Delft University of Technology

Bachelor thesis

Bachelor Applied Physics & Applied Mathematics

Finite Element Method Applied to the One-dimensional Westervelt Equation

Author:
Bas Dirkse

Supervisors:
Dr. Domenico J. P. Lahaye
Dr. ir. Martin D. Verweij

Delft University of Technology,

Faculty of Applied Sciences &
Faculty of Electrical Engineering, Mathematics and Computer Science,

Section of Acoustical Wavefield Imaging &
Department of Numerical Analysis

February 13, 2014

TU Delft Delft University of Technology

Summary

In this thesis we researched the applicability, properties and efficiency of the finite element method to solve the one-dimensional Westervelt equation, which describes nonlinear plane wave propagation. The goal was to investigate whether this lesser-known solution method has advantages or disadvantages compared to more commonly used solution techniques. We developed an understanding of nonlinear wave propagation by analyzing the Burgers equation, which we used to benchmark solutions. We used the commercial finite element software package COMSOL to calculate first solutions, where we found that numerical errors occur as the wave propagates through the shock wave formation distance. We examined the effect of several numerical parameters and concluded that reducing the element size decreases the overall error of the solution, both near the shock wave front and elsewhere. This also helps reduce numerical oscillations if present. Increasing the element order also improved the solution. The time stepping algorithm was found to have a strong connection to the element size. The maximum time step depends strongly on the minimum element size. Reducing physical parameters such as the amplitude of the source, or adding damping, were also researched but were shown to have little effect on reducing the numerical error around the shock wave front. The finite element method can solve inhomogeneous domains with relative ease compared to homogeneous domains, which may be an advantage over other methods.

We then developed our own Matlab implementation of Galerkin's finite element method for the Westervelt equation to get more insight into the algorithms behind this method and get a better understanding of the effect of numerical parameters. We implemented two different time solvers and we concluded that our specific choice of backward differential formulas was producing more accurate results than more general build-in time solvers that come with COMSOL or Matlab. Furthermore we saw that the accuracy of the solution does not only depend on spatial numerical parameters, but also on the time solving parameters. Different time solving techniques can yield different degrees of accuracy and efficiency, and must therefore be chosen with care.

We finally turned to adaptive finite element method techniques in order to improve overall accuracy and efficiency. We have shown that a simple form of adaptiveness can help improve the accuracy of the solution, but its efficiency depends on the implementation and the number of spatial dimensions in which the equation is solved. The finite element method provides different types of adaptiveness, such as local refinement/coarsening, node movement and local change of the order of the basis functions, which may be combined together. We showed the advantages and disadvantages of a node movement implementation based on the MMPDE-6 algorithm. We concluded that more research can be put in incorporating (combined types of) adaptiveness to solve the Westervelt equation.

Contents

1 Introduction

1.1 Nonlinear wave propagation

In most fields of acoustics it is sufficient to model wave propagation with the linear wave equation. The analytical solution to the homogeneous linear wave equation is well known. Therefore pressure fields can usually be found quite easily. In acoustic ultrasound imaging however, this situation is different, because high frequency and high amplitude sources are used. These large and rapid pressure changes cause the wave to propagate nonlinearly.

The nonlinear effect (in fluids) has two causes. The first cause is intrinsic. It arises due to the cumulative effect of distortion of the wave profile by convection, introduced by the particle velocity that itself constitutes the sound wave. Larger particle velocities propagate faster than slower ones, leading to distortion of an acoustic wave upon propagation [1]. The second cause has to do with the equations of state. Compression of the medium leads to change in local propagation speeds, which follows from the equation of states for compressible fluids. For such fluids, this second cause has the largest contribution to the nonlinearity, especially for large acoustic pressures.

This nonlinear effect can be neglected under normal circumstances, because at small sound pressures both causes to nonlinearity have negligible effects. This is especially true when attenuation and scattering effects are also taken into account. The propagation speed does not change significantly with the pressure because the medium has negligible compression. At very large pressures or frequencies the causes of nonlinearity become significant and therefore high pressure and/or frequency wave propagation has to be described by a nonlinear wave equation.

There are various derivations and approximations available which lead to slightly different nonlinear wave equations, depending on the particular use of interest. This is not the subject of this thesis and we will therefore directly state an equation which describes lossless nonlinear wave propagation very generally. It is known as the Westervelt equation and is given by [2]

$$\nabla^2 p - \frac{1}{c_0^2}\frac{\partial^2 p}{\partial t^2} = -\frac{\beta}{\rho_0 c_0^4}\frac{\partial^2 p^2}{\partial t^2}, \tag{1.1}$$

where p is the unknown pressure field, β is the coefficient of nonlinearity, ρ_0 is the ambient density of mass and c_0 is the small signal sound speed. The subscript 0 emphasizes (for all materials) that we mean the *small signal* sound propagation speed in case of c_0 and the *ambient* density of mass in case of ρ_0, which must not be confused with the local sound propagation speed and the local density of mass. β is a dimensionless material property that accounts for both nonlinear effects. It is theoretically derived from the equation of states and its values are usually obtained empirically. More discussion on this coefficient can be found in [1] or [2].

The form of (1.1) is standard in the literature, since we have the linear wave operator acting on p on the left-hand-side and the nonlinear term on the right-hand-side.

If we set $\beta = 0$ in (1.1), the right-hand-side vanishes and we are left with the equation for linear wave propagation. We can see that the nonlinear term depends on specific material properties and is proportional to $\frac{\partial^2 p^2}{\partial t^2}$. Is is clear that this partial derivative becomes larger with increasing wave amplitude and frequency.

If we focus on the one-dimensional version of (1.1), i.e. ∇^2 is replaced by $\frac{\partial^2}{\partial x^2}$, and introduce a comoving or retarded time $\tau = t - \frac{x}{c_0}$ we get

$$\frac{\partial^2 p}{\partial x^2} - \frac{2}{c_0}\frac{\partial^2 p}{\partial \tau \partial x} = -\frac{\beta}{\rho_0 c_0^4}\frac{\partial^2 p^2}{\partial \tau^2}.$$

The variation in pressure with respect to x is small in the comoving frame and therefore the second-order partial derivative can be neglected [2] in the equation above. If then the equation is integrated with respect to τ we obtain the Burgers equation [2]

$$\frac{\partial p}{\partial x} = \frac{\beta}{\rho_0 c_0^3} p \frac{\partial p}{\partial \tau}. \tag{1.2}$$

This is the simplest and most used equation to model nonlinear effects of lossless propagating plane waves.

1.2 Numerical methods for solving the Westervelt equation

The Westervelt equation (1.1) does not have a known analytical solution, and because of its nonlinearity the superposition principle doesn't generally apply. Hence its solution must be calculated numerically. There are three common methods for solving the Westervelt equation. The first method is by the use of finite difference schemes. This is the most straight forward way, in which derivatives are approximated by finite differences and a resulting system of equations is solved in time iteratively. The second method is by the use of Greens functions, which can only obtain exact solutions for linear operators with a right-hand-size forcing function. It is used to solve the Westervelt equation iteratively, where the nonlinear term is viewed as a source function of the linear wave equation. The last method is the finite element method (FEM). In this method a weak formulation of the problem is solved for a specific test function. The first two methods are commonly applied in solving the Westervelt equation, but much less is known about solving it using the finite element method.

The FEM has proven to be a successful solving method throughout physics, for example in structural analysis problems or fluid flow problems. It is usually applied when geometries are complicated and/or lots of inhomogeneities are present. Such conditions may be present in diagnostic or treatment focused ultrasound applications. Therefore we want to research how applicable the finite element method is to medical ultrasound problems.

In this thesis we will research the applicability of the FEM to the Westervelt equation. We examine how this method behaves when certain known problems arise, for example when approaching the shock wave formation distance. We research which parameters have an effect on the accuracy of the solution and the efficiency of the scheme.

We will also try to comprehend which parameters effect the accuracy and efficiency of the calculation. We will use the Burgers equation (1.2) and its solution as a benchmark to the one-dimensional Westervelt equation. Therefore we study the Burgers equation, and one-dimensional nonlinear wave propagation in chapter 2.

1.3 Structure of this thesis

In chapter 2 we study the Burgers equation along with its implicit solution. We develop a scheme to approximate an explicit solution and implement it in Matlab, so we can use it as a benchmark solution. We study why shock wave formation occurs and what the distance from the source is at which it first occurs. We show some examples of nonlinear propagation as an illustration. Finally, we state the known analytical solution to the Burgers equation valid for single frequency sources and study how multiple higher-order harmonics occur in the frequency domain as a result of the nonlinearity. This is the basis for our understanding of nonlinear wave propagation.

In chapter 3 we start our research by solving the Westervelt equation with a commercial FEM software package called COMSOL. This provides us with general information on FEM solutions for the Westervelt equation and can help us understand its limitations and strengths. The advantage is that we use an existing FEM code, and even an existing model in COMSOL [7]. The disadvantage is that it is difficult to understand the underlying algorithms and it is therefore not very clear how certain computational parameters influence the accuracy and computation time. We will research the effect of both physical and numerical parameters on the solution. First, we study what effect the source amplitude has on the accuracy of the solution. We will also examine if adding physical damping to the Westervelt equation can reduce known numerical errors. A study on a inhomogeneous medium will also be performed to examine if the FEM can resolve that. Numerical parameters that we will examine are the element size and the order of the basis functions. Those two characteristic parameters partly determine the accuracy up to which COMSOL can solve the Westervelt equation. We finally also took a look at the correlation between the time step taken in the time solver and the (smallest) element size in space.

In chapter 4 we continue our research by developing a self-made FEM implementation in Matlab, based on Galerkins finite element method. This should give us more insight in the effect of varying (numerical) parameters. We started by implementing a FEM scheme to solve a Poisson equation to gain experience developing FEM code and to verify its solution with a simple analytical solution. We did this for first- and second-order elements. We then developed code for solving the linear wave equation, which could also be verified analytically. After incorporating the nonlinear term we made a full implementation for the Westervelt equation. The implementation was made with first-order basis functions, and an attempt was made to improve it to second-order basis functions. We did not succeed to get a stable solution. After spatial discretization we solved the time depending system in two distinct ways: one implementation with a self-developed backward differential formula (BDF) and the other with the use of a Matlab build-in ODE solver. Our goal is to design a more accurate and efficient implementation of a finite element scheme and to have more control over numerical parameters influencing the solution. We can now compare the three available options to obtain a solution.

In chapter 5 we examine the hypothesis that higher accuracy can be obtained more efficiently by implementing some form of adaptiveness in the finite element method. First we discuss the various types of adaptiveness available to the finite element method and argue which type is most favorable when solving the Westervelt equation. Because of the transient nature of wave propagation, we have chosen to implement r-adaptiveness. In this type the nodes are allowed to move along the domain, refining the mesh wherever it is most needed. We then discuss the criteria on which the adaptivity bases its adjustments. We discuss error approximation as such a criterion and argue that the difference between solutions calculated with second- and first-order basis functions is a good error approximation to use. We also discuss that a monitor function, which is based on the magnitude of the spatial derivative, is a good criterion for adaptivity. The advantage is that there is no need to calculate a higher-order solution. We finally show two different implementations of r-adaptiveness. The first is known in literature [12] as MMPDE-6. It is based on a monitor function, which describes where on the domain the spatial derivative of the solution is large, and a mesh distribution is constructed that refines where the derivatives are large and coarsens where they are small. This is done by equally dividing the monitor function over each element, so that elements become small wherever the monitor function (and thus the derivative of the solution) is large. This is a sort of automatic mesh refinement where, apart from two initial parameters, we have little to no control over the future movement of the nodes. Nodes may approach each other very closely. Also since the mesh depends on the solution and the solution depends on the mesh, a more difficult, tangled system of equations arises which may be quite stiff. The second method that we implemented is a more straightforward co-traveling mesh. Based on the knowledge about the domain, the sound propagation speed and the pulse-like nature of the source function, we constructed a type of mesh where the refined area follows the pulse as it propagates along the domain. The advantages here are that it is a simple implementation in which we have control over the element sizes at all time.

2 Analysis and solution of the Burgers equation

For the Burgers equation (1.2) an implicit solution is known (Blackstock, 1962) as derived in [3]. It is of the form

$$p(x,\tau) = f(\phi), \quad \phi = \tau + \frac{\beta}{\rho_0 c_0^3}(x - x_0)p(x,\tau), \tag{2.1}$$

where $p(x_0,\tau) = f(\tau)$ is the source pressure as a function of the comoving time τ located at $x = x_0$. In principle, this equation can be approximated numerically to an explicit solution given any source pressure. This solution can be used to compare with results from numerical methods.

2.1 Behavior of nonlinear wave propagation and shock formation

In deriving the Burgers equation (and the Westervelt equation for that matter) we find that any phase point on the waveform (a point of constant pressure) travels with velocity [2]

$$\frac{dx}{dt}\Big|_p = c_0 + \frac{\beta}{\rho_0 c_0} p, \tag{2.2}$$

where we can see that points of high pressure travel faster than points of low pressure. In Figure 2.1 it is shown that the high pressure points have traveled faster than the low pressure points, when compared to linear propagation. As a result any phase point where $\frac{\partial p}{\partial x} < 0$ moves faster than a phase point a small distance further on and at some point they may approach each other, i.e. $\frac{\partial p}{\partial x} \to -\infty$. At this point a shock wave is formed. The Burgers equation is only valid until this shock formation distance.

The shock formation distance is given by [3]

$$x_{sh} = \frac{\rho_0 c_0^3}{\beta f'_{max}}, \tag{2.3}$$

where f'_{max} is the maximum of the derivative of the source pressure function f with respect to its independent variable. If the source is a single frequency sine wave (i.e. $f(t) = P_0 \sin(2\pi f_0 t)$), then $f'_{max} = P_0 2\pi f_0$ and thus (2.3) reduces to

$$x_{sh} = \frac{\rho_0 c_0^3}{\beta P_0 2\pi f_0}, \tag{2.4}$$

where P_0 is the amplitude and f_0 the frequency of such a single frequency sine wave.

Usually in one-dimensional nonlinear wave propagation, distance is given in a dimensionless quantity $\sigma = \frac{x}{x_{sh}}$. The Burgers equation is then valid from $0 < \sigma < 1$. This is convenient when comparing plots and data for different shock wave distances.

2.1.1 Behavior of nonlinear wave propagation in the retarded time domain

In the previous section we saw how the pressure was distorted at a certain time over a domain in x. In this section we will look at the solution $p(x,\tau)$ to (1.2) at a fixed point in space as a function of τ. This is another way of looking at the same solution and shock wave formation. To do so, we look at the slowness, defined as the inverse of the propagation speed, with respect to the retarded time, which is given by [3]

$$\frac{\partial \tau}{\partial x}\Big|_p = -\frac{\beta}{\rho_0 c_0^3} p. \tag{2.5}$$

From this equation we can see how τ decreases as x increases for fixed pressure points. So in particular, high pressure points arrive earlier than low pressure points (since they were moving faster). This is illustrated in Figure 2.2.

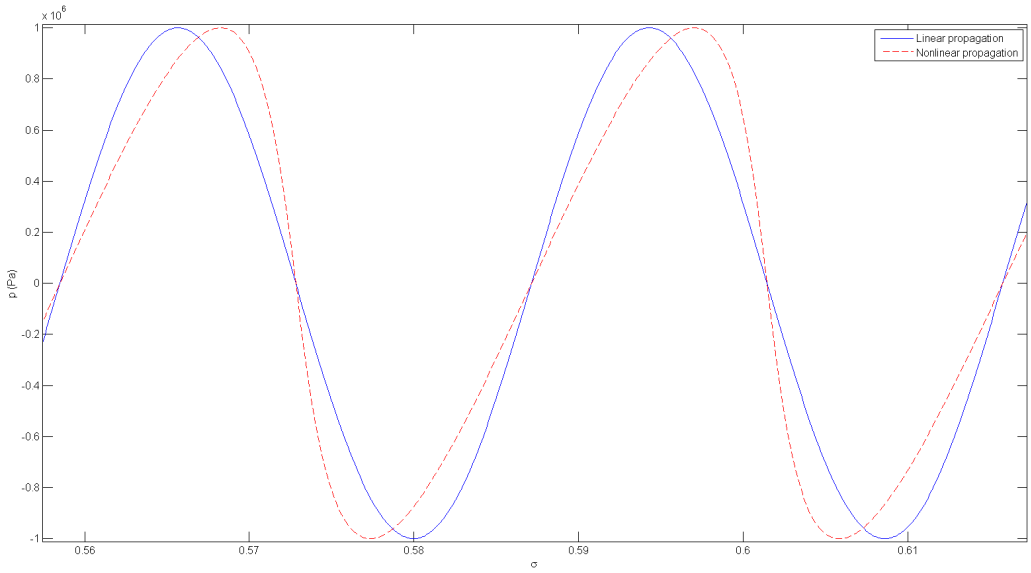


Figure 2.1: *Nonlinear wave propogation compared with linear wave propogation at some fixed time t in a domain x somewhere between the source and the shock formation distance. Note that $\sigma = \frac{x}{x_{sh}}$ and $x_{sh} = 0.517$ m. It is clear that the high pressure phase points traveled faster than the low pressure phase points. This becomes most clear when compared with the linear propagation. The source pressure in this example is a* 100 kHz *sine wave with amplitude of* 1 MPa. *$c_0 = 1481\ \frac{\text{m}}{\text{s}}$, $\rho_0 = 999.6\ \frac{\text{kg}}{\text{m}^3}$ and $\beta = 10$.*

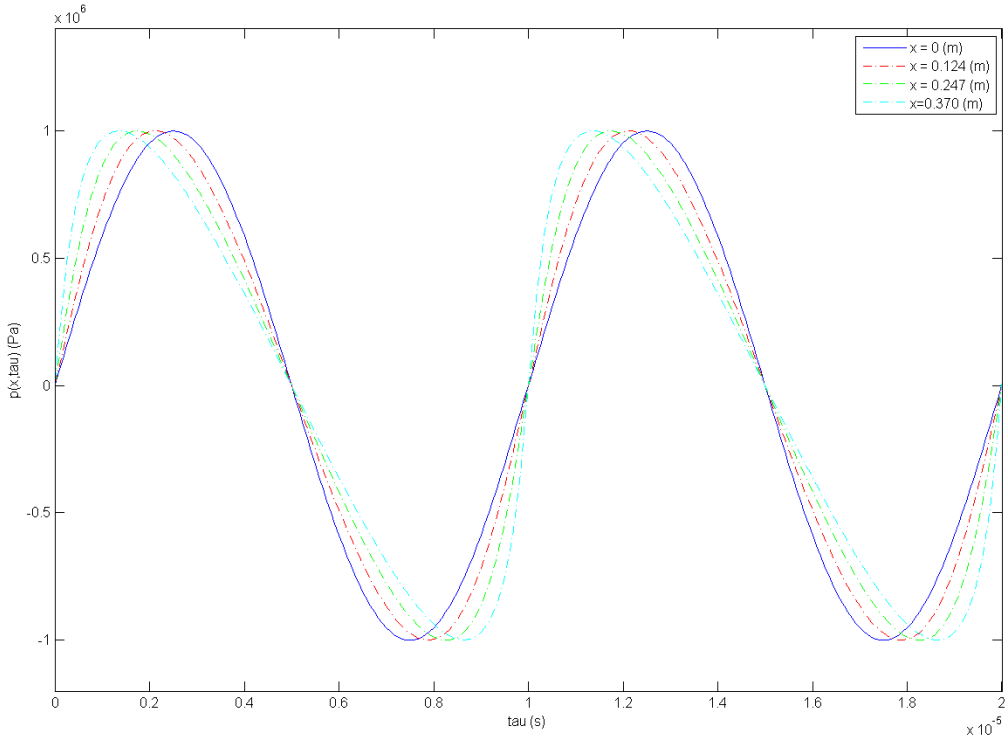


Figure 2.2: *Nonlinear pressure variation in time occurs at certain points x from the source (located at $x = 0$), where $0 < x < x_{sh} \approx 0.517$ m. The source pressure in this example is a* 100 kHz *sine wave with amplitude of* 1 MPa. *$c_0 = 1481\ \frac{\text{m}}{\text{s}}$, $\rho_0 = 999.6\ \frac{\text{kg}}{\text{m}^3}$ and $\beta = 10$.*

2.2 Numerical approximation of the implicit solution to the Burgers equation

Using a Taylor expansion we find that we can approximate (2.1) explicitly for $p(x, \tau)$ by the following numerical scheme [4]

$$p(x_{i+1},\tau) \cong p\Big(x_i, \tau + \frac{\beta}{\rho c_0^3}(x_{i+1}-x_i)p(x_i,\tau)\Big). \tag{2.6}$$

This approximation is only valid as long as p varies slowly enough over the distance $\Delta x = x_{i+1} - x_i$, since this is a first-order Taylor expansion approximation. Thus we can obtain the pressure at $x = x_{i+1}$ from the pressure at $x = x_i$ by transforming the time basis of $p(x_i, \tau)$ with a shift that depends on the magnitude of the pressure field at each time instant [4].

