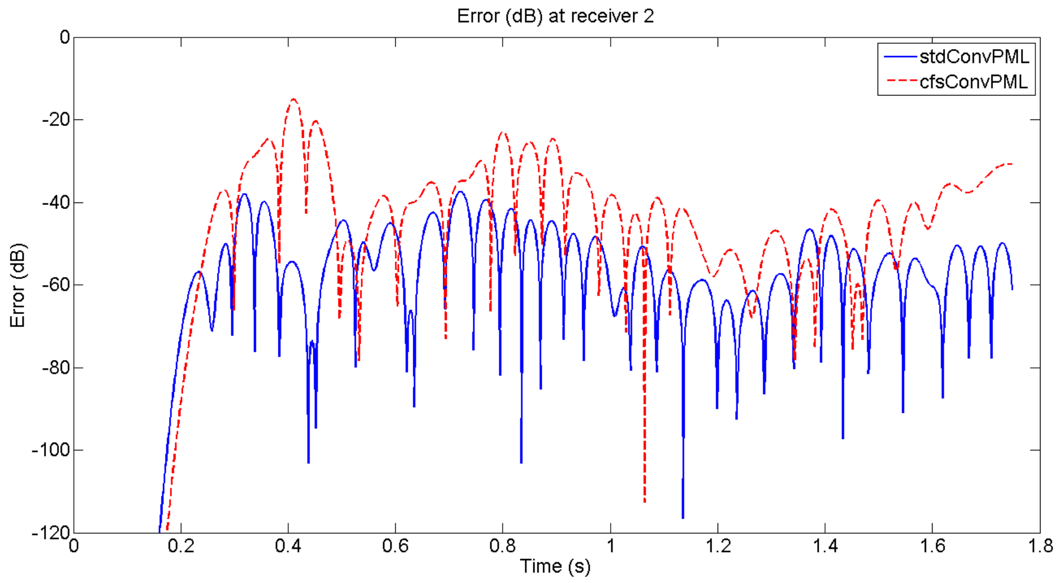


Figure 21: Error in dB at receiver 2 at different times for the stdConvMPML and the cfsConvMPML.

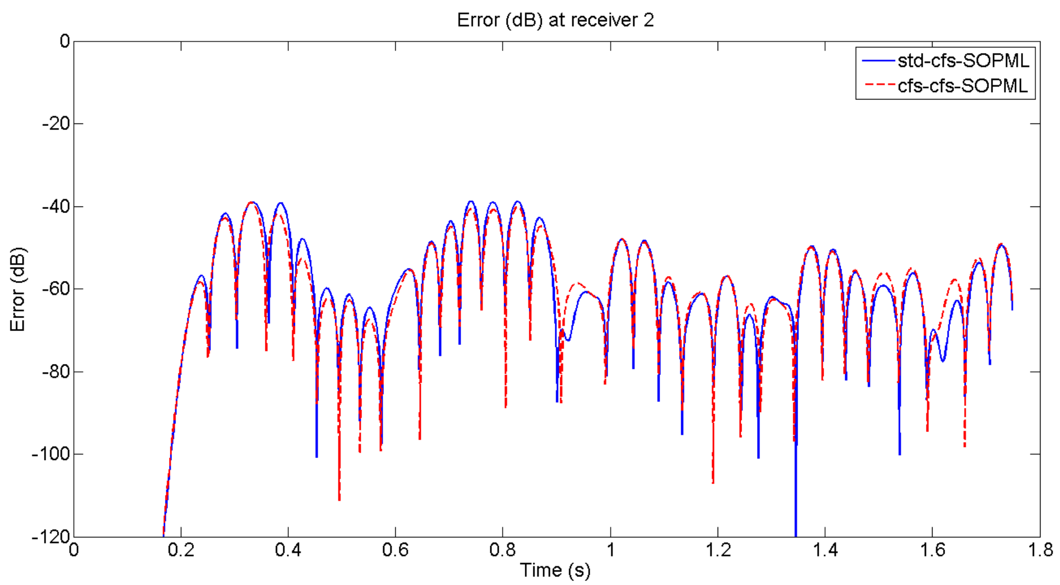


Figure 22: Error in dB at receiver 2 at different times for the std-cfs-MPML and the cfs-cfs-MPML.

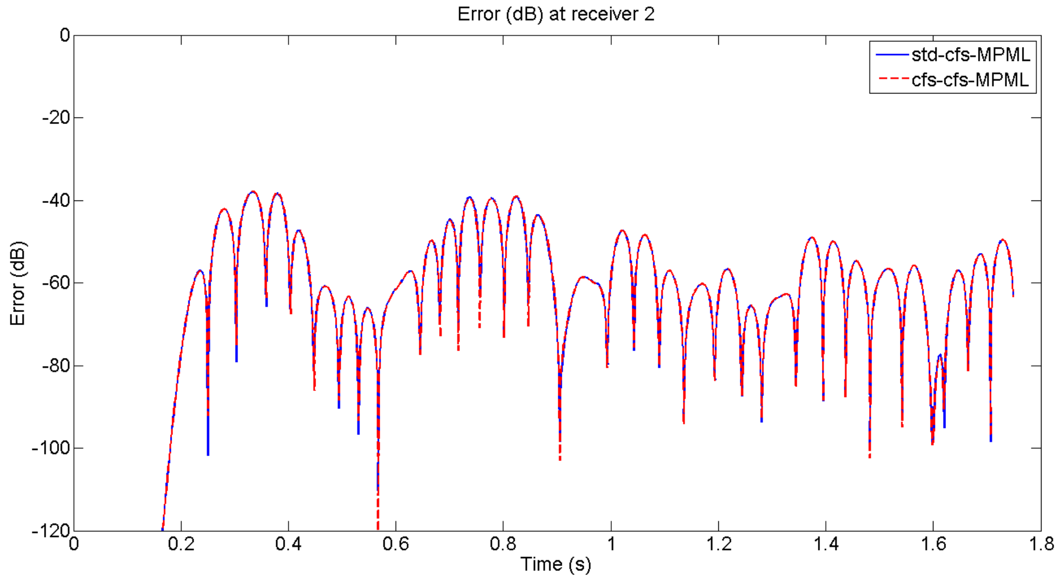


Figure 23: Error in dB at receiver 2 at different times for the std-cfs-SOPML and the cfs-cfs-SOPML.

We see that none of our PML's perform significantly better than the other. The error seems to fluctuate around approximately -60dB for time instances 0.3 to 1.8 seconds. There seems to be less of a decline in the error at later times compared to the dB error of the square model. However the receiver location is now different compared to the square model.

To give a better comparison between performance of a PML in a rectangular domain compared to a square domain we make some changes to our square model. We remove receivers 1 and 3 and only keep receiver 2 at the same location. All other parameters are kept the same with the exemption of the number of cells in the x-direction. We change the number of cells in the x-direction to 40 cells (of which 10 are PML cells on both sides). This way we have created a rectangular model inside our square model. Next we compute the error in dB at receiver 2 in this adapted "square" model. We use 10 PML cells and a std-cfs-MPML. The results are shown in Figure 24 below. We notice that there seems to be more overlap of reflections at the boundary with the outgoing wave. This can be seen as a larger error at time before 0.5 seconds. Also the reflection around 0.75 seconds has in error increase of almost 70 dB. This is almost a factor 12 increase in error at this points in time. This reflection is a reflection caused by the near-grazing angle incident waves. Far away from the source in the rectangular domain the outgoing wave seems to travel almost parallel to the boundary.

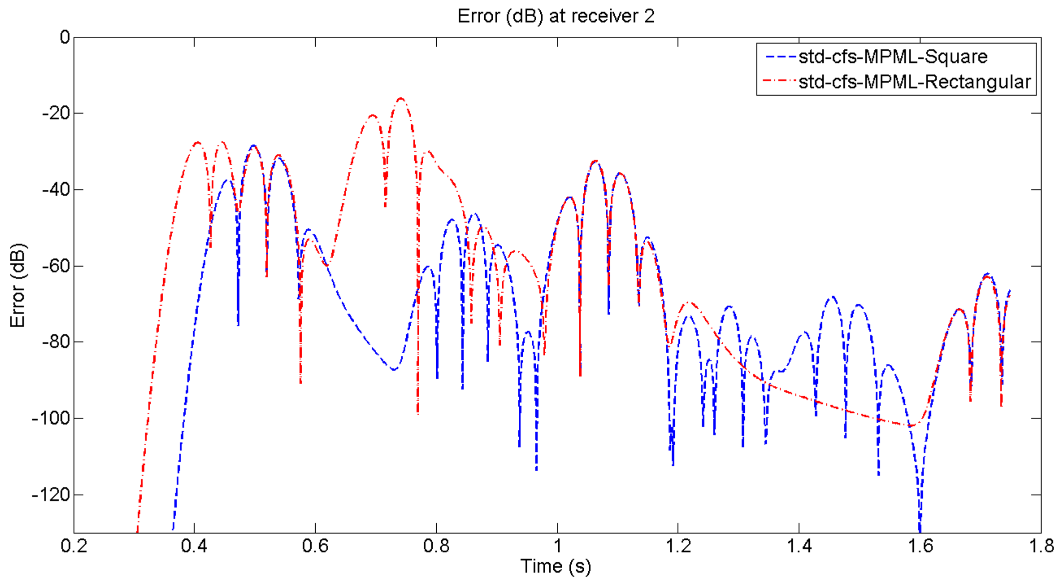


Figure 24: The error in dB at receiver 2 for the square model and the adjusted "square" model.

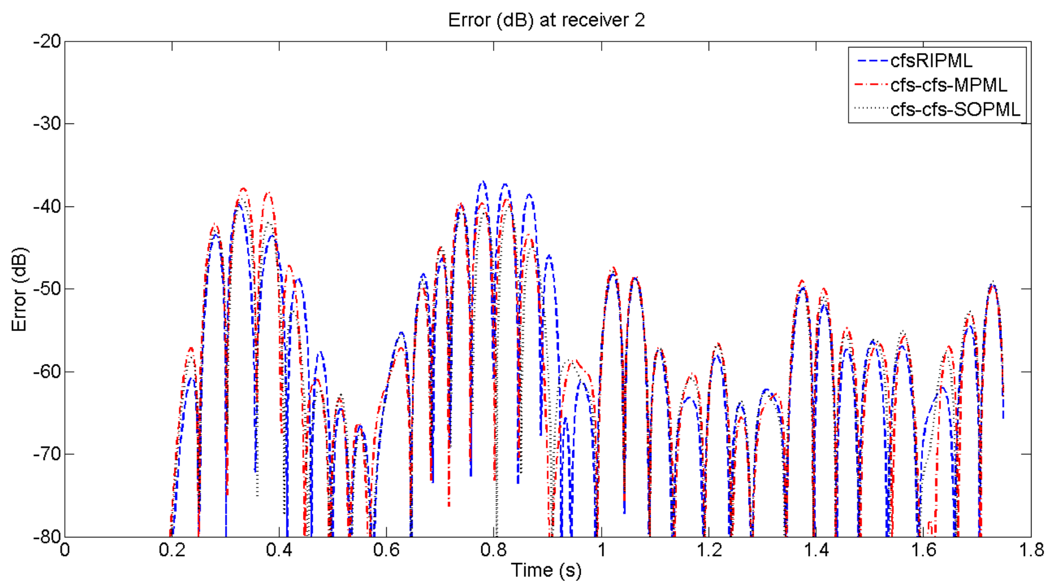


Figure 25: Error in dB at receiver 2 at different times for the std-cfs-MPML and the cfs-cfs-MPML.

We see in Figure 25 that indeed all the different PML perform approximately equally well on a rectangular grid with the difference in errors between them being 1-4 dB.

Lastly we look again at the influence of the number of PML cells on the performance of the attenuation. The results are shown in Figures 26 and 27.

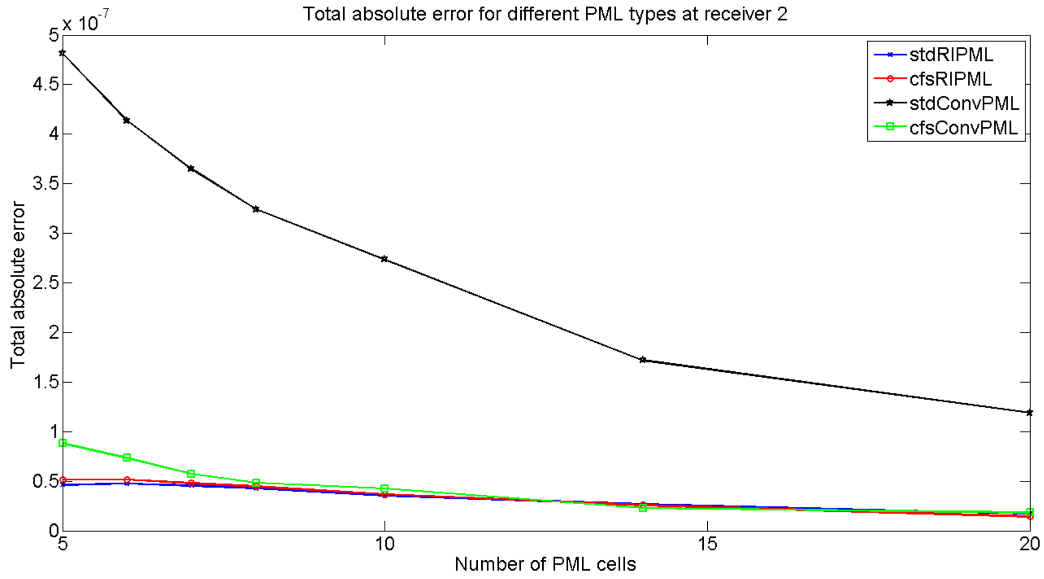


Figure 26: Total absolute error relative to the maximum amplitude at receiver 2.

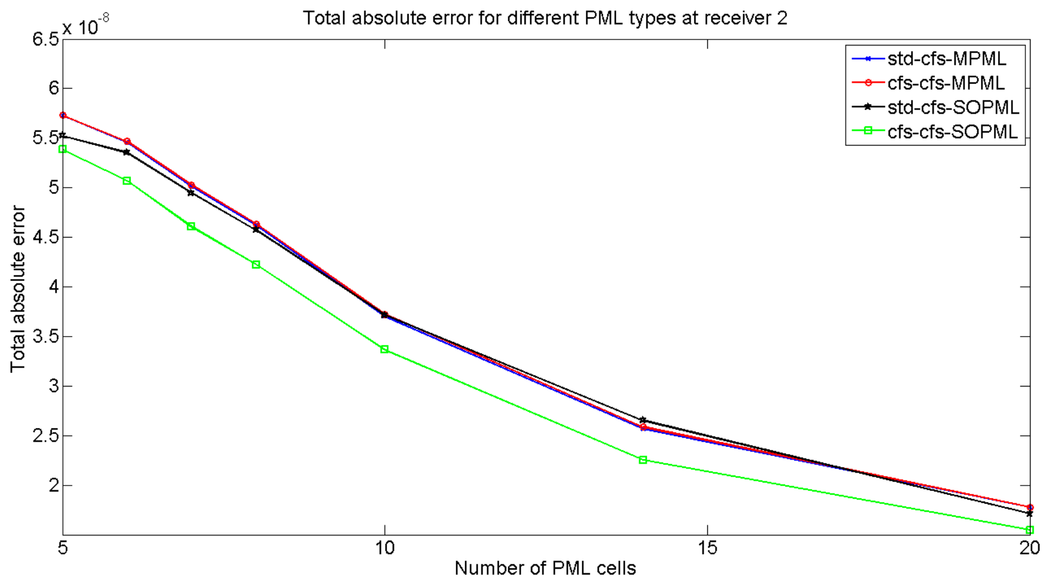


Figure 27: Total absolute error relative to the maximum amplitude at receiver 2.

We see that with an increase in the number of PML cells the error seems to be going down towards a certain asymptote. What is interesting is that when we look at the total maximum error the cfsConvPML produces a better results than its std counterpart indicating the importance of α_j in the attenuation of reflections of low angle waves.

4.3 Optimal attenuation functions for each PML

Through trial and error, we found that small alterations had to be made to the maximum attenuation coefficients in order for the computation the remain stable or perform better. The results are shown in Table 2. It should be noted that the picking of correct attenuation coefficients MPML took considerable less time for the MPML compared to the SOPML. Especially for the cfs-cfs-SOPML the picking of good attenuation coefficients was extremely time consuming.

Table 2: Different maximum attenuation coefficients used for each PML.

Method	$d_{1_{j_0}}$	$\kappa_{1_{j_0}}$	$\alpha_{1_{j_0}}$	$d_{2_{j_0}}$	$\kappa_{2_{j_0}}$	$\alpha_{2_{j_0}}$
stdRIPML	$d_{1_{j_0}}/1.5$					
cfsRIPML	$d_{1_{j_0}}/2$	$\kappa_{1_{j_0}}/1.5$	$\alpha_{1_{j_0}}/1.5$			
stdConv	$d_{1_{j_0}}/2750$					
cfsConv	$d_{1_{j_0}}/1300$	$3\kappa_{1_{j_0}}$	$\alpha_{1_{j_0}}/400$			
std-cfs-SOPML	$d_{1_{j_0}}/1.5$			$d_{1_{j_0}}/30$	$1.5\kappa_{1_{j_0}}$	$100\alpha_{2_{j_0}}$
cfs-cfs-SOPML	$d_{1_{j_0}}/2$	$\kappa_{1_{j_0}}/1.5$	$\alpha_{1_{j_0}}/1.5$	$d_{1_{j_0}}/30$	$1.5\kappa_{1_{j_0}}$	$20\alpha_{1_{j_0}}$
std-cfs-MPML	$d_{1_{j_0}}$	$\kappa_{1_{j_0}}$	$\alpha_{1_{j_0}}$	$d_{1_{j_0}}/100$		
cfs-cfs-MPML	$d_{1_{j_0}}$	$\kappa_{1_{j_0}}$	$\alpha_{1_{j_0}}$	$d_{1_{j_0}}/100$	$15\kappa_{1_{j_0}}$	$200\alpha_{1_{j_0}}$

We found for both the SOPML the rule that $d_{2_{j_0}} < d_{1_{j_0}}$ for the method to be stable. Also we found that the influence of $\alpha_{2_{j_0}}$ for the MPML was almost negligible. This made the optimization of the MPML easier than the optimization of the SOPML. For the cfs-cfs-SOPML it was found that $\kappa_{2_{j_0}}$ had to be bigger than $\kappa_{1_{j_0}}$ for the results to improve. It must be mentioned that in comparison to the SOPML the optimization of the MPML was easier. Whereas with the SOPML the method would sometimes become unstable with a change of an attenuation coefficient by a factor of 10 , the MPML remained stable r with most changes in the attenuation coefficients.

4.4 Computation time

In Table 3 a comparison of the computation time of the different PML 's is shown. Only the values for the cfsRIPML, cfsConvPML and the cfs-cfs-MPML are given. The cfs-cfs-SOPML has the same amount of storage space as the cfs-cfs-MPML and from our computations we found that its computation time is very similar to that of the cfs-cfs-MPML. The same goes for the stdConv and stdRIPML.

Table 3 : Comparison of the computation time difference between the cfsRIPML and the ConvPML and the calculation time difference between the cfsRIPML and the cfs-cfs-MPML. Negative times indicate the method is faster and positive times indicate the method is slower.

Type	5 cell	6 cell	7 cell	8 cell	10 cell	14 cell	20 cell
ConvPML	-43%	-44%	-31%	-24%	-42.0167%	-42%	-50%
cfs-cfs-MPML	40%	38%	58 %	66 %	31 %	44 %	41 %

It can be seen that just as we expected the calculation time for MPML (and therefore also the SOPML) is larger than the calculation time of the cfsRIPML. It does however seems to oscillate for different number of PML cells. In Table 4 we show the CPU time differences for the computations.

Table 4: Comparison of the CPU time difference between the cfsRIPML and the ConvPML and the calculation time difference between the cfsRIPML and the cfs-cfs-MPML. Negative times indicate the method is faster and positive times indicate the method is slower.

Type	5 cell	6 cell	7 cell	8 cell	10 cell	14 cell	20 cell
ConvPML	-42%	-44%	-24%	-22%	-41%	-40%	-48%
cfs-cfs-MPML	33%	28%	56%	65%	26%	41%	44%

We can see that the results for the CPU times are fairly similar to the results of the computation times meaning that our computation happens fairly efficient.

4.5 Evanescent waves

Lastly we wanted to see how well our model could handle evanescent waves like surface waves. Therefore the following model was created which has also been tested in [12]. The model is shown in Figure 27. The source used was a 1.5 Hz Ricker wavelet. The time step used was 3.2 ms and the spatial step size was 2.5 m. The top 12.5 meters of the model was computed to be air by setting Lamé’s first and second parameters to zero Pa and the density to $1/10^{-6}$ kg/m^3 in this region. This way a free-surface was created. Therefore we now switch to a RSG grid in order to have a stable solution with this sudden shift in elastic parameters. By setting the source very close to this free-surface evanescent waves are created. By also setting the receiver very close to the free-surface these evanescent waves should be visible in our results. It was found in [28] that evanescent wave can cause large reflections at the boundaries. The evanescent wave can be compared with the near-grazing angle waves of the rectangular domain. The stdRIPML, cfsRIPML, std-cfs-SOPML, std-cfs-MPML and cfs-cfs-MPML were tested on this model. The vertical particle velocity was measured at the receiver location.

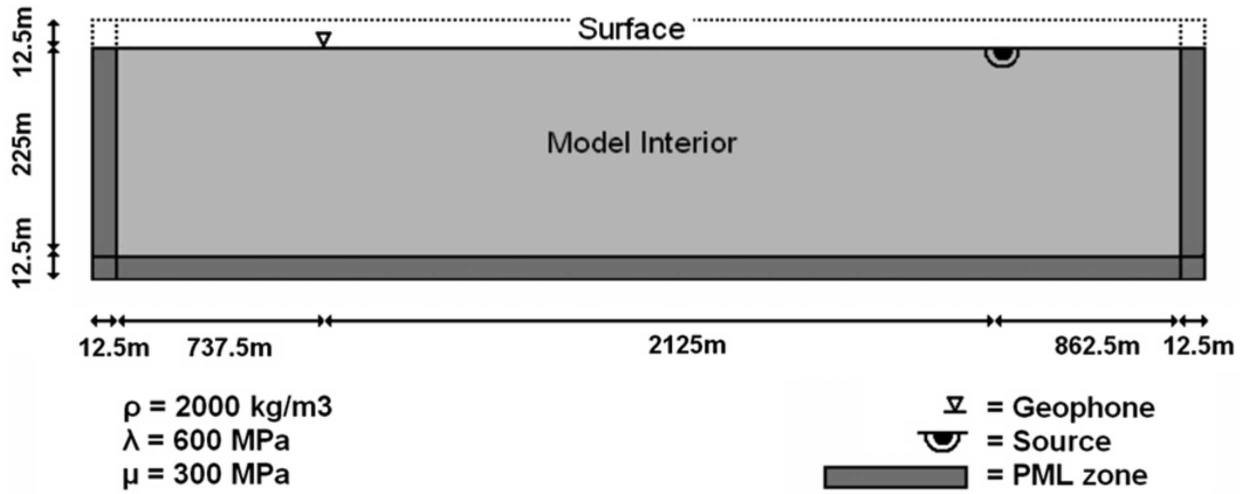


Figure 28: Schematic view of the model for testing how the MPML deals with evanescent waves.

The resulting seismograms are shown in Figures 29 and 30.

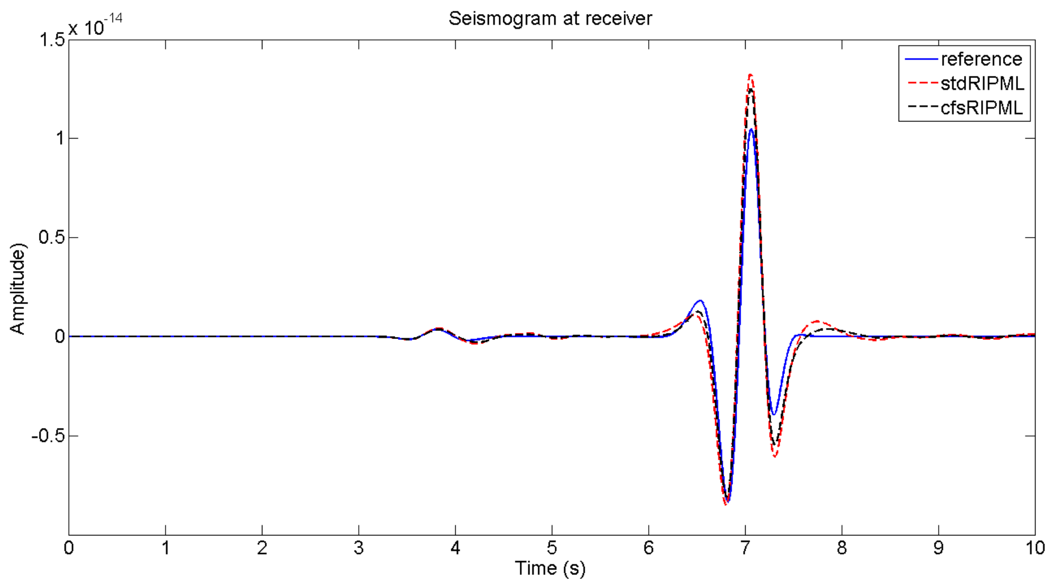


Figure 29: Seismogram at the receiver location for the evanescent model for different PML types.