An implementation in Matlab code of such a numerical scheme can be found in Appendix A.

2.3 The single frequency source function and its analytical solutions

2.3.1 Fubini's solution

For a single frequency source there is also an explicit solution known to the Burgers equation (1.2) found by Fubini in 1935 [3]. That is, if the source has the form of

$$p(0,t) = P_0 \sin(\omega t),$$

then the solution is given by

$$p(\sigma,\tau) = P_0 \sum_{n=1}^{\infty} \frac{2}{n\sigma} J_n(n\sigma)\sin(n\omega\tau), \tag{2.7}$$

where $\sigma = x/x_{sh}$ is the dimensionless measure of length as defined before and J_n is the ordinary Bessel function of order n. It is only valid for $0 < \sigma < 1$ [5]. This solution shows how energy is passed over to higher harmonics as the wave travels along x.

This is an important observation that does not only apply to single frequency sources. For each frequency component in the source function, higher-order harmonics will form as the wave progresses.

2.3.2 Fay's solution

Post shock formation distance there is also an analytical solution known which was first found by Fay (1931) for a singe frequency source. The solution is given as [3] [5]

$$p(x,t) = P_0 \frac{2}{\Gamma} \sum_{n=1}^{\infty} \frac{\sin(n\omega\tau)}{\sinh[n(1+\sigma)/\Gamma]}, \tag{2.8}$$

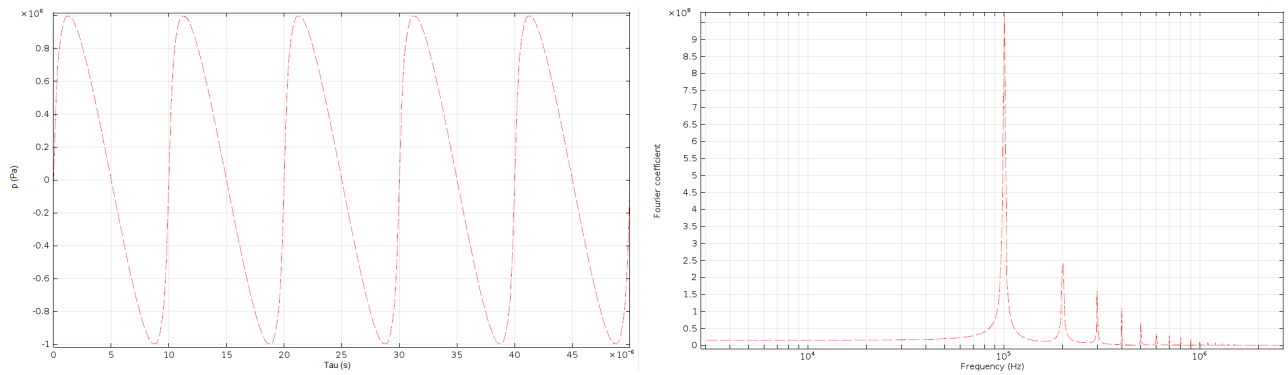
where Γ is a group of constants characterizing the importance of nonlinearity relative to that of dissipation.

It is important to note that this is a solution to the Burgers equation in which dissipation was taken into account. This is necessary to describe post shock wave propagation accurately. The solution is accurate from $\sigma > 3$ as shown by Blackstock in 1964 [3].

2.4 The frequency spectrum of Fubini's solution

As mentioned in Section 2.3.1, one of the most important results of the nonlinear propagation is the transfer of energy to higher-order harmonics. This applies not only to single frequency sources, but to any source function in general. It is important to understand that nonlinear behavior in the time (or spatial) domain is associated with an energy shift towards higher-order harmonics in the frequency domain. The closer to the shock wave distance we come, the steeper the wave becomes in the time (or spatial) domain, and the larger the amplitudes of the higher-order harmonics become in the frequency domain. Such high frequencies can give difficulties in any numerical approach to the problem.

In figure 2.3 this effect is shown for a single frequency source.



(a) *Pressure field*

(b) *Frequency spectrum*

Figure 2.3: *Plot of the frequency spectrum (b) of a pressure field (a) at a distance* $\sigma = 0.77$ *from the source. Again,* $x_{sh} = 0.517$ m. *We can see energy being passed to the higher-order harmonics with frequency* nf_0. *The source (at* $x = 0$*) is a* 1 MPa, 100 kHz *sine wave.* $c_0 = 1481\ \frac{\text{m}}{\text{s}}$, $\rho_0 = 999.6\ \frac{\text{kg}}{\text{m}^3}$ *and* $\beta = 10$.

3 FEM solutions to the Westervelt equation using COMSOL

In this section we discuss the results that we find when we solve the one-dimensional Westervelt equation with the finite element method software package COMSOL[1]. This software package is designed for multipurpose engineering and combines physics with geometry.

The simulations in this section are based on the model of the one-dimensional Westervelt equation provided by COMSOL in its model library [7]. The model is modified to examine the effect of varying specific parameters on the obtained solution. We want to solve the Westervelt equation numerically up to a desired accuracy, without numerical errors occurring. This is especially difficult when approaching the shock wave formation distance. In this part of our study, we try to look which parameters can help us to solve more accurately and at what cost.

In section 3.1 we define the problem and set our simulation parameters. In section 3.2 we look at the effect of different parameters on the solution. We look at the pressure amplitude, the mesh size and the order of the basis functions. We then examine if incorporating (physical) damping resolves some numerical problems and we conclude with a simulation on a inhomogeneous domain.

3.1 Simulation parameters and settings

3.1.1 Initial and boundary conditions on the domain

We solve the Westervelt equation on a one-dimensional domain of length L, which is initially at rest, i.e.

$$p(x,0) = 0 \quad \text{and} \quad \frac{\partial p}{\partial t}\Big|_{t=0} = 0 \qquad \text{for} \quad x \in (0, L].$$

A source is located at $x = 0$. In ultrasound imaging, this source typically emits a very short pulse. Such a pulse is modeled by an single frequency sine with a Gaussian envelope. The source function is given by

$$p(0,t) = P_0 \sin[2\pi f_0(t - T_d)] \exp\Big[-\Big(\frac{t - T_d}{T_w/2}\Big)^2 \Big] \big[H(t) - H(t - T_{end})\big], \tag{3.1}$$

where P_0 is the pressure amplitude, f_0 is the frequency of the source, T_d is a time delay, T_w is a pulse width and T_{end} is the end of the pulse. $H(t)$ is the Heaviside step function and is added for computational purposes.[2] We set, as is customary in ultrasound imaging pulses, $T_d = \frac{6}{f_0}$, $T_w = \frac{3}{f_0}$ and $T_{end} = 2T_d$ [4]. In Figure 3.1 the source function is shown.

3.1.2 Material and simulation parameters

The material parameters are chosen to be the physical properties of water. We set $\rho_0 = 999.6\ \frac{\text{kg}}{\text{m}^3}$ and $c_0 = 1481.44\ \frac{\text{m}}{\text{s}}$. Water typically has a coefficient of nonlinearity β of about 3.5, but to speeds up computations we choose $\beta = 10$ in all our simulations. This shortens the shock wave distance about a factor 3 and therefore speed up the computation (which runs from $x = 0$ to about the shock wave distance) by a rough factor 3.

The domain length L is set to be $0.95 \cdot x_{sh} + T_{end} \cdot c_0$, where the second term is added to assure the pulse has passed the 95% shock wave distance. This domain is divided into an equidistant mesh, with element size $dx = \frac{c_0}{18 f_0}$, so that there are 18 elements per period. The number of elements is therefore L/dx. Quartic elements are used since they are proven most accurate [7].

COMSOL is set to use a BDF time differentiation method of an order between 1 and 5. We have set the maximum time step to be $dt = 0.05\frac{dx}{c_0}$, yielding 20 times more points per period in time than in space. It is to be emphasized that dt is a maximum time step and that COMSOL chooses a time step based on a tolerance setting, which can be arbitrarily much smaller than dt. It also chooses the most suitable order of BDF and it estimates errors on the go.

[1] COMSOL Multiphysics version 4.3b, released 2013, see http://www.comsol.com/

[2] Actually a smooth approximation to the step function is used to assure the existence of second-order derivatives. The step function is added to have the values for $f_0 t > 12$ be identically zero for computational purposes. Note that $|p(0, 12/f_0)/P_0| \approx 10^{-22}$.

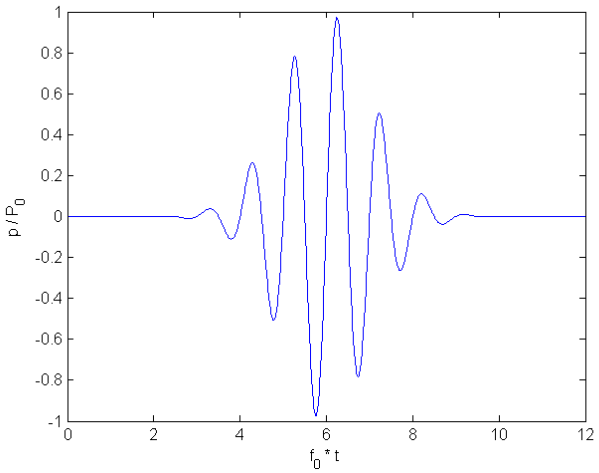


Figure 3.1: *A single frequency source function with a Gaussian envelope as described by* (3.1)*. The delay and width parameters are chosen so that a pulse of about six periods is generated. In this case* $T_d = \frac{6}{f_0}$ *and* $T_w = \frac{3}{f_0}$.

We have chosen for a frequency of 100 kHz. This is an order smaller than typical ultrasound imaging frequencies, but COMSOL choses dt to be on average about a 100 times larger at this frequency than at a frequency of 1 MHz, so for computation time purposes we chose for 100 kHz.

P_0 is the maximum pressure amplitude. We set it to be 1 MPa, which is typical in ultrasound imaging. We have varied P_0 around this value to see what the effects are on the computations. P_0 influences the shock wave distance and the value of the (time) derivatives near this distance, without effecting any other computation related parameters.

All parameters are summerized in Table 1. These parameters are used in the simulations unless specified differently.

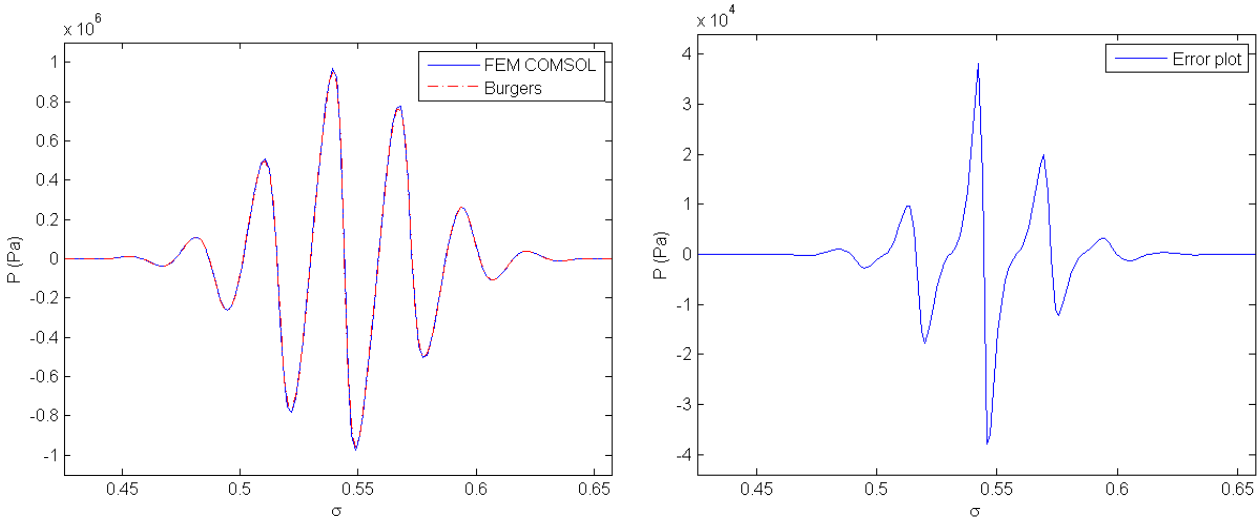
Table 1: *A summary of the parameters used for simulation of the one-dimensional Westervelt equation in COMSOL. These parameters are used, unless specified differently.*

ρ_0	999.6	$\frac{\text{kg}}{\text{m}^3}$	Material ambient density
c_0	1481.44	$\frac{\text{m}}{\text{s}}$	Small signal sound speed
β	10		Coefficient of nonlinearity
P_0	10^6	Pa	Maximum pressure amplitude
f_0	10^5	Hz	Single source frequency
T_d	$\frac{6}{f_0}$	s	Source envelope delay
T_w	$\frac{3}{f_0}$	s	Source envelope width
T_{end}	$2 \cdot T_d = \frac{12}{f_0}$	s	Source end time
x_{sh}	$\frac{\rho_0 c_0^3}{\beta P_0 2\pi f_0}$	m	Shock formation distance
L	$0.95 \cdot x_{sh} + T_{end} \cdot c_0$	m	Domain length
dx	$\frac{c_0}{18 f_0}$	m	Mesh element size
dt	$\frac{dx}{20 c_0}$	s	Maximum time step
	4		Element order
	equidistant		Mesh type

3.2 Results

First simulations were run with all parameters set as in Table 1. With these simulations, we obtain smooth and seemingly accurate solutions. We will quantify this by comparing it with Burgers solution (2.1).

In Figure 3.2a we plotted the solution at some intermediate time $t = 0.25$ ms, when the pulse traveled about halfway to the shock wave distance. We can see that the simulation agrees quite well with the Burgers solution, but difficulties already start arising near the peaks. That is where the second-order derivatives become large and an error is being made. The relative error is already in the order of 10^{-2}. In Figure 3.2b we plotted the difference between the Burgers solution and the simulated solution versus σ. It can be seen clearly that the simulation overshoots near the peaks.



(a) *Pressure field comparison*

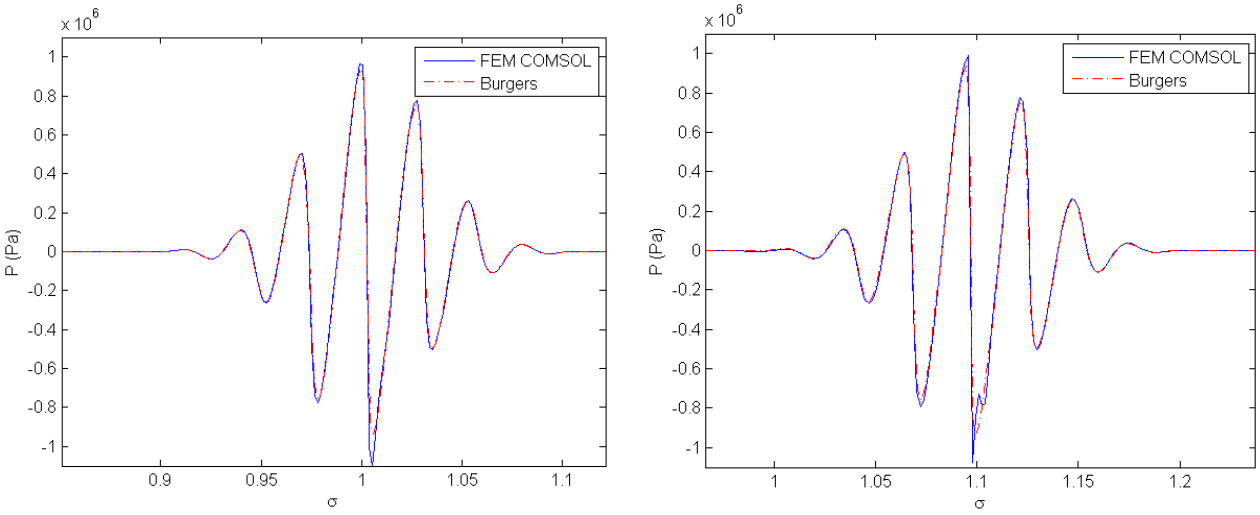
(b) *Calculated error*

Figure 3.2: *The pressure field as function of the distance from the source at time* $t = 0.25$ ms. *Parameters are all set as in Table 1.* $x_{sh} = 0.52$ m *for reference. Figure (a) shows the simulated pressure field as well as the calculated field with Burgers solution and Figure (b) shows the difference between the two solutions, which is notably large near the peaks.*

If we take a look at the solution at later times, when the shock wave distance is approached or just passed, we see that large errors are being made around the peaks. Figure 3.3 shows these situations. It is clear that the overshoot at the peaks is now very large. This is because of the very large (time and spatial) derivatives.

Now if we increase the maximum pressure amplitude to $P_0 = 2.2$ MPa, we see that this effect becomes even larger. The error is very large near the bottom peak and the solution is starting to oscillate there. This oscillation dies out rather slowly. In Figure 3.4 we plotted the simulated pressure field just after the shock wave distance.

This small sawtooth-like oscillation, known as numerical oscillation, turns around at each vertex of an element. The solution is smooth to the right of the deepest peak, and oscillates to the left of it. Oscillations occur around the bottom peak and die out very slowly. These observations can lead us to believe that the used mesh is not fine enough to suppress numerical oscillations. We examine this in Section 3.2.1.



(a) $t = 0.41$ ms **(b)** $t = 0.443$ ms

Figure 3.3: *The pressure field as function of the distance from the source at two specific times. Parameters are all set as in Table 1.* $x_{sh} = 0.52$ m *for reference. In Figure (a) there is a large overshoot near the peaks. In Figure (b) this overshoot is only corrected after several spatial steps.*

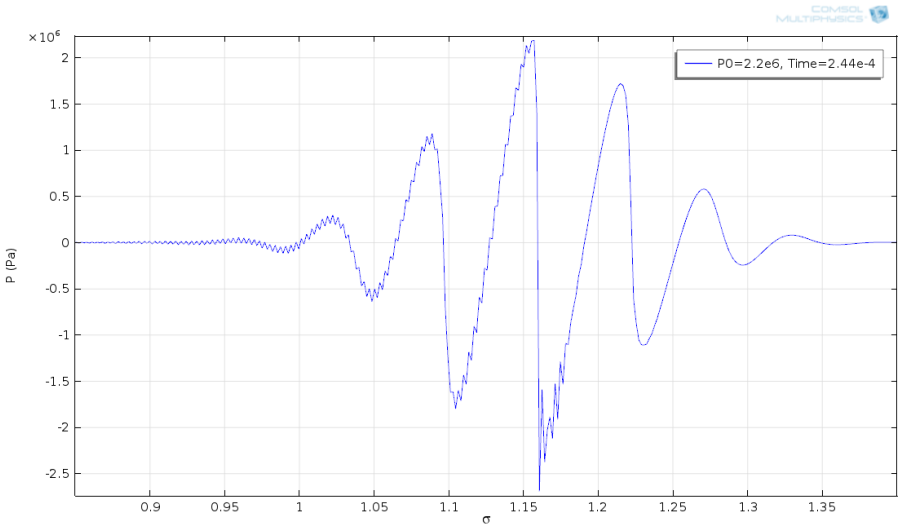


Figure 3.4: *The pressure field as function of the distance from the source at time* $t = 0.244$ ms. *Parameters are all set as in Table 1, except for* $P_0 = 2.2$ MPa. $x_{sh} = 0.235$ m *for reference. Sawtooth-like numerical oscillations occur near the bottom peak and they die out very slowly.*

3.2.1 Comparison of mesh size

The previous result justifies us to research the effect of changing the mesh size dx. To magnify this effect, we now set $P_0 = 3.4$ MPa, about 1.5 times larger then when we first noticed the numerical oscillations. We calculated a solution for element size $dx \approx 9.12 \cdot 10^{-4} \approx \frac{c_0}{16.2 f_0}$ m and $dx \approx 3.35 \cdot 10^{-4} \approx \frac{c_0}{44.2 f_0}$ m, a factor 2.72 finer mesh. Quartic elements are still used.