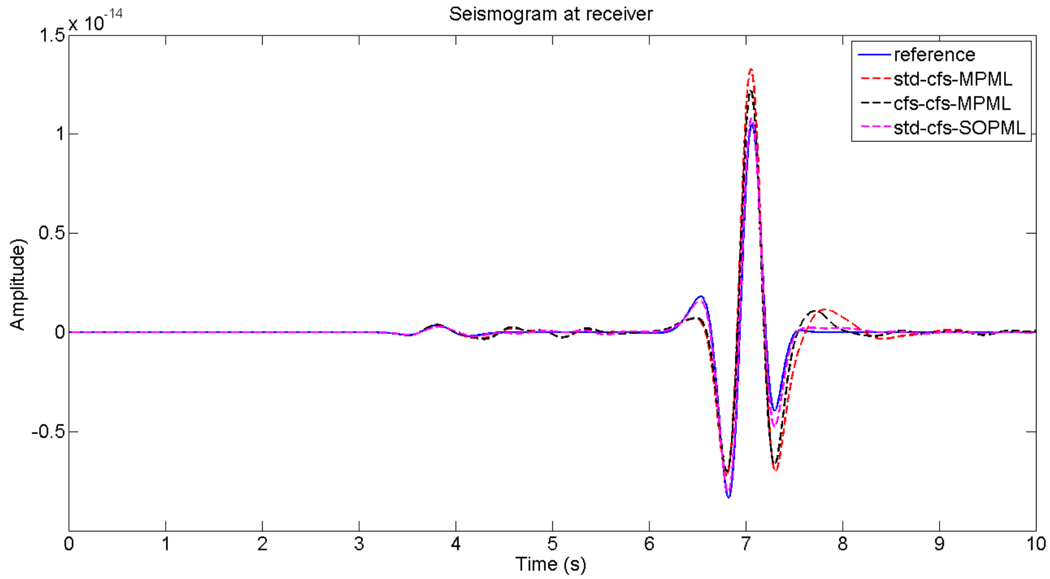


Figure 30: Seismogram at the receiver location for the evanescent model for different PML types.

One can directly see the difficulties the program has with the evanescent waves. It has trouble getting the correct amplitude of the outgoing wave due the interference of evanescent waves.

In Figures 31 and 32 the errors in dB are shown for the different PML types.

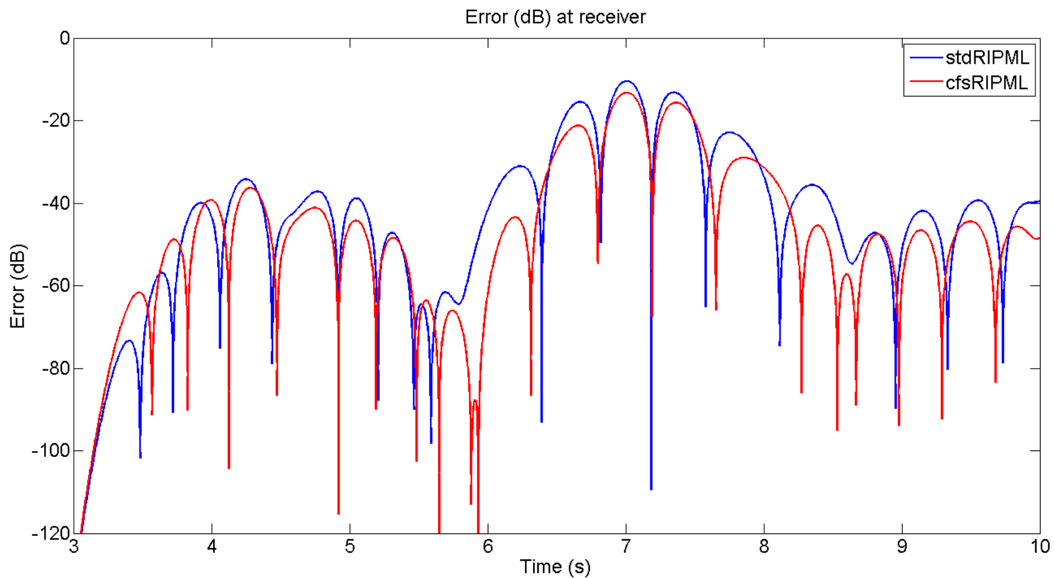


Figure 31: The error at the receiver location in dB for the stdRIPML and the cfsRIPML.

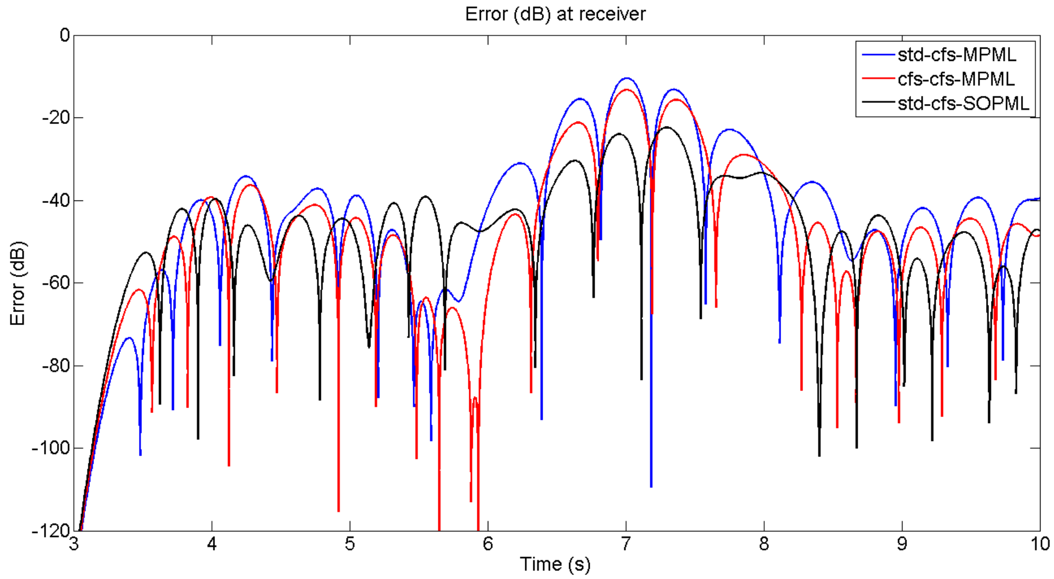


Figure 32: The error at the receiver location in dB for the std-cfs-MPML, cfs-cfs-MPML and the std-cfs-SOPML.

From Figure 31 we observe that the cfsRIPML outperforms the stdRIPML by approximately a factor 2. This shows the importance of the α_j attenuation function for the absorption of evanescent waves. We see in Figure 32 that an cfs-cfs-MPML outperforms an std-cfs-PML by approximately 8 dB. This again shows that the α_j attenuation functions (since we have now a MPML) can greatly improve the performance of the PML in absorbing evanescent waves.

5. Conclusion

It is possible to create a stable and working multi-pole PML (MPML) for the modeling of seismic waves. Using a recursive integration technique the new stretching function that consist of the sum of different stretching functions can be implement in the velocity-stress finite difference time domain wave equations. The MPML did not outperform any of the other tested PML's but gave similar results. More testing on different models will be needed to find out if it can outperform other PML's for specific scenarios. The MPML can perform in square models, rectangular models and can deal with evanescent waves. It was found that the optimization of the MPML was easier compared to the optimization of the SOPML. With an increase in PML cell numbers the performance of the MPML improved. The memory space needed for a MPML compared to a RIPML is increased by a factor 2 but with the increasing computer power of these days this extra memory space seems less and less important. The computation time of the MPML was longer than the computation time of a RIPML but it fluctuated for different numbers of PML cells and no real conclusion could be made on this topic.

References

[1] (Stanford Rock Physics Laboratory, n.d.) "Conceptual Overview of Rock and Fluid Factors that Impact Seismic Velocity and Impedance" Retrieved from https://openei.org/wiki/Seismic_Techniques

[2] B. A. Bolt (1993), Earthquakes

[3] https://www.ucl.ac.uk/EarthSci/people/lidunka/GEOL2014/Geophysics4%20-%20Seismic%20waves/SEISMOLOGY%20.htm

[4] C.P.A. Wapenaar and A.J. Berkhout , 1989, "Elastic wave field extrapolation"

[5] J. Virieux, 1986, "P-SV wave propagation in heterogeneous media: Velocity-stress finite-difference method"

[6] E. H. Saenger, 2000, Wave propagation in fractured media: theory and application of the rotated staggered fininte-difference grid.

[7] P. Moczo, J. Kristek, L. Halada, 2004, The Finite-Difference Method for Seismologists An Introduction.

[8] J. P. Bérenger, 1994, A perfectly matched layer for absorption of electromagnetic waves.

[9] S. D. Gedney, 1998, The perfectly matched layer absorbing medium in Advances in Computational Electrodynamics: the Finite-Difference Time-Domain Method.

[10] T. Wang, X. Tang, 2003, Finite-difference modeling of elastic wave propagation: A nonsplitting perfectly matched layer approach.

[11] J. A. Roden, S. D. Gedney (2000), Convolutional PML (CPML): An efficient FDTD implementation of the CFS-PML for arbitrary media.

[12] F. H. Drossaert, A. Giannopoulos, 2007, Complex frequency shifted convolution PML for FDTD modeling of elastic waves.

[13] W. C. Chew and W. H. Weedon, 1994,"A 3d perfectly matched medium from modified Maxwell's equations with stretched coordinates," Microwave and Optical Tech. Lett., vol. 7, no. 13, pp. 599–604.

[14] W.C. Chew, Q. H. Liu, 1996, Perfectly matched layers for elastodynamics: A new absorbing boundary condition.

[15] M. Kuzuoglu, R. Mittra, 1996, Frequency dependence of the constitutive parameters of the causal perfectly matched anisotropic absorbers.

[16] A. Giannopoulos, 2012, "Unsplit implementation of higher order PMLs," IEEE Trans. Antennas Propag., vol. 60, no. 3, pp. 1479–1485.

[17] A. Giannopoulos, 2008, “An improved new implementation of complex frequency shifted PML for the FDTD method,” IEEE Trans. Antennas Propag., vol. 56, no. 9, pp. 2995–3000.

[18] A. Giannopoulos, 2018, Multipole Perfectly Matched Layer for Finite-Difference Time-Domain Electromagnetic Modeling.

[19] D. P. Connolly, A. Giannopoulos, M. C. Forde, 2015, A higher order perfectly matched layer formulation for finite-difference time-domain seismic wave modeling

[20] F. Collino, C. Tsogska, 2001, Application of the perfectly matched absorbing layer model to linear elastadynamic problem in anisotropic heterogeneous media.

[21] H. Feng, Wie Zhang, Jie Zhang, X. Chen, Importance of double-pole CFS-PML for broad-band seismic wave simulation ad optimal parameter selection.

[22] A. Taflove, 1980, Application of the finite-difference time-domain method to sinusoidal steady state electromagnetic penetration problems.

[23] J. von Neumann, R. D. Richtmeyer, 1950, A method for the numerical calculation of hydrodynamic shocks.

[24] K. Yee, 1966, Numerical solution of inital boundary value problems involving maxwell's equations in isotropic media

[25] D. Lam, 1969, Finite difference methods for electromagnetic scattering problems.

[26] G. Mur, 1981, Absorbing boundary conditions for the finite-difference approximation of the time-domain electromagnetic field equations.

[27] O. Ramadan, 2003, Auxiliary differential equation formulation: an efficient implementation of the perfectly matched layer.

[28] J. P. Bérenger, 2002, Application of the CFS PML to the absorption of evanescent waves in waveguides.

Appendix

Appendix A. Matlab codes

A.1 MPML on a Virieux grid

```
% double pole MPML on a Virieux grid
clear all
close all
clc

% grid parameters
npml = 10; % number of PML cells
N = 20; % number of gridpoints per minimum wavelength
nx = 250; % number of cells in x-direction
nz = 250; % number of cells in y-direction
vp0 = 3000; % P-wave velocity in subsurface
vs0 = vp0/2; % S-wave velocity in subsurface
rho = 2000; % density in subsurface
buo0 = 1/rho; % buoyancy in the subsurface
nt = 1000; % number of time steps

% adding pml cells to nx and nz
nx = nx+2*npml;
nz = nz+2*npml;

% Source parameters
fc = 10; % center frequency Ricker-wavelet
isx = int16(nx/2); % source x-position in gridpoints
isz = int16(nz/2); % source y-position in gridpoints

% Receiver locations
irx1 = isx;
irz1 = isz + 5;
irx2 = isx;
irz2 = isz + 115;
irx3 = isx + 50;
irz3 = isz + 50;

% Lame's constants
mu0 = rho*vs0*vs0; % Lame's second parameter
lambda0 = rho*vp0*vp0-2*mu0; % Lame's first parameter

% Stability criteria
dx = vs0/(N*fc); % 43print43ia for spatial frame
dz = dx;
dt = 0.99/(vp0*sqrt(1/dx*1/dx+1/dz*1/dz)); % stability for time frame

% Source Ricker wavelet
tsour=1/fc;
t = (0:nt-1)*dt;
```

```
t0=tsour*1.5;
T0=tsour*1.5;
tau=pi*(t-t0)/T0;
a=4;
fs=(1-a*tau.*tau).*exp(-2*tau.*tau);
fs = fs/(dx*dz); % dividing the source by the gridsize to get the stress

% PML parameters
pmlfac = 2; % order of 44print44ial scaling
R = 10^(-((log10(npml)-1)/log10(2))-3); % reflection coefficient for a normal
incident P-wave
dmax1 = -(3*vp0)/(2*npml*dx)*logI; % d1j attenuation coefficient
kappamax1 = 2*(vs0/(dx*fc))/N; % k1j attenuation coefficient
alfamax1 = pi*fc; % alfa1j attenuation coefficient
dmax2 = dmax1/100; % dmax2j attenuation coefficient
kappamax2 = kappamax1*15; % kappamax2j attenuation coefficient
alfamax2 = alfamax1*200; % alfamax2j attenuation coefficient

% creating stress and velocity matrices
vx = zeros(nx+1,nz+1); % velocity matrix x-direction
vz = zeros(nx+1,nz+1); % velocity matrix y-direction
txx = zeros(nx+1,nz+1); % stress matrix xx
tzz = zeros(nx+1,nz+1); % stress matrix zz
txz = zeros(nx+1,nz+1); % stress matrix xz
lambda = zeros(nx+1,nz+1); % Lame's first parameter matrix
mu = zeros(nx+1,nz+1); % Lame's second parameter matrix
buo = zeros(nx+1,nz+1); % buoyancy in the matrix
lambda(:,☺ = lambda0; % filling the lambda matrix with the constant
mu(:,☺ = mu0; % filling the mu matrix with the constant
buo(:,☺ = buo0; % filling the buo matrix with the constant

% PML correction term matrices
Rax = ones(nx+1,nz+1); % Rax matrix
RB1x = ones(nx+1,nz+1); % RB1x matrix
RE1x = ones(nx+1,nz+1); % RE1x matrix
RF1x = zeros(nx+1,nz+1); % RF1x matrix
RB2x = ones(nx+1,nz+1); % RB2x matrix
RE2x = ones(nx+1,nz+1); % RE2x matrix
RF2x = zeros(nx+1,nz+1); % RF2x matrix
% PML correction term matrices
Raz = ones(nx+1,nz+1); % Raz matrix
RB1z = ones(nx+1,nz+1); % RB1z matrix
RE1z = ones(nx+1,nz+1); % RE1z matrix
RF1z = zeros(nx+1,nz+1); % RF2z matrix
RB2z = ones(nx+1,nz+1); % RB2z matrix
RE2z = ones(nx+1,nz+1); % RE2z matrix
RF2z = zeros(nx+1,nz+1); % RF2z matrix

% vectors to store receiver seismograms
prvx1 = zeros(1,nt);
prvz1 = zeros(1,nt);
prtxx1 = zeros(1,nt);
prtxz1 = zeros(1,nt);
prtzz1 = zeros(1,nt);

prvx2 = zeros(1,nt);
```

```
prvz2 = zeros(1,nt);
prtxx2 = zeros(1,nt);
prtxz2 = zeros(1,nt);
prtzz2 = zeros(1,nt);

prvx3 = zeros(1,nt);
prvz3 = zeros(1,nt);
prtxx3 = zeros(1,nt);
prtxz3 = zeros(1,nt);
prtzz3 = zeros(1,nt);

% filling PML matrices using the maximum attenuation coefficients
% different attenuation functions can be used if wanted
for I = 1:npml
        d1 = (dmax1*(i/npml)^pmlfac);
        k1 = (1+(kappamax1-1)*(i/npml)^pmlfac);
        a1 = alfamax1*(1-(i/npml));
        RB1x(npml+1-I,☺ = 2/(2+a1*dt);
        RB1z(:,npml+1-i) = 2/(2+a1*dt);
        RE1x(npml+1-I,☺ = (2-a1*dt)/(2+a1*dt);
        RE1z(:,npml+1-i) = (2-a1*dt)/(2+a1*dt);
        RF1x(npml+1-I,☺ = (2*dt*d1)/(2+a1*dt);
        RF1z(:,npml+1-i) = (2*dt*d1)/(2+a1*dt);
        d2 = (dmax2*(i/npml)^pmlfac);
        k2 = (1+(kappamax2-1)*(i/npml)^pmlfac);
        a2 = alfamax2*(1-(i/npml));
        Rax(npml+1-I,☺ = k1+(d1*dt)/(2+a1*dt)+(d2*dt)/(2+a2*dt);
        Raz(:,npml+1-i) = k1+(d1*dt)/(2+a1*dt)+(d2*dt)/(2+a2*dt);
        RB2x(npml+1-I,☺ = 2/(2+a2*dt);
        RB2z(:,npml+1-i) = 2/(2+a2*dt);
        RE2x(npml+1-I,☺ = (2-a2*dt)/(2+a2*dt);
        RE2z(:,npml+1-i) = (2-a2*dt)/(2+a2*dt);
        RF2x(npml+1-I,☺ = (2*dt*d2)/(2+a2*dt);
        RF2z(:,npml+1-i) = (2*dt*d2)/(2+a2*dt);
end

% filling all the PML zones with the correct attenuation value
Rax(end-npml+1:end,☺ = Rax(npml:-1:1,☺;
Raz(:,end-npml+1:end) = Raz(:,npml:-1:1);
RB1x(end-npml+1:end,☺ = RB1x(npml:-1:1,☺;
RB1z(:,end-npml+1:end) = RB1z(:,npml:-1:1);
RE1x(end-npml+1:end,☺ = RE1x(npml:-1:1,☺;
RE1z(:,end-npml+1:end) = RE1z(:,npml:-1:1);
RF1x(end-npml+1:end,☺ = RF1x(npml:-1:1,☺;
RF1z(:,end-npml+1:end) = RF1z(:,npml:-1:1);
RB2x(end-npml+1:end,☺ = RB2x(npml:-1:1,☺;
RB2z(:,end-npml+1:end) = RB2z(:,npml:-1:1);
RE2x(end-npml+1:end,☺ = RE2x(npml:-1:1,☺;
RE2z(:,end-npml+1:end) = RE2z(:,npml:-1:1);
RF2x(end-npml+1:end,☺ = RF2x(npml:-1:1,☺;
RF2z(:,end-npml+1:end) = RF2z(:,npml:-1:1);

% creating the memory matrices phi for the velocity and stress fields
Jphixzdz1 = zeros(nx+1,nz+1);
Jphixxdx1 = zeros(nx+1,nz+1);
```

```
Jphixzdx1 = zeros(nx+1,nz+1);
Jphizzdz1 = zeros(nx+1,nz+1);
Mphixxzdz1 = zeros(nx+1,nz+1);
Mphixxxdx1 = zeros(nx+1,nz+1);
Mphixzzdx1 = zeros(nx+1,nz+1);
Mphixzxdz1 = zeros(nx+1,nz+1);
Mphizzzdz1 = zeros(nx+1,nz+1);
Mphizzxdx1 = zeros(nx+1,nz+1);
Jphixzdz2 = zeros(nx+1,nz+1);
Jphixxdx2 = zeros(nx+1,nz+1);
Jphixzdx2 = zeros(nx+1,nz+1);
Jphizzdz2 = zeros(nx+1,nz+1);
Mphixxzdz2 = zeros(nx+1,nz+1);
Mphixxxdx2 = zeros(nx+1,nz+1);
Mphixzzdx2 = zeros(nx+1,nz+1);
Mphixzxdz2 = zeros(nx+1,nz+1);
Mphizzzdz2 = zeros(nx+1,nz+1);
Mphizzxdx2 = zeros(nx+1,nz+1);

%% starting loop

tic % timing the computation time
cpustart = cputime; % timing the cputime

for t = 1:nt % loop over all t

        % adding the source to the stress field
        txx(isx,isz) = txx(isx,isz)+fs(t)*dt*buo(isx,isz);
        tzz(isx,isz) = tzz(isx,isz)+fs(t)*dt*buo(isx,isz);

        % updating vx
        % correction term update
        Jphixzdzchange =
RB1z(2:end,2:end).*Jphixzdz1(2:end,2:end)+RB2z(2:end,2:end).*Jphixzdz2(2:end,
2:end);
        Jphixxdxchange =
RB1x(2:end,2:end).*Jphixxdx1(2:end,2:end)+RB2x(2:end,2:end).*Jphixxdx2(2:end,
2:end);

        % spatial derivatives
        txxdx = (txx(2:end,2:end)-txx(1:nx,2:end))./dx;
        txzdz = (txz(2:end,2:end)-txz(2:end,1:nz))./dz;

        % correction term calculation
        Jxzdz = (1./Raz(2:end,2:end)-1).*txzdz-
(1./Raz(2:end,2:end).*Jphixzdzchange);
        Jxdx = (1./Rax(2:end,2:end)-1).*txxdx-
(1./Rax(2:end,2:end).*Jphixxdxchange);

        % updating vx
        vx(2:end,2:end) =
vx(2:end,2:end)+buo(2:end,2:end).*dt.*(txxdx+Jxdx)+buo(2:end,2:end).*dt.*(txz
dz+Jxzdz);

        % updating phi
```

```
        Jphixzdz1(2:end,2:end) =
RE1z(2:end,2:end).*Jphixzdz1(2:end,2:end)+RF1z(2:end,2:end)./Raz(2:end,2:end)
.*(txzdz-Jphixzdzchange);
        Jphixxdx1(2:end,2:end) =
RE1x(2:end,2:end).*Jphixxdx1(2:end,2:end)+RF1x(2:end,2:end)./Rax(2:end,2:end)
.*(txxdx-Jphixxdxchange);
        Jphixzdz2(2:end,2:end) =
RE2z(2:end,2:end).*Jphixzdz2(2:end,2:end)+RF2z(2:end,2:end)./Raz(2:end,2:end)
.*(txzdz-Jphixzdzchange);
        Jphixxdx2(2:end,2:end) =
RE2x(2:end,2:end).*Jphixxdx2(2:end,2:end)+RF2x(2:end,2:end)./Rax(2:end,2:end)
.*(txxdx-Jphixxdxchange);

        % doing the same for vz

        Jphixzdxchange =
RB1x(1:nx,1:nz).*Jphixzdx1(1:nx,1:nz)+RB2x(1:nx,1:nz).*Jphixzdx2(1:nx,1:nz);
        Jphizzdzchange =
RB1z(1:nx,1:nz).*Jphizzdz1(1:nx,1:nz)+RB2z(1:nx,1:nz).*Jphizzdz2(1:nx,1:nz);

        tzzdz = (tzz(1:nx,2:end)-tzz(1:nx,1:nz))./dz;
        txzdx = (txz(2:end,1:nz)-txz(1:nx,1:nz))./dx;

        Jxzdx = (1./Rax(1:nx,1:nz)-1).*txzdx-
(1./Rax(1:nx,1:nz).*Jphixzdxchange);
        Jzdz = (1./Raz(1:nx,1:nz)-1).*tzzdz-
(1./Raz(1:nx,1:nz).*Jphizzdzchange);

        vz(1:nx,1:nz) =
vz(1:nx,1:nz)+buo(1:nx,1:nz).*dt.*(tzzdz+Jzdz)+buo(1:nx,1:nz).*dt.*(txzdx+Jxz
dx);

        Jphixzdx1(1:nx,1:nz) =
RE1x(1:nx,1:nz).*Jphixzdx1(1:nx,1:nz)+RF1x(1:nx,1:nz)./Rax(1:nx,1:nz).*(txzdx
-Jphixzdxchange);
        Jphizzdz1(1:nx,1:nz) =
RE1z(1:nx,1:nz).*Jphizzdz1(1:nx,1:nz)+RF1z(1:nx,1:nz)./Raz(1:nx,1:nz).*(tzzdz
-Jphizzdzchange);
        Jphixzdx2(1:nx,1:nz) =
RE2x(1:nx,1:nz).*Jphixzdx2(1:nx,1:nz)+RF2x(1:nx,1:nz)./Rax(1:nx,1:nz).*(txzdx
-Jphixzdxchange);
        Jphizzdz2(1:nx,1:nz) =
RE2z(1:nx,1:nz).*Jphizzdz2(1:nx,1:nz)+RF2z(1:nx,1:nz)./Raz(1:nx,1:nz).*(tzzdz
-Jphizzdzchange);

        % doing the same for txx

        Mphixxzdzchange =
RB1z(1:nx,2:end).*Mphixxzdz1(1:nx,2:end)+RB2z(1:nx,2:end).*Mphixxzdz2(1:nx,2:
end);
        Mphixxxdxchange =
RB1x(1:nx,2:end).*Mphixxxdx1(1:nx,2:end)+RB2x(1:nx,2:end).*Mphixxxdx2(1:nx,2:
end);

        vxdx = (vx(2:end,2:end)-vx(1:nx,2:end))./dx;
```

```
        vzdz = (vz(1:nx,2:end)-vz(1:nx,1:nz))./dz;

        Mxxzdz = (1./Raz(1:nx,2:end)-1).*vzdz-
(1./Raz(1:nx,2:end).*Mphixxzdzchange);
        Mxxdx = (1./Rax(1:nx,2:end)-1).*vxdx-
(1./Rax(1:nx,1:nz).*Mphixxxdxchange);

        txx(1:nx,2:end) =
txx(1:nx,2:end)+(lambda(1:nx,2:end)+2.*mu(1:nx,2:end)).*dt.*(vxdx+Mxxdx)+lamb
da(1:nx,2:end).*dt.*(vzdz+Mxxzdz);

        Mphixxzdz1(1:nx,2:end) =
RE1z(1:nx,2:end).*Mphixxzdz1(1:nx,2:end)+RF1z(1:nx,2:end)./Raz(1:nx,2:end).*(
vzdz-Mphixxzdzchange);
        Mphixxxdx1(1:nx,2:end) =
RE1x(1:nx,2:end).*Mphixxxdx1(1:nx,2:end)+RF1x(1:nx,2:end)./Rax(1:nx,2:end).*(
vxdx-Mphixxxdxchange);
        Mphixxzdz2(1:nx,2:end) =
RE2z(1:nx,2:end).*Mphixxzdz2(1:nx,2:end)+RF2z(1:nx,2:end)./Raz(1:nx,2:end).*(
vzdz-Mphixxzdzchange);
        Mphixxxdx2(1:nx,2:end) =
RE2x(1:nx,2:end).*Mphixxxdx2(1:nx,2:end)+RF2x(1:nx,2:end)./Rax(1:nx,2:end).*(
vxdx-Mphixxxdxchange);

        % doing the same for txz

        Mphixzzdxchange =
RB1x(2:end,1:nz).*Mphixzzdx1(2:end,1:nz)+RB2x(2:end,1:nz).*Mphixzzdx2(2:end,1
:nz);
        Mphixzxdzchange =
RB1z(2:end,1:nz).*Mphixzxdz1(2:end,1:nz)+RB2z(2:end,1:nz).*Mphixzxdz2(2:end,1
:nz);

        vzdx = (vz(2:end,1:nz)-vz(1:nx,1:nz))./dx;
        vxdz = (vx(2:end,2:end)-vx(2:end,1:nz))./dz;

        Mxzzdx = (1./Rax(2:end,1:nz)-1).*vzdx-
(1./Rax(2:end,1:nz).*Mphixzzdxchange);
        Mxzxdz = (1./Raz(2:end,1:nz)-1).*vxdz-
(1./Raz(2:end,1:nz).*Mphixzxdzchange);

        txz(2:end,1:nz) =
txz(2:end,1:nz)+mu(2:end,1:nz).*dt.*(vzdx+Mxzzdx)+mu(2:end,1:nz).*dt.*(vxdz+M
xzxdz);

        Mphixzzdx1(2:end,1:nz) =
RE1x(2:end,1:nz).*Mphixzzdx1(2:end,1:nz)+RF1x(2:end,1:nz)./Rax(2:end,1:nz).*(
vzdx-Mphixzzdxchange);
        Mphixzxdz1(2:end,1:nz) =
RE1z(2:end,1:nz).*Mphixzxdz1(2:end,1:nz)+RF1z(2:end,1:nz)./Raz(2:end,1:nz).*(
vxdz-Mphixzxdzchange);
        Mphixzzdx2(2:end,1:nz) =
RE2x(2:end,1:nz).*Mphixzzdx2(2:end,1:nz)+RF2x(2:end,1:nz)./Rax(2:end,1:nz).*(
vzdx-Mphixzzdxchange);
```

```
        Mphixzxdz2(2:end,1:nz) =
RE2z(2:end,1:nz).*Mphixzxdz2(2:end,1:nz)+RF2z(2:end,1:nz)./Raz(2:end,1:nz).*(
vxdz-Mphixzxdzchange);

        % doing the same for tzz
        Mphizzzdzchange =
RB1z(1:nx,2:end).*Mphizzzdz1(1:nx,2:end)+RB2z(1:nx,2:end).*Mphizzzdz2(1:nx,2:
end);
        Mphizzxdxchange =
RB1x(1:nx,2:end).*Mphizzxdx1(1:nx,2:end)+RB2x(1:nx,2:end).*Mphizzxdx2(1:nx,2:
end);

        Mzzxdx = (1./Rax(1:nx,2:end)-1).*vxdx-
(1./Rax(1:nx,2:end).*Mphizzxdxchange);
        Mzzzdz = (1./Raz(1:nx,2:end)-1).*vzdz-
(1./Raz(1:nx,2:end).*Mphizzzdzchange);

        tzz(1:nx,2:end) =
tzz(1:nx,2:end)+lambda(1:nx,2:end).*dt.*(vxdx+Mzzxdx)+(lambda(1:nx,2:end)+2.*
mu(1:nx,2:end)).*dt.*(vzdz+Mzzzdz);

        Mphizzzdz1(1:nx,2:end) =
RE1z(1:nx,2:end).*Mphizzzdz1(1:nx,2:end)+RF1z(1:nx,2:end)./Raz(1:nx,2:end).*(
vzdz-Mphizzzdzchange);
        Mphizzxdx1(1:nx,2:end) =
RE1x(1:nx,2:end).*Mphizzxdx1(1:nx,2:end)+RF1x(1:nx,2:end)./Rax(1:nx,2:end).*(
vxdx-Mphizzxdxchange);
        Mphizzzdz2(1:nx,2:end) =
RE2z(1:nx,2:end).*Mphizzzdz2(1:nx,2:end)+RF2z(1:nx,2:end)./Raz(1:nx,2:end).*(
vzdz-Mphizzzdzchange);
        Mphizzxdx2(1:nx,2:end) =
RE2x(1:nx,2:end).*Mphizzxdx2(1:nx,2:end)+RF2x(1:nx,2:end)./Rax(1:nx,2:end).*(
vxdx-Mphizzxdxchange);


%          % plotting stress fields if wanted
%          if (mod(t,10)==5)
%          clf;
%          imagesc(tzz');
%          hold on
%          plot(irx1, irz1,'ob')
%          plot(irx2, irz2,'ob')
%          plot(irx3, irz3,'ob')
%          plot(isx, isz,'xb')
%          plot(npml:nx+1-npml,ones(1,numel(npml:nx+1-npml))*npml,'-r')
%          plot(npml:nx+1-npml,ones(1,numel(npml:nx+1-npml))*(nz+1-npml),'-r')
%          plot(ones(1,numel(npml:nz+1-npml))*npml,npml:nz+1-npml,'-r')
%          plot(ones(1,numel(npml:nz+1-npml))*(nx+1-npml),npml:nz+1-npml,'-r')
%          xlim([0 nx])
%          ylim([0 nz])
%          %caxis([-0.2 0.2]); % setting limits for colorbar
%          colorbar;
%          title(49print('Time %f ms',t*dt*1000));
%          pause(0.01)
%          end
```

```
        % storing the measured field variable at the different receiver
        % locations
        prvx1(t) = vx(irx1,irz1);
        prvz1(t) = vz(irx1,irz1);
        prtxx1(t) = txx(irx1,irz1);
        prtxz1(t) = txz(irx1,irz1);
        prtzz1(t) = tzz(irx1,irz1);

        prvx2(t) = vx(irx2,irz2);
        prvz2(t) = vz(irx2,irz2);
        prtxx2(t) = txx(irx2,irz2);
        prtxz2(t) = txz(irx2,irz2);
        prtzz2(t) = tzz(irx2,irz2);

        prvx3(t) = vx(irx3,irz3);
        prvz3(t) = vz(irx3,irz3);
        prtxx3(t) = txx(irx3,irz3);
        prtxz3(t) = txz(irx3,irz3);
        prtzz3(t) = tzz(irx3,irz3);
end
cpuend = cputime;
time = toc;
cputimespend = cpuend-cpustart;
```