We can clearly see from Figure 3.5, in which both solutions are plotted, that this numerical oscillation does not occur when choosing a finer mesh. The solution also seems to be more accurate overall, although here no comparison is made with the Burgers solution.

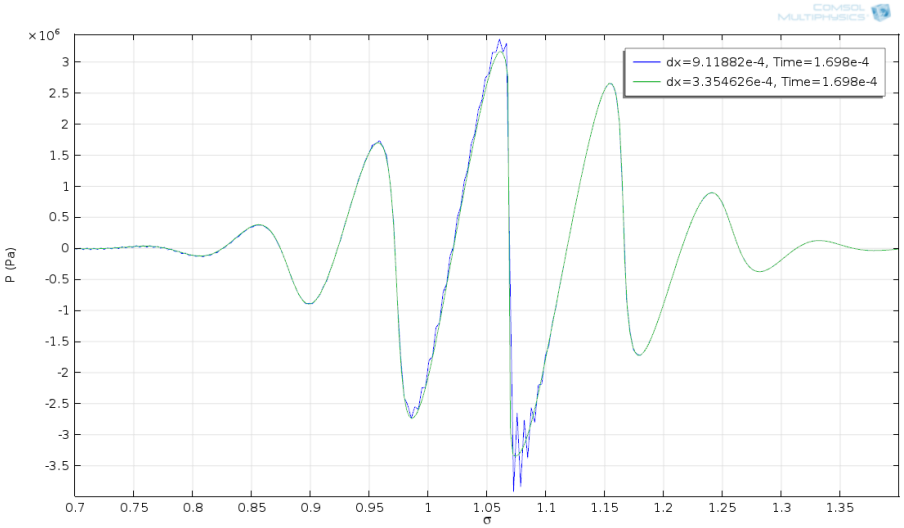


Figure 3.5: *The pressure field as function of the distance from the source at time* $t = 0.17$ ms. *Parameters are all set as in Table 1, except for* $P_0 = 3.4$ MPa *and* dx, *which is set as shown in the legend.* $x_{sh} = 0.152$ m *for reference. We conclude that a finer mesh suppresses the sawtooth-like oscillations.*

It has to be noted that a finer mesh (i.e. smaller dx) results directly in a finer time step (smaller dt) taken by the time solver. This is a necessity that we will see during all of our simulations and we discuss it more elaborately further on in this thesis. In conclusion, refining the mesh will result in suppressing the numerical oscillations and obtaining more accurate solutions overall. The cost is however, that each time step a larger (nonlinear) system of equations is solved and that more time steps must be taken to solve until a specific time (since smaller time steps dt were taken).

3.2.2 First-order mesh elements

Up until now we have only used quartic elements, and in the previous section we have seen that numerical oscillations are suppressed quickly when refining the mesh. In this subsection we examine if numerical errors can be reduced in similar ways for linear elements and we check how accurate the solution is when using linear (spatial) elements. In Figure 3.6 we have plotted a simulation with linear elements for various element sizes dx.

In Figure 3.6 we do not see any numerical oscillations occur as with quartic elements in Figure 3.5. We see that for the largest dx linear elements fail to accurately calculate the nonlinear effects and the solution is valid almost nowhere. Refining the mesh shows that the solution becomes more accurate, when comparing with Burgers solution, but still a large error is made around the peaks for the finest mesh ($dx \approx 0.123$ mm), which is suppressed only after a few elements. Comparing Figure 3.5 (quartic elements) with Figure 3.6 (linear elements) for $dx \approx 0.335$ mm, we conclude that quartic elements are much more accurate overall than linear elements (using the same number of elements n), when compared to the Burgers solution to this problem. A side note to this conclusion is that quartic element simulations have about 4 times as much degrees of freedom than linear element simulations, which results in a 4 times larger nonlinear system of equations that is solved for each time step. The element order has little to no effect on the time step taken however. In section 5.2 we will discuss this some more.

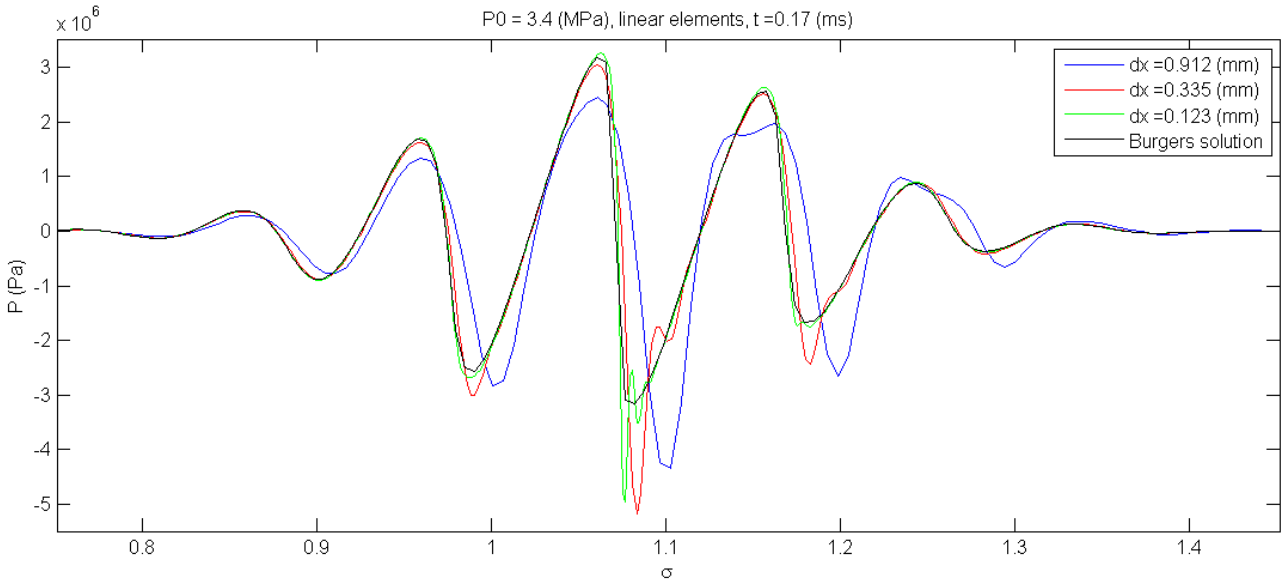


Figure 3.6: *The pressure field as function of the distance from the source at time* $t = 0.17$ ms. *Parameters are all set as in Table 1, except for* $P_0 = 3.4$ MPa *and* dx, *which is set as shown in the legend.* $x_{sh} = 0.152$ m *for reference. Linear elements need a finer mesh size than quartic elements to become accurate.*

3.2.3 Modeling with a damping term

In some numerical schemes a damping term is added to avoid numerical errors such as those arising in Figure 3.4. Up until now we have only studied the Westervelt equation without a damping term. We will also simulate a solution with a (physical) damping term. Acoustic attenuation is usually modeled with the power law and the attenuation coefficient is then given by [2] [4]

$$\alpha = \alpha_0 f^{\alpha_1}, \tag{3.2}$$

where α is the attenuation coefficient in $\frac{\text{Np}}{\text{m}}$, α_0 is a material property with the appropriate unit and α_1 is a unitless material property. For water, $\alpha_1 = 2$ and thus the attenuation scales quadratically with the frequency. This is exactly the thermoviscous model of attenuation, which is just a special case of the power law model of attenuation [2]. The power law gives attenuation in the frequency domain, which is difficult to implement in a time domain calculation, but the thermoviscous model has a time domain form. The complete (one-dimensional) Westervelt equation with thermoviscous attenuation is [2]

$$\frac{\partial^2 p}{\partial x^2} - \frac{1}{c_0^2}\frac{\partial^2 p}{\partial t^2} + \frac{\delta}{c_0^2}\frac{\partial^3 p}{\partial x^2 \partial t} = -\frac{\beta}{\rho_0 c_0^4}\frac{\partial^2 p^2}{\partial t^2}, \tag{3.3}$$

where δ is a different attenuation coefficient in $\frac{\text{m}^2}{\text{s}}$, which is related to α_0. Clearly the third term on the left hand side contributes (linearly) for attenuation. Physically, δ is a material property depending on several thermal and viscous material properties, such as thermal conductivity, both heat capacities and both viscosities (bulk and dynamic). See [2] for details. δ is related to α_0 by the following equation

$$\delta = \frac{\alpha_0 c_0^3}{2\pi^2}, \tag{3.4}$$

which is valid if $\alpha \ll k = \frac{2\pi f_0}{c_0}$. This requirement is usually met.

It is unclear from (3.3) that the damping increases quadratically with the frequency but this should be remembered, since it is equivalent to (3.2) with $\alpha_1 = 2$. This means that the higher-order harmonics are attenuated more quickly and thus attenuation is a counter effect that tends to weaken the nonlinear effect. We shall inspect this phenomenon for different damping coefficients δ. For water,

$\alpha_0 = 2.5 \cdot 10^{-14}$ $\frac{\text{Np}}{\text{m}\cdot\text{Hz}^2}$ [4], and it follows from (3.4) that $\delta = 4.1 \cdot 10^{-6}$ $\frac{\text{m}^2}{\text{s}}$. In Figure 3.7 we have plotted several simulations where we varied the damping coefficient.

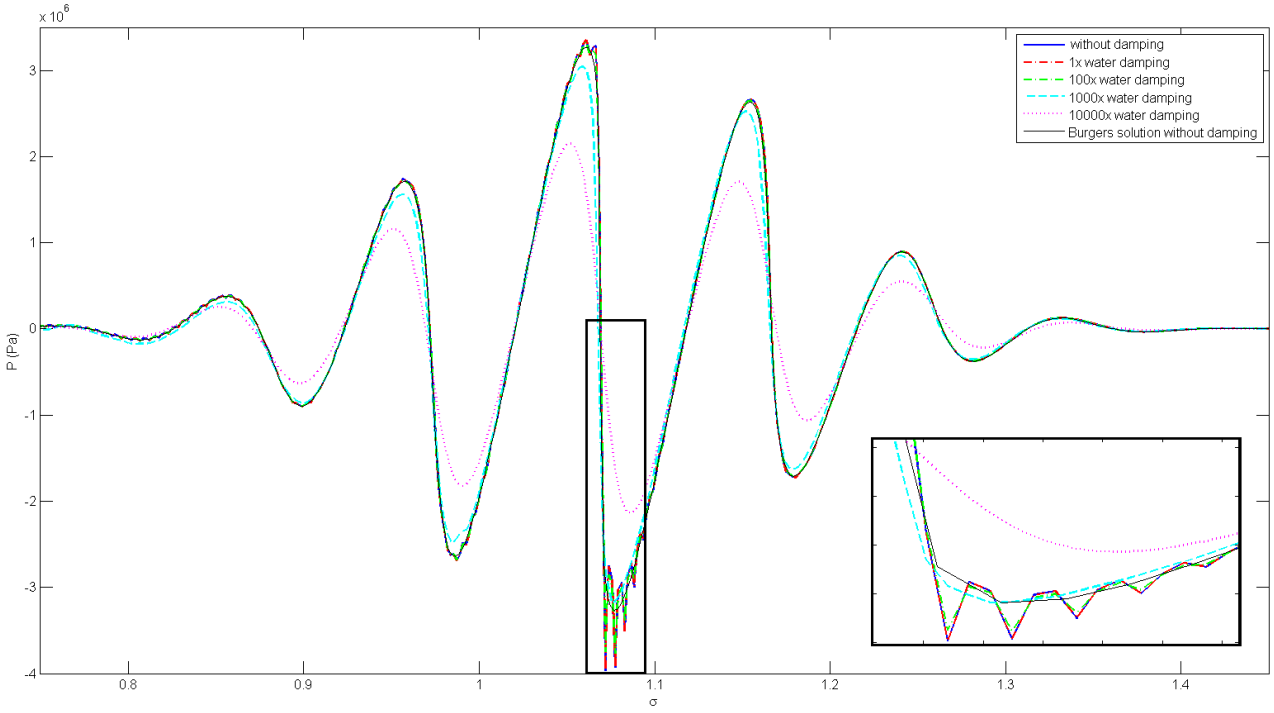


Figure 3.7: *The pressure field as a function of the distance from the source at time* $t = 0.17$ ms. *Parameters are all set as in Table 1, except for* $P_0 = 3.4$ MPa. $x_{sh} = 0.152$ m *for reference. The model used in this simulation is the Westervelt equation with a thermoviscous damping term added* (3.3). *The attenuation coefficient* δ *for water is* $\delta = 4.1 \cdot 10^{-6}$ $\frac{\text{m}^2}{\text{s}}$. *We see that physical damping does not make a difference in the numerical simulation. Strong damping reduces errors near the peaks, but damps out the nonlinear effect and the entire solution.*

We can conclude from this figure that damping is only a marginal solution to the numerical problems that occur near the shock wave distance. As can be seen in Figure 3.7, damping with regular strength (1x water) does not affect the solution at all. A 100 times stronger damping only very slightly reduces the problem near the bottom peak. At a damping of 1,000 times stronger we finally see the solution smoothening, but already at the cost of a reduction in the nonlinearity and in the amplitude. At a very strong 10,000 times damping, we see that the solution almost completely fails to simulate the nonlinear effects as well as a great reduction in amplitude.

From this we can conclude that physical realistic damping does not help us resolve numerical problems around the peaks of the solution near the shock wave distance. If we choose a physically realistic damping coefficient, we see that the solution is almost not affected at all. However with very strong damping, propagation trough the shock wave formation distance can be calculated smoothly, but it suppresses the nonlinear effect strongly and also attenuates the entire solution.

3.2.4 Inhomogeneous domains

One of the advantages of FEM simulations is that it is quite easy to incorporate location dependent physical properties. In other words, it is easy to calculate a solution on a Inhomogeneous domain. Usually the variation of these physical properties with respect to the position is in a discontinuous, step-like way, but not necessarily; for example if you have a certain temperature distribution along your domain and the physical properties are functions of the temperature. In that case the variation can be continuous (if, of course, the physical properties depend in a continuous way on the temperature).

In nonlinear wave propagation, the relevant physical properties are ρ_0, c_0 and β. So if more than one different material is used as a propagation medium, these properties will vary with position. We take

a look at an example where we used two layers of our regular material (water, but then with $\beta = 10$) and in between those layers one layer of a material with a smaller density and sound propagation speed, and a higher coefficient of nonlinearity (i.e. like fat, with a higher nonlinear coefficient). So now ρ_0, c_0 and β are (discontinuous, step-like) functions of x and we have set them in this example to:

$$\begin{aligned} c_0(x) &= \begin{cases} 1500\ (\frac{\mathrm{m}}{\mathrm{s}}), & \text{if } 0 < x < \frac{L}{3} \text{ or } \frac{2L}{3} < x < L \\ 1400\ (\frac{\mathrm{m}}{\mathrm{s}}), & \text{if } \frac{L}{3} < x < \frac{2L}{3} \end{cases} \\ \rho_0(x) &= \begin{cases} 1000\ (\frac{\mathrm{kg}}{\mathrm{m}^3}), & \text{if } 0 < x < \frac{L}{3} \text{ or } \frac{2L}{3} < x < L \\ 900\ (\frac{\mathrm{kg}}{\mathrm{m}^3}), & \text{if } \frac{L}{3} < x < \frac{2L}{3} \end{cases} \\ \beta(x) &= \begin{cases} 10, & \text{if } 0 < x < \frac{L}{3} \text{ or } \frac{2L}{3} < x < L \\ 13, & \text{if } \frac{L}{3} < x < \frac{2L}{3}. \end{cases} \end{aligned} \tag{3.5}$$

Note that the subscript 0 does not indicate whether it belongs to a certain material, but it indicates (for all materials) the small signal sound propagation speed in case of c_0 and the ambient density in case of ρ_0.

Recognize that we have wedged in a slab of length $L/3$ with different physical properties in the middle of our domain. Whenever the acoustic impedance $Z_0 = \rho_0 c_0$ is a discontinuous function of position, both reflection and transmission occur at the point of discontinuity.

The reflection coefficient (as a ratio of amplitudes) is defined as

$$R = \frac{Z_+ - Z_-}{Z_+ + Z_-}, \tag{3.6}$$

where Z_+ indicates the acoustic impedance of the medium towards which the wave is traveling and Z_- is the acoustic impedance of the medium in which the wave is currently traveling (and being reflected in).

So at the first boundary, $R = -0.087$ and at the second boundary $R = 0.087$. The minus sign indicates a phase change of 180 degrees at the reflection. This occurs since the wave is being reflected by a medium with a lower acoustic impedance, in which case the phase is always reversed. So in both reflections, the amplitude of the reflected wave has a magnitude a factor R times the amplitude of the incoming wave.

In Figure 3.8 we have plotted the solution at a time somewhere between the moment of the first and the second reflection. Since the incoming wave has an amplitude of $P_0 = 1$ MPa, we expect the reflected wave to have an amplitude of about 87 kPa, which is quite accurately solved for. Note that it is unnatural to speak of a certain shock wave distance x_{sh} for inhomogeneous problems, therefore the pressure field is plotted versus x instead of the usual $\sigma = x/x_{sh}$. We have therefore also set the domain length L to some estimated but fixed length.

3.3 Conclusions on the COMSOL solutions

In this section we collect the results and conclusions from all simulations in this chapter. Most of these conclusions will be qualitative, but they give us much insight into finite element solutions to the Westervelt equation. It is the basis for further research done in this thesis.

We see that as the pulse propagates, the nonlinear effect increases the magnitude of the spatial and time derivatives (both of first- and second-order), and where these are largest, numerical errors occur and solutions are inaccurate or even completely invalid. Calculating close to, or even through the shock wave results in some drastic errors, such as those in Figure 3.3.

These increasingly large derivatives (which scale with P_0) can even lead to numerical oscillations throughout the rest of the solution, a situation of which Figure 3.4 is an example.