A.2 RIPML code on Virieux grid

```
% single pole MPML
clear all
close all
clc

% grid parameters
npml = 10; % number of PML cells
N = 20; % number of gridpoints per minimum wavelength
nx = 250; % number of cells in x-direction
nz = 250; % number of cells in y-direction
vp0 = 3000; % P-wave velocity in subsurface
vs0 = vp0/2; % S-wave velocity in subsurface
rho = 2000; % density in subsurface
buo0 = 1/rho; % buoyancy in the subsurface
nt = 1000; % number of time steps

% adding pml cells to nx and nz
nx = nx+2*npml;
nz = nz+2*npml;

% Source parameters
fc = 10; % center frequency Ricker-wavelet
isx = int16(nx/2); % source x-position in gridpoints
isz = int16(nz/2); % source y-position in gridpoints

% Receiver locations
irx1 = isx;
```

```
irz1 = isz + 5;
irx2 = isx;
irz2 = isz + 115;
irx3 = isx + 50;
irz3 = isz + 50;

% Lame's constants
mu0 = rho*vs0*vs0; % Lame's second parameter
lambda0 = rho*vp0*vp0-2*mu0; % Lame's first parameter

% Stability criteria
dx = vs0/(N*fc); % stabilty for spatial frame
dz = dx;
dt = 0.99/(vp0*sqrt(1/dx*1/dx+1/dz*1/dz)); % stability for time frame

% Source Ricker wavelet
tsour=1/fc;
t = (0:nt-1)*dt;
t0=tsour*1.5;
T0=tsour*1.5;
tau=pi*(t-t0)/T0;
a=4;
fs=(1-a*tau.*tau).*exp(-2*tau.*tau);
fs = fs/(dx*dz); % dividing the source by the gridsize to get the stress

% PML parameters
pmlfac = 2; % order of polynomal scaling
R = 10^(-((log10(npml)-1)/log10(2))-3); % reflection coefficient for a normal
incident P-wave
dmax = -(3*vp0)/(2*npml*dx)*log(R); % dj attenuation coefficient
kappamax = 2*(vs0/(dx*fc))/N; % kj attenuation coefficient
alfamax = pi*fc; % alfaj attenuation coefficient

% creating stress and velocity matrices
vx = zeros(nx+1,nz+1); % velocity matrix x-direction
vz = zeros(nx+1,nz+1); % velocity matrix y-direction
txx = zeros(nx+1,nz+1); % stress matrix xx
tzz = zeros(nx+1,nz+1); % stress matrix zz
txz = zeros(nx+1,nz+1); % stress matrix xz
lambda = zeros(nx+1,nz+1); % Lame's first parameter matrix
mu = zeros(nx+1,nz+1); % Lame's second parameter matrix
buo = zeros(nx+1,nz+1); % buoyancy in the matrix
lambda(:,:) = lambda0; % filling the lambda matrix with the constant
mu(:,:) = mu0; % filling the mu matrix with the constant
buo(:,:) = buo0; % filling the buo matrix with the constant

% PML correction term matrices
RAx = ones(nx+1,nz+1);
RBx = ones(nx+1,nz+1);
REx = ones(nx+1,nz+1);
RFx = zeros(nx+1,nz+1);
RAz = ones(nx+1,nz+1);
RBz = ones(nx+1,nz+1);
REz = ones(nx+1,nz+1);
RFz = zeros(nx+1,nz+1);
```

```
% vectors to store receiver seismograms
prvx1 = zeros(1,nt);
prvz1 = zeros(1,nt);
prtxx1 = zeros(1,nt);
prtxz1 = zeros(1,nt);
prtzz1 = zeros(1,nt);

prvx2 = zeros(1,nt);
prvz2 = zeros(1,nt);
prtxx2 = zeros(1,nt);
prtxz2 = zeros(1,nt);
prtzz2 = zeros(1,nt);

prvx3 = zeros(1,nt);
prvz3 = zeros(1,nt);
prtxx3 = zeros(1,nt);
prtxz3 = zeros(1,nt);
prtzz3 = zeros(1,nt);

% filling PML matrices using the maximum attenuation coefficients
% different attenuation functions can be used if wanted
for i = 1:npml
        d = (dmax*(i/npml)^pmlfac);
        k = (1+(kappamax-1)*(i/npml)^pmlfac);
        a = alfamax*(1-(i/npml));
        RAx(npml+1-i,:) = k+(d*dt)/(2+a*dt);
        RAz(:,npml+1-i) = k+(d*dt)/(2+a*dt);
        RBx(npml+1-i,:) = 2/(2+a*dt);
        RBz(:,npml+1-i) = 2/(2+a*dt);
        REx(npml+1-i,:) = (2-a*dt)/(2+a*dt);
        REz(:,npml+1-i) = (2-a*dt)/(2+a*dt);
        RFx(npml+1-i,:) = (2*dt*d)/(2+a*dt);
        RFz(:,npml+1-i) = (2*dt*d)/(2+a*dt);
end

% filling all the PML zones with the correct attenuation value
RAx(end-npml+1:end,:) = RAx(npml:-1:1,:);
RAz(:,end-npml+1:end) = RAz(:,npml:-1:1);
RBx(end-npml+1:end,:) = RBx(npml:-1:1,:);
RBz(:,end-npml+1:end) = RBz(:,npml:-1:1);
REx(end-npml+1:end,:) = REx(npml:-1:1,:);
REz(:,end-npml+1:end) = REz(:,npml:-1:1);
RFx(end-npml+1:end,:) = RFx(npml:-1:1,:);
RFz(:,end-npml+1:end) = RFz(:,npml:-1:1);

% creating the memory matrices phi for the velocity and stress fields
Jphixzdz = zeros(nx+1,nz+1);
Jphixxdx = zeros(nx+1,nz+1);
Jphixzdx = zeros(nx+1,nz+1);
Jphizzdz = zeros(nx+1,nz+1);
Mphixxzdz = zeros(nx+1,nz+1);
Mphixxxdx = zeros(nx+1,nz+1);
Mphixzzdx = zeros(nx+1,nz+1);
Mphixzxdz = zeros(nx+1,nz+1);
Mphizzzdz = zeros(nx+1,nz+1);
```

```
Mphizzxdx = zeros(nx+1,nz+1);

%% starting loop

tic % timing the computation time
cpustart = cputime; % timing the cputime

for t = 1:nt % loop over all t

        % adding the source to the stress field
        txx(isx,isz) = txx(isx,isz)+fs(t)*dt*buo(isx,isz);
        tzz(isx,isz) = tzz(isx,isz)+fs(t)*dt*buo(isx,isz);

        % updating vx
        % spatial derivatives
        txxdx = (txx(2:end,2:end)-txx(1:nx,2:end))./dx;
        txzdz = (txz(2:end,2:end)-txz(2:end,1:nz))./dz;

        % correction term update
        Jxzdz = (1./RAz(2:end,2:end)-1).*txzdz-
(1./RAz(2:end,2:end).*RBz(2:end,2:end).*Jphixzdz(2:end,2:end));
        Jxdx = (1./RAx(2:end,2:end)-1).*txxdx-
(1./RAx(2:end,2:end).*RBx(2:end,2:end).*Jphixxdx(2:end,2:end));

        % updating vx
        vx(2:end,2:end) =
vx(2:end,2:end)+buo(2:end,2:end).*dt.*(txxdx+Jxdx)+buo(2:end,2:end).*dt.*(txz
dz+Jxzdz);

        % updating phi
        Jphixzdz(2:end,2:end) =
REz(2:end,2:end).*Jphixzdz(2:end,2:end)+RFz(2:end,2:end)./RAz(2:end,2:end).*(
txzdz-RBz(2:end,2:end).*Jphixzdz(2:end,2:end));
        Jphixxdx(2:end,2:end) =
REx(2:end,2:end).*Jphixxdx(2:end,2:end)+RFx(2:end,2:end)./RAx(2:end,2:end).*(
txxdx-RBx(2:end,2:end).*Jphixxdx(2:end,2:end));

        % doing the same for vz
        tzzdz = (tzz(1:nx,2:end)-tzz(1:nx,1:nz))./dz;
        txzdx = (txz(2:end,1:nz)-txz(1:nx,1:nz))./dx;

        Jxzdx = (1./RAx(1:nx,1:nz)-1).*txzdx-
(1./RAx(1:nx,1:nz).*RBx(1:nx,1:nz).*Jphixzdx(1:nx,1:nz));
        Jzdz = (1./RAz(1:nx,1:nz)-1).*tzzdz-
(1./RAz(1:nx,1:nz).*RBz(1:nx,1:nz).*Jphizzdz(1:nx,1:nz));

        vz(1:nx,1:nz) =
vz(1:nx,1:nz)+buo(1:nx,1:nz).*dt.*(tzzdz+Jzdz)+buo(1:nx,1:nz).*dt.*(txzdx+Jxz
dx);

        Jphixzdx(1:nx,1:nz) =
REx(1:nx,1:nz).*Jphixzdx(1:nx,1:nz)+RFx(1:nx,1:nz)./RAx(1:nx,1:nz).*(txzdx-
RBx(1:nx,1:nz).*Jphixzdx(1:nx,1:nz));
```

```
        Jphizzdz(1:nx,1:nz) =
REz(1:nx,1:nz).*Jphizzdz(1:nx,1:nz)+RFz(1:nx,1:nz)./RAz(1:nx,1:nz).*(tzzdz-
RBz(1:nx,1:nz).*Jphizzdz(1:nx,1:nz));

        % doing the same for txx
        vxdx = (vx(2:end,2:end)-vx(1:nx,2:end))./dx;
        vzdz = (vz(1:nx,2:end)-vz(1:nx,1:nz))./dz;

        Mxxzdz = (1./RAz(1:nx,2:end)-1).*vzdz-
(1./RAz(1:nx,2:end).*RBz(1:nx,2:end).*Mphixxzdz(1:nx,2:end));
        Mxxdx = (1./RAx(1:nx,2:end)-1).*vxdx-
(1./RAx(1:nx,1:nz).*RBx(1:nx,2:end).*Mphixxxdx(1:nx,2:end));

        txx(1:nx,2:end) =
txx(1:nx,2:end)+(lambda(1:nx,2:end)+2.*mu(1:nx,2:end)).*dt.*(vxdx+Mxxdx)+lamb
da(1:nx,2:end).*dt.*(vzdz+Mxxzdz);

        Mphixxzdz(1:nx,2:end) =
REz(1:nx,2:end).*Mphixxzdz(1:nx,2:end)+RFz(1:nx,2:end)./RAz(1:nx,2:end).*(vzd
z-RBz(1:nx,2:end).*Mphixxzdz(1:nx,2:end));
        Mphixxxdx(1:nx,2:end) =
REx(1:nx,2:end).*Mphixxxdx(1:nx,2:end)+RFx(1:nx,2:end)./RAx(1:nx,2:end).*(vxd
x-RBx(1:nx,2:end).*Mphixxxdx(1:nx,2:end));

        % doing the same for txz
        vzdx = (vz(2:end,1:nz)-vz(1:nx,1:nz))./dx;
        vxdz = (vx(2:end,2:end)-vx(2:end,1:nz))./dz;

        Mxzzdx = (1./RAx(2:end,1:nz)-1).*vzdx-
(1./RAx(2:end,1:nz).*RBx(2:end,1:nz).*Mphixzzdx(2:end,1:nz));
        Mxzxdz = (1./RAz(2:end,1:nz)-1).*vxdz-
(1./RAz(2:end,1:nz).*RBz(2:end,1:nz).*Mphixzxdz(2:end,1:nz));

        txz(2:end,1:nz) =
txz(2:end,1:nz)+mu(2:end,1:nz).*dt.*(vzdx+Mxzzdx)+mu(2:end,1:nz).*dt.*(vxdz+M
xzxdz);

        Mphixzzdx(2:end,1:nz) =
REx(2:end,1:nz).*Mphixzzdx(2:end,1:nz)+RFx(2:end,1:nz)./RAx(2:end,1:nz).*(vzd
x-RBx(2:end,1:nz).*Mphixzzdx(2:end,1:nz));
        Mphixzxdz(2:end,1:nz) =
REz(2:end,1:nz).*Mphixzxdz(2:end,1:nz)+RFz(2:end,1:nz)./RAz(2:end,1:nz).*(vxd
z-RBz(2:end,1:nz).*Mphixzxdz(2:end,1:nz));

        % doing the same for tzz
        Mzzxdx = (1./RAx(1:nx,2:end)-1).*vxdx-
(1./RAx(1:nx,2:end).*RBx(1:nx,2:end).*Mphizzxdx(1:nx,2:end));
        Mzzzdz = (1./RAz(1:nx,2:end)-1).*vzdz-
(1./RAz(1:nx,2:end).*RBz(1:nx,2:end).*Mphizzzdz(1:nx,2:end));

        tzz(1:nx,2:end) =
tzz(1:nx,2:end)+lambda(1:nx,2:end).*dt.*(vxdx+Mzzxdx)+(lambda(1:nx,2:end)+2.*
mu(1:nx,2:end)).*dt.*(vzdz+Mzzzdz);
```

```
        Mphizzzdz(1:nx,2:end) =
REz(1:nx,2:end).*Mphizzzdz(1:nx,2:end)+RFz(1:nx,2:end)./RAz(1:nx,2:end).*(vzd
z-RBz(1:nx,2:end).*Mphizzzdz(1:nx,2:end));
        Mphizzxdx(1:nx,2:end) =
REx(1:nx,2:end).*Mphizzxdx(1:nx,2:end)+RFx(1:nx,2:end)./RAx(1:nx,2:end).*(vxd
x-RBx(1:nx,2:end).*Mphizzxdx(1:nx,2:end));

        % plotting stress fields
%         if (mod(t,10)==5)
%         clf;
%         imagesc(tzz');
%         hold on
%         plot(irx1, irz1,'ob')
%         plot(irx2, irz2,'ob')
%         plot(irx3, irz3,'ob')
%         plot(isx, isz,'xb')
%         plot(npml:nx+1-npml,ones(1,numel(npml:nx+1-npml))*npml,'-r')
%         plot(npml:nx+1-npml,ones(1,numel(npml:nx+1-npml))*(nz+1-npml),'-r')
%         plot(ones(1,numel(npml:nz+1-npml))*npml,npml:nz+1-npml,'-r')
%         plot(ones(1,numel(npml:nz+1-npml))*(nx+1-npml),npml:nz+1-npml,'-r')
%         xlim([0 nx])
%         ylim([0 nz])
%         caxis([-8*10^-12 8*10^-12]);
%         colorbar;
%         title(sprintf('Time %f ms',t*dt*1000));
%         pause(0.01)
%         end

        prvx1(t) = vx(irx1,irz1);
        prvz1(t) = vz(irx1,irz1);
        prtxx1(t) = txx(irx1,irz1);
        prtxz1(t) = txz(irx1,irz1);
        prtzz1(t) = tzz(irx1,irz1);

        prvx2(t) = vx(irx2,irz2);
        prvz2(t) = vz(irx2,irz2);
        prtxx2(t) = txx(irx2,irz2);
        prtxz2(t) = txz(irx2,irz2);
        prtzz2(t) = tzz(irx2,irz2);

        prvx3(t) = vx(irx3,irz3);
        prvz3(t) = vz(irx3,irz3);
        prtxx3(t) = txx(irx3,irz3);
        prtxz3(t) = txz(irx3,irz3);
        prtzz3(t) = tzz(irx3,irz3);
end
cpuend = cputime;
time = toc;
cputimespend = cpuend-cpustart;
```

A.3 ConvPML on Virieux grid

```
% ConvPML
clear all
close all
clc

% more information on this type of PML can be found in: F. H. Drossaert,
% A. Giannopoulos, 2007, Complex frequency shifted convolution PML for
% FDTD modeling of elastic waves.
% grid parameters
npml = 10; % number of PML cells
N = 20; % number of gridpoints per minimum wavelength
nx = 250; % number of cells in x-direction
nz = 250; % number of cells in y-direction
vp0 = 3000; % P-wave velocity in subsurface
vs0 = vp0/2; % S-wave velocity in subsurface
rho = 2000; % density in subsurface
buo0 = 1/rho; % buoyancy in the subsurface
nt = 1000; % number of time steps

% adding pml cells to nx and nz
nx = nx+2*npml;
nz = nz+2*npml;

% Source parameters
fc = 10; % center frequency Ricker-wavelet
isx = int16(nx/2); % source x-position in gridpoints
isz = int16(nz/2); % source y-position in gridpoints

% Receiver locations
irx1 = isx;
irz1 = isz + 5;
irx2 = isx;
irz2 = isz + 115;
irx3 = isx + 50;
irz3 = isz + 50;

% Lame's constants
mu0 = rho*vs0*vs0; % Lame's second parameter
lambda0 = rho*vp0*vp0-2*mu0; % Lame's first parameter

% Stability criteria
dx = vs0/(N*fc); % stabilty for spatial frame
dz = dx;
dt = 0.99/(vp0*sqrt(1/dx*1/dx+1/dz*1/dz)); % stability for time frame

% Source Ricker wavelet
tsour=1/fc;
t = (0:nt-1)*dt;
t0=tsour*1.5;
T0=tsour*1.5;
tau=pi*(t-t0)/T0;
a=4;
```

```
fs=(1-a*tau.*tau).*exp(-2*tau.*tau);
fs = fs/(dx*dz); % dividing the source by the gridsize to get the stress

% PML parameters
pmlfac = 2; % order of polynomal scaling
R = 10^(-((log10(npml)-1)/log10(2))-3); % reflection coefficient for a normal
incident P-wave
sigmamax = -(3*vp0)/(2*npml*dx)*log(R)/2750; % dj attenuation coefficient
kappamax = 1; %2*(vs0/(dx*fc))/N; % kj attenuation coefficient
alfamax = 0; %pi*fc; % alfaj attenuation coefficient

% creating stress and velocity matrices
vx = zeros(nx+1,nz+1); % velocity matrix x-direction
vz = zeros(nx+1,nz+1); % velocity matrix y-direction
txx = zeros(nx+1,nz+1); % stress matrix xx
tzz = zeros(nx+1,nz+1); % stress matrix zz
txz = zeros(nx+1,nz+1); % stress matrix xz
lambda = zeros(nx+1,nz+1); % Lame's first parameter matrix
mu = zeros(nx+1,nz+1); % Lame's second parameter matrix
buo = zeros(nx+1,nz+1); % buoyancy in the matrix
lambda(:,:) = lambda0; % filling the lambda matrix with the constant
mu(:,:) = mu0; % filling the mu matrix with the constant
buo(:,:) = buo0; % filling the buo matrix with the constant

% vectors to store receiver signals
prvx1 = zeros(1,nt);
prvz1 = zeros(1,nt);
prtxx1 = zeros(1,nt);
prtxz1 = zeros(1,nt);
prtzz1 = zeros(1,nt);

prvx2 = zeros(1,nt);
prvz2 = zeros(1,nt);
prtxx2 = zeros(1,nt);
prtxz2 = zeros(1,nt);
prtzz2 = zeros(1,nt);

prvx3 = zeros(1,nt);
prvz3 = zeros(1,nt);
prtxx3 = zeros(1,nt);
prtxz3 = zeros(1,nt);
prtzz3 = zeros(1,nt);

% creating convolution operator matrices
omegaxx = zeros(nx+1, nz+1);
omegaxz = zeros(nx+1, nz+1);
omegazx = zeros(nx+1, nz+1);
omegazz = zeros(nx+1, nz+1);

phixx = zeros(nx+1, nz+1);
phixz = zeros(nx+1, nz+1);
phizx = zeros(nx+1, nz+1);
phizz = zeros(nx+1, nz+1);

% creating multiplication factor matrices
```

```
bx = ones(nx, nz);
cx = zeros(nx, nz);
ddx = zeros(nx, nz);
bz = ones(nx, nz);
cz = zeros(nx, nz);
ddz = zeros(nx, nz);

% filling multiplication factor matrices using the maximum attenuation
% coefficients
for i = 1:npml
        sigma = (sigmamax*(i/npml)^pmlfac);
        k = (1+(kappamax-1)*(i/npml)^pmlfac);
        a = alfamax*(1-(i/npml));
        bx(npml+1-i,:) = exp(-1*(sigma/k+a));
        cx(npml+1-i,:) = (sigma/(sigma*k+k*k*a))*(exp(-1*(sigma/k+a))-1);
        ddx(npml+1-i,:) = (1-k)/k;
        bz(:,npml+1-i) = exp(-1*(sigma/k+a));
        cz(:,npml+1-i) = (sigma/(sigma*k+k*k*a))*(exp(-1*(sigma/k+a))-1);
        ddz(:,npml+1-i) = (1-k)/k;
end

% filling the multiplication factor matrices
bx(end-npml+1:end,:) = bx(npml:-1:1,:);
cx(end-npml+1:end,:) = cx(npml:-1:1,:);
ddx(end-npml+1:end,:) = ddx(npml:-1:1,:);
bz(:,end-npml+1:end) = bz(:,npml:-1:1);
cz(:,end-npml+1:end) = cz(:,npml:-1:1);
ddz(:,end-npml+1:end) = ddz(:,npml:-1:1);

%% starting loop

tic % timing the computation time
cpustart = cputime; % timing the cputime

for t = 1:nt % loop over all t

        % adding the source to the stress field
        txx(isx,isz) = txx(isx,isz)+fs(t)*dt*buo(isx,isz);
        tzz(isx,isz) = tzz(isx,isz)+fs(t)*dt*buo(isx,isz);

        % updating vx and vz
        % spatial derivatives
        txxdx = (txx(2:end,2:end)-txx(1:nx,2:end))./dx;
        txzdz = (txz(2:end,2:end)-txz(2:end,1:nz))./dz;
        tzzdz = (tzz(1:nx,2:end)-tzz(1:nx,1:nz))./dz;
        txzdx = (txz(2:end,1:nz)-txz(1:nx,1:nz))./dx;

        % updating velocity without any corrections
        vxup =
vx(2:end,2:end)+buo(2:end,2:end).*dt.*txxdx+buo(2:end,2:end).*dt.*txzdz;
        vzup =
vz(1:nx,1:nz)+buo(1:nx,1:nz).*dt.*tzzdz+buo(1:nx,1:nz).*dt.*txzdx;

        % calculating velocity convolution matrices
        phixx(2:end,2:end) = bx.*phixx(2:end,2:end)+cx.*txxdx;
```

```
        phixz(2:end,2:end) = bz.*phixz(2:end,2:end)+cz.*txzdz;
        phizx(1:nx,1:nz) = bx.*phizx(1:nx,1:nz)+cx.*txzdx;
        phizz(1:nx,1:nz) = bz.*phizz(1:nx,1:nz)+cz.*tzzdz;

        % pertubating the velocity fields
        vx(2:end,2:end) =
vxup+buo(2:end,2:end).*dt.*(ddx.*txxdx+phixx(2:end,2:end))+buo(2:end,2:end).*
dt.*(ddz.*txzdz+phixz(2:end,2:end));
        vz(1:nx,1:nz) =
vzup+buo(1:nx,1:nz).*dt.*(ddz.*tzzdz+phizz(1:nx,1:nz))+buo(1:nx,1:nz).*dt.*(d
dx.*txzdx+phizx(1:nx,1:nz));

        vxdx = (vx(2:end,2:end)-vx(1:nx,2:end))./dx;
        vzdz = (vz(1:nx,2:end)-vz(1:nx,1:nz))./dz;
        vzdx = (vz(2:end,1:nz)-vz(1:nx,1:nz))./dx;
        vxdz = (vx(2:end,2:end)-vx(2:end,1:nz))./dz;

        % doing the same for the stress fields
        txxup =
txx(1:nx,2:end)+(lambda(1:nx,2:end)+2.*mu(1:nx,2:end)).*dt.*vxdx+lambda(1:nx,
2:end).*dt.*vzdz;
        txzup =
txz(2:end,1:nz)+mu(2:end,1:nz).*dt.*vzdx+mu(2:end,1:nz).*dt.*vxdz;
        tzzup =
tzz(1:nx,2:end)+lambda(1:nx,2:end).*dt.*vxdx+(lambda(1:nx,2:end)+2.*mu(1:nx,2
:end)).*dt.*vzdz;

        omegaxx(1:nx,2:end) = bx.*omegaxx(1:nx,2:end)+cx.*vxdx;
        omegaxz(2:end,1:nz) = bz.*omegaxz(2:end,1:nz)+cz.*vzdz;
        omegazx(2:end,1:nz) = bx.*omegazx(2:end,1:nz)+cx.*vxdx;
        omegazz(1:nx,2:end) = bz.*omegazz(1:nx,2:end)+cz.*vzdz;

        txx(1:nx,2:end) =
txxup+(lambda(1:nx,2:end)+2.*mu(1:nx,2:end)).*dt.*(ddx.*vxdx+omegaxx(1:nx,2:e
nd))+lambda(1:nx,2:end).*dt.*(ddz.*vzdz+omegazz(1:nx,2:end));
        txz(2:end,1:nz) =
txzup+mu(2:end,1:nz).*dt.*(ddx.*vzdx+omegazx(2:end,1:nz))+mu(2:end,1:nz).*dt.
*(ddz.*vxdz+omegaxz(2:end,1:nz));
        tzz(1:nx,2:end) =
tzzup+lambda(1:nx,2:end).*dt.*(ddx.*vxdx+omegaxx(1:nx,2:end))+(lambda(1:nx,2:
end)+2.*mu(1:nx,2:end)).*dt.*(ddz.*vzdz+omegazz(1:nx,2:end));

%          % plotting stress fields
%          if (mod(t,10)==5)
%          clf;
%          imagesc(tzz');
%          hold on
%          plot(irx1, irz1,'ob')
%          plot(irx2, irz2,'ob')
%          plot(irx3, irz3,'ob')
%          plot(isx, isz,'xb')
%          plot(npml:nx+1-npml,ones(1,numel(npml:nx+1-npml))*npml,'-r')
%          plot(npml:nx+1-npml,ones(1,numel(npml:nx+1-npml))*(nz+1-npml),'-r')
%          plot(ones(1,numel(npml:nz+1-npml))*npml,npml:nz+1-npml,'-r')
%          plot(ones(1,numel(npml:nz+1-npml))*(nx+1-npml),npml:nz+1-npml,'-r')
%          xlim([0 nx])
```

```
%         ylim([0 nz])
%         %caxis([-0.2 0.2]);
%         colorbar;
%         title(sprintf('Time %f ms',t*dt*1000));
%         pause(0.01)
%         end


        % storing the measured field variable at the different receiver
        % locations
        prvx1(t) = vx(irx1,irz1);
        prvz1(t) = vz(irx1,irz1);
        prtxx1(t) = txx(irx1,irz1);
        prtxz1(t) = txz(irx1,irz1);
        prtzz1(t) = tzz(irx1,irz1);

        prvx2(t) = vx(irx2,irz2);
        prvz2(t) = vz(irx2,irz2);
        prtxx2(t) = txx(irx2,irz2);
        prtxz2(t) = txz(irx2,irz2);
        prtzz2(t) = tzz(irx2,irz2);

        prvx3(t) = vx(irx3,irz3);
        prvz3(t) = vz(irx3,irz3);
        prtxx3(t) = txx(irx3,irz3);
        prtxz3(t) = txz(irx3,irz3);
        prtzz3(t) = tzz(irx3,irz3);
end
cpuend = cputime;
time = toc;
cputimespend = cpuend-cpustart;
```

A.4 SOPML on Virieux grid

```
% SOPML
clear all
close all
clc
% More information on this type of PML can be found in: D. P. Connolly,
% A. Giannopoulos, M. C. Forde, 2015, A higher order perfectly matched
% layer formulation for finite-difference time-domain seismic wave modeling
% grid parameters
npml = 10; % number of PML cells
N = 20; % number of gridpoints per minimum wavelength
nx = 250; % number of cells in x-direction
nz = 250; % number of cells in y-direction
vp0 = 3000; % P-wave velocity in subsurface
vs0 = vp0/2; % S-wave velocity in subsurface
rho = 2000; % density in subsurface
buo0 = 1/rho; % buoyancy in the subsurface
nt = 1000; % number of time steps

% adding pml cells to nx and nz
nx = nx+2*npml;
```

```
nz = nz+2*npml;

% Source parameters
fc = 10; % center frequency Ricker-wavelet
isx = int16(nx/2); % source x-position in gridpoints
isz = int16(nz/2); % source y-position in gridpoints

% Receiver locations
irx1 = isx;
irz1 = isz + 5;
irx2 = isx;
irz2 = isz + 115;
irx3 = isx + 50;
irz3 = isz + 50;

% Lame's constants
mu0 = rho*vs0*vs0; % Lame's second parameter
lambda0 = rho*vp0*vp0-2*mu0; % Lame's first parameter

% Stability criteria
dx = vs0/(N*fc); % stabilty for spatial frame
dz = dx;
dt = 0.99/(vp0*sqrt(1/dx*1/dx+1/dz*1/dz)); % stability for time frame

% Source Ricker wavelet
tsour=1/fc;
t = (0:nt-1)*dt;
t0=tsour*1.5;
T0=tsour*1.5;
tau=pi*(t-t0)/T0;
a=4;
fs=(1-a*tau.*tau).*exp(-2*tau.*tau);
fs = fs/(dx*dz); % dividing the source by the gridsize to get the stress

% PML parameters
pmlfac1 = 2; % order of polynomal scaling
R = 10^(-((log10(npml)-1)/log10(2))-3); % reflection coefficient for a normal
incident P-wave
dmax1 = -(3*vp0)/(2*npml*dx)*log(R); % d1j attenuation coefficient
kappamax1 = 2*(vs0/(dx*fc))/N; % k1j attenuation coefficient
alfamax1 = pi*fc; % alfa1j attenuation coefficient
dmax2 = dmax1/100; % dmax2j attenuation coefficient
kappamax2 = kappamax1*15; % kappamax2j attenuation coefficient
alfamax2 = alfamax1*200; % alfamax2j attenuation coefficient

% creating stress and velocity matrices
vx = zeros(nx+1,nz+1); % velocity matrix x-direction
vz = zeros(nx+1,nz+1); % velocity matrix y-direction
txx = zeros(nx+1,nz+1); % stress matrix xx
tzz = zeros(nx+1,nz+1); % stress matrix zz
txz = zeros(nx+1,nz+1); % stress matrix xz
lambda = zeros(nx+1,nz+1); % Lame's first parameter matrix
mu = zeros(nx+1,nz+1); % Lame's second parameter matrix
buo = zeros(nx+1,nz+1); % buoyancy in the matrix
lambda(:,:) = lambda0; % filling the lambda matrix with the constant
```

```
mu(:,:) = mu0; % filling the mu matrix with the constant
buo(:,:) = buo0; % filling the buo matrix with the constant

% PML correction term matrices for x-direction
RA1x = ones(nx+1,nz+1);
RB1x = ones(nx+1,nz+1);
RE1x = ones(nx+1,nz+1);
RF1x = zeros(nx+1,nz+1);
RA2x = ones(nx+1,nz+1);
RB2x = ones(nx+1,nz+1);
RE2x = ones(nx+1,nz+1);
RF2x = zeros(nx+1,nz+1);
% PML correction term matrices for z-direction
RA1z = ones(nx+1,nz+1);
RB1z = ones(nx+1,nz+1);
RE1z = ones(nx+1,nz+1);
RF1z = zeros(nx+1,nz+1);
RA2z = ones(nx+1,nz+1);
RB2z = ones(nx+1,nz+1);
RE2z = ones(nx+1,nz+1);
RF2z = zeros(nx+1,nz+1);

% vectors to store receiver seismograms
prvx1 = zeros(1,nt);
prvz1 = zeros(1,nt);
prtxx1 = zeros(1,nt);
prtxz1 = zeros(1,nt);
prtzz1 = zeros(1,nt);

prvx2 = zeros(1,nt);
prvz2 = zeros(1,nt);
prtxx2 = zeros(1,nt);
prtxz2 = zeros(1,nt);
prtzz2 = zeros(1,nt);

prvx3 = zeros(1,nt);
prvz3 = zeros(1,nt);
prtxx3 = zeros(1,nt);
prtxz3 = zeros(1,nt);
prtzz3 = zeros(1,nt);

% filling PML matrices using the maximum attenuation coefficients
% different attenuation functions can be used if wanted
for i = 1:npml
        d1 = (dmax1*(i/npml));
        k1 = (1+(kappamax1-1)*(i/npml));
        a1 = alfamax1*(1-(i/npml));
        RA1x(npml+1-i,:) = (2+dt*a1)/(2*k1+dt*(a1*k1+d1));
        RA1z(:,npml+1-i) = (2+dt*a1)/(2*k1+dt*(a1*k1+d1));
        RB1x(npml+1-i,:) = (2*k1)/(2*k1+dt*(a1*k1+d1));
        RB1z(:,npml+1-i) = (2*k1)/(2*k1+dt*(a1*k1+d1));
        RE1x(npml+1-i,:) = (2*k1-dt*(a1*k1+d1))/(2*k1+dt*(a1*k1+d1));
        RE1z(:,npml+1-i) = (2*k1-dt*(a1*k1+d1))/(2*k1+dt*(a1*k1+d1));
        RF1x(npml+1-i,:) = (2*d1*dt)/((2*k1+dt*(a1*k1+d1))*k1);
        RF1z(:,npml+1-i) = (2*d1*dt)/((2*k1+dt*(a1*k1+d1))*k1);
        d2 = (dmax2*(i/npml)^pmlfac1);
```

```
        k2 = (1+(kappamax2-1)*(i/npml)^pmlfac1);
        a2 = alfamax2*(1-(i/npml));
        RA2x(npml+1-i,:) = (2+dt*a2)/(2*k2+dt*(a2*k2+d2));
        RA2z(:,npml+1-i) = (2+dt*a2)/(2*k2+dt*(a2*k2+d2));
        RB2x(npml+1-i,:) = (2*k2)/(2*k2+dt*(a2*k2+d2));
        RB2z(:,npml+1-i) = (2*k2)/(2*k2+dt*(a2*k2+d2));
        RE2x(npml+1-i,:) = (2*k2-dt*(a2*k2+d2))/(2*k2+dt*(a2*k2+d2));
        RE2z(:,npml+1-i) = (2*k2-dt*(a2*k2+d2))/(2*k2+dt*(a2*k2+d2));
        RF2x(npml+1-i,:) = (2*d2*dt)/((2*k2+dt*(a2*k2+d2))*k2);
        RF2z(:,npml+1-i) = (2*d2*dt)/((2*k2+dt*(a2*k2+d2))*k2);
end

% filling all the PML zones with the correct attenuation value
RA1x(end-npml+1:end,:) = RA1x(npml:-1:1,:);
RA1z(:,end-npml+1:end) = RA1z(:,npml:-1:1);
RB1x(end-npml+1:end,:) = RB1x(npml:-1:1,:);
RB1z(:,end-npml+1:end) = RB1z(:,npml:-1:1);
RE1x(end-npml+1:end,:) = RE1x(npml:-1:1,:);
RE1z(:,end-npml+1:end) = RE1z(:,npml:-1:1);
RF1x(end-npml+1:end,:) = RF1x(npml:-1:1,:);
RF1z(:,end-npml+1:end) = RF1z(:,npml:-1:1);
RA2x(end-npml+1:end,:) = RA2x(npml:-1:1,:);
RA2z(:,end-npml+1:end) = RA2z(:,npml:-1:1);
RB2x(end-npml+1:end,:) = RB2x(npml:-1:1,:);
RB2z(:,end-npml+1:end) = RB2z(:,npml:-1:1);
RE2x(end-npml+1:end,:) = RE2x(npml:-1:1,:);
RE2z(:,end-npml+1:end) = RE2z(:,npml:-1:1);
RF2x(end-npml+1:end,:) = RF2x(npml:-1:1,:);
RF2z(:,end-npml+1:end) = RF2z(:,npml:-1:1);

% creating the memory matrices phi for the velocity and stress fields
Jphixzdz1 = zeros(nx+1,nz+1);
Jphixxdx1 = zeros(nx+1,nz+1);
Jphizzdx1 = zeros(nx+1,nz+1);
Jphizzdz1 = zeros(nx+1,nz+1);
Mphixxzdz1 = zeros(nx+1,nz+1);
Mphixxxdx1 = zeros(nx+1,nz+1);
Mphixzzdx1 = zeros(nx+1,nz+1);
Mphizxzdz1 = zeros(nx+1,nz+1);
Mphizzzdz1 = zeros(nx+1,nz+1);
Mphizzxdx1 = zeros(nx+1,nz+1);
Jphixzdz2 = zeros(nx+1,nz+1);
Jphixxdx2 = zeros(nx+1,nz+1);
Jphizzdx2 = zeros(nx+1,nz+1);
Jphizzdz2 = zeros(nx+1,nz+1);
Mphixxzdz2 = zeros(nx+1,nz+1);
Mphixxxdx2 = zeros(nx+1,nz+1);
Mphixzzdx2 = zeros(nx+1,nz+1);
Mphizxzdz2 = zeros(nx+1,nz+1);
Mphizzzdz2 = zeros(nx+1,nz+1);
Mphizzxdx2 = zeros(nx+1,nz+1);

%% starting loop

tic % timing the computation time
cpustart = cputime; % timing the cputime
```

```
for t = 2:nt

        % adding the source to the stress field
        txx(isx,isz) = txx(isx,isz)+fs(t)*dt*buo(isx,isz);
        tzz(isx,isz) = tzz(isx,isz)+fs(t)*dt*buo(isx,isz);

        % updating vx
        % calcuting the derivatives
        txxdx = (txx(2:end,2:end)-txx(1:nx,2:end))./dx;
        txzdz = (txz(2:end,2:end)-txz(2:end,1:nz))./dz;

        % calculating the correction term
        Jxzdz = (RA1z(2:end,2:end).*RA2z(2:end,2:end)-
1).*txzdz+RA2z(2:end,2:end).*RB1z(2:end,2:end).*Jphixzdz1(2:end,2:end)+RB2z(2
:end,2:end).*Jphixzdz2(2:end,2:end);
        Jxdx = (RA1x(2:end,2:end).*RA2x(2:end,2:end)-
1).*txxdx+RA2x(2:end,2:end).*RB1x(2:end,2:end).*Jphixxdx1(2:end,2:end)+RB2x(2
:end,2:end).*Jphixxdx2(2:end,2:end);

        % updating vx
        vx(2:end,2:end) =
vx(2:end,2:end)+buo(2:end,2:end).*dt.*(txxdx+Jxdx)+buo(2:end,2:end).*dt.*(txz
dz+Jxzdz);

        % updating phi
        Jphixzdz2(2:end,2:end) = RE2z(2:end,2:end).*Jphixzdz2(2:end,2:end)-
RF2z(2:end,2:end).*(RA1z(2:end,2:end).*txzdz+RB1z(2:end,2:end).*Jphixzdz1(2:e
nd,2:end));
        Jphixxdx2(2:end,2:end) = RE2x(2:end,2:end).*Jphixxdx2(2:end,2:end)-
RF2x(2:end,2:end).*(RA1x(2:end,2:end).*txxdx+RB1x(2:end,2:end).*Jphixxdx1(2:e
nd,2:end));
        Jphixzdz1(2:end,2:end) = RE1z(2:end,2:end).*Jphixzdz1(2:end,2:end)-
RF1z(2:end,2:end).*txzdz;
        Jphixxdx1(2:end,2:end) = RE1x(2:end,2:end).*Jphixxdx1(2:end,2:end)-
RF1x(2:end,2:end).*txxdx;

        % doing the same for vz
        tzzdz = (tzz(1:nx,2:end)-tzz(1:nx,1:nz))./dz;
        txzdx = (txz(2:end,1:nz)-txz(1:nx,1:nz))./dx;

        Jxzdx = (RA1x(1:nx,1:nz).*RA2x(1:nx,1:nz)-
1).*txzdx+RA2x(1:nx,1:nz).*RB1x(1:nx,1:nz).*Jphixzdx1(1:nx,1:nz)+RB2x(1:nx,1:
nz).*Jphixzdx2(1:nx,1:nz);
        Jzdz = (RA1z(1:nx,1:nz).*RA2z(1:nx,1:nz)-
1).*tzzdz+RA2z(1:nx,1:nz).*RB1z(1:nx,1:nz).*Jphizzdz1(1:nx,1:nz)+RB2z(1:nx,1:
nz).*Jphizzdz2(1:nx,1:nz);

        vz(1:nx,1:nz) =
vz(1:nx,1:nz)+buo(1:nx,1:nz).*dt.*(tzzdz+Jzdz)+buo(1:nx,1:nz).*dt.*(txzdx+Jxz
dx);

        Jphixzdx2(1:nx,1:nz) = RE2x(1:nx,1:nz).*Jphixzdx2(1:nx,1:nz)-
RF2x(1:nx,1:nz).*(RA1x(1:nx,1:nz).*txzdx+RB1x(1:nx,1:nz).*Jphixzdx1(1:nx,1:nz
));
```

```
        Jphizzdz2(1:nx,1:nz) = RE2z(1:nx,1:nz).*Jphizzdz2(1:nx,1:nz)-
RF2z(1:nx,1:nz).*(RA1z(1:nx,1:nz).*tzzdz+RB1z(1:nx,1:nz).*Jphizzdz1(1:nx,1:nz
));
        Jphixzdx1(1:nx,1:nz) = RE1x(1:nx,1:nz).*Jphixzdx1(1:nx,1:nz)-
RF1x(1:nx,1:nz).*txzdx;
        Jphizzdz1(1:nx,1:nz) = RE1z(1:nx,1:nz).*Jphizzdz1(1:nx,1:nz)-
RF1z(1:nx,1:nz).*tzzdz;

        % doing the same for txx
        vxdx = (vx(2:end,2:end)-vx(1:nx,2:end))./dx;
        vzdz = (vz(1:nx,2:end)-vz(1:nx,1:nz))./dz;

        Mxxzdz = (RA1z(1:nx,2:end).*RA2z(1:nx,2:end)-
1).*vzdz+RA2z(1:nx,2:end).*RB1z(1:nx,2:end).*Mphixxzdz1(1:nx,2:end)+RB2z(1:nx
,2:end).*Mphixxzdz2(1:nx,2:end);
        Mxxdx = (RA1x(1:nx,2:end).*RA2x(1:nx,2:end)-
1).*vxdx+RA2x(1:nx,2:end).*RB1x(1:nx,2:end).*Mphixxxdx1(1:nx,2:end)+RB2x(1:nx
,2:end).*Mphixxxdx2(1:nx,2:end);

        txx(1:nx,2:end) =
txx(1:nx,2:end)+(lambda(1:nx,2:end)+2.*mu(1:nx,2:end)).*dt.*(vxdx+Mxxdx)+lamb
da(1:nx,2:end).*dt.*(vzdz+Mxxzdz);

        Mphixxzdz2(1:nx,2:end) = RE2z(1:nx,2:end).*Mphixxzdz2(1:nx,2:end)-
RF2z(1:nx,2:end).*(RA1z(1:nx,2:end).*vzdz+RB1z(1:nx,2:end).*Mphixxzdz1(1:nx,2
:end));
        Mphixxxdx2(1:nx,2:end) = RE2x(1:nx,2:end).*Mphixxxdx2(1:nx,2:end)-
RF2x(1:nx,2:end).*(RA1x(1:nx,2:end).*vxdx+RB1x(1:nx,2:end).*Mphixxxdx1(1:nx,2
:end));
        Mphixxzdz1(1:nx,2:end) = RE1z(1:nx,2:end).*Mphixxzdz1(1:nx,2:end)-
RF1z(1:nx,2:end).*vzdz;
        Mphixxxdx1(1:nx,2:end) = RE1x(1:nx,2:end).*Mphixxxdx1(1:nx,2:end)-
RF1x(1:nx,2:end).*vxdx;

        % doing the same for txz
        vzdx = (vz(2:end,1:nz)-vz(1:nx,1:nz))./dx;
        vxdz = (vx(2:end,2:end)-vx(2:end,1:nz))./dz;

        Mxzzdx = (RA1x(2:end,1:nz).*RA2x(2:end,1:nz)-
1).*vzdx+RA2x(2:end,1:nz).*RB1x(2:end,1:nz).*Mphixzzdx1(2:end,1:nz)+RB2x(2:en
d,1:nz).*Mphixzzdx2(2:end,1:nz);
        Mxzxdz = (RA1z(2:end,1:nz).*RA2z(2:end,1:nz)-
1).*vxdz+RA2z(2:end,1:nz).*RB1z(2:end,1:nz).*Mphixzxdz1(2:end,1:nz)+RB2z(2:en
d,1:nz).*Mphixzxdz2(2:end,1:nz);

        txz(2:end,1:nz) =
txz(2:end,1:nz)+mu(2:end,1:nz).*dt.*(vzdx+Mxzzdx)+mu(2:end,1:nz).*dt.*(vxdz+M
xzxdz);

        Mphixzzdx2(2:end,1:nz) = RE2x(2:end,1:nz).*Mphixzzdx2(2:end,1:nz)-
RF2x(2:end,1:nz).*(RA1x(2:end,1:nz).*vzdx+RB1x(2:end,1:nz).*Mphixzzdx1(2:end,
1:nz));
        Mphixzxdz2(2:end,1:nz) = RE2z(2:end,1:nz).*Mphixzxdz2(2:end,1:nz)-
RF2z(2:end,1:nz).*(RA1z(2:end,1:nz).*vxdz+RB1z(2:end,1:nz).*Mphixzxdz1(2:end,
1:nz));
```

```
        Mphixzzdx1(2:end,1:nz) = RE1x(2:end,1:nz).*Mphixzzdx1(2:end,1:nz)-
RF1x(2:end,1:nz).*vzdx;
        Mphixzxdz1(2:end,1:nz) = RE1z(2:end,1:nz).*Mphixzxdz1(2:end,1:nz)-
RF1z(2:end,1:nz).*vxdz;

        % doing the same for tzz
        Mzzxdx = (RA1x(1:nx,2:end).*RA2x(1:nx,2:end)-
1).*vxdx+RA2x(1:nx,2:end).*RB1x(1:nx,2:end).*Mphizzxdx1(1:nx,2:end)+RB2x(1:nx
,2:end).*Mphizzxdx2(1:nx,2:end);
        Mzzzdz = (RA1z(1:nx,2:end).*RA2z(1:nx,2:end)-
1).*vzdz+RA2z(1:nx,2:end).*RB1z(1:nx,2:end).*Mphizzzdz1(1:nx,2:end)+RB2z(1:nx
,2:end).*Mphizzzdz2(1:nx,2:end);

        tzz(1:nx,2:end) =
tzz(1:nx,2:end)+lambda(1:nx,2:end).*dt.*(vxdx+Mzzxdx)+(lambda(1:nx,2:end)+2.*
mu(1:nx,2:end)).*dt.*(vzdz+Mzzzdz);

        Mphizzzdz2(1:nx,2:end) = RE2z(1:nx,2:end).*Mphizzzdz2(1:nx,2:end)-
RF2z(1:nx,2:end).*(RA1z(1:nx,2:end).*vzdz+RB1z(1:nx,2:end).*Mphizzzdz1(1:nx,2
:end));
        Mphizzxdx2(1:nx,2:end) = RE2x(1:nx,2:end).*Mphizzxdx2(1:nx,2:end)-
RF2x(1:nx,2:end).*(RA1x(1:nx,2:end).*vxdx+RB1x(1:nx,2:end).*Mphizzxdx1(1:nx,2
:end));
        Mphizzzdz1(1:nx,2:end) = RE1z(1:nx,2:end).*Mphizzzdz1(1:nx,2:end)-
RF1z(1:nx,2:end).*vzdz;
        Mphizzxdx1(1:nx,2:end) = RE1x(1:nx,2:end).*Mphizzxdx1(1:nx,2:end)-
RF1x(1:nx,2:end).*vxdx;

%         % plotting stress fields
%         if (mod(t,10)==5)
%         clf;
%         imagesc(tzz');
%         hold on
%         plot(irx1, irz1,'ob')
%         plot(irx2, irz2,'ob')
%         plot(irx3, irz3,'ob')
%         plot(isx, isz,'xb')
%         plot(npml:nx+1-npml,ones(1,numel(npml:nx+1-npml))*npml,'-r')
%         plot(npml:nx+1-npml,ones(1,numel(npml:nx+1-npml))*(nz+1-npml),'-r')
%         plot(ones(1,numel(npml:nz+1-npml))*npml,npml:nz+1-npml,'-r')
%         plot(ones(1,numel(npml:nz+1-npml))*(nx+1-npml),npml:nz+1-npml,'-r')
%         xlim([0 nx])
%         ylim([0 nz])
%         %caxis([-0.2 0.2]); % setting limits for colorbar
%         colorbar;
%         title(sprintf('Time %f ms',t*dt*1000));
%         pause(0.01)
%         end

        % storing the measured field variable at the different receiver
        % locations
        prvx1(t) = vx(irx1,irz1);
        prvz1(t) = vz(irx1,irz1);
        prtxx1(t) = txx(irx1,irz1);
        prtxz1(t) = txz(irx1,irz1);
        prtzz1(t) = tzz(irx1,irz1);
```

```
        prvx2(t) = vx(irx2,irz2);
        prvz2(t) = vz(irx2,irz2);
        prtxx2(t) = txx(irx2,irz2);
        prtxz2(t) = txz(irx2,irz2);
        prtzz2(t) = tzz(irx2,irz2);

        prvx3(t) = vx(irx3,irz3);
        prvz3(t) = vz(irx3,irz3);
        prtxx3(t) = txx(irx3,irz3);
        prtxz3(t) = txz(irx3,irz3);
        prtzz3(t) = tzz(irx3,irz3);
end
cpuend = cputime;
time = toc;
cputimespend = cpuend-cpustart;
```