Reducing the mesh size dx can contribute to a more accurate overall solution and can help reduce errors in the bottom peak. When decreasing the mesh size, we also have to decrease the size of the time step dt, so that the propagation is still calculated accurately. Reduction of both computational

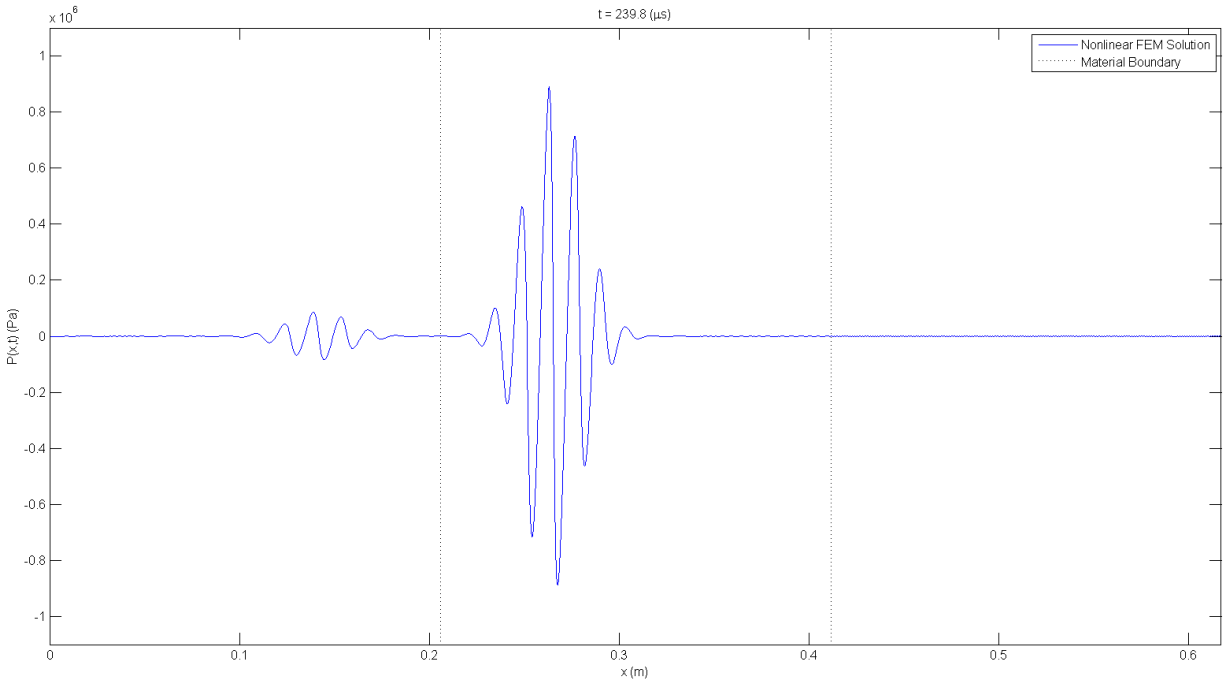


Figure 3.8: *The pressure field as a function of position at time* $t = 239.6$ μs*, a time somewhere in between the moment of the first reflection and the moment of the second reflection. Reflections occur due to discontinuities in the acoustic impedance, which are at* $x \approx 0.206$ m *and* $x \approx 4.12$ m*. All properties are set as in Table 1, except for those defined in* (3.5) *and* $L \approx 0.617$ m*. We see that a reflection has occurred which is described by* (3.6)*.*

parameters leads to smoother and more accurate results, but at a cost of computation time. This cost is twofold. A smaller mesh size directly leads to a larger number of elements n (on a fixed domain) and therefore increases the size of the nonlinear system of equations. The reduction in the time step taken leads to an increasing number of iterations to solve up to a certain final time, which means the nonlinear system of equations needs to be solved more times. In chapter 4 we return to this trade-off.

We then took a look at the effect of reducing the element order. Linear elements solve the Westervelt equation less accurate than quartic elements for the same number of elements n, both overall and when resolving the deep peaks. But more interestingly, it seams that a larger reduction factor in mesh size is needed to generate a more accurate solution than it did for quartic elements. We study this effect more quantitatively in section 5.2. Quartic elements however use 4 times as many degrees of freedom than linear elements, making it a more computationally demanding problem. It does not, or very little, affect the time step taken. Therefore the increase in computation time is only due to the larger system solved for per time step.

We also investigated if adding a damping term to the Westervelt equation reduces the error around the bottom peak. This can be predicted, since attenuation is a counter effect to the nonlinear effect. Using thermoviscous damping for water, we ran simulations for different values of the attenuation coefficient, which results have been collected in Figure 3.7. From this we conclude that damping up to 100 times stronger than physical damping in water, does not at all affect the solution. Even stronger damping, such as 1,000 or 10,000 times water damping, resulted in smoothness around the bottom peaks with, as a (large) trade-off, the reduction in solution amplitude and in nonlinearity. We conclude from this that it is not beneficial in general to incorporate acoustic attenuation. At this point we make the side note that some numerical implementations use numerical damping, which is something different than the physical damping studied here.

Finally, we solved a problem involving a inhomogeneous domain. Finite element simulations are very suitable for calculations on these sort of domains, especially when the physical quantities do not depend (directly or indirectly) on time. The inhomogeneity is then easily embedded in the spatial finite element formulation and solutions are simulated in roughly the same amount of computation time. We conclude from our calculated example (presented in Figure 3.8) that inhomogeneity does

not or very little affect the computation time and accuracy of the solution.

All this information obtained by running many COMSOL simulations, motivates us to look more closely into finite element methods specifically applied to the Westervelt equation. We further investigate the correlation between dx and dt and simulate with different time solving methods. We propose some form of local mesh refinement as a computationally more efficient way of reducing errors and increasing accuracy.

4 Using Galerkin's Finite Element Method for solving the Westervelt equation

In this section we formulate and implement a finite element method to solve the one-dimensional Westervelt equation. We do this to further analyse and resolve numerical problems which we encountered in Section 3.

4.1 Weak Formulation

We want to cast the one-dimensional Westervelt equation into its weak formulation. To do so, we start with the Westervelt equation accompanied by suitable initial and boundary conditions:

$$\begin{cases} \frac{\partial^2 p}{\partial x^2} - \frac{1}{c_0^2}\frac{\partial^2 p}{\partial t^2} = -\frac{\beta}{\rho_0 c_0^4}\frac{\partial^2 p^2}{\partial t^2} \\ p(0,t) = f(t) \\ p(L,t) = 0 \\ p(x,0) = 0 \\ \frac{\partial p}{\partial t}\Big|_{t=0} = 0. \end{cases} \tag{4.1}$$

Note that $f(t)$ is the source function at $x = 0$. Then we multiply the PDE by a (arbitrary) test function $v(x)$ and integrate over the domain to get

$$\int_0^L \frac{\partial^2 p}{\partial x^2} v(x)dx - \frac{1}{c_0^2}\int_0^L \frac{\partial^2 p}{\partial t^2} v(x)dx = -\frac{\beta}{\rho_0 c_0^4}\int_0^L \frac{\partial^2 p^2}{\partial t^2} v(x)dx, \tag{4.2}$$

which is the first weak formulation. We need to add a constraint to (4.2) in order to make it equivalent to (4.1). We require that v is sufficiently smooth and satisfies $v(0) = v(L) = 0$, since p has essential boundary conditions at $x = 0$ and $x = L$ [8]. Now if we integrate the first term by parts to reduce the highest order spatial derivative present in the weak formulation, we get

$$\int_0^L \frac{\partial^2 p}{\partial x^2} v(x)dx = \left[\frac{\partial p}{\partial x} v(x)\right]_0^L - \int_0^L \frac{\partial p}{\partial x}\frac{dv}{dx}dx = -\int_0^L \frac{\partial p}{\partial x}\frac{dv}{dx}dx, \tag{4.3}$$

where the last equality follows if we note that v vanishes at $x = 0$ and $x = L$. Plugging (4.3) into (4.2) and multiplying by -1 we get the desired weak formulation

$$\int_0^L \frac{\partial p}{\partial x}\frac{dv}{dx}dx + \frac{1}{c_0^2}\int_0^L \frac{\partial^2 p}{\partial t^2} v(x)dx = \frac{\beta}{\rho_0 c_0^4}\int_0^L \frac{\partial^2 p^2}{\partial t^2} v(x)dx, \tag{4.4}$$

which is now only valid for all smooth v satisfying $v(0) = v(L) = 0$, since we used this condition in (4.3).

4.2 Discretization of the linear wave equation

We start by taking a look at discretizing the linear wave equation, i.e. $\beta = 0$ in (4.1). So we start by discretization of the left-hand-side of (4.4).

4.2.1 Discretization in space

We can discretize the domain into n elements, and hence $n + 1$ nodes in a one-dimensional problem, where $x_0 = 0$ and $x_n = L$. The unknown solution p can then be expanded as a linear combination

$$p(x,t) \approx \sum_{i=0}^{n} u_i(t)\phi_i(x) = u_0(t)\phi_0(x) + u_n(t)\phi_n(x) + \sum_{i=1}^{n-1} u_i(t)\phi_i(x), \tag{4.5}$$

where $\{\phi_i : i = 0, 1, ..., n\}$ is a set of suitable basis functions around x_i. Note that u_0 and u_n are known from the Dirichlet boundary conditions. We choose ϕ_i to be a piecewise linear function such that

$$\phi_i(x_j) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}. \tag{4.6}$$

In Figure 4.1 we have shown two of such basis functions.

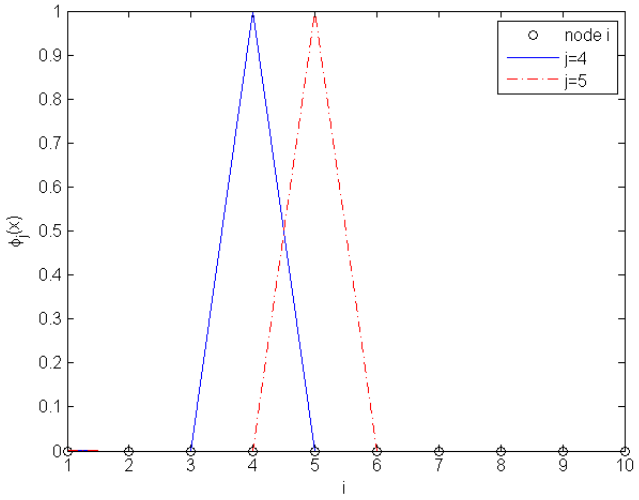


Figure 4.1: *Example of two adjacent basis functions*

If we assume that the test function v can also be represented by (linear combinations of) the set $\{\phi_i : i = 0, 1, ..., n\}$ [8], we obtain the following finite element representation of the weak form:

$$\begin{cases} \text{Find the set } \{u_1(t), ..., u_{n-1}(t)\} \text{ such that,} \\ \sum_{i=0}^{n} \left(u_i \int_0^L \phi_i' \phi_j' dx + \frac{1}{c_0^2} \ddot{u}_i \int_0^L \phi_i \phi_j dx \right) = 0 \qquad \forall j \in \{1, ..., n-1\}. \end{cases} \tag{4.7}$$

It is important to note that $\int_0^L \phi_i' \phi_j' dx$ and $\int_0^L \phi_i \phi_j dx$ vanish if $|j - i| > 1$, because of the choice of the basis functions.

Looking more closely to the case $j = 1$ in (4.7), we see that there is a non-vanishing term containing u_0, which accounts for the (time-dependent) Dirichlet boundary condition. Since $\phi_0(0) = 1$ it follows that $u_0(t) = f(t)$, the source function which is known. Hence, we can bring that term to the right-hand-side of the equation and obtain

$$\sum_{i=1}^{n} \left(u_i \int_0^L \phi_i' \phi_1' dx + \frac{1}{c_0^2} \ddot{u}_i \int_0^L \phi_i \phi_1 dx \right) = -u_0 \int_0^L \phi_0' \phi_1' dx - \frac{1}{c_0^2} \ddot{u}_0 \int_0^L \phi_0 \phi_1 dx. \tag{4.8}$$

Of course when $u_0(t)$ is known, $\ddot{u}_0$ can either be calculated analytically or can be approximated numerically to an arbitrary accuracy.

The case $j = n - 1$ is similar to $j = 1$, where a non-vanishing term containing u_n exists. u_n accounts for the homogeneous Dirichlet boundary condition at $x = L$ and therefore $u_n(t) = 0$. That term can just be eliminated from the set of equations.

Recognizing that the set of equations described by (4.7) can be written in matrix vector notation, we get

$$\mathbf{K}\mathbf{u} + \mathbf{M}\ddot{\mathbf{u}} = \mathbf{g}, \tag{4.9}$$

where the elements of the mass matrix $\mathbf{M}$ and the stiffness matrix $\mathbf{K}$ are

$$\begin{aligned} \mathbf{K}(i,j) &= \int_0^L \phi_i' \phi_j' dx \qquad \text{and} \\ \mathbf{M}(i,j) &= \frac{1}{c_0^2} \int_0^L \phi_i \phi_j dx \quad \forall i, j \in \{1, ..., n-1\} \end{aligned} \tag{4.10}$$

respectively, and the elements of $\mathbf{g}$ are

$$\mathbf{g}(j) = \begin{cases} -u_0(t) \int_0^L \phi_0' \phi_1' dx - \frac{1}{c_0^2} \ddot{u}_0(t) \int_0^L \phi_0 \phi_1 dx, & \text{if } j = 1 \\ 0, & \text{if } j \neq 1. \end{cases} \tag{4.11}$$

Note again that $\mathbf{M}(i,j)$ and $\mathbf{K}(i,j)$ equal zero if $|j-i|>1$ and thus $\mathbf{M}$ and $\mathbf{K}$ are very sparse $(n-1)\times(n-1)$ matrices. The right-hand-side vector $\mathbf{g}$ is a column vector of length $n-1$.

4.2.2 Discretization in time

The semi-discretized system (4.9) still needs to be discretized in time. We can do that by use of finite difference methods. We would like to use a backward differential formula accurate up to $\mathcal{O}\big((\Delta t)^2\big)$. This is given by (see Appendix B for derivation)

$$\mathbf{K}\mathbf{u}^{\mathbf{k}}+\mathbf{M}\frac{2\mathbf{u}^{\mathbf{k}}-5\mathbf{u}^{\mathbf{k-1}}+4\mathbf{u}^{\mathbf{k-2}}-\mathbf{u}^{\mathbf{k-3}}}{(\Delta t)^2}=\mathbf{g}^{\mathbf{k}}, \tag{4.12}$$

where $\mathbf{u}^{\mathbf{k}}$ is the vector containing the coefficients $\{u_1,...,u_{n+1}\}$ at time t_k. Solving (4.12) for $\mathbf{u}^{\mathbf{k}}$, we arrive at

$$((\Delta t)^2\mathbf{K}+2\mathbf{M})\mathbf{u}^{\mathbf{k}}=5\mathbf{M}\mathbf{u}^{\mathbf{k-1}}-4\mathbf{M}\mathbf{u}^{\mathbf{k-2}}+\mathbf{M}\mathbf{u}^{\mathbf{k-3}}+(\Delta t)^2\mathbf{g}^{\mathbf{k}}, \tag{4.13}$$

which can be solved quickly for $\mathbf{u}^{\mathbf{k}}$ by Matlab's backslash operator without inverting the matrix $((\Delta t)^2\mathbf{K}+2\mathbf{M})$. So $\mathbf{u}^{\mathbf{k}}$ can be calculated from the values of $\mathbf{u}$ at times t_{k-1}, t_{k-2} and t_{k-3}. Using iteration the solution can be solved in evolving time.

We identify $\mathbf{u}^{\mathbf{0}}$ as the initial condition, discretized in space. But in order to initiate the above scheme, we need an $\mathbf{u}^{\mathbf{1}}$ and a $\mathbf{u}^{\mathbf{2}}$. We can get $\mathbf{u}^{\mathbf{1}}$ from the initial condition on the first time derivative of the pressure field. If we discretize $\frac{\partial p}{\partial t}|_{t=0}$ as $\dot{\mathbf{u}}^{\mathbf{0}}$, then we can estimate $\mathbf{u}^{\mathbf{1}}$ by

$$\mathbf{u}^{\mathbf{1}}=\mathbf{u}^{\mathbf{0}}+\Delta t\dot{\mathbf{u}}^{\mathbf{0}}. \tag{4.14}$$

To solve for $\mathbf{u}^{\mathbf{2}}$, we start with one time step of first-order accuracy (i.e. use the third equation in (B.6)), which only depends on the previous two time steps. Now iterating over k steps gives the solution at $T=t_k=k\Delta t$. If some time between t_k and t_{k-1} is of interest, interpolation can be used.

A Matlab code implementing this finite element scheme can be found in Appendix C.1.

4.3 Discretization of the nonlinear term

4.3.1 Discretization in space

We start by expanding the time derivative in the right-hand-side term of (4.4), which gives us

$$\frac{\partial^2 p^2}{\partial t^2}=\frac{\partial}{\partial t}\left(\frac{\partial}{\partial t}p^2\right)=\frac{\partial}{\partial t}\left(2p\frac{\partial p}{\partial t}\right)=2\left(\frac{\partial p}{\partial t}\right)^2+2p\frac{\partial^2 p}{\partial t^2}. \tag{4.15}$$

Using the same basis function and linear expansion of $p(x,t)$ as in Section 4.2, we get the following set of terms on the right-hand-side of (4.7)

$$\frac{2\beta}{\rho_0 c_0^4}\int_0^L\Big(\sum_{i=0}^{n}\dot{u}_i\phi_i\Big)\Big(\sum_{m=0}^{n}\dot{u}_m\phi_m\Big)\phi_j dx+\frac{2\beta}{\rho_0 c_0^4}\int_0^L\Big(\sum_{i=0}^{n}\ddot{u}_i\phi_i\Big)\Big(\sum_{m=0}^{n}u_m\phi_m\Big)\phi_j dx, \tag{4.16}$$

for the j^{th} equation, where $j\in\{1,2,...,n-1\}$. Interchanging summation over i and integration, we get

$$\sum_{i=0}^{n}\dot{u}_i\frac{2\beta}{\rho_0 c_0^4}\int_0^L\Big(\sum_{m=0}^{n}\dot{u}_m\phi_m\Big)\phi_i\phi_j dx+\sum_{i=0}^{n}\ddot{u}_i\frac{2\beta}{\rho_0 c_0^4}\int_0^L\Big(\sum_{m=0}^{n}u_m\phi_m\Big)\phi_i\phi_j dx. \tag{4.17}$$

Now if we recognize that both integrals vanish if $m\neq j-1,j,j+1$ (because of the choice of the basis functions), we can eliminate the summation over m to arrive at

$$\begin{array}{l}\sum_{i=0}^{n}\dot{u}_i\left(\dot{u}_{j-1}\frac{2\beta}{\rho_0 c_0^4}\int_0^L\phi_{j-1}\phi_i\phi_j dx+\dot{u}_j\frac{2\beta}{\rho_0 c_0^4}\int_0^L\phi_j\phi_i\phi_j dx+\dot{u}_{j+1}\frac{2\beta}{\rho_0 c_0^4}\int_0^L\phi_{j+1}\phi_i\phi_j dx\right)\quad+\\ \sum_{i=0}^{n}\ddot{u}_i\left(u_{j-1}\frac{2\beta}{\rho_0 c_0^4}\int_0^L\phi_{j-1}\phi_i\phi_j dx+u_j\frac{2\beta}{\rho_0 c_0^4}\int_0^L\phi_j\phi_i\phi_j dx+u_{j+1}\frac{2\beta}{\rho_0 c_0^4}\int_0^L\phi_{j+1}\phi_i\phi_j dx\right),\end{array}\tag{4.18}$$

for the j^{th} equation, where again $j\in\{1,...,n-1\}$. (4.18) is the term that should be added to the right-hand-side of (4.7).

Identifying three new matrices $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$ with elements

$$\begin{array}{rcll}\mathbf{C_1}(i,j) &=& \frac{2\beta}{\rho_0 c_0^4}\int_0^L\phi_{j-1}\phi_i\phi_j dx, & \\ \mathbf{C_2}(i,j) &=& \frac{2\beta}{\rho_0 c_0^4}\int_0^L\phi_j\phi_i\phi_j dx & \text{and}\\ \mathbf{C_3}(i,j) &=& \frac{2\beta}{\rho_0 c_0^4}\int_0^L\phi_{j+1}\phi_i\phi_j dx & \forall i,j\in\{1,...,n-1\}\end{array}\tag{4.19}$$

respectively, we can express (4.18) more compactly as

$$\begin{array}{l}\sum_{i=0}^{n}\dot{u}_i\left(\dot{u}_{j-1}\mathbf{C_1}(i,j)+\dot{u}_j\mathbf{C_2}(i,j)+\dot{u}_{j+1}\mathbf{C_3}(i,j)\right)\quad+\\ \sum_{i=0}^{n}\ddot{u}_i\left(u_{j-1}\mathbf{C_1}(i,j)+u_j\mathbf{C_2}(i,j)+u_{j+1}\mathbf{C_3}(i,j)\right).\end{array}\tag{4.20}$$

In each of the equations $j=1,...,n-1$ where the term between brackets does not vanish for $i=0$ and $i=n$ (this is only the case for $j=1$ and $j=n-1$ because of the choice of basis functions), those contributions to the summation, which account for the Dirichlet boundary conditions, are separated and absorbed into the $\mathbf{g}$ vector as mentioned in Section 4.2.

Now if we look carefully at the first term of (4.20) we see that for a fixed j^{th} equation, all elements $\mathbf{C_1}(i,j)\quad\forall i\in\{1,...,n-1\}$ are multiplied by $\dot{u}_{j-1}$. All elements $\mathbf{C_2}(i,j)\quad\forall i\in\{1,...,n-1\}$ are multiplied by $\dot{u}_j$ and all elements $\mathbf{C_3}(i,j)\quad\forall i\in\{1,...,n-1\}$ are multiplied by $\dot{u}_{j+1}$. The second term of (4.20) is similar except each row of a $\mathbf{C}$ matrix is multiplied by u_{j-1}, u_j and u_{j+1} respectively. This applies for all $j=1,...,n-1$. Therefore (4.20) can be written in full matrix form as

$$\begin{array}{l}\left(\begin{bmatrix}\dot{u}_0&0&\cdots&0\\0&\dot{u}_1&\ddots&\vdots\\\vdots&\ddots&\ddots&0\\0&\cdots&0&\dot{u}_{n-2}\end{bmatrix}\mathbf{C_1}+\begin{bmatrix}\dot{u}_1&0&\cdots&0\\0&\dot{u}_2&\ddots&\vdots\\\vdots&\ddots&\ddots&0\\0&\cdots&0&\dot{u}_{n-1}\end{bmatrix}\mathbf{C_2}+\begin{bmatrix}\dot{u}_2&0&\cdots&0\\0&\dot{u}_3&\ddots&\vdots\\\vdots&\ddots&\ddots&0\\0&\cdots&0&\dot{u}_n\end{bmatrix}\mathbf{C_3}\right)\dot{\mathbf{u}}+\\ \left(\begin{bmatrix}u_0&0&\cdots&0\\0&u_1&\ddots&\vdots\\\vdots&\ddots&\ddots&0\\0&\cdots&0&u_{n-2}\end{bmatrix}\mathbf{C_1}+\begin{bmatrix}u_1&0&\cdots&0\\0&u_2&\ddots&\vdots\\\vdots&\ddots&\ddots&0\\0&\cdots&0&u_{n-1}\end{bmatrix}\mathbf{C_2}+\begin{bmatrix}u_2&0&\cdots&0\\0&u_3&\ddots&\vdots\\\vdots&\ddots&\ddots&0\\0&\cdots&0&u_n\end{bmatrix}\mathbf{C_3}\right)\ddot{\mathbf{u}}.\end{array}\tag{4.21}$$

Note that all explicitly given matrices in (4.21) are diagonal matrices, such that when matrix multiplication is used, the first row of $\mathbf{C_x}$ is multiplied by the first term on the diagonal and so on. Also note that we know u_0 and u_n from the given Dirichlet boundary conditions and there derivatives can therefore be calculated analytically or numerically.

Now name the matrices in the first row of (4.21) from left to right $\mathbf{H_{-1}}(\dot{\mathbf{u}})$, $\mathbf{H_0}(\dot{\mathbf{u}})$, $\mathbf{H_1}(\dot{\mathbf{u}})$ and in the second row $\mathbf{I_{-1}}(\mathbf{u})$, $\mathbf{I_0}(\mathbf{u})$, $\mathbf{I_1}(\mathbf{u})$, where the bracket notation means that $\mathbf{H}$ is a function of $\dot{\mathbf{u}}$ and $\mathbf{I}$ is a function of $\mathbf{u}$.

Defining

$$\begin{aligned} \mathbf{N_1}(\dot{\mathbf{u}}) &= \mathbf{H_{-1}}(\dot{\mathbf{u}})\mathbf{C_1} + \mathbf{H_0}(\dot{\mathbf{u}})\mathbf{C_2} + \mathbf{H_1}(\dot{\mathbf{u}})\mathbf{C_3} \quad \text{and} \\ \mathbf{N_2}(\mathbf{u}) &= \mathbf{I_{-1}}(\mathbf{u})\mathbf{C_1} + \mathbf{I_0}(\mathbf{u})\mathbf{C_2} + \mathbf{I_1}(\mathbf{u})\mathbf{C_3} \quad , \end{aligned} \tag{4.22}$$

we can finally state the semi-discrete matrix form of the finite element formulation of the one-dimensional Westervelt equation:

$$\mathbf{Ku} + \mathbf{M}\ddot{\mathbf{u}} = \mathbf{N_1}(\dot{\mathbf{u}})\dot{\mathbf{u}} + \mathbf{N_2}(\mathbf{u})\ddot{\mathbf{u}} + \mathbf{g}, \tag{4.23}$$

where $\mathbf{g}$ now also includes the Dirichlet boundary condition contributions of the nonlinear term:

$$\mathbf{g}(j) = \begin{cases} -\int_0^L \phi_0'\phi_1' dx - \frac{1}{c_0^2}\ddot{u}_0 \int_0^L \phi_0\phi_1 dx \\ +\dot{u}_0 \left(\dot{u}_0 \frac{2\beta}{\rho_0 c_0^4} \int_0^L \phi_0\phi_0\phi_1 dx + \dot{u}_1 \frac{2\beta}{\rho_0 c_0^4} \int_0^L \phi_1\phi_0\phi_1 dx \right) \\ +\ddot{u}_0 \left(u_0 \frac{2\beta}{\rho_0 c_0^4} \int_0^L \phi_0\phi_0\phi_1 dx + u_1 \frac{2\beta}{\rho_0 c_0^4} \int_0^L \phi_1\phi_0\phi_1 dx \right), & \text{if } j = 1 \\ 0, & \text{if } j \neq 1. \end{cases} \tag{4.24}$$

4.3.2 Discretization in time

Now we focus on the time discretization of (4.23). As in the linear case, we use a second-order backward differential scheme. Derivation of expressions for the derivatives can be found in Appendix B. Applying them to (4.23) gives us

$$\begin{aligned} &\mathbf{Ku^k} + \mathbf{M}\frac{\mathbf{2u^k - 5u^{k-1} + 4u^{k-2} - u^{k-3}}}{(\Delta t)^2} = \\ &\mathbf{N_1}\left(\frac{\mathbf{3u^k - 4u^{k-1} + u^{k-2}}}{\mathbf{2\Delta t}}\right)\frac{\mathbf{3u^k - 4u^{k-1} + u^{k-2}}}{2\Delta t} + \mathbf{N_2}(\mathbf{u^k})\frac{\mathbf{2u^k - 5u^{k-1} + 4u^{k-2} - u^{k-3}}}{(\Delta t)^2} + \mathbf{g^k}, \end{aligned} \tag{4.25}$$

where just as before we find $\mathbf{u^1}$ from the given initial condition $\mathbf{u^0}$ and its derivative $\dot{\mathbf{u}}^\mathbf{0}$, and find $\mathbf{u^2}$ from a single first-order BDF time step.

(4.25) is an implicit system of nonlinear equations in unknown $\mathbf{u^k}$ which can not be made explicit in a straight forward way. Therefore we solve it by the method of successive approximation. Because the time step Δt is small, we expect this method to converge in just a few iterations.

During each time step we start by estimating $\mathbf{u^k}$, i.e. by ${}^\mathbf{0}\mathbf{u^k} = \mathbf{u^k_{linear}}$, where $\mathbf{u^k_{linear}}$ is the solution to (4.13) and the left superscript 0 means that it is an initial guess for $\mathbf{u^k}$. We can then, based on this guess, calculate the matrices $\mathbf{N_1}$ and $\mathbf{N_2}$. Then we can solve (4.25) and obtain ${}^\mathbf{1}\mathbf{u^k}$. Now we can repeat calculating $\mathbf{N_1}$ and $\mathbf{N_2}$, and solving (4.25) until $\max(|{}^\mathbf{r}\mathbf{u^k} - {}^{\mathbf{r-1}}\mathbf{u^k}|) < \epsilon$, where ϵ is a small number defining the allowed tolerance. Then we set $\mathbf{u^k} = {}^\mathbf{r}\mathbf{u^k}$ and we have finished this time step. This process, which is called successive approximation, is summarized step by step below:

1. Initiate by guessing ${}^\mathbf{0}\mathbf{u^k} = \mathbf{u^k_{linear}}$.
2. Calculate $\mathbf{N_1}$ and $\mathbf{N_2}$.
3. Solve (4.25) for ${}^\mathbf{r}\mathbf{u^k}$
4. If $\max(|{}^\mathbf{r}\mathbf{u^k} - {}^{\mathbf{r-1}}\mathbf{u^k}|) < \epsilon$ go to 5.
   Otherwise go to 2.
5. set $\mathbf{u^k} = {}^\mathbf{r}\mathbf{u^k}$.

4.4 Using a Matlab build-in time solver

Instead of using a second-order accurate BDF formula with constant time step, we can also choose to use one of the more sophisticated build-in time solvers in Matlab. Solutions can be best obtained with the solver ode15s [13], which solves (systems of) stiff differential equations. The set of equations (4.23) is or can become quite stiff towards the shock wave formation distance.

Since all Matlab's solvers solve equations of the form

$$M(t,y)\dot{y} = f(t,y), \tag{4.26}$$

where $M(t,y)$ is an optional mass matrix depending on t and the solution $y(t)$, and $f(t,y)$ is the right hand side function, which may also depend on t and $y(t)$. To solve (4.23) with ode15s, we need to get it into the above form.

We start by rewriting (4.23) as a system of first-order differential equations. If we substitute $\mathbf{y_1} = \mathbf{u}$ and $\mathbf{y_2} = \dot{\mathbf{u}}$, and recognizing that $\dot{\mathbf{y}}_\mathbf{1} = \mathbf{y_2}$ we get

$$\begin{aligned} \mathbf{K}\mathbf{y_1} \;+\; \mathbf{M}\dot{\mathbf{y}}_\mathbf{2} \;&=\; \mathbf{N_1}(\mathbf{y_2})\mathbf{y_2} \;+\; \mathbf{N_2}(\mathbf{y_1})\dot{\mathbf{y}}_\mathbf{2} \;+\; \mathbf{g}(t) \\ \dot{\mathbf{y}}_\mathbf{1} \;&=\; \mathbf{y_2}. \end{aligned} \tag{4.27}$$

Collecting the terms of the first equation, and putting the above equations in one system (of $(2n-2)\times(2n-2)$ equations), we arrive at

$$\begin{bmatrix} \mathbf{O} & \mathbf{M}-\mathbf{N_2} \\ \mathbf{I} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{y}}_\mathbf{1} \\ \dot{\mathbf{y}}_\mathbf{2} \end{bmatrix} = \begin{bmatrix} -\mathbf{K} & \mathbf{N_1} \\ \mathbf{O} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{y_1} \\ \mathbf{y_2} \end{bmatrix} + \begin{bmatrix} \mathbf{g} \\ \mathbf{0} \end{bmatrix}, \tag{4.28}$$

where $\mathbf{I}$ is the $(n-1)\times(n-1)$ identity matrix, $\mathbf{O}$ is the zero matrix and $\mathbf{0}$ is a zero column vector of length $n-1$. The left-hand-side matrix is identified as the mass matrix of this problem, which depends on $\mathbf{y}$ via $\mathbf{N_2}$. The complete right-hand-side is identified as $\mathbf{f}(t,\mathbf{y})$, where the direct dependence on t is via $\mathbf{g}$. Also note that the Jacobian is non-constant, since $\mathbf{N_1}$ depends on $\mathbf{y}$. Using sparsity and providing sparsity patterns for the Jacobian and the Mass matrices can speed up the calculations significantly.

4.5 Results

In this section we discuss the results obtained by the implementations as discussed in this chapter. We have made implementations both for solving (4.25), the scheme for the second-order BDF time solver, and for solving (4.28), the scheme that solves the semi-discretized system of equations with Matlab's ode15s. Both implementations were made in Matlab, and some data from COMSOL is imported into Matlab for comparison.

In these simulations, slightly different parameters were used. Without further detail we present these parameters in Table 2, except for dt and dx. We now set this to $\frac{dx}{4c_0}$ which is somewhat larger than before. This can be done since we use linear elements instead of quartic elements. This is because in simulations with quartic elements 3 auxiliary points between the element nodes are introduced, which effectively reduces the distance between calculation points by a factor 4. Hence, the time step needs to be about a factor 4 smaller than with linear elements to have the same number of time steps per spatial separation. The element size dx is somewhat smaller than before, since first-order elements are used throughout. This is to ensure simulations are still accurate. We now set $dx = \frac{c_0}{72f0}$, a factor 4 times finer than we used with quartic elements. Note that there are now 72 nodes per period (in space). The source function is unchanged and is still given by (3.1).

In Figure 4.2 we have plotted the result obtained with the second-order BDF implementation as discussed in Section 4.3. We have used all parameters as set in Table 2. The solution is plotted at the shock wave distance. We have run a FEM simulation of both the linear wave equation and the Westervelt equation. The linear FEM solution in this simulation is very accurate when compared against the analytical solution. From this we can conclude that the mesh size dx and time step dt were sufficiently small to calculate the wave propagation correctly. This can also be seen when we compare the nonlinear FEM solution to Burgers solution. The propagation is calculated nicely and

Table 2: *A summary of the parameters used for simulations of the one-dimensional Westervelt equation in all Matlab and COMSOL simulations presented in chapter 4. These parameters are used, unless specified differently.*

ρ_0	1000	$\frac{\text{kg}}{\text{m}^3}$	Material ambient density
c_0	1500	$\frac{\text{m}}{\text{s}}$	Small signal sound speed
β	10		Coefficient of nonlinearity
P_0	10^6	Pa	Maximum pressure amplitude
f_0	10^5	Hz	Single source frequency
T_d	$\frac{6}{f_0}$	s	Source envelope delay
T_w	$\frac{3}{f_0}$	s	Source envelope width
T_{end}	$2 \cdot T_d = \frac{12}{f_0}$	s	Source end time
x_{sh}	$\frac{\rho_0 c_0^3}{\beta P_0 2\pi f_0}$	m	Shock formation distance
L	$x_{sh} + T_{end} \cdot c_0$	m	Domain length
dx	$\frac{c_0}{72 f_0}$	m	Mesh element size
dt	$\frac{dx}{4c_0}$	s	Maximum time step
ϵ	$10^{-9} \cdot P_0$	Pa	Absolute tolerance for successive approximation
	1		Element order
	equidistant		Mesh type

the solution is reasonably accurate where the first and second-order derivatives are relatively small. It is again near the bottom peaks where the solution becomes erroneous.

In this implementation we used the method of successive approximation to solve the nonlinear system of equations for each time step. Note that we choose to set the absolute tolerance $\epsilon = 10^{-9} \cdot P_0 = 10^{-3}$ Pa. In Section 4.3.2 we argued that the number of iterations this would take during each time step was going to be small, because the time step taken is small and the nonlinear term is small compared to the linear terms in the Westervelt equation. To verify this, we plotted the number of iterations needed to solve the nonlinear system of equations for each time step versus the time step. The result is shown in Figure 4.3. From this figure we can see that the number of iterations needed is at most 3. This justifies the implementation of successive approximation over more complicated methods such as the Newton-Raphson method.

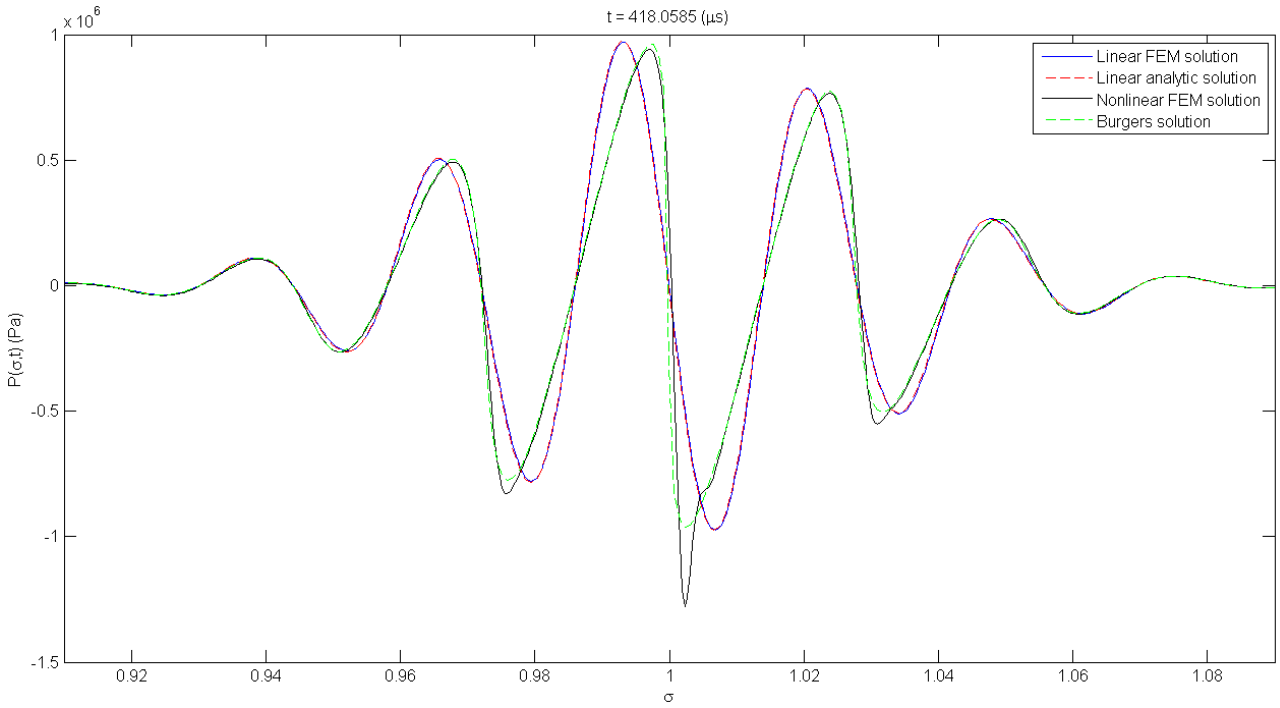


Figure 4.2: *Solution of the linear wave equation and the Westervelt equation calculated with our own implementation using the BDF time scheme as discussed in Section 4.3.2. Parameters were set as in Table 2.*

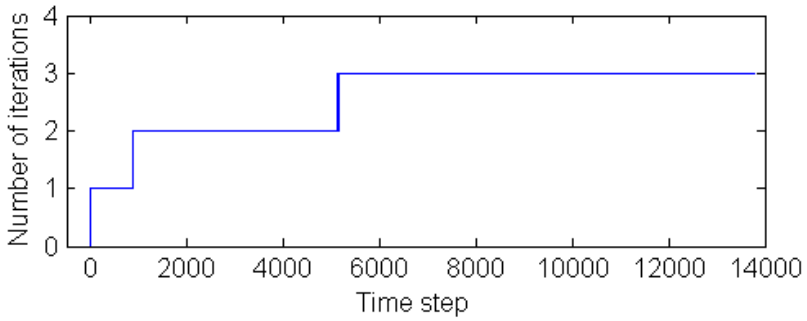


Figure 4.3: *Number of iterations that the successive approximation algorithm took to solve the nonlinear wave equation up to a desired tolerance per time step versus the number of the time steps. No more than three iterations were needed.*

In Figure 4.4 we have plotted the results we obtained with the Matlab build-in time solver ode15s as discussed in Section 4.4. We used the same parameters as before. This time, the solution algorithm also takes time steps based on a tolerance setting and estimates the best order of BDF it can use. It is similar to the time solver that COMSOL uses, and control is lost over the effective time step taken. We use it to test the performance of our BDF time solver. We can see that the linear wave propagation is still calculated nicely, but it seams to be just a little bit more off than with the BDF algorithm. The Westervelt equation was also solved and seems to have errors near the bottom peaks as well as the middle top peak. It is clearly less accurate than the BDF algorithm.