A.5 MPML on RSG

```
% double pole MPML on RSG grid
clear all
close all
clc

% grid parameters
npml = 10; % number of PML cells
N = 20; % number of gridpoints per minimum wavelength
nx = 250; % number of cells in x-direction
nz = 50; % number of cells in y-direction
vp0 = 3000; % P-wave velocity in subsurface
vs0 = vp0/2; % S-wave velocity in subsurface
rho = 2000; % density in subsurface
buo0 = 1/rho; % buoyancy in the subsurface
nt = 1000; % number of time steps

% adding pml cells to nx and nz
nx = nx+2*npml;
nz = nz+2*npml;

% Source parameters
fc = 10; % center frequency Ricker-wavelet
isx = int16(nx/2); % source x-position in gridpoints
isz = int16(nz/2); % source y-position in gridpoints

% Receiver locations
irx1 = isx + 5;
irz1 = isz;
irx2 = isx + 20;
irz2 = isz;
irx3 = isx + 10;
irz3 = isz + 10;

% Lame's constants
```

```
mu0 = rho*vs0*vs0; % Lame's second parameter
lambda0 = rho*vp0*vp0-2*mu0; % Lame's first parameter

% Stability criteria
dx = vs0/(N*fc); % stabilty for spatial frame
dz = dx;
dt = 0.99/(vp0*sqrt(1/dx*1/dx+1/dz*1/dz)); % stability for time frame

% Source Ricker wavelet
tsour=1/fc;
t = (0:nt-1)*dt;
t0=tsour*1.5;
T0=tsour*1.5;
tau=pi*(t-t0)/T0;
a=4;
fs=(1-a*tau.*tau).*exp(-2*tau.*tau);
fs = fs/(dx*dz); % dividing the source by the gridsize to get the stress
ss = 1; % spreading of the source to prevent chequerboard pattern

% PML parameters
pmlfac = 2; % order of polynomal scaling
R = 10^(-((log10(npml)-1)/log10(2))-3); % reflection coefficient for a normal
incident P-wave
dmax1 = -(3*vp0)/(2*npml*dx)*log(R); % d1j attenuation coefficient
kappamax1 = 2*(vs0/(dx*fc))/N; % k1j attenuation coefficient
alfamax1 = pi*fc; % alfa1j attenuation coefficient
dmax2 = dmax1/100; % dmax2j attenuation coefficient
kappamax2 = kappamax1*15; % kappamax2j attenuation coefficient
alfamax2 = alfamax1*200; % alfamax2j attenuation coefficient

% creating stress and velocity matrices
vx = zeros(nx+1,nz+1); % velocity matrix x-direction
vz = zeros(nx+1,nz+1); % velocity matrix y-direction
txx = zeros(nx+1,nz+1); % stress matrix xx
tzz = zeros(nx+1,nz+1); % stress matrix zz
txz = zeros(nx+1,nz+1); % stress matrix xz
lambda = zeros(nx+1,nz+1); % Lame's first parameter matrix
mu = zeros(nx+1,nz+1); % Lame's second parameter matrix
buo = zeros(nx+1,nz+1); % buoyancy in the matrix
lambda(:,:) = lambda0; % filling the lambda matrix with the constant
mu(:,:) = mu0; % filling the mu matrix with the constant
buo(:,:) = buo0; % filling the buo matrix with the constant

% PML correction term matrices
RAx = ones(nx+1,nz+1); % RAx matrix
RB1x = ones(nx+1,nz+1); % RB1x matrix
RE1x = ones(nx+1,nz+1); % RE1x matrix
RF1x = zeros(nx+1,nz+1); % RF1x matrix
RB2x = ones(nx+1,nz+1); % RB2x matrix
RE2x = ones(nx+1,nz+1); % RE2x matrix
RF2x = zeros(nx+1,nz+1); % RF2x matrix
% PML correction term matrices
RAz = ones(nx+1,nz+1); % RAz matrix
RB1z = ones(nx+1,nz+1); % RB1z matrix
RE1z = ones(nx+1,nz+1); % RE1z matrix
RF1z = zeros(nx+1,nz+1); % RF2z matrix
```

```
RB2z = ones(nx+1,nz+1); % RB2z matrix
RE2z = ones(nx+1,nz+1); % RE2z matrix
RF2z = zeros(nx+1,nz+1); % RF2z matrix

% vectors to store receiver seismograms
prvx1 = zeros(1,nt);
prvz1 = zeros(1,nt);
prtxx1 = zeros(1,nt);
prtxz1 = zeros(1,nt);
prtzz1 = zeros(1,nt);

prvx2 = zeros(1,nt);
prvz2 = zeros(1,nt);
prtxx2 = zeros(1,nt);
prtxz2 = zeros(1,nt);
prtzz2 = zeros(1,nt);

prvx3 = zeros(1,nt);
prvz3 = zeros(1,nt);
prtxx3 = zeros(1,nt);
prtxz3 = zeros(1,nt);
prtzz3 = zeros(1,nt);

% filling PML matrices using the maximum attenuation coefficients
% different attenuation functions can be used if wanted
for i = 1:npml
        d1 = (dmax1*(i/npml)^pmlfac);
        k1 = (1+(kappamax1-1)*(i/npml)^pmlfac);
        a1 = alfamax1*(1-(i/npml));
        RB1x(npml+1-i,:) = 2/(2+a1*dt);
        RB1z(:,npml+1-i) = 2/(2+a1*dt);
        RE1x(npml+1-i,:) = (2-a1*dt)/(2+a1*dt);
        RE1z(:,npml+1-i) = (2-a1*dt)/(2+a1*dt);
        RF1x(npml+1-i,:) = (2*dt*d1)/(2+a1*dt);
        RF1z(:,npml+1-i) = (2*dt*d1)/(2+a1*dt);
        d2 = (dmax2*(i/npml)^pmlfac);
        k2 = (1+(kappamax2-1)*(i/npml)^pmlfac);
        a2 = alfamax2*(1-(i/npml));
        RAx(npml+1-i,:) = k1+(d1*dt)/(2+a1*dt)+(d2*dt)/(2+a2*dt);
        RAz(:,npml+1-i) = k1+(d1*dt)/(2+a1*dt)+(d2*dt)/(2+a2*dt);
        RB2x(npml+1-i,:) = 2/(2+a2*dt);
        RB2z(:,npml+1-i) = 2/(2+a2*dt);
        RE2x(npml+1-i,:) = (2-a2*dt)/(2+a2*dt);
        RE2z(:,npml+1-i) = (2-a2*dt)/(2+a2*dt);
        RF2x(npml+1-i,:) = (2*dt*d2)/(2+a2*dt);
        RF2z(:,npml+1-i) = (2*dt*d2)/(2+a2*dt);
end

 % filling all the PML zones with the correct attenuation value
RAx(end-npml+1:end,:) = RAx(npml:-1:1,:);
RAz(:,end-npml+1:end) = RAz(:,npml:-1:1);
RB1x(end-npml+1:end,:) = RB1x(npml:-1:1,:);
RB1z(:,end-npml+1:end) = RB1z(:,npml:-1:1);
RE1x(end-npml+1:end,:) = RE1x(npml:-1:1,:);
RE1z(:,end-npml+1:end) = RE1z(:,npml:-1:1);
RF1x(end-npml+1:end,:) = RF1x(npml:-1:1,:);
```

```
RF1z(:,end-npml+1:end) = RF1z(:,npml:-1:1);
RB2x(end-npml+1:end,:) = RB2x(npml:-1:1,:);
RB2z(:,end-npml+1:end) = RB2z(:,npml:-1:1);
RE2x(end-npml+1:end,:) = RE2x(npml:-1:1,:);
RE2z(:,end-npml+1:end) = RE2z(:,npml:-1:1);
RF2x(end-npml+1:end,:) = RF2x(npml:-1:1,:);
RF2z(:,end-npml+1:end) = RF2z(:,npml:-1:1);

% creating the memory matrices phi for the velocity and stress fields
Jphixzdz1 = zeros(nx+1,nz+1);
Jphixxdx1 = zeros(nx+1,nz+1);
Jphixzdx1 = zeros(nx+1,nz+1);
Jphizzdz1 = zeros(nx+1,nz+1);
Mphixxzdz1 = zeros(nx+1,nz+1);
Mphixxxdx1 = zeros(nx+1,nz+1);
Mphixzzdx1 = zeros(nx+1,nz+1);
Mphixzxdz1 = zeros(nx+1,nz+1);
Mphizzzdz1 = zeros(nx+1,nz+1);
Mphizzxdx1 = zeros(nx+1,nz+1);
Jphixzdz2 = zeros(nx+1,nz+1);
Jphixxdx2 = zeros(nx+1,nz+1);
Jphixzdx2 = zeros(nx+1,nz+1);
Jphizzdz2 = zeros(nx+1,nz+1);
Mphixxzdz2 = zeros(nx+1,nz+1);
Mphixxxdx2 = zeros(nx+1,nz+1);
Mphixzzdx2 = zeros(nx+1,nz+1);
Mphixzxdz2 = zeros(nx+1,nz+1);
Mphizzzdz2 = zeros(nx+1,nz+1);
Mphizzxdx2 = zeros(nx+1,nz+1);

%% starting loop

tic % timing the computation time
cpustart = cputime; % timing the cputime

for t = 2:nt

        % adding source to the stress fields
        % in order to prevent a chequerboard pattern the source has the be
        % spread out since we are not using a rotated staggered grid
         for tsx = isx-ss:isx+ss
                for tsz = isz-ss:isz+ss
                        distancex = double(isx-tsx)*dx;
                        distancez = double(isz-tsz)*dz;
                        distance =
sqrt(distancex*distancex+distancez*distancez);
                        tzz(tsx,tsz) = tzz(tsx,tsz)+fs(t)*dt/rho*exp(-
0.5*(distance/dx)^2);
                        txx(tsx,tsz) = txx(tsx,tsz)+fs(t)*dt/rho*exp(-
0.5*(distance/dx)^2);
                end
         end

        % updating vx
        % correction term update
```

```
        Jphixzdzchange =
RB1z(2:end,2:end).*Jphixzdz1(2:end,2:end)+RB2z(2:end,2:end).*Jphixzdz2(2:end,
2:end);
        Jphixxdxchange =
RB1x(2:end,2:end).*Jphixxdx1(2:end,2:end)+RB2x(2:end,2:end).*Jphixxdx2(2:end,
2:end);

        % spatial derivatives
        txxdx = (txx(2:end,2:end)-txx(1:nx,1:nz)+txx(2:end,1:nz)-
txx(1:nx,2:end))./(2*dx);
        txzdz = (txz(2:end,2:end)-txz(1:nx,1:nz)+txz(1:nx,2:end)-
txz(2:end,1:nz))./(2*dz);

        % correction term calculation
        Jxzdz = (1./RAz(2:end,2:end)-1).*txzdz-
(1./RAz(2:end,2:end).*Jphixzdzchange);
        Jxdx = (1./RAx(2:end,2:end)-1).*txxdx-
(1./RAx(2:end,2:end).*Jphixxdxchange);

        % updating vx
        vx(2:end,2:end) =
vx(2:end,2:end)+buo(2:end,2:end).*dt.*(txxdx+Jxdx)+buo(2:end,2:end).*dt.*(txz
dz+Jxzdz);

        % updating phi
        Jphixzdz1(2:end,2:end) =
RE1z(2:end,2:end).*Jphixzdz1(2:end,2:end)+RF1z(2:end,2:end)./RAz(2:end,2:end)
.*(txzdz-Jphixzdzchange);
        Jphixxdx1(2:end,2:end) =
RE1x(2:end,2:end).*Jphixxdx1(2:end,2:end)+RF1x(2:end,2:end)./RAx(2:end,2:end)
.*(txxdx-Jphixxdxchange);
        Jphixzdz2(2:end,2:end) =
RE2z(2:end,2:end).*Jphixzdz2(2:end,2:end)+RF2z(2:end,2:end)./RAz(2:end,2:end)
.*(txzdz-Jphixzdzchange);
        Jphixxdx2(2:end,2:end) =
RE2x(2:end,2:end).*Jphixxdx2(2:end,2:end)+RF2x(2:end,2:end)./RAx(2:end,2:end)
.*(txxdx-Jphixxdxchange);

        % doing the same for vz
        Jphixzdxchange =
RB1x(2:end,2:end).*Jphixzdx1(2:end,2:end)+RB2x(2:end,2:end).*Jphixzdx2(2:end,
2:end);
        Jphizzdzchange =
RB1z(2:end,2:end).*Jphizzdz1(2:end,2:end)+RB2z(2:end,2:end).*Jphizzdz2(2:end,
2:end);

        tzzdz = (tzz(2:end,2:end)-tzz(1:nx,1:nz)+tzz(1:nx,2:end)-
tzz(2:end,1:nz))./(2*dz);
        txzdx = (txz(2:end,2:end)-txz(1:nx,1:nz)+txz(2:end,1:nz)-
txz(1:nx,2:end))./(2*dx);

        Jxzdx = (1./RAx(2:end,2:end)-1).*txzdx-
(1./RAx(2:end,2:end).*Jphixzdxchange);
        Jzdz = (1./RAz(2:end,2:end)-1).*tzzdz-
(1./RAz(2:end,2:end).*Jphizzdzchange);
```

```
        vz(2:end,2:end) =
vz(2:end,2:end)+buo(2:end,2:end).*dt.*(tzzdz+Jzdz)+buo(2:end,2:end).*dt.*(txz
dx+Jxzdx);

        Jphixzdx1(2:end,2:end) =
RE1x(2:end,2:end).*Jphixzdx1(2:end,2:end)+RF1x(2:end,2:end)./RAx(2:end,2:end)
.*(txzdx-Jphixzdxchange);
        Jphizzdz1(2:end,2:end) =
RE1z(2:end,2:end).*Jphizzdz1(2:end,2:end)+RF1z(2:end,2:end)./RAz(2:end,2:end)
.*(tzzdz-Jphizzdzchange);
        Jphixzdx2(2:end,2:end) =
RE2x(2:end,2:end).*Jphixzdx2(2:end,2:end)+RF2x(2:end,2:end)./RAx(2:end,2:end)
.*(txzdx-Jphixzdxchange);
        Jphizzdz2(2:end,2:end) =
RE2z(2:end,2:end).*Jphizzdz2(2:end,2:end)+RF2z(2:end,2:end)./RAz(2:end,2:end)
.*(tzzdz-Jphizzdzchange);

        % doing the same for txx
        Mphixxzdzchange =
RB1z(1:nx,1:nz).*Mphixxzdz1(1:nx,1:nz)+RB2z(1:nx,1:nz).*Mphixxzdz2(1:nx,1:nz)
;
        Mphixxxdxchange =
RB1x(1:nx,1:nz).*Mphixxxdx1(1:nx,1:nz)+RB2x(1:nx,1:nz).*Mphixxxdx2(1:nx,1:nz)
;

        vxdx = (vx(2:end,2:end)-vx(1:nx,1:nz)+vx(2:end,1:nz)-
vx(1:nx,2:end))./(2*dx);
        vzdz = (vz(2:end,2:end)-vz(1:nx,1:nz)+vz(1:nx,2:end)-
vz(2:end,1:nz))./(2*dz);

        Mxxzdz = (1./RAz(1:nx,1:nz)-1).*vzdz-
(1./RAz(1:nx,1:nz).*Mphixxzdzchange);
        Mxxdx = (1./RAx(1:nx,1:nz)-1).*vxdx-
(1./RAx(1:nx,1:nz).*Mphixxxdxchange);

        txx(1:nx,1:nz) =
txx(1:nx,1:nz)+(lambda(1:nx,1:nz)+2.*mu(1:nx,1:nz)).*dt.*(vxdx+Mxxdx)+lambda(
1:nx,1:nz).*dt.*(vzdz+Mxxzdz);

        Mphixxzdz1(1:nx,1:nz) =
RE1z(1:nx,1:nz).*Mphixxzdz1(1:nx,1:nz)+RF1z(1:nx,1:nz)./RAz(1:nx,1:nz).*(vzdz
-Mphixxzdzchange);
        Mphixxxdx1(1:nx,1:nz) =
RE1x(1:nx,1:nz).*Mphixxxdx1(1:nx,1:nz)+RF1x(1:nx,1:nz)./RAx(1:nx,1:nz).*(vxdx
-Mphixxxdxchange);
        Mphixxzdz2(1:nx,1:nz) =
RE2z(1:nx,1:nz).*Mphixxzdz2(1:nx,1:nz)+RF2z(1:nx,1:nz)./RAz(1:nx,1:nz).*(vzdz
-Mphixxzdzchange);
        Mphixxxdx2(1:nx,1:nz) =
RE2x(1:nx,1:nz).*Mphixxxdx2(1:nx,1:nz)+RF2x(1:nx,1:nz)./RAx(1:nx,1:nz).*(vxdx
-Mphixxxdxchange);

        % doing the same for txz
```

```
        Mphixzzdxchange =
RB1x(1:nx,1:nz).*Mphixzzdx1(1:nx,1:nz)+RB2x(1:nx,1:nz).*Mphixzzdx2(1:nx,1:nz)
;
        Mphixzxdzchange =
RB1z(1:nx,1:nz).*Mphixzxdz1(1:nx,1:nz)+RB2z(1:nx,1:nz).*Mphixzxdz2(1:nx,1:nz)
;

        vxdz = (vx(2:end,2:end)-vx(1:nx,1:nz)+vx(1:nx,2:end)-
vx(2:end,1:nz))./(2*dz);
        vzdx = (vz(2:end,2:end)-vz(1:nx,1:nz)+vz(2:end,1:nz)-
vz(1:nx,2:end))./(2*dx);

        Mxzzdx = (1./RAx(1:nx,1:nz)-1).*vzdx-
(1./RAx(1:nx,1:nz).*Mphixzzdxchange);
        Mxzxdz = (1./RAz(1:nx,1:nz)-1).*vxdz-
(1./RAz(1:nx,1:nz).*Mphixzxdzchange);

        txz(1:nx,1:nz) =
txz(1:nx,1:nz)+mu(1:nx,1:nz).*dt.*(vzdx+Mxzzdx)+mu(1:nx,1:nz).*dt.*(vxdz+Mxzx
dz);

        Mphixzzdx1(1:nx,1:nz) =
RE1x(1:nx,1:nz).*Mphixzzdx1(1:nx,1:nz)+RF1x(1:nx,1:nz)./RAx(1:nx,1:nz).*(vzdx
-Mphixzzdxchange);
        Mphixzxdz1(1:nx,1:nz) =
RE1z(1:nx,1:nz).*Mphixzxdz1(1:nx,1:nz)+RF1z(1:nx,1:nz)./RAz(1:nx,1:nz).*(vxdz
-Mphixzxdzchange);
        Mphixzzdx2(1:nx,1:nz) =
RE2x(1:nx,1:nz).*Mphixzzdx2(1:nx,1:nz)+RF2x(1:nx,1:nz)./RAx(1:nx,1:nz).*(vzdx
-Mphixzzdxchange);
        Mphixzxdz2(1:nx,1:nz) =
RE2z(1:nx,1:nz).*Mphixzxdz2(1:nx,1:nz)+RF2z(1:nx,1:nz)./RAz(1:nx,1:nz).*(vxdz
-Mphixzxdzchange);

        % doing the same for tzz
        Mphizzzdzchange =
RB1z(1:nx,1:nz).*Mphizzzdz1(1:nx,1:nz)+RB2z(1:nx,1:nz).*Mphizzzdz2(1:nx,1:nz)
;
        Mphizzxdxchange =
RB1x(1:nx,1:nz).*Mphizzxdx1(1:nx,1:nz)+RB2x(1:nx,1:nz).*Mphizzxdx2(1:nx,1:nz)
;

        Mzzxdx = (1./RAx(1:nx,1:nz)-1).*vxdx-
(1./RAx(1:nx,1:nz).*Mphizzxdxchange);
        Mzzzdz = (1./RAz(1:nx,1:nz)-1).*vzdz-
(1./RAz(1:nx,1:nz).*Mphizzzdzchange);

        tzz(1:nx,1:nz) =
tzz(1:nx,1:nz)+lambda(1:nx,1:nz).*dt.*(vxdx+Mzzxdx)+(lambda(1:nx,1:nz)+2.*mu(
1:nx,1:nz)).*dt.*(vzdz+Mzzzdz);

        Mphizzzdz1(1:nx,1:nz) =
RE1z(1:nx,1:nz).*Mphizzzdz1(1:nx,1:nz)+RF1z(1:nx,1:nz)./RAz(1:nx,1:nz).*(vzdz
-Mphizzzdzchange);
```

```
        Mphizzxdx1(1:nx,1:nz) =
RE1x(1:nx,1:nz).*Mphizzxdx1(1:nx,1:nz)+RF1x(1:nx,1:nz)./RAx(1:nx,1:nz).*(vxdx
-Mphizzxdxchange);

%          % plotting stress fields
%          if (mod(t,10)==5)
%          clf;
%          imagesc(tzz');
%          hold on
%          plot(irx1, irz1,'ob')
%          plot(irx2, irz2,'ob')
%          plot(irx3, irz3,'ob')
%          plot(isx, isz,'xb')
%          plot(npml:nx+1-npml,ones(1,numel(npml:nx+1-npml))*npml,'-r')
%          plot(npml:nx+1-npml,ones(1,numel(npml:nx+1-npml))*(nz+1-npml),'-r')
%          plot(ones(1,numel(npml:nz+1-npml))*npml,npml:nz+1-npml,'-r')
%          plot(ones(1,numel(npml:nz+1-npml))*(nx+1-npml),npml:nz+1-npml,'-r')
%          xlim([0 nx])
%          ylim([0 nz])
% %          caxis([-4e-10 4e-10]); % setting limits for colorbar
%          colorbar;
%          title(sprintf('Time %f ms',t*dt*1000));
%          pause(0.01)
%          end

        % storing the measured field variable at the different receiver
        % locations
        prvx1(t) = vx(irx1,irz1);
        prvz1(t) = vz(irx1,irz1);
        prtxx1(t) = txx(irx1,irz1);
        prtxz1(t) = txz(irx1,irz1);
        prtzz1(t) = tzz(irx1,irz1);

        prvx2(t) = vx(irx2,irz2);
        prvz2(t) = vz(irx2,irz2);
        prtxx2(t) = txx(irx2,irz2);
        prtxz2(t) = txz(irx2,irz2);
        prtzz2(t) = tzz(irx2,irz2);

        prvx3(t) = vx(irx3,irz3);
        prvz3(t) = vz(irx3,irz3);
        prtxx3(t) = txx(irx3,irz3);
        prtxz3(t) = txz(irx3,irz3);
        prtzz3(t) = tzz(irx3,irz3);
end
cpuend = cputime;
time = toc;
cputimespend = cpuend-cpustart;
```