To better compare this notion, we have put together all Westervelt solutions we have obtained so far with parameters set as in Table 2. In Figure 4.5 we show a solution calculated by COMSOL, our BDF algorithm solution and our ode15s solution. We compare them all with the Burgers solution. Surprisingly we see that our solution with ode15s is roughly the same as COMSOL's solution. Apparently the time solver COMSOL uses is very much like the ode15s solver in Matlab. The Matlab implementation was somewhat more efficient however. We can also see that the BDF algorithm is the most accurate solver we have tested so far and it is also the fastest implementation of the three. We can conclude that for this specific problem it can be useful to closely look at what time solver you

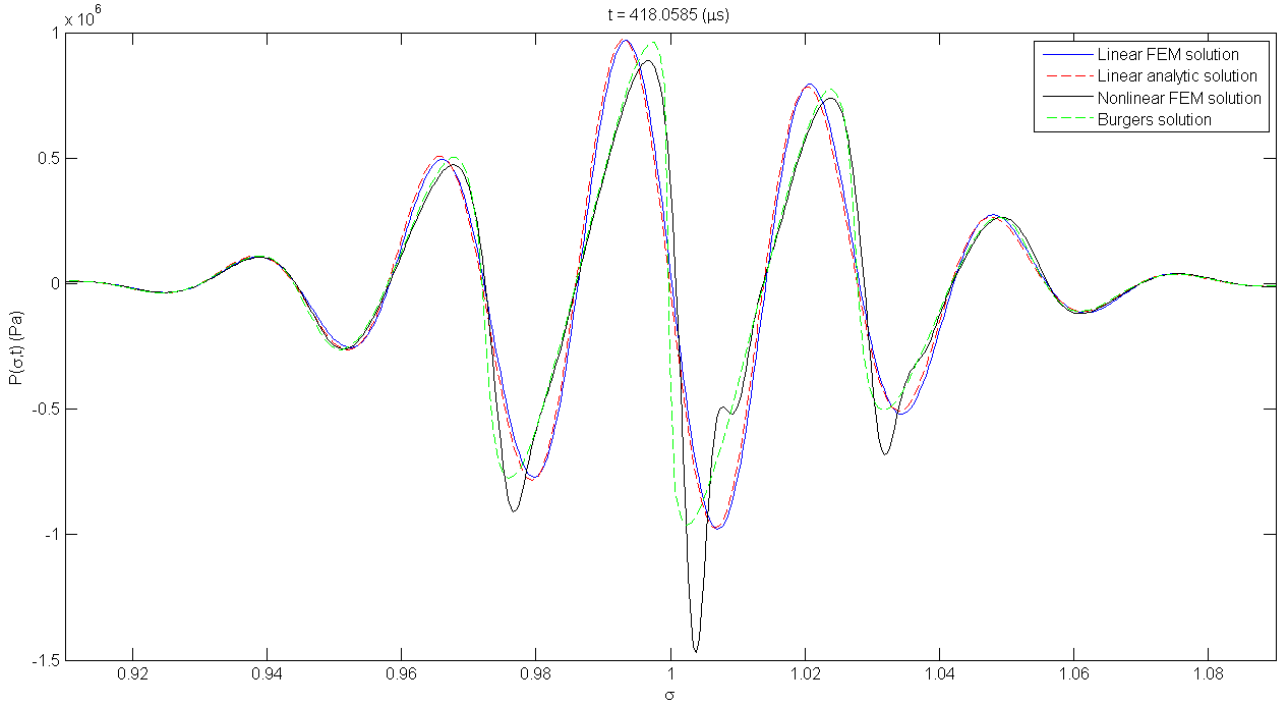


Figure 4.4: *Solution of the linear wave equation and the Westervelt equation calculated with our own implementation using the ode15s time scheme as discussed in Section 4.4. Parameters were set as in Table 2.*

want to use.

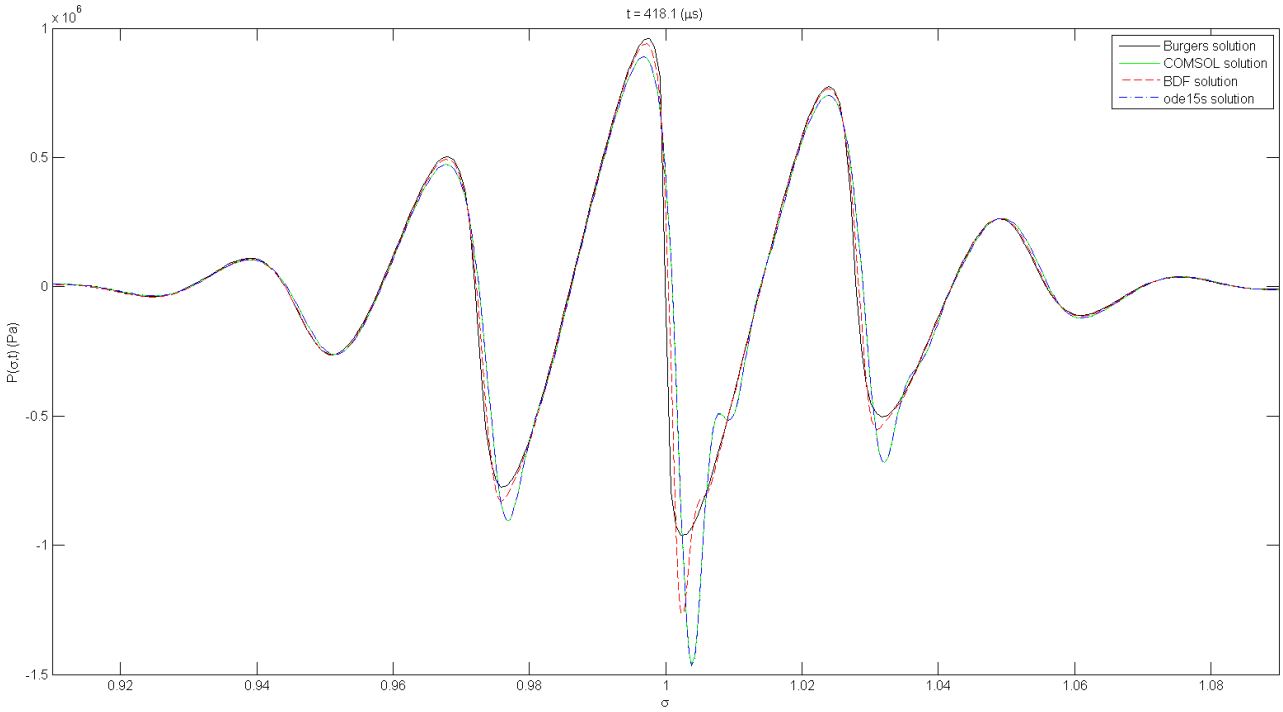


Figure 4.5: *Plot of the three nonlinear solvers we have seen so far as indicated in the legend. Parameters were set as in Table 2. Surprisingly the COMSOL and ode15s solutions coincide and the BDF solution is most accurate.*

We then used our BDF algorithm to see what the effect of reducing the mesh size is on the solution and to check if we see the same as with COMSOL. We refined the mesh by a factor 2 and ran another simulation, which we plotted in Figure 4.6. Just as we expected, the error decreased when we

decreased the mesh size. The error reduction in the middle bottom peak around $\sigma = 1$ is only marginal (especially if you consider the amount of extra computation time), but around the other peaks the solution clearly improved. This is probably due to the fact that the solution is still somewhat more smooth there.

In chapter 5 we discuss how adaptiveness can improve the accuracy of the solution only there where it is mostly needed.

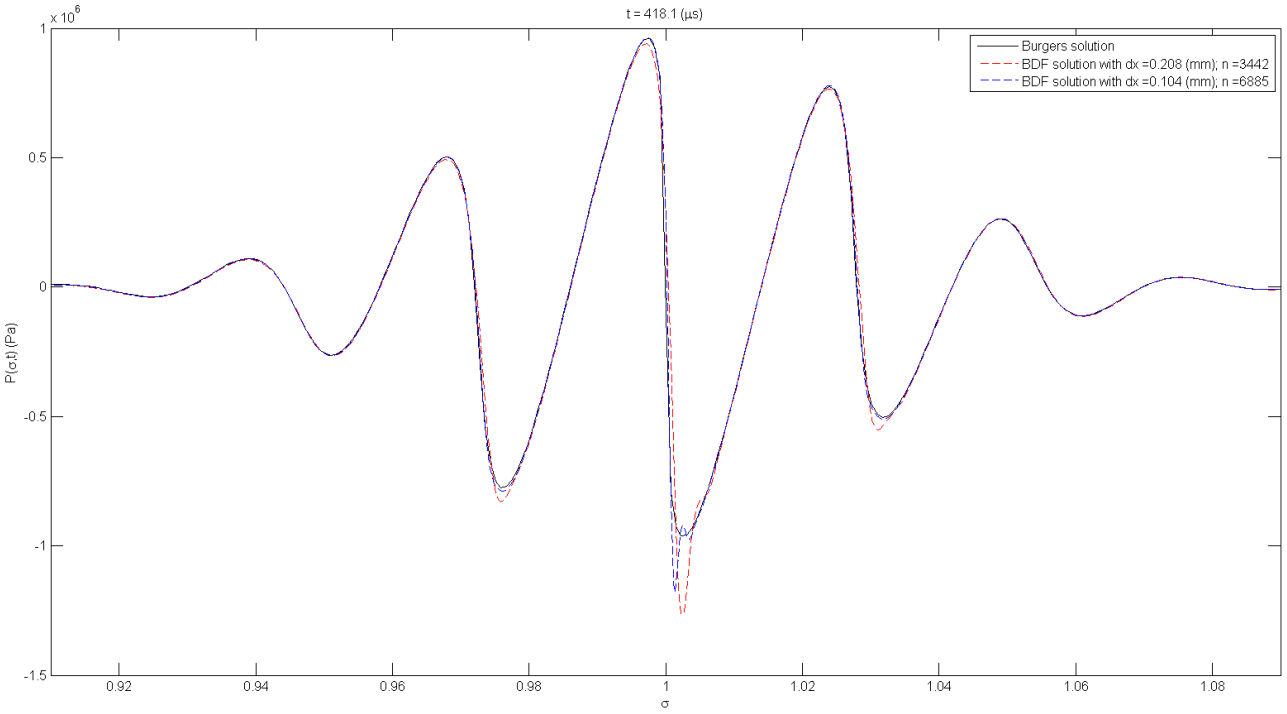


Figure 4.6: *Solutions obtained with our BDF implementation where we varied the element size dx as shown in the legend. The rest of the parameters were set as in Table 2.*

5 Adaptive Mesh simulations

In this chapter we research the types of spatial adaptiveness available in the FEM. We argue which type is applicable to the Westervelt equation and research the benefits that adaptiveness has on the solution. Adaptiveness tries to adjust the mesh, or the solution representation on this mesh, in such a way to optimally compute a solution up to a certain accuracy. Generally speaking, adaptive FEM computes a solution more efficiently than an ordinary FEM given a specified error.

5.1 Types of spatial adaptiveness

There are three common and distinct types of spatial adaptiveness available in the FEM [14] [15]. The first type of adaptiveness is the local refinement or coarsening of a mesh. This is known as h-adaptiveness. In this method nodal points are removed at points where the solution is estimated to be accurate and added where it is inaccurate. This method does not in general keep the number of elements, and therefore the number of degrees of freedom, constant. It is the most commonly used method because of its relative simplicity, but it is less applicable to very transient problems such as (nonlinear) wave propagation because it is incapable of following an evolving phenomenon in time. H-adaptiveness can be very useful in problems involving singularities however, since mesh refinement can be applied locally near the singularity. With respect to nonlinear wave propagation, this type of adaptiveness may be advantageous in standing wave problems.

The second type of adaptiveness is based on relocating or moving a mesh, which is known as r-adaptiveness. It leaves the number of elements and degrees of freedom constant, but moves the nodes along the mesh towards the location where they are mostly needed. Of course when the initial mesh is to coarse to calculate up to a desired accuracy, r-adaptiveness alone can not resolve the problem, since no degrees of freedom can be added. The other way around, if a mesh is to fine, i.e. it calculates more accurately than desired, no nodes can be removed to increase computational efficiency. Therefore the number of elements and degrees of freedom must be chosen carefully. R-adaptiveness is however extremely useful in problems where the elements need to follow a phenomenon that evolves in time, such as wave propagation.

The third type of adaptiveness varies the polynomial degree of the bases locally. This is called p-adaptiveness. Based on error estimations over each element, the order of the basis function is increased or decreased to calculate accurate solutions more optimally. This method keeps the number of elements constant, but varies the number of degrees of freedom. P-adaptiveness is very useful when the solution is sufficiently smooth. Also it is very efficient to implement, since the element stiffness and mass matrices only need to be recalculated if the order of the basis functions is changed. Thus, parts of the overall stiffness and mass matrices can remain unchanged, while the parts representing the elements that have changed can just be altered in the existing matrices. As mentioned before, this type of adaptiveness is by itself only useful on smooth solutions and therefore not applicable to nonlinear wave propagation.

All of the three adaptiveness methods can however be combined, even though this can significantly complicate the algorithm. The most commonly used is hp-adaptiveness which can efficiently solve problems where the solution varies only slowly in time but where singularities may be present. In the case of nonlinear wave propagation, we at least want to incorporate r-adaptivity to follow the time-evolving propagation, but both h- or p-adaptiveness may help to resolve the shock wave front that forms as the wave propagates. In this research we focus on r-adaptivity alone, but make sure that a large enough number of elements is chosen overall so that nodes are more free to move around as time evolves.

5.2 Different implementations of r-adaptiveness

In r-adaptivity (and also in h-adaptivity) the objective is to have the mesh evolve towards a certain equidistribution. An equidistribution is a node distribution which equally distributes the error approximation or monitor function. In this distribution, the mesh controlling quantity is equal over each element. There are many strategies in adaptive FEM to find the equidistribution and many of them

are based on a certain error approximation. In the equidistribution the (approximation of the) error attains the same maximum on each element, and thus calculating a solution in the most efficient way.

One of such available error estimations is the difference between the solutions calculated with quadratic and linear elements, given that a fine enough mesh is used. Wherever the difference is large, the solution is assumed to have a large error with respect to the actual (unknown) solution. That is exactly the location where r- or h-refinement needs to occur in order to reduce the global error and/or improve efficiency.

In order to show that this approach is successful, we first show that quadratic element solutions are more accurate than linear element solutions and that they converge more quickly. We do this by solving a (time-independent) Poisson equation with our own Galerkin's FEM implementation. The problem has the form

$$-\frac{d^2u}{dx^2} = g(x), \quad u(0) = 0 \quad \text{and} \quad \frac{du}{dx}\Big|_{x=1} = 0 \tag{5.1}$$

where $g(x)$ is a known forcing function. Of course an analytical solution can be obtained by direct integration and applying the boundary conditions. Figure 5.1 shows the (maximum) absolute error between the FEM solution and the analytical solution for both linear and quadratic elements as a function of the number of elements n. The straight slope in this double-logarithmic plot suggest a convergence of $\mathcal{O}(n^{-a})$. Also we can conclude that a is larger for second-order elements than for first-order elements, since the slope is larger. It is clear that for all number of elements n that the second-order elements are more accurate than the first-order elements.

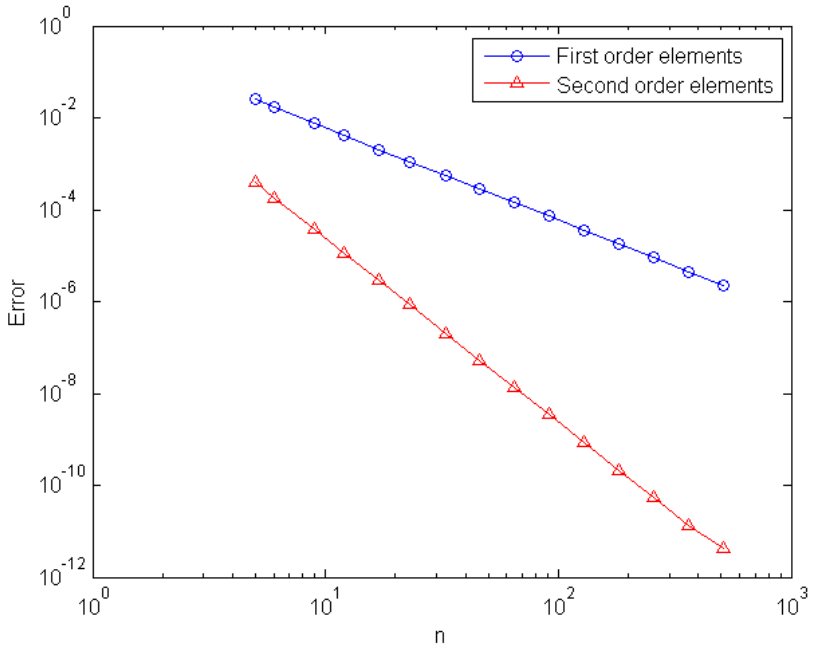


Figure 5.1: *Double-logarithmic plot of the (maximum) absolute error between the calculated and analytical solution versus the number of elements used in solving the Poisson equation* (5.1) *for a particular choice of* $g(x)$*. A comparison is drawn between linear and quadratic (first-order and second-order) elements both on an equidistant mesh.*

We then used this knowledge to better understand how the error evolves when solving the Westervelt equation. We used COMSOL to calculate a solution using linear and quadratic elements on a equidistant mesh with parameters set as in Table 2. Figure 5.2 shows the difference between second and first-order element solutions at five different times. Figure 5.3 shows both solutions for reference. We can see that an error is made at the location where the pulse is in the domain. Also the error increases as time evolves and the wave propagates (thus becoming more nonlinear). If, based on this knowledge, we can adapt to where the error is large and move our grid towards these areas, we can

reduce the overall error of the solution and calculate more efficiently. The error may also evolve slower in time.

Thus if we would implement a r-adaptiveness based on an error estimator that is the difference between the solutions calculated with different orders of basis functions, we would likely see a reduction in the global error, an increment in computational efficiency and a reduction in the way the error evolves in time. This method does however require that a solution is calculated using higher-order basis functions and this will always cost computation time, especially with a highly transient problem like this.

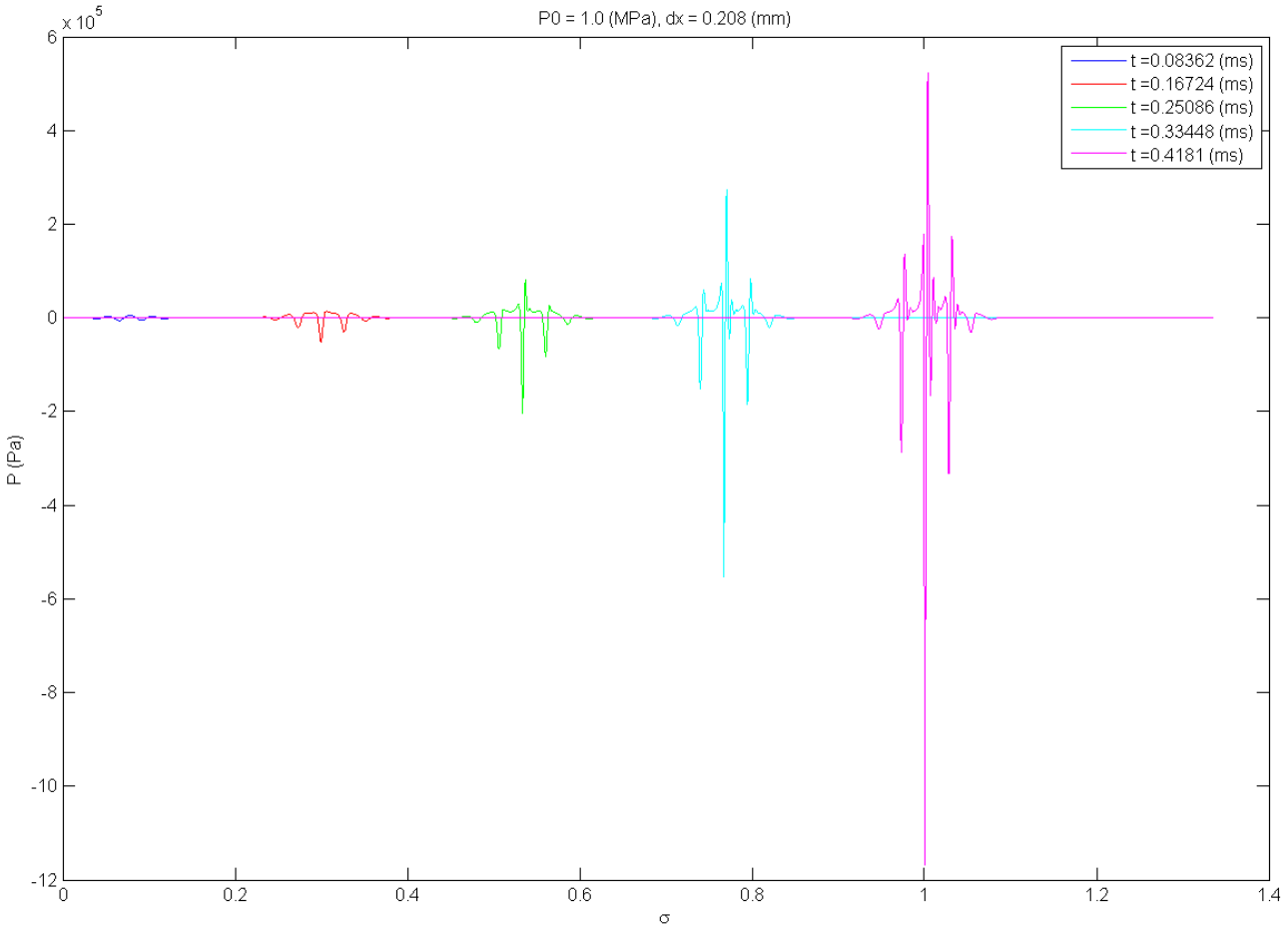


Figure 5.2: *Plot of the difference between the solution of the Westervelt equation using second- and first-order elements respectively at certain times t. Solutions were obtained with COMSOL and parameters were set as in Table 2. The solutions are shown in Figure 5.3.*

By comparing Figure 5.2 and Figure 5.3 we see that the error is large in the regions where the pulse is located and especially where the spatial derivatives are large. This motivates us to look into a type of r-adaptiveness that uses this information instead of an error approximation. In this way it is not needed to calculate a solution with higher-order elements. Data from only one solution (such as the magnitude of its first-order derivative in space) can be used. This can be a huge advantage in computation time and difficulty of the algorithm. In Sections 5.3 and 5.4 we discuss two of these r-adaptiveness implementations.