Appendix B. Figures

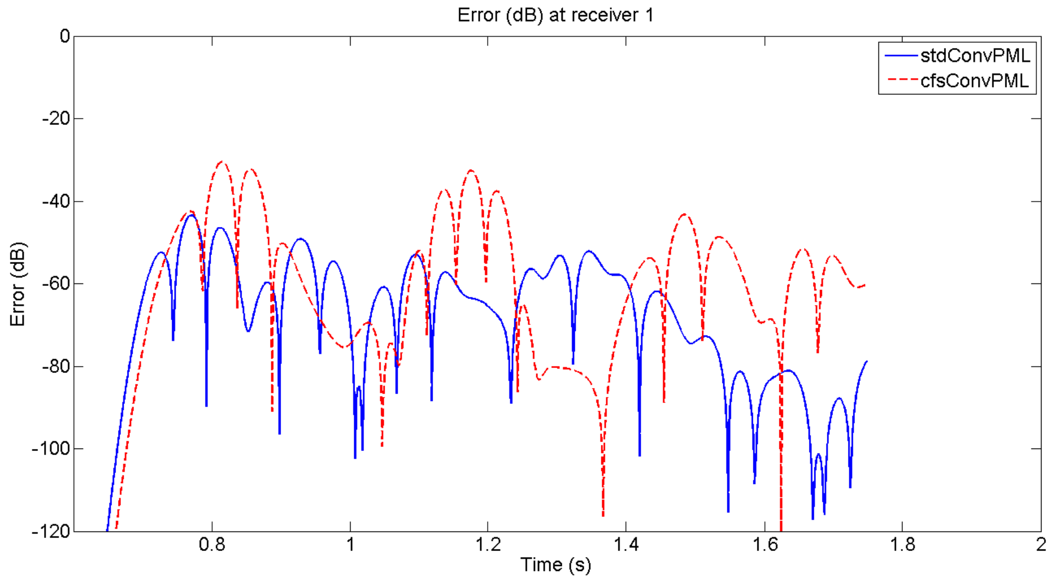


Figure B.1: Error in dB for two different types of PML's on the square model at receiver 1.

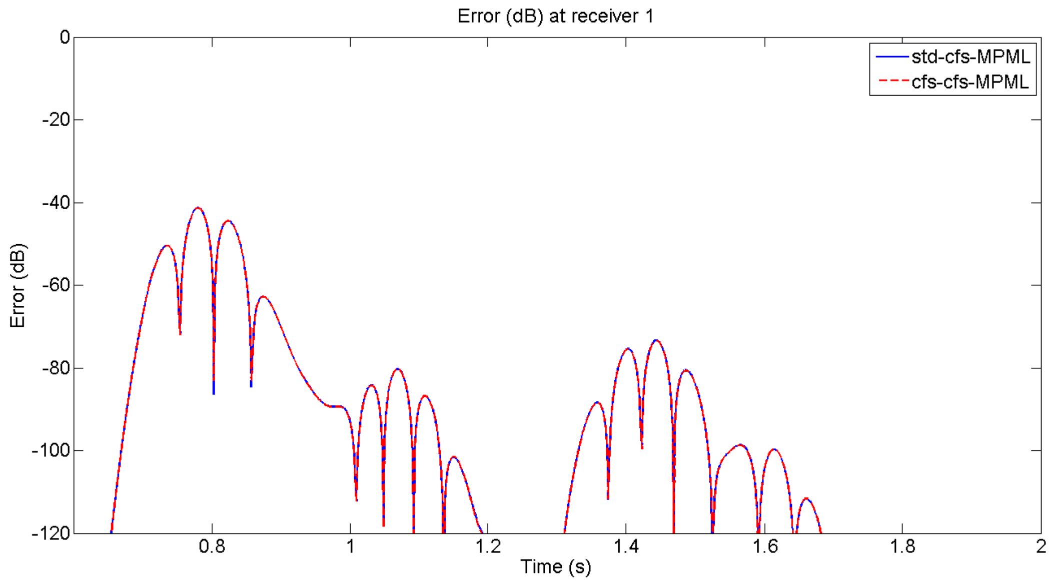


Figure B.2: Error in dB for two different types of PML's on the square model at receiver 1.

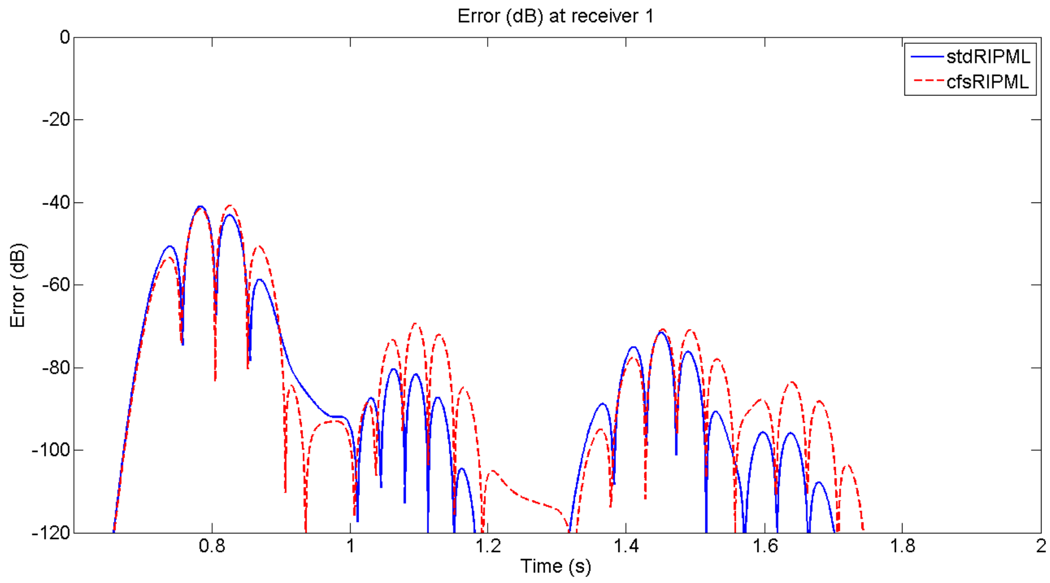


Figure B.3: Error in dB for two different types of PML's on the square model at receiver 1.

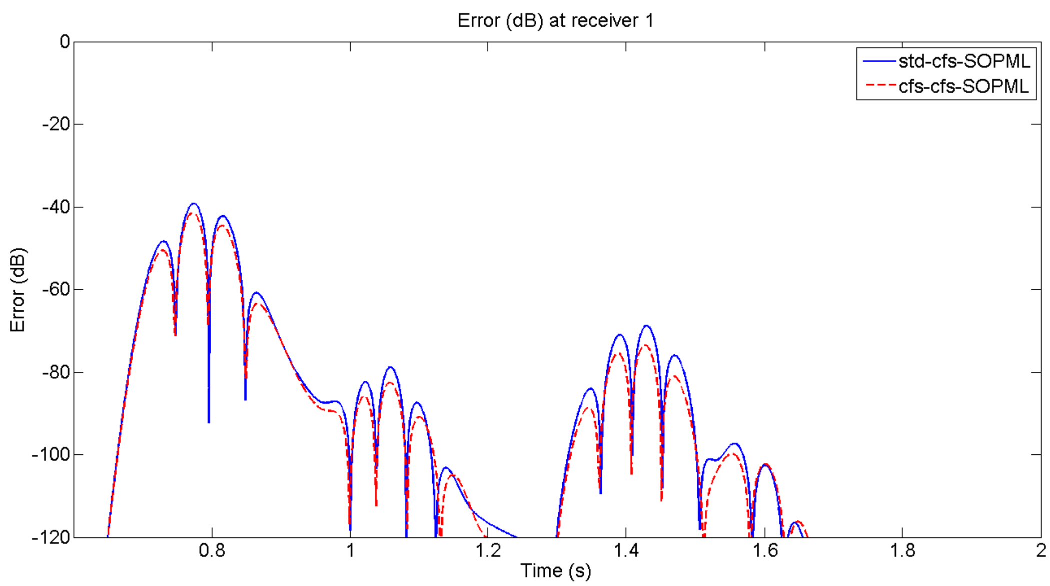


Figure B.4: Error in dB for two different types of PML's on the square model at receiver 1.

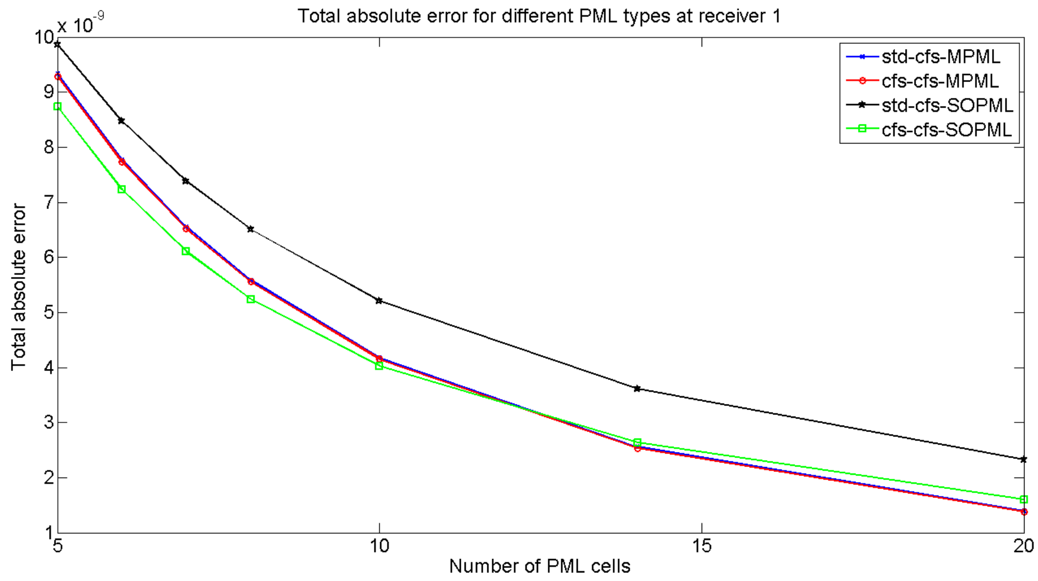


Figure B.5: Total absolute error for different numbers of PML cells at receiver 1 for different PML types on the square model.

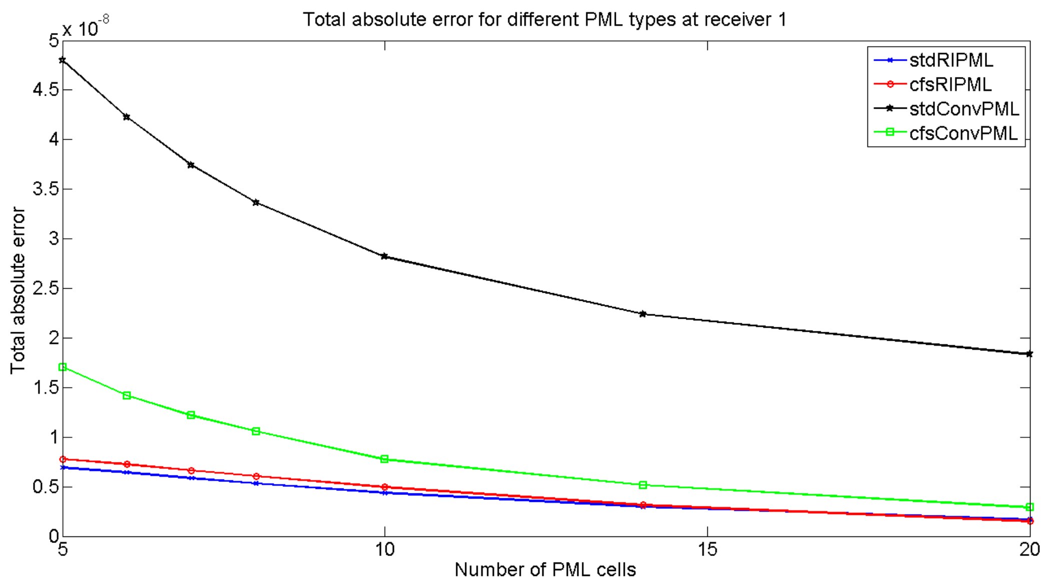


Figure B.6: Total absolute error for different numbers of PML cells at receiver 1 for different PML types on the square model.

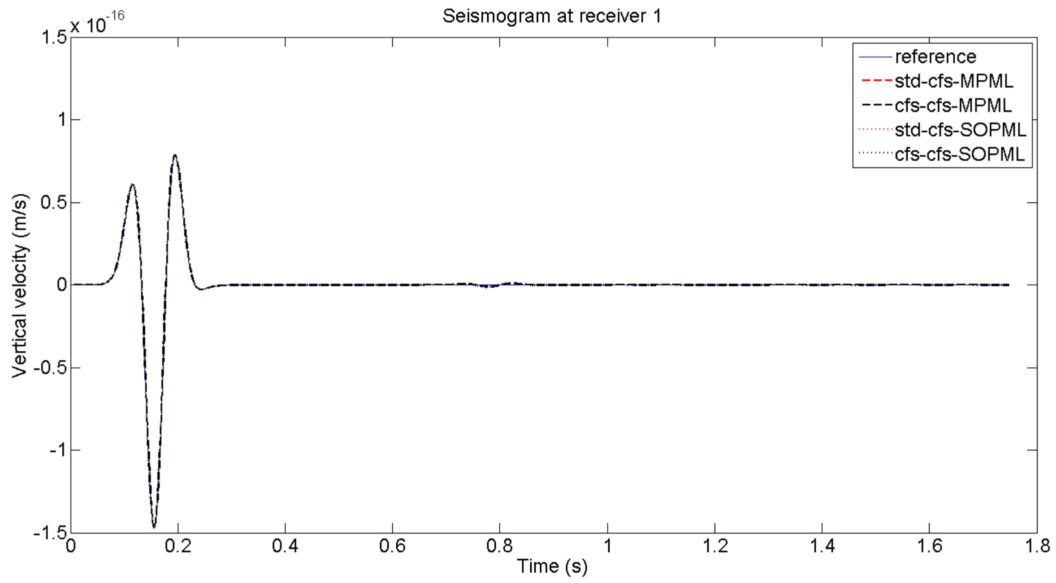


Figure B.7: Seismogram at receiver 1 for different PML types on the square model.

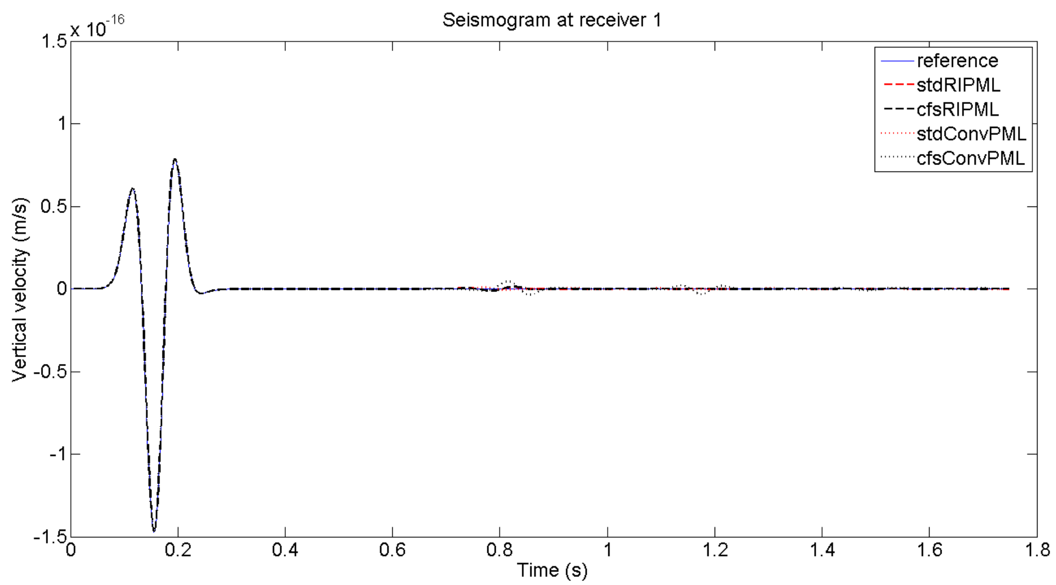


Figure B.8: Seismogram at receiver 1 for different PML types on the square model.

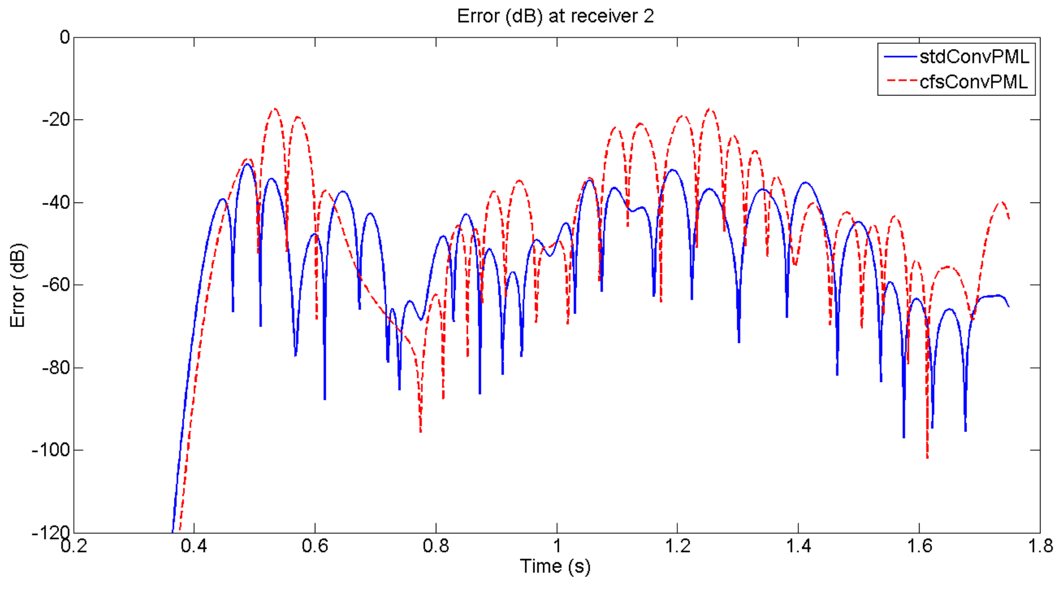


Figure B.9: Error in dB for two different types of PML's on the square model at receiver 2.

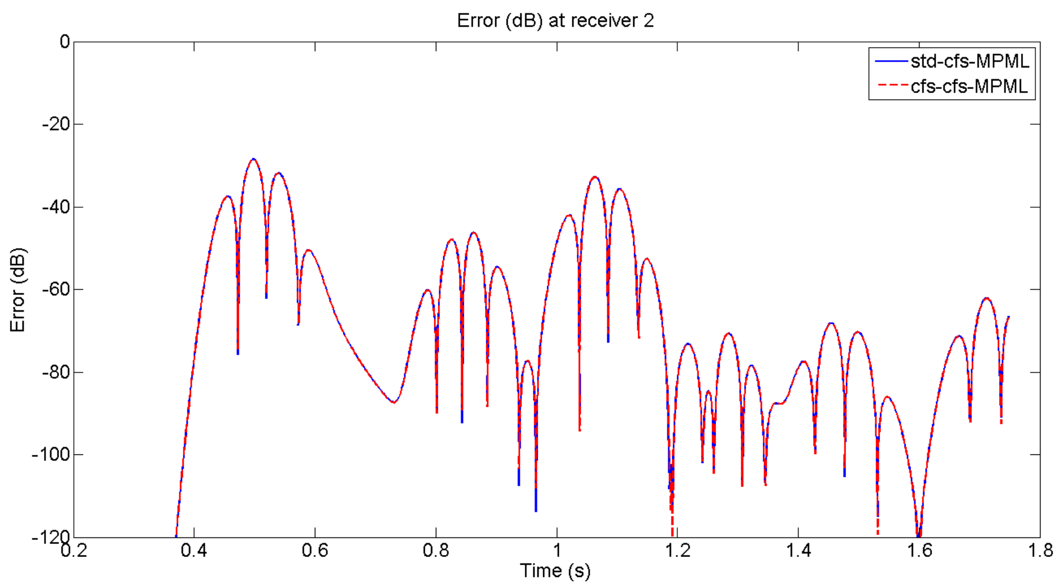


Figure B.10: Error in dB for two different types of PML's on the square model at receiver 2.

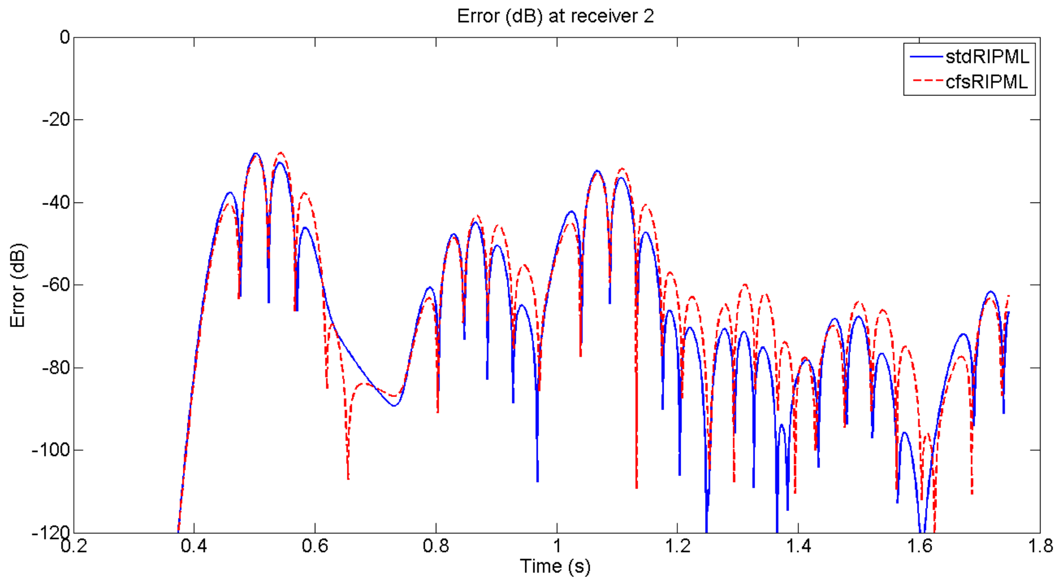


Figure B.11: Error in dB for two different types of PML's on the square model at receiver 2.

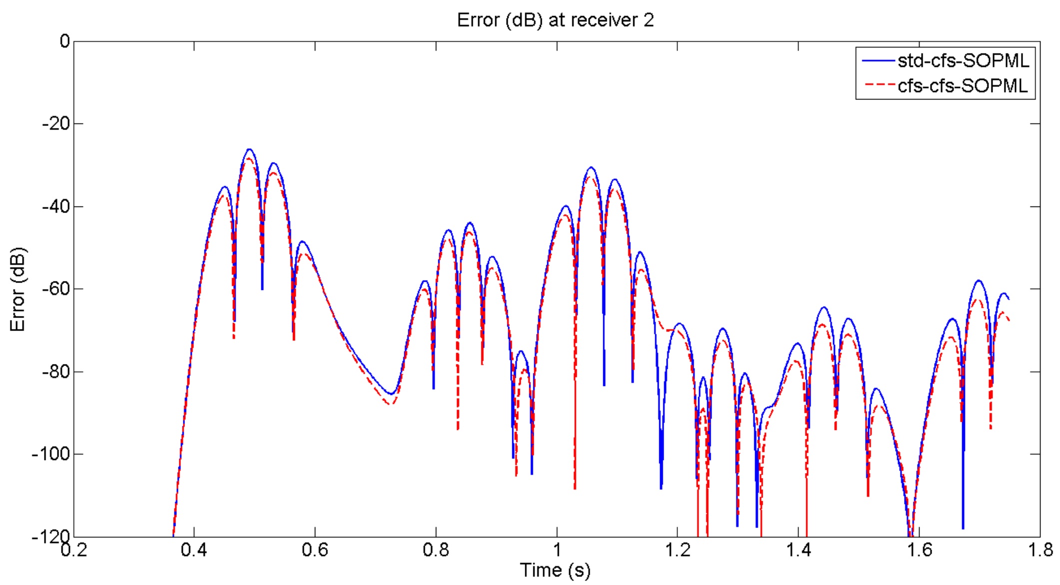


Figure B.12: Error in dB for two different types of PML's on the square model at receiver 2.

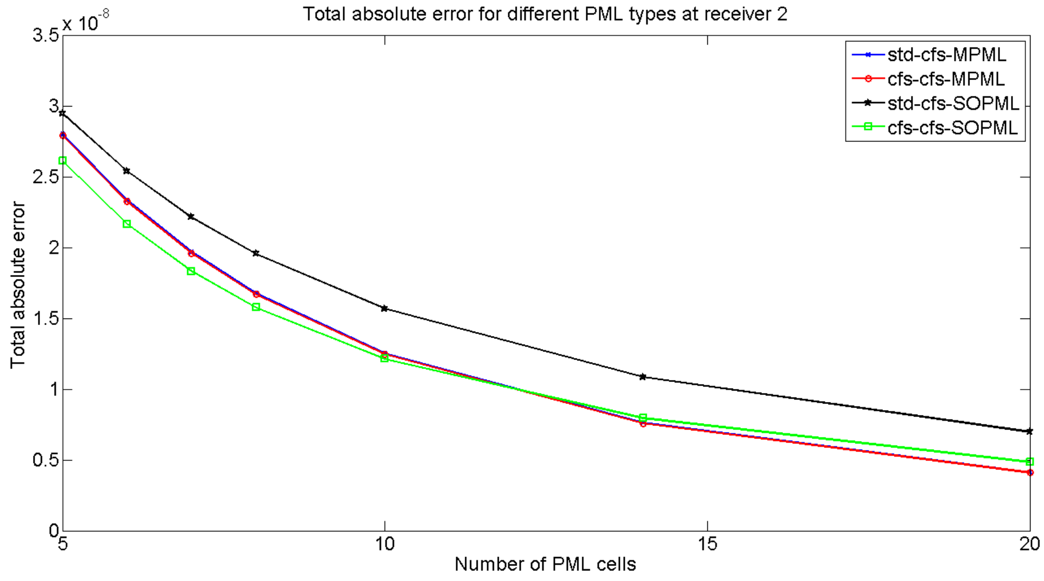


Figure B.13: Total absolute error for different number of PML cells at receiver 2 on the square model.

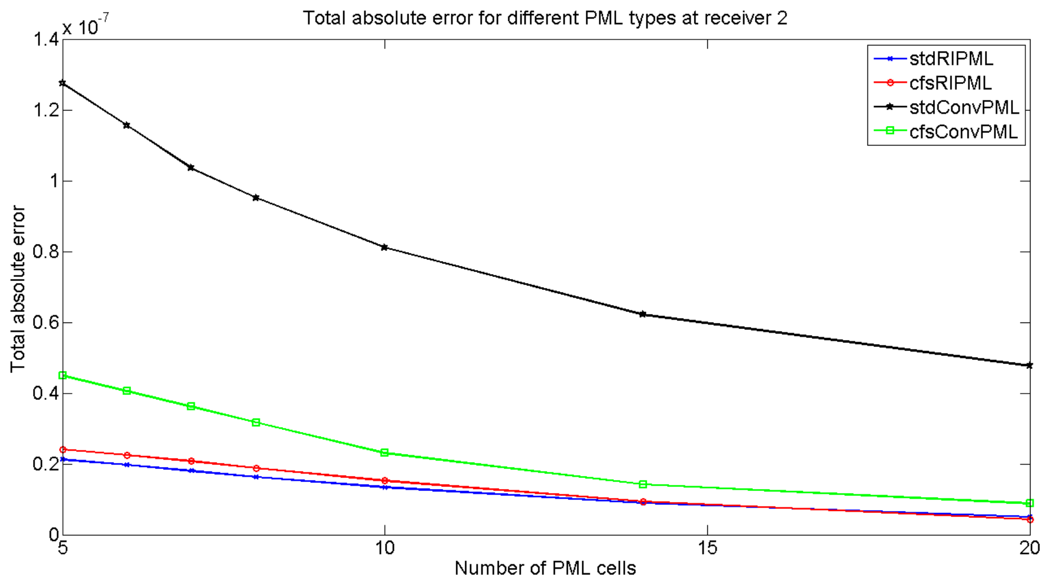


Figure B.14: Total absolute error for different number of PML cells at receiver 2 on the square model.

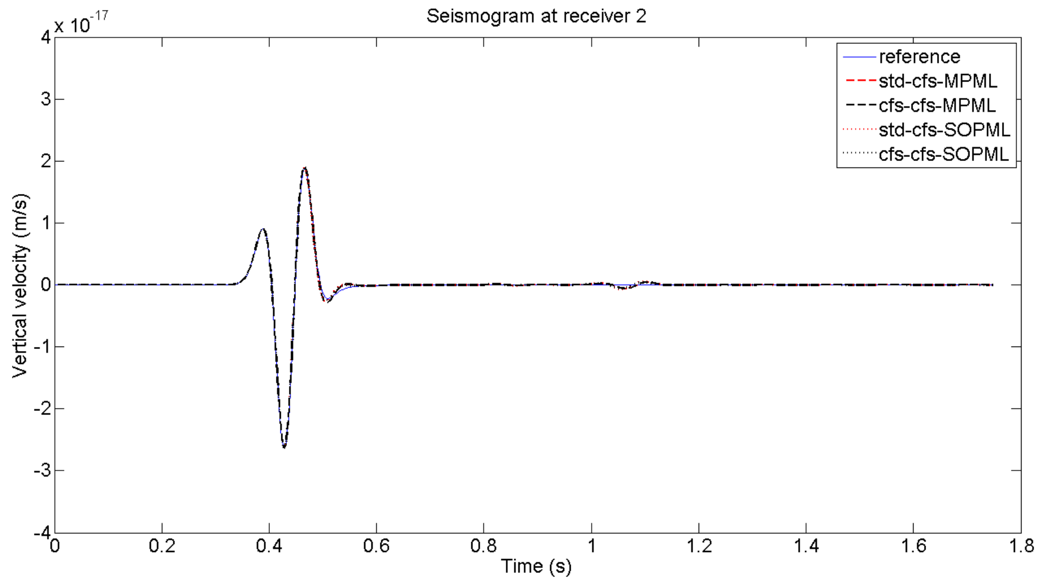


Figure B.15: Seismogram for different PML types at receiver 2 on the square model.

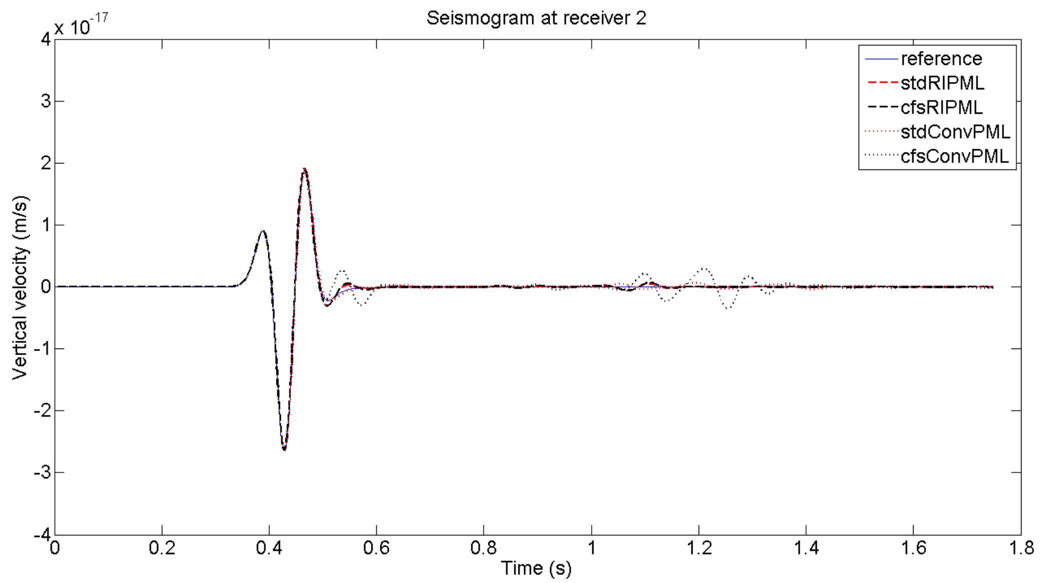


Figure B.16: Seismogram for different PML types at receiver 2 on the square model.

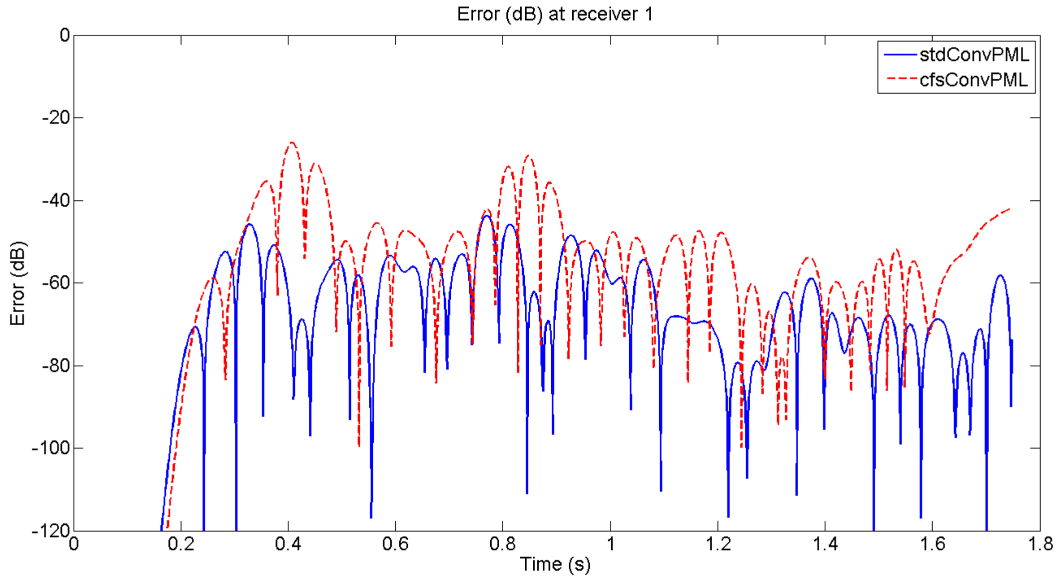


Figure B.17: Error in dB for two different types of PML's on the rectangular model at receiver 1.

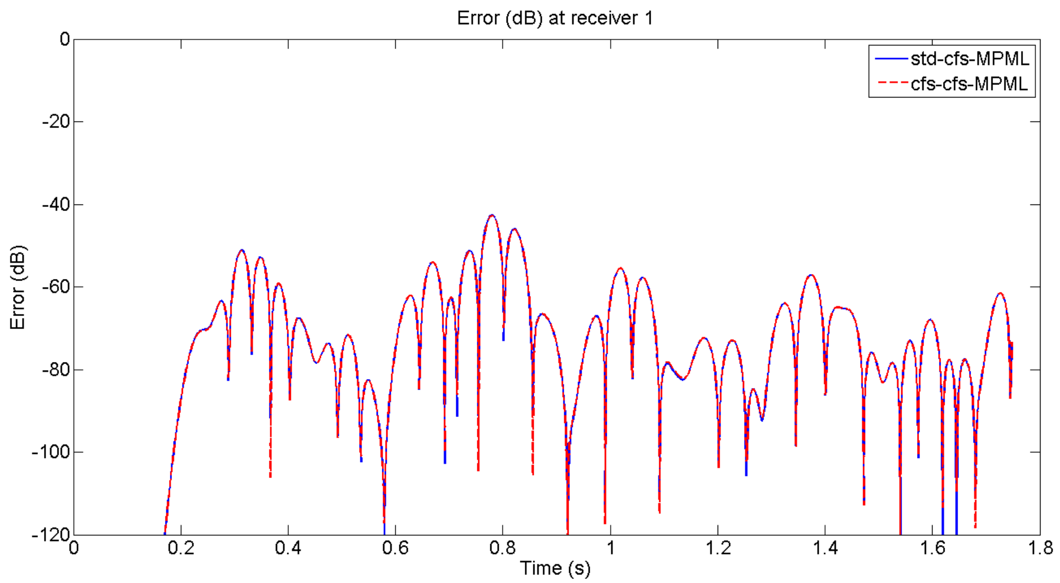


Figure B.18: Error in dB for two different types of PML's on the rectangular model at receiver 1.

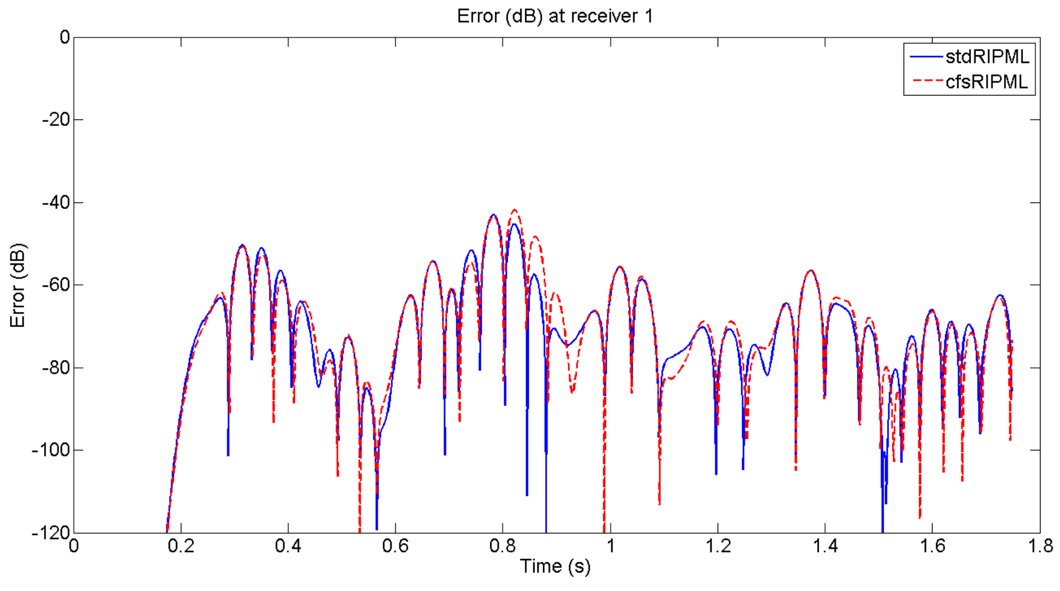


Figure B.19: Error in dB for two different types of PML's on the rectangular model at receiver 1.

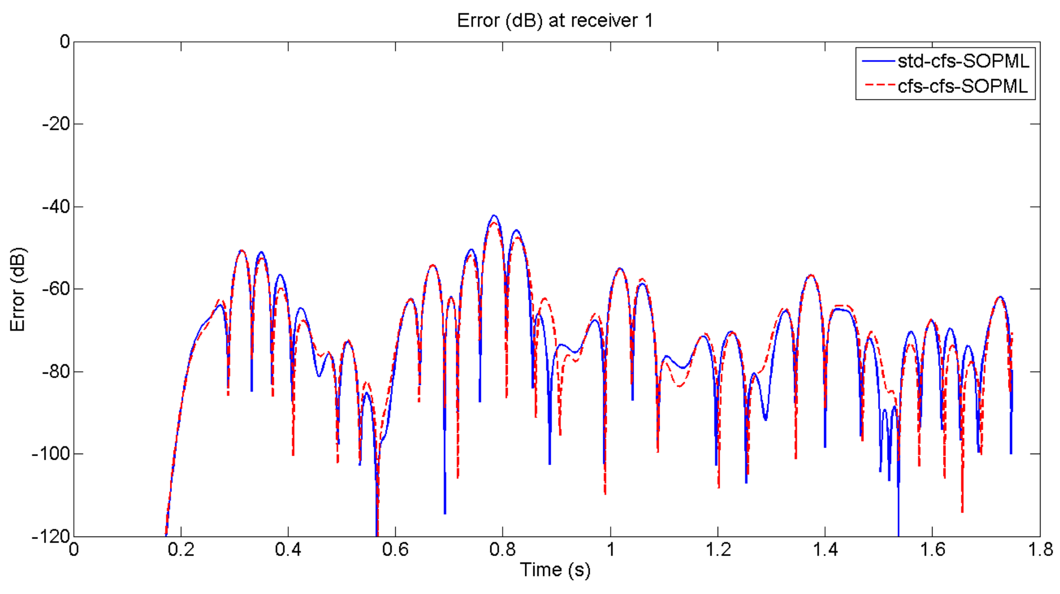


Figure B.20: Error in dB for two different types of PML's on the rectangular model at receiver 1.