5.3 Implementation of MMPDE-6

In this section we discuss the implementation of a very specific type of r-adaptiveness called MMPDE-6 [12]. Instead of equally distributing an error estimator, this algorithm equidistributes a monitor function, which is based on the first-order spatial derivative of the solution. Its advantage is that no additional solutions need to be calculated and the method can thus be used on the fly. We have also

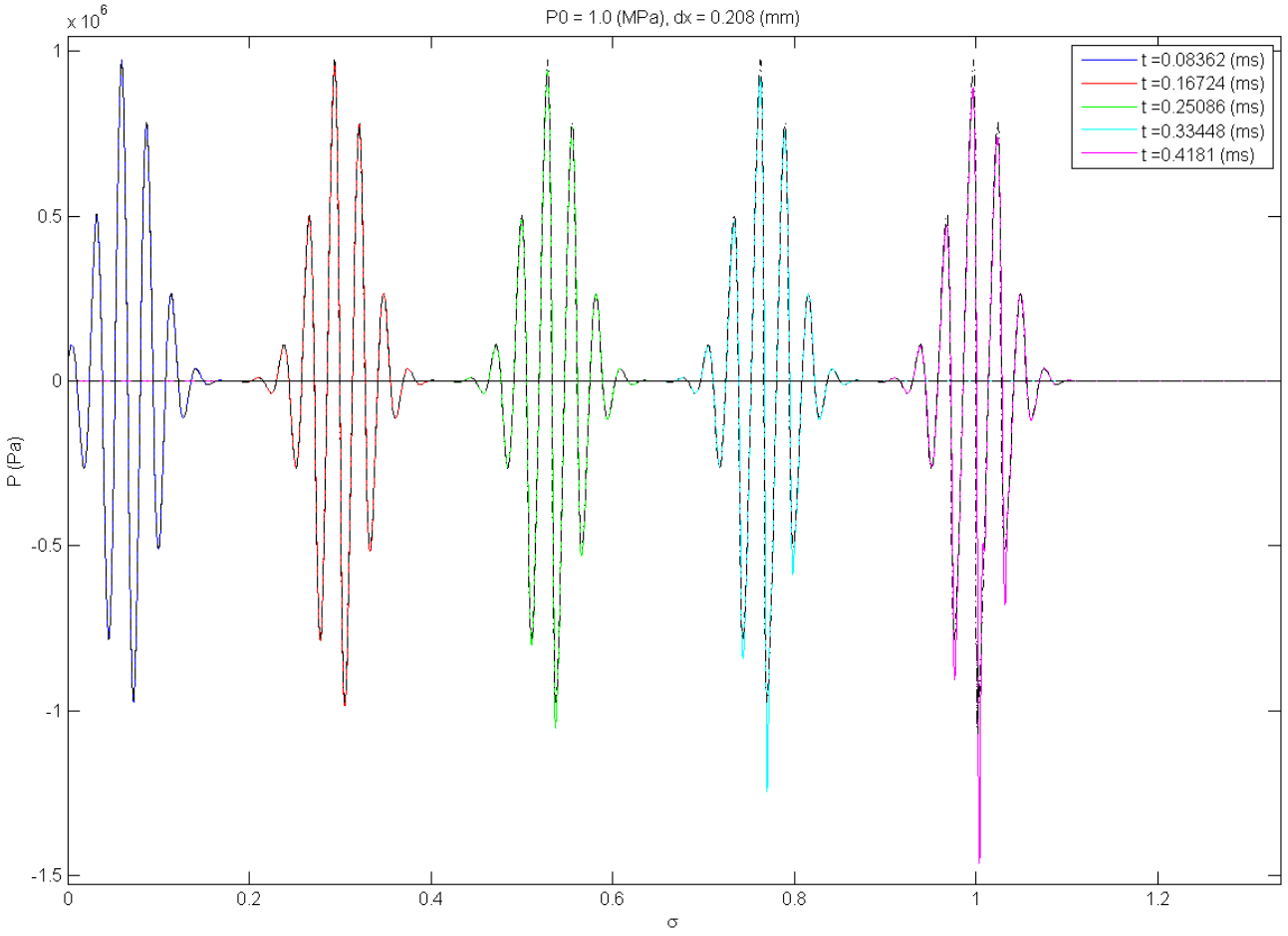


Figure 5.3: *Plot of the solution calculated with COMSOL using first-order elements (colored solid lines) and second-order elements (black dashed lines) at different times t. Parameters were set as in Table 2. Their difference is plotted in Figure 5.2.*

seen that errors are being made in solving the Westervelt equation in regions were the derivative is large.

The specific choice of the monitor function influences the adaptiveness and its accuracy, and much analysis can be put into motivating a certain monitor function. That is beyond the scope of this thesis and we use a widely used monitor function, also suggested by [12] and its references, which is the 'arc-length' monitor function which is defined as

$$M = \sqrt{1 + \left|\frac{du}{dx}\right|^2}. \tag{5.2}$$

Let us take a closer look into what we mean by equidistributing this monitor function. What our overall goal is, is to find out how the nodes evolve towards an equidistribution and how the equidistribution evolves in time (so that the nodes can follow this distribution). Let $\Omega_C = [0 \quad n]$ be the computational domain (now still continuous) and let $\Omega_P = [0 \quad L]$ be the physical domain. We denote $\xi \in \Omega_C$ as the computational variable and $x \in \Omega_P$ be the physical variable. We want to find the map

$$x(\xi, t): \quad \Omega_C \mapsto \Omega_P, \quad x(1,t) = 0 \quad \text{and} \quad x(n,t) = L,$$

that equidistributes M. One equation, which is known as MMPDE-6, which approximately equidistributes M is defined as [12]

$$\epsilon\Big(1 - \gamma\frac{\partial^2}{\partial\xi^2}\Big)\dot{x} = \frac{\partial}{\partial\xi}\Big(M\frac{\partial x}{\partial\xi}\Big), \tag{5.3}$$

where ϵ is a small parameter controlling how close the distribution $x(\xi,t)$ stays to the equidistribution as time evolves, and γ is a smoothing parameter. Now the solution of equation (5.3) $x(\xi,t)$ is the desired distribution that depends also via the monitor function on $\frac{\partial u}{\partial x}$.

If the right-hand-side of (5.3) is equated to zero, we obtain the equation that exactly equidistributes M over x at a given time t. Of course it is not always possible to find an exact solution $x(\xi)$. We want the equidistribution to evolve smoothly in time the same way as the monitor function evolves in time. To do this, we add the left-hand-side of (5.3). The distribution of the mesh is then kept ϵ-close to the initial distribution, while being smoothed due to the $\gamma\frac{\partial^2}{\partial\xi^2}$ term.

If the computational domain is then discretized, for example by choosing $\xi = 0, 1, 2, ..., n$, we also make the physical domain discrete via the introduced map. We then need to approximate (5.3) with finite difference and a typical discretization of (5.3) is given by [12]

$$\epsilon\Big(\mathbf{I} - \frac{\gamma}{(\Delta\xi)^2}\mathbf{A}\Big)\dot{\mathbf{x}} = \mathbf{E}, \tag{5.4}$$

where $\mathbf{A}$ is the $(n-1)\times(n-1)$ matrix with -2 on the main diagonal, 1 on the diagonals left and right of the main diagonal, such that $\frac{1}{(\Delta\xi)^2}\mathbf{Ax}$ is a central approximation for the second-order spatial derivative of x. $\mathbf{E}$ is the vector representation of the discretization of the right-hand-size of (5.3), which is calculated element by element as [12]

$$\begin{aligned} E_i &= \frac{2}{(\Delta\xi)^2}\left(M_{i+\frac{1}{2}}\Big(x_{i+1}-x_i\Big) - M_{i-\frac{1}{2}}\Big(x_i - x_{i-1}\Big)\right), \\ M_{i+\frac{1}{2}} &= \tfrac{1}{2}\Big(M_i + M_{i+1}\Big), \\ M_i &= \sqrt{1+\left|\frac{u_{i+1}-u_{i-1}}{x_{i+1}-x_{i-1}}\right|^2}, \qquad i = 1, 2, ..., n-1. \end{aligned} \tag{5.5}$$

Note that in our case $\Delta\xi = 1$. We now need to solve this (semi-discrete) system of equations simultaneously to the Westervelt equation. We can easily do this, by expanding the implementation of Section 4.4. If we expand the system of equations (4.28) by letting $\mathbf{y_3} = \mathbf{x}$, we now solve the system

$$\begin{bmatrix} \mathbf{O} & \mathbf{M}-\mathbf{N_2} & \mathbf{O} \\ \mathbf{I} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \epsilon(\mathbf{I}-\frac{\gamma}{(\Delta\xi)^2}\mathbf{A}) \end{bmatrix}\begin{bmatrix}\dot{\mathbf{y}}_\mathbf{1}\\ \dot{\mathbf{y}}_\mathbf{2}\\ \dot{\mathbf{y}}_\mathbf{3}\end{bmatrix} = \begin{bmatrix} -\mathbf{K} & \mathbf{N_1} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O}\end{bmatrix}\begin{bmatrix}\mathbf{y_1}\\ \mathbf{y_2}\\ \mathbf{y_3}\end{bmatrix} + \begin{bmatrix}\mathbf{g}\\ \mathbf{0}\\ \mathbf{E}(\mathbf{y_1},\mathbf{y_3})\end{bmatrix}, \tag{5.6}$$

where for clarity it is indicated that $\mathbf{E}$ depends on $\mathbf{y_1} = \mathbf{u}$ and $\mathbf{y_3} = \mathbf{x}$. It is important to understand that the mass matrix $\mathbf{M}$ and the stiffness matrix $\mathbf{K}$ now also depend on $\mathbf{y_3} = \mathbf{x}$. $\mathbf{A}$ is now the only matrix that does not depend on $\mathbf{y}$, and this is therefore a much stiffer and more nonlinear system of equations.

5.4 Implementation of Co-traveling refinement

A completely different, but effective, form of refinement can be implemented for our problem, since we follow only a small pulse traveling over the domain. Strictly speaking, this would be another form of r-adaptiveness, since no nodes or degrees of freedom are added. The idea is that we construct a refined area around the pulse at the expense of coarsening the areas where the solution is zero. This method can also be motivated by examining the Figures 5.2 and 5.3. We see that the error is being made somewhere within the pulse, while the error is very small in the zero parts of the solution. We can therefore afford to relocate nodes from the zero part of the solution towards the pulse.

This is not real adaptivity in the sense that information about the solution is used to update the mesh accordingly, but rather a more quasi-adaptiveness which is constructed with a priori knowledge only. By knowing the width of the pulse and the propagation speed we calculate for all future time where the pulse is going to be, and just refine the mesh there.

We know from the source function $f(t)$ that its duration (the amount of time it is nonzero) $D = \frac{12}{f_0}$. Hence the maximum pulse width $W = \frac{12\cdot c_0}{f_0}$. With the pulse starting at $t = 0$ we now have enough

information to divide the domain into three separate intervals I_1, I_2 and I_3, such that the pulse is always in I_2. We do this by defining $0 \leq a \leq b \leq L$ as

$$a = \max(c_0 \cdot t - W, 0) \quad \text{and} \quad b = \min(c_0 \cdot t, L).$$

We can now define

$$I_1 = [0 \quad a], \qquad I_2 = [a \quad b], \qquad I_3 = [b \quad L].$$

In the sub-domain I_2 we want to accomplish refinement with control over the minimum element size dx_{min}. In this algorithm we can not even take dx_{min} arbitrarily small without changing the number of elements n. Let's say we fix the number of elements n. On an equidistant grid, each element would have length $dx = \frac{L}{n}$. Since we can now refine with at most $n-2$ elements in I_2 the smallest element size dx_{min} satisfies

$$\frac{L}{n} = dx \geq dx_{min} \geq \frac{W}{L}\frac{n}{n-2}dx = \frac{W}{n-2}, \tag{5.7}$$

where W is the maximum length of I_2. The maximum refinement achieved is then defined as

$$q \equiv \frac{dx}{dx_{min}} \leq \frac{L}{W}\frac{n-2}{n} \leq \frac{L}{W} = \frac{\frac{\rho_0 c_0^3}{\beta P_0 2\pi f_0} + \frac{12c_0}{f_0}}{\frac{12c_0}{f_0}} = 1 + \frac{\rho_0 c_0^2}{\beta P_0 24\pi}. \tag{5.8}$$

Note that q is the ratio between the average element size and the minimum element size. Also, even for very large n, there is a maximum refinement possible depending on physical parameters. In practice it is easier to define dx_{min} via the refinement factor q. Having chosen a $q \geq 1$ or a dx that satisfies (5.7) and (5.8), we divide I_2 in an equidistant mesh with element size dx_{min}, which will then consist of fewer than n elements because of the above constraints, and we use the remainder of the elements to create equidistant meshes on I_1 and I_3.

In Figure 5.4 we have shown an example of the co-moving mesh with all parameters set as usual, but $n = 60$. We choose $q = 2$, which satisfies

$$1 \leq q = 2 \leq \frac{L}{W}\frac{n-2}{n} \approx 3.8.$$

We implemented this algorithm in our BDF scheme for the Westervelt equation, and interpolated the needed previous solution to the current mesh. Each time step we need to do three interpolations and recalculate all the matrices $\mathbf{M}$, $\mathbf{K}$, $\mathbf{C_1}$, $\mathbf{C_2}$ and $\mathbf{C_3}$. This may slow the down the time integration significantly.

5.5 Results

First we tried an implementation with the MMPDE-6 algorithm on linear wave propagation. The algorithm has two controlling properties ϵ and γ. γ is a parameter that controls smoothing, where a larger γ produces a smoother mesh. ϵ controls, together with the magnitude of the monitor function M, the speed at which nodes move towards an equidistribution and, when solved in the same physical time as the problem, also the size of the smallest element. This control is however marginal and some problems arise with this algorithm for the Westervelt equation.

First it leaves us little control over the smallest element size, practically letting nodes approach each other arbitrarily close. This in turn has the effect that the time step that can be taken will also reduce until no practical time step can be taken anymore. The algorithm can move the nodes so close to each other that the needed time step is so small, that a final solution is never going to be reached.

Also if nodes move too fast the solution can become unstable. This can occur because the mass and stiffness matrices change to rapidly over time near a specific point that is moving fast. This has its effect on the time solver and the overall stability of the solution. Also, when nodes move to rapidly, larger interpolation errors can be made when interpolating previous solutions to the current mesh.

By increasing ϵ or norming (reducing) the monitor function M, we can reduce this effect but it reduces the overall speed at which nodes move toward the needed areas. It can still follow the wave

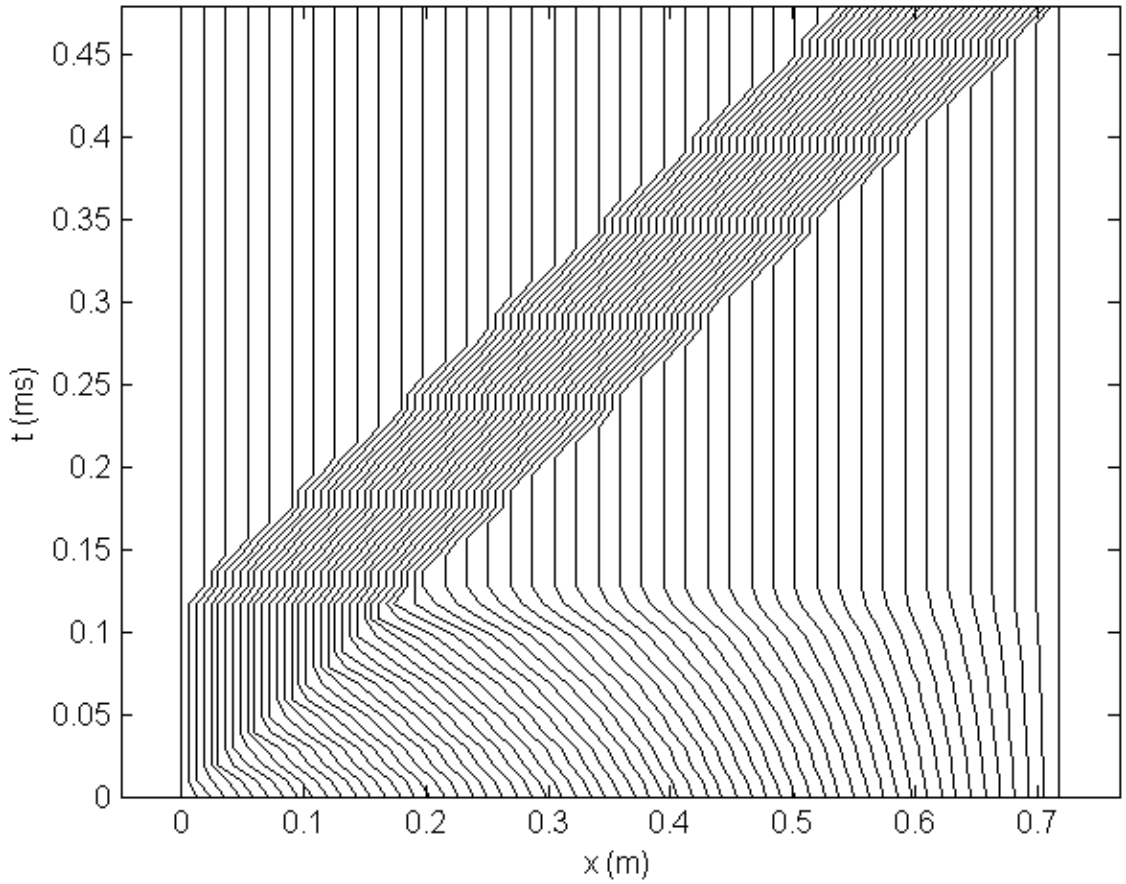


Figure 5.4: *Plot that shows how the nodes move in time with the co-traveling algorithm. Each line represents a node. Parameters were set as in Table 2, except for $n = 60$ (which overrides dx). We choose $q = 2$ for refinement.*

propagation, but a local refinement within the pulse is hard to accomplish. It also does require a better initial mesh.

The trade-off which needs to be made, has to take all three aspects into account. Nodes must not move to fast, in order to keep the solution stable. They must also not approach each other arbitrarily close or the time solver will get stuck. Finally, the adaptiveness must be quick enough in order to have a significant reduction in mesh size where needed.

It may be difficult for the algorithm to start from a zero solution and have the pulse enter the domain of interest via the boundary condition. Therefore it may be worthwhile to first solve on a fixed mesh until (at $t = \frac{12}{f_0}$) the pulse has fully entered the domain. Then, we can solve (5.3) for an artificial time variable to move the mesh close to the equidistribution. If the solution is then interpolated to this distribution we can start the MMPDE-6 algorithm from a distribution close to the equidistribution.

We have tried all of the above, but only for a large ϵ and a scaled M-function we can get a stable and accurate solution to the linear wave equation. However this leads to a very marginal refinement, which we will show in Figure 5.5. The cost of solving such a complicated and stiff system of equations does not outweigh the small refinement that is accomplished. Note that this is at best a refinement of 24%, and on average only 9.3%, compared to a uniform mesh.

We then turned to the co-traveling mesh refinement, which we developed specifically for this problem. It has the advantage that it gives full control over the smallest element size, and we can thus implement it safely in our BDF scheme without having to worry that our time step is not going to be small enough. dt will then of course depend on dx_{min}. Using the relation between dx_{min} and the refinement q as defined in section 5.4, we have control over all of our numerical parameters: the local refinement, the (minimum) element size and the time step. The drawback is that we can not achieve a refinement larger than $\frac{W}{L} \approx 4$ for our problem. Also the mesh is locally (within the area of the pulse) still uniform, which is not very advantageous towards the shock wave distance. Also, the

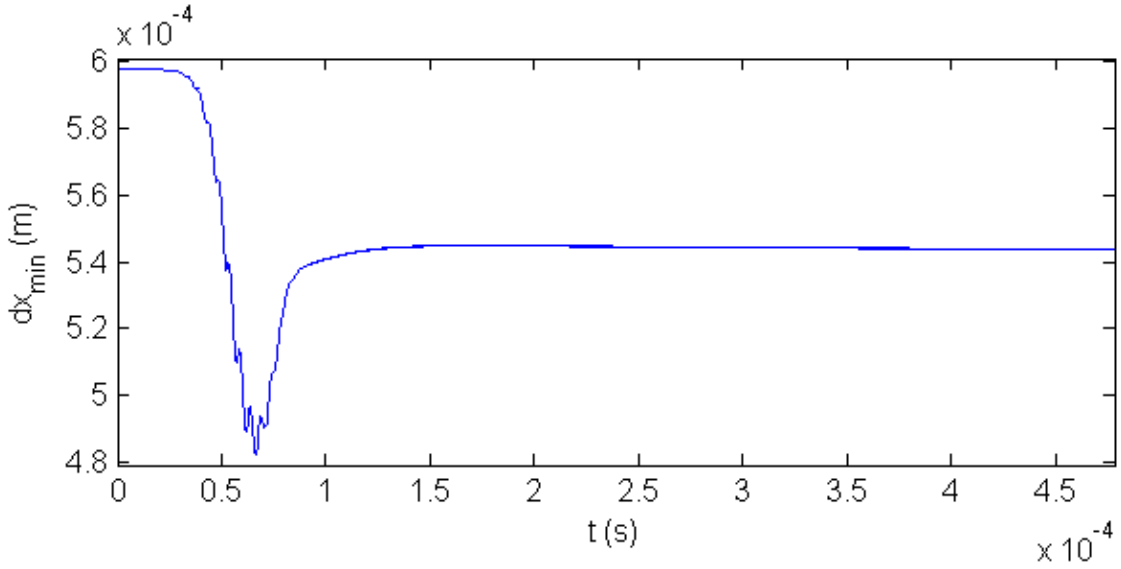


Figure 5.5: *Plot that shows the smallest element size in the domain versus the time for a MMPDE-6 simulation that was stable for all t solving the linear wave equation.*

nodes only move as fast as needed to follow the solution.

What this algorithm basically does is start with an equidistant mesh of $q \cdot n$ elements and then, at each time step, removes $(q-1) \cdot n$ nodes that are outside of the pulse. This reduces the size of the system by a factor q, but at the expense of interpolations and recalculations of all matrices each time step.

The current implementation works, although it is in its current form significantly slower than a calculation done on an equidistant mesh with $q \cdot n$ elements. This can be because of the specific implementation in Matlab; perhaps it may work more efficiently in other programming environments. But the advantage becomes larger when co-traveling refinement is used in three spatial dimensions.

In Figure 5.6 we have added a co-traveling mesh solution to Figure 4.6 to compare with solutions on finer but equidistant meshes. The conclusion is as predicted before, i.e. that the accuracy with n elements is the same as a simulation done on a equidistant mesh with $q \cdot n$ elements.

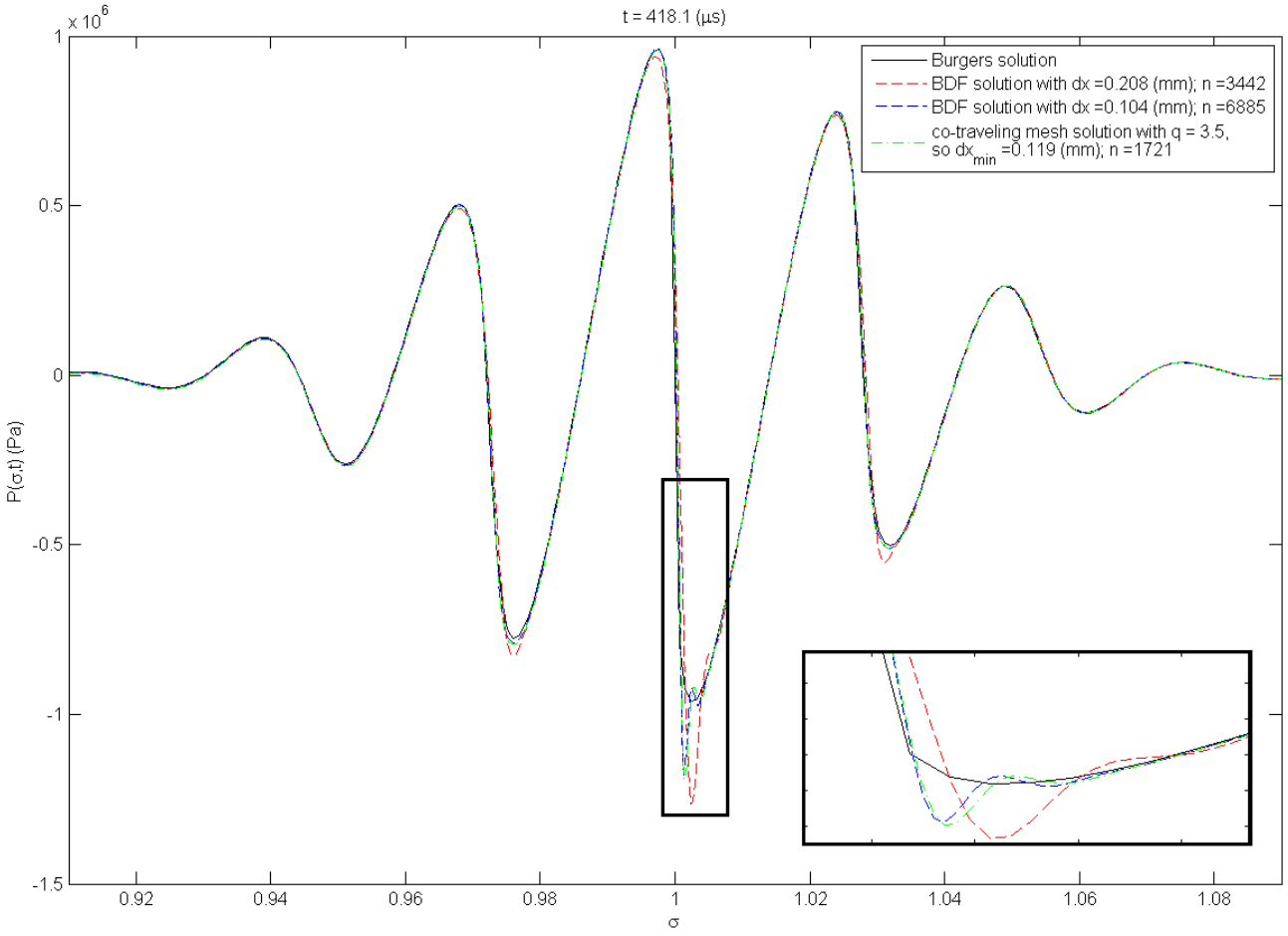


Figure 5.6: *The solution calculated with a co-traveling mesh compared with solutions calculated on an equidistant mesh. Parameters are all set as in Table 2, except for dx which is overridden by the n setting, found in the legend. Note that this is Figure 4.6 with a co-moving mesh solution added. The reduction of elements is nicely compensated by the local refinement.*

6 Conclusion and Discussion

In chapter 1 we formulated the somewhat broad question of how applicable FEM is to the Westervelt equation. We have used different approaches in trying to get a firm understanding of the behavior of the solution found by FEM techniques. We first collect observations and conclusions on specific parts of our research and then present an overall conclusion on the subject.

6.1 The causes and reduction of numerical errors

First we investigated the behavior of FEM solutions close to or just after the shock wave formation distance. We see that numerical oscillations may occur when the mesh is chosen too coarse. A large overshoot occurs when the first-order (spatial) derivative become extremely large, i.e. when a shock wave is starting to form. This effect is enlarged if we increase the source amplitude. That means that for even stronger sources, the numerical problems become more difficult to resolve.

These numerical errors are a known effect which also occur when solving the Westervelt equation by other methods [4]. We find that these numerical problems reduce when the mesh is refined or when the order of the polynomial basis is increased.

Unfortunately when decreasing the element size (refining the mesh), the time step needs to be decreased to in order for the propagation to be solved correctly. This is a requirement that is independent of the nonlinear effect, since it also holds for linear wave propagation. A certain time step needs to be small enough for the spatial derivative to have not changed to much (due to the wave propagation), else a propagation error will be made. The conclusion is that refining the mesh can be very costly in terms of computation time, because each time step a larger system of equations must be solved and more time steps must be taken to reach some final time.

Also, when using a non-uniform mesh, the time step depends on the smallest element size needed to calculate the propagation correctly at that point in space. So choosing a better mesh distribution reduces the time needed to solve the system of equations in space at each time step, but the number of time steps needed still depends on the smallest element size. Even despite this fact, it is possible that the most optimal mesh is a non-uniform one. Generally speaking a choice of mesh has an influence on the accuracy of the solution as well as the computational efficiency, especially in three dimensions where many more choices on meshes can be made. It requires some experience with FEM to optimize a mesh for certain problems. More research can be put into obtaining a optimal mesh distribution for this specific problem.

It is in this light more advantageous to use higher-order basis functions, since they seem to affect the time step less. Higher-order basis functions also have the huge advantage that they reduce the relative error by a larger factor than low-order basis functions for a certain refinement. This is a known fact for FEM, which was illustrated in Figure 5.1. The disadvantage is that increasing the order of basis functions will also increase the number of degrees of freedom, and thus increase the system of equations that is solved at each time step.

6.2 The evaluation of different time solvers

We examined different time solvers for the semi-discrete FEM system of equations. The first were build-in ode solvers that come with Matlab and COMSOL, which use adaptive time stepping techniques based on tolerance settings. The advantage is that these methods have proven to work for many different problems and that they estimate time steps on the go. More research can be put into which settings of these solvers can help improve the solution. These solvers reduced the time step when approaching the shock wave formation distance, indicating that the solution is becoming more difficult to achieve. They did manage to solve past the shock wave formation without having to reduce the time step to an impractically small step.

The second time solver was a Matlab implementation of a backward differential scheme up to second-order accuracy. Some research was put into the choice of this scheme, because of the difficulty of the problem at hand. As time evolves, the system becomes more and more stiff, since it is approaching a shock wave formation. In general, implicit and backward methods are more suitable for solving stiff

problems as they posses better stability properties. The order was chosen based on accuracy and stability of the solution by trial of various orders.

In this implementation we solved the nonlinear system of equations by the method of successive approximation. This is a very straightforward method that generally has slower convergence properties than more advanced methods such as Newton-Raphson. Successive approximation was however very efficient in this case, because the time step taken was very small and the nonlinear term is small compared to the linear terms in the equation. It solved the equation up to a relative tolerance of 10^{-3} within at most 3 iterations.

We see that different degrees of accuracy can be obtained by different time solving techniques. It is difficult to compare the two methods. The build-in type solvers have a wide variety of properties that can affect the accuracy of the solution, such as absolute or relative tolerance, solver type and order. The BDF scheme we developed only had a single parameter: the size of the time step.

We conclude that the specific choice of the time solving method also has an important effect on the accuracy of the solution. More research can be put into finding and implementing an optimal time solver for the Westervelt equation.

6.3 The effects of damping

As we know from the literature (see for example [3], [4] or [10]), damping is a counter effect to the nonlinearity and therefore it may reduce the numerical problems that we encounter. Physical damping is often described in the frequency by a the power law. Thermoviscous damping, described in the time domain, is just a special case of this, which is an excellent description of attenuation in water.

We studied the Westervelt equation with a thermoviscous damping term, where we varied the damping coefficient. We conclude that physically realistic damping does not solve the numerical problems that occur near the peaks when approaching the shock wave distance. This is because the solution is not significantly affected by the introduced physical damping. If we choose a very large damping coefficient however (i.e. 1,000 times the damping coefficient for water), we see that the numerical problems are significantly reduced, but at the expense of losing much nonlinearity and a reduction in amplitude. A too strongly damped solution does not accurately describe nonlinear wave propagation. We thus conclude that physically realistic damping does not help us reduce the numerical problems, while extremely strong damping does, but it fails to represent the nonlinear propagation we are interested in.

In the frequency domain it may help if frequencies above a certain cut-off are strongly attenuated, so that the shock wave front remains a little smooth. It basically filters the higher-order harmonics in the solution to ensure a certain amount of smoothness of the solution at the cost of accuracy. These frequency domain filters can be difficult to implement in a FEM scheme.

In conclusion, we recommend that more research can be put into physically or numerically damping nonlinear wave propagation. It may lead to smoother shock wave approximations which can be solved more accurately using FEM. See for example [10]. An interesting question is if it is possible to incorporate a sort of frequency based filtering in FEM simulations of nonlinear wave propagation.

6.4 Inhomogeneous domains or different domain shapes

In this research we only solved one-dimensional wave propagation and therefore domains are relatively simple. We have researched the possibility of inhomogeneous domains, which in general is a strong suit of FEM. In our case FEM has quite accurately solved reflections that occurred due to inhomogeneity. From the literature we know that these results generalize to three dimension, at least for linear wave propagation and certainly also for other types of physical problems. It is thus safe to assume that these results will also apply to three dimensions for the Westervelt equation. In three dimensions we can also solve for complicated domain shapes more easily with FEM than for example with finite difference methods.

Our research did not focus on inhomogeneity and domain problems and more research can be done to investigate if FEM has an advantage over other solving methods when difficult domain geometries

and inhomogeneities are present. If it has, it may be an important factor when choosing a method for solving nonlinear wave propagation.

6.5 Adaptive solution methods

In chapter 1 we also formulated the hypothesis that (spatial) adaptiveness can improve the efficiency and accuracy of the solution. We examined the different types of adaptivity available in FEM. Generally speaking there are three types of adaptiveness. The first locally refines/coarsens the mesh, the second resizes elements by moving the nodes, and the last increases/decreases the order of the basis functions locally. These types of adaptivity can also be combined in any way, but it may drastically complicate implementation.

The method were nodes move along the domain is an interesting option since it can be capable of following the wave propagation as well as refining it where the derivatives become large. In [12] this type of adaptiveness, known as r-adaptiveness, was used in an example of solving the Burgers equation. We implemented such an adaptiveness to examine if it improved efficiency and accuracy.

We found that this type of adaptiveness can have negative effects on the stability of the solution and the size of the time step taken by the solver. The last can be explained because the specific implementation we chose (MMPDE-6) did not give us full control over the element size. When the elements became extremely small around certain areas, the time step needed became too small to practically advance the solution in time. Various implementations of r-adaptiveness are available and a specific choice should be made with a lot of care.

We then used our a priori knowledge of the solution, the boundary conditions and the domain, to implement a co-moving refinement, which is an adaptivity that leaves us with full control over element size and relocation speed, but it is not a full adaptive method since it does not respond to the solution evolving in time. The method we designed still has a locally uniform mesh (within the area where the pulse is located) and does not refine around the steep slopes. We did manage to calculate a solution to the same accuracy with 3.5 times less elements. This is just the first hint that adaptivity can help us improve efficiency and accuracy.

The expense of in time changing meshes comes from the need to recalculate (parts of) the system matrices each time (part of) the mesh changes. Also it may be necessary to perform interpolation operations on previous solutions. Thus the adaptivity must at least improve enough to compensate for these drawbacks. When any form of adaptivity is implemented, one should make sure it makes use of all its possibilities.

We can conclude that simple adaptivity already helped improve the accuracy of the solution and it is recommended that more research is put into adaptivity for solving nonlinear wave propagation with FEM. Among the things that can be researched is using combinations of adaptivity types, since locally increasing the number of elements or the order of the basis functions can help resolve shock wave fronts. We can also look into combining spatial and temporal adaptiveness to achieve optimal efficiency and restore the tight correlation between the spatial and temporal numerical parameters. We further predict that adaptivity becomes increasingly more important in three-dimensional problems, since the profit made by adaptivity can be significantly larger in three dimensions. For example, pulse following can reduce the amount of elements in each dimension by a factor 3.5, so the total reduction in three dimensions may be in the order of a factor $3.5^3 \approx 40$.

6.6 Overall evaluation of FEM as a method for solving the Westervelt equation

What we have seen so far in this research is that FEM is capable of solving nonlinear wave propagation in one dimension. Its results are comparable to those obtained by other methods used to solve the Westervelt equation. Problems that arose were similar to those encountered in other methods and certain specific solutions were suggested in this thesis.

Investigating whether filtering or other forms of (artificial) damping can be performed with FEM can be worthwhile, since they have proven to work in some cases for other solving methods [4].

We recommend research to be done to investigate whether FEM may be used as a preferred method if the domains involved in the problem become very complicated of shape or structure (i.e.

very inhomogeneous). Scaling to three dimensions is recommended as that can fully show the power of FEM.

We even stronger recommend researching adaptivity in FEM. Adaptiveness is a relatively young branch of research in the area of FEM and we have shown that it can be advantageous in nonlinear wave propagation. In FEM there are many different types of spatial adaptiveness available which can also be combined with temporal adaptiveness.

At this point we have reason to believe that FEM can become the preferred choice of solution method for nonlinear wave propagation under certain circumstances. More research can be done on this subject and good starting points are for example [10] or [16].

Appendices

A Matlab code for approximating the implicit Burgers solution

```
clc; clear all; close all; tic;

%medium parameters (in apropriate SI units)
c0=1500; rho=1000; beta=10;

%typical ultrasound imaging wave properties
P0 = 1e6; % Pa
f0 = 1e5; %Hz

%calculating shock wave formation distance
xsh = rho*c0^3/(beta*P0*2*pi*f0);  %m

L = xsh +12*c0/f0; %m
dx = c0/f0/36;

dtau = 0.25*dx/c0; %s
Td = 6/f0;
Tw = 3/f0;
TauEnd = 2*Td;
tau = [0:dtau:TauEnd];

%defining the source signal, in this case a single frequency sine wave with a ...
    Gaussian envelope.
p0 = P0.*sin(2*pi*f0*tau).*exp(-(tau-Td).^2./(Tw/2)^2); %Source signal

%initialize scheme
pburg(1,:) = p0; % pburg(x=0,tau)

for j=2:ceil(L/dx)
    tt = tau + beta/rho/c0^3 * dx * pburg(j-1,:); %calculating the shifted times
    pburg(j,:) = interp1(tau,pburg(j-1,:),tt); %interpolating in time to find the ...
        values
end
toc;
```

B Derivation of the second-order backward differential formulas

In this section we briefly derive the second-order backward differential equations used in the time stepping sequence. In this appendix we drop vector notation, such as bold font, but the derivation applies to a single equation as well as to systems of (time depending) equations.

Suppose we want to estimate the second time derivative at time t_k (denoted by a superscript)

$$\ddot{u}^k \approx \frac{au^k + bu^{k-1} + cu^{k-2} + du^{k-3}}{(\Delta t)^2}, \tag{B.1}$$

for unknown coefficients a, b, c, d. Using Taylor expansion on each u around t_k we get

$$\begin{array}{rcccccccccc} u^k & = & u^k & & & & & & & & \\ u^{k-1} & = & u^k & - & \