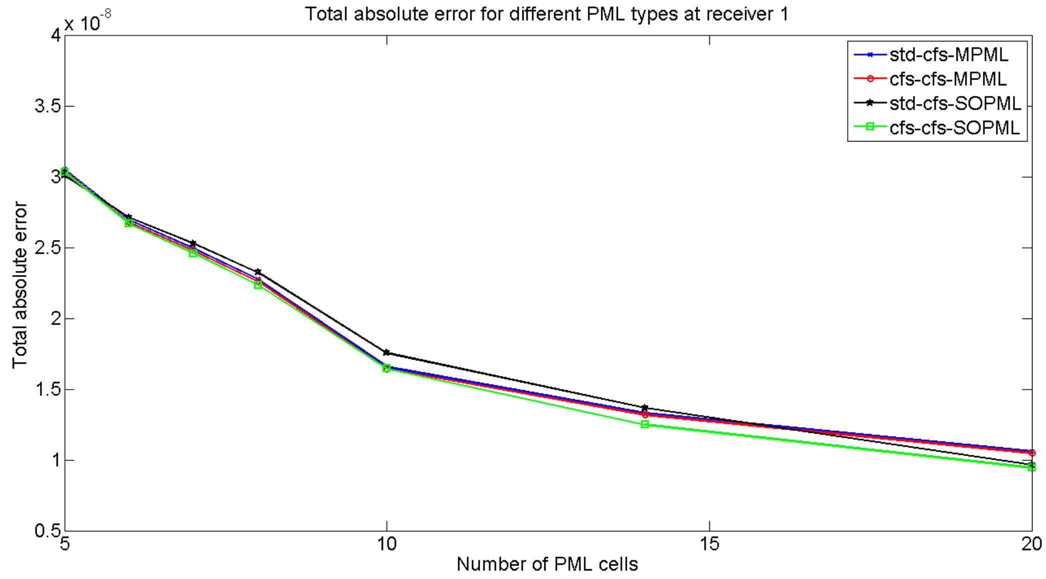


Figure B.21: Total absolute error for different numbers of PML cells at receiver 1 for different PML types on the rectangular model.

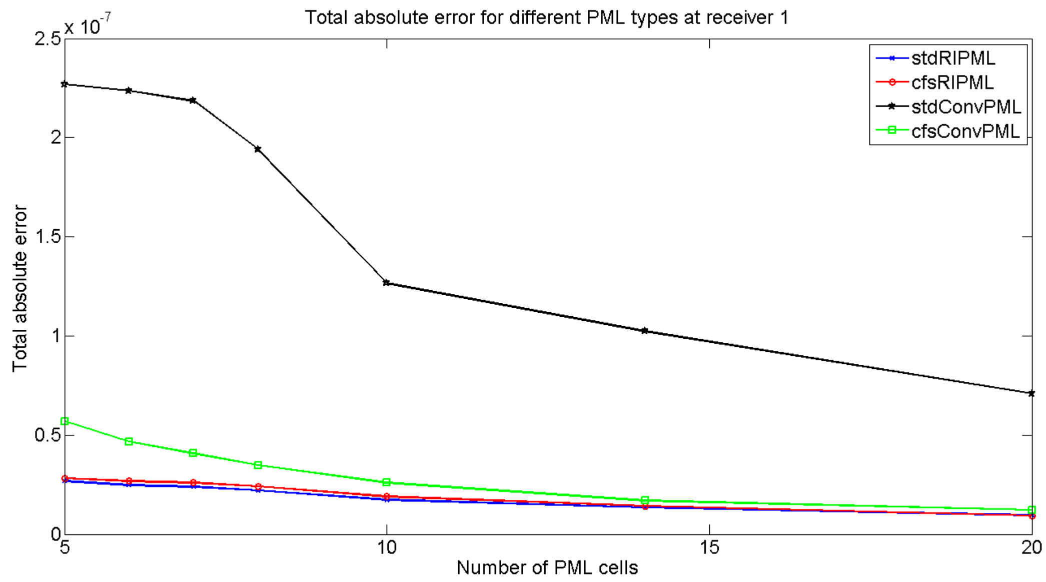


Figure B.22: Total absolute error for different numbers of PML cells at receiver 1 for different PML types on the rectangular model.

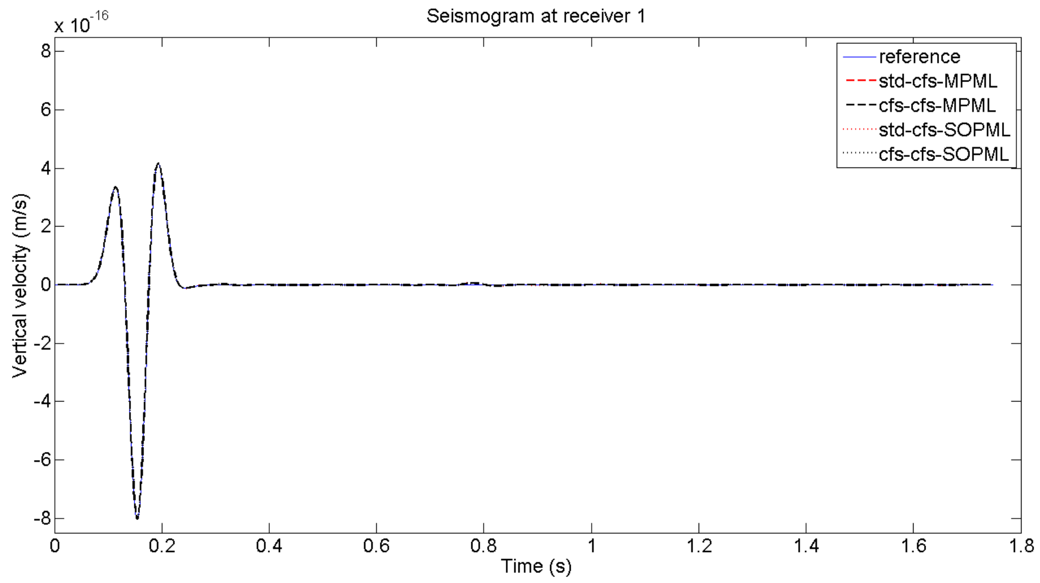


Figure B.23: Seismogram at receiver 1 for different PML types on the rectangular model.

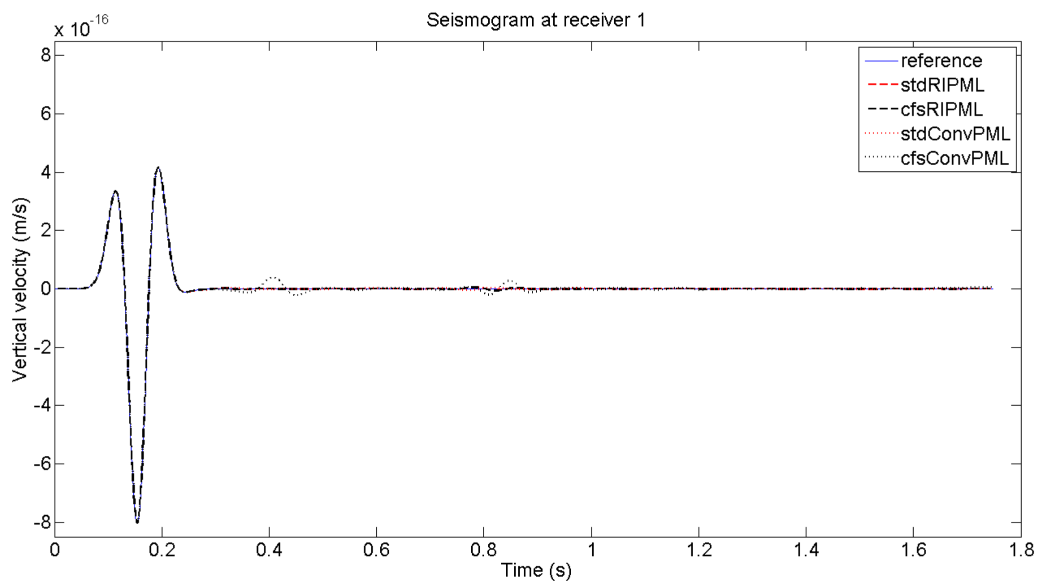


Figure B.24: Seismogram at receiver 1 for different PML types on the rectangular model.

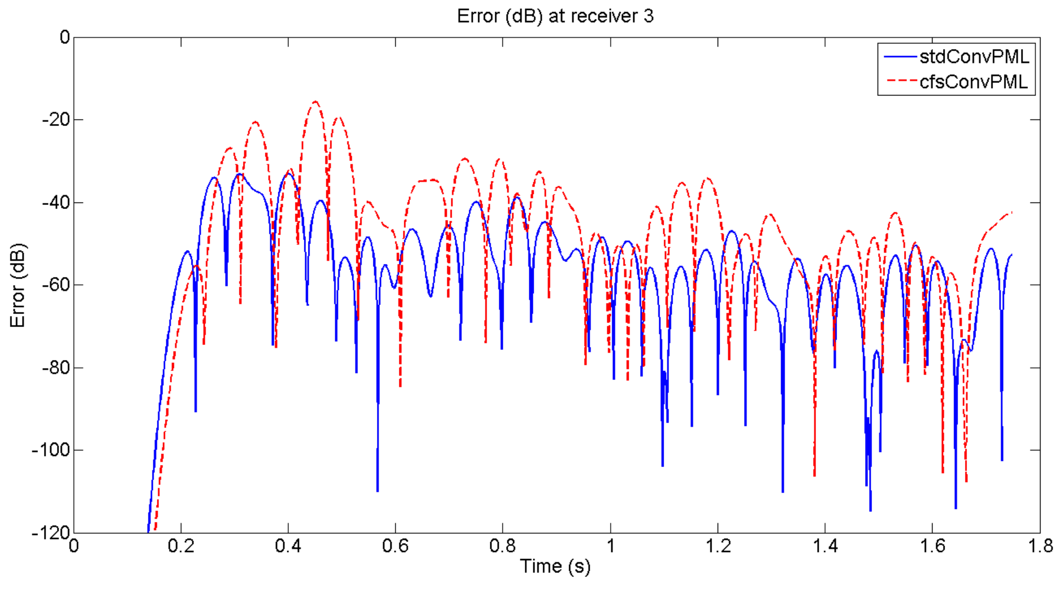


Figure B.25: Error in dB for two different types of PML's on the rectangular model at receiver 3.

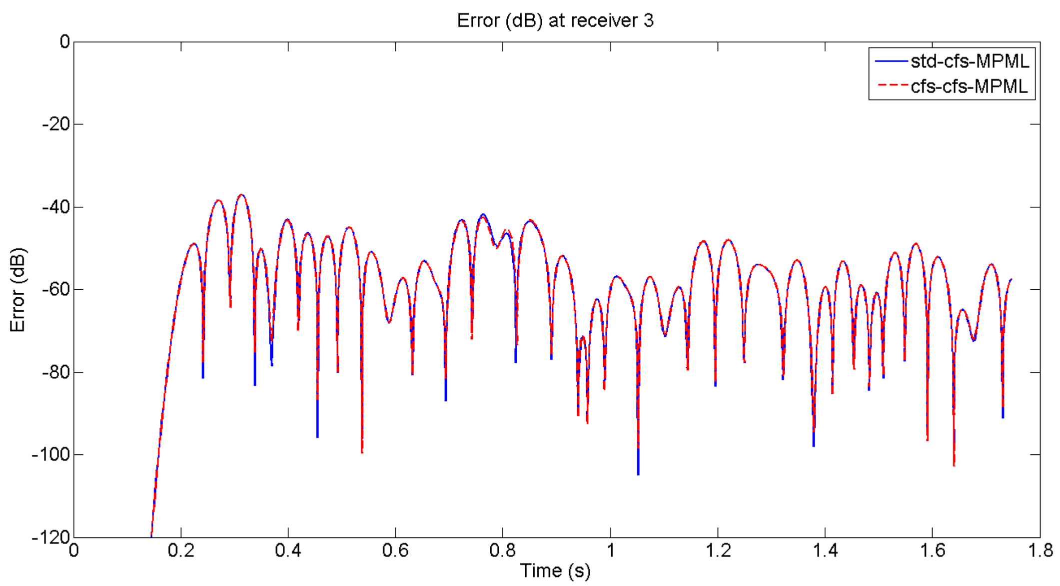


Figure B.26: Error in dB for two different types of PML's on the rectangular model at receiver 3.

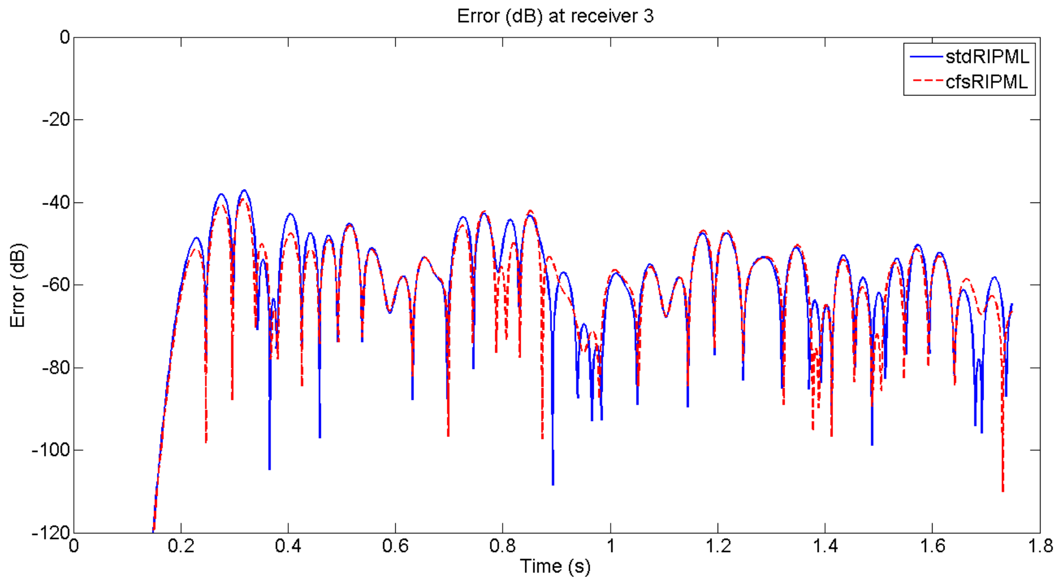


Figure B.27: Error in dB for two different types of PML's on the rectangular model at receiver 3.

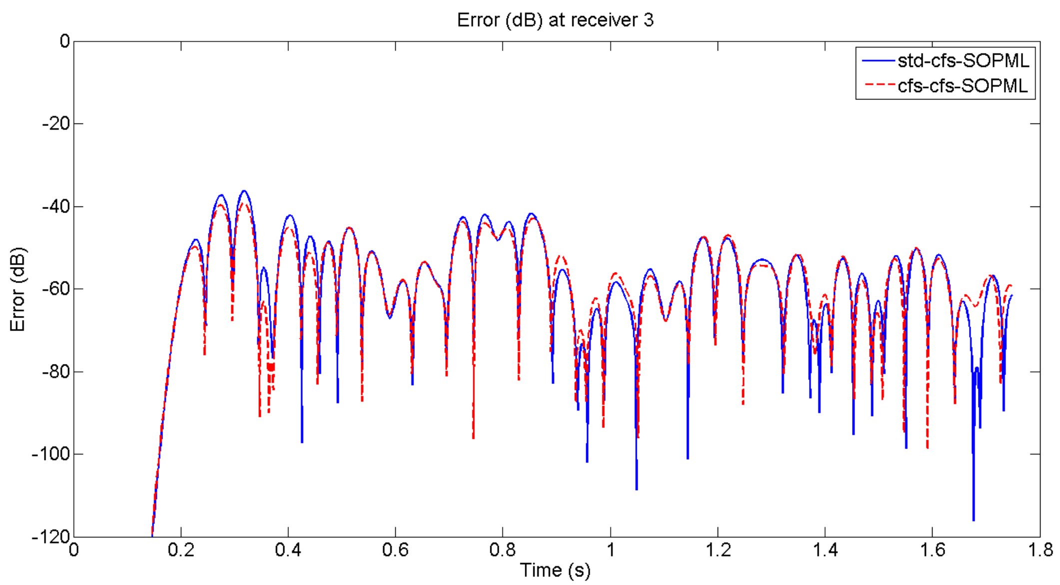


Figure B.28: Error in dB for two different types of PML's on the rectangular model at receiver 3.

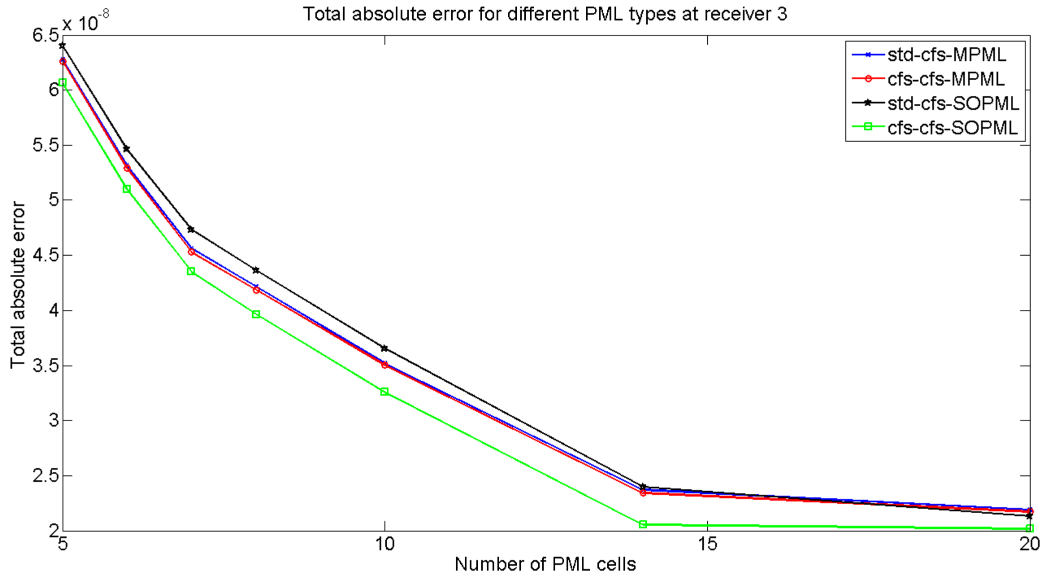


Figure B.29: Total absolute error for different number of PML cells at receiver 3 on the rectangular model.

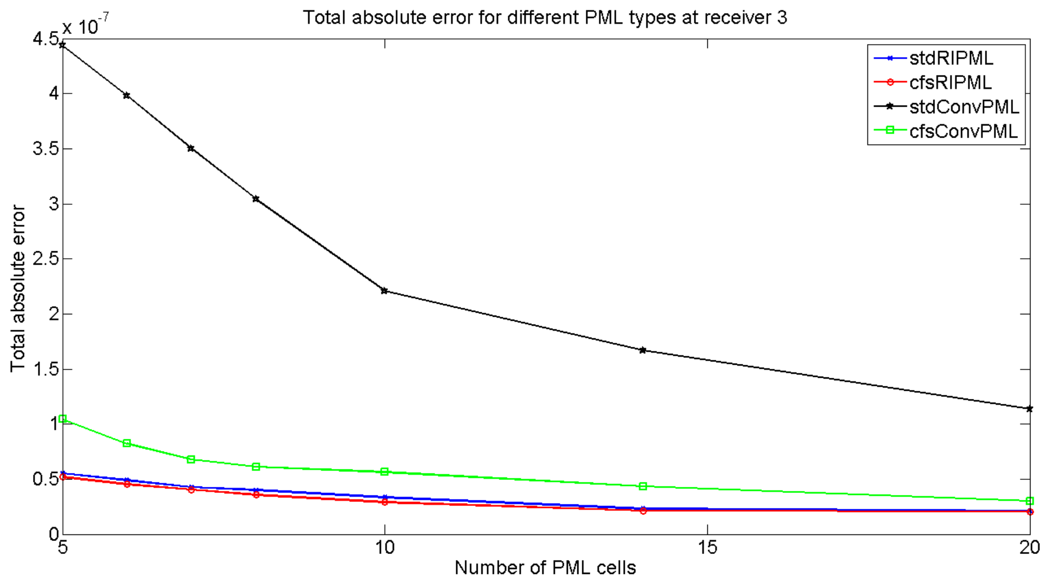


Figure B.30: Total absolute error for different number of PML cells at receiver 3 on the rectangular model.

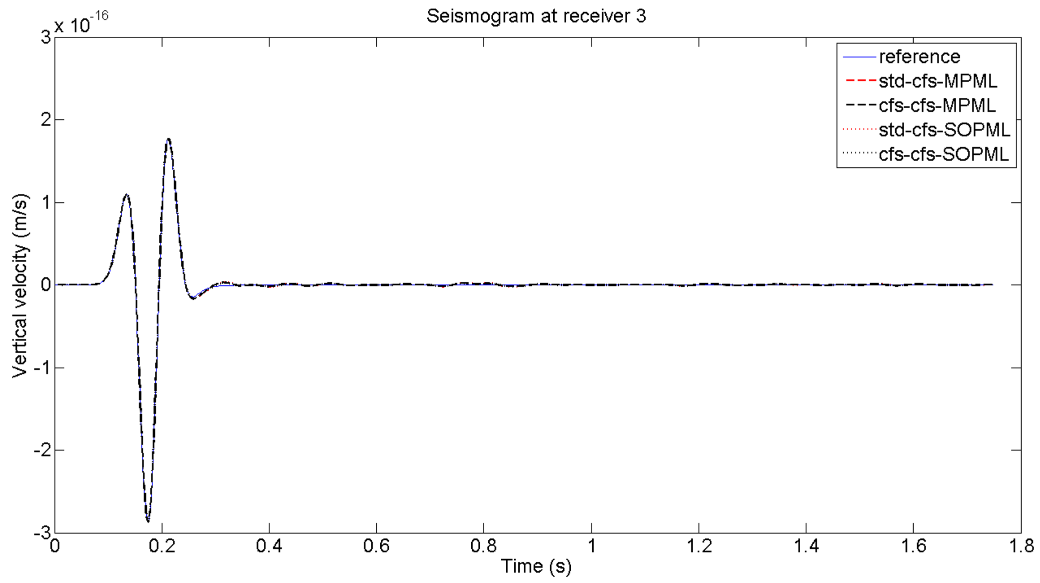


Figure B.31: Seismogram for different PML types at receiver 3 on the rectangular model.

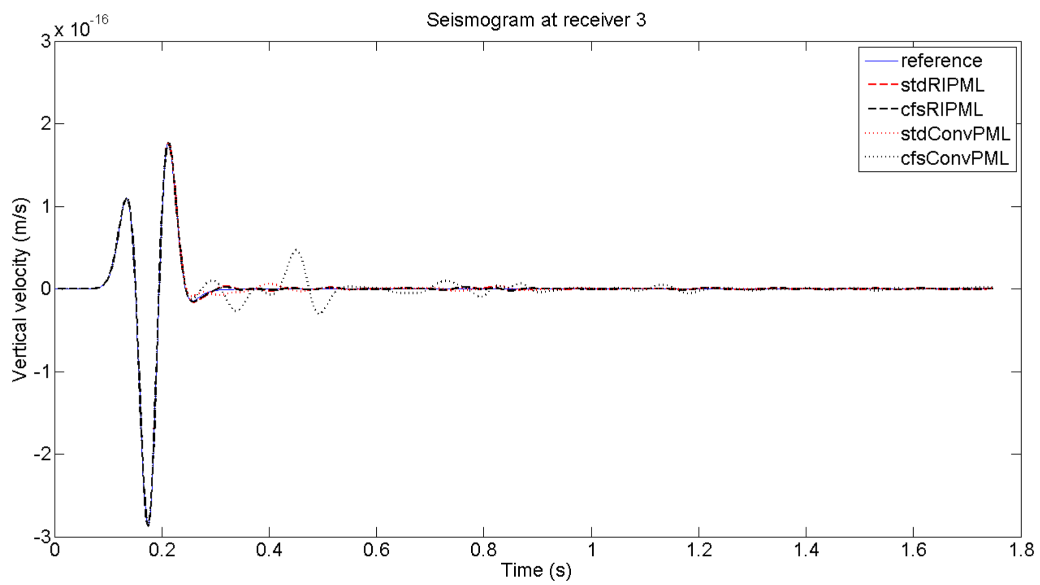


Figure B.32: Seismogram for different PML types at receiver 3 on the rectangular model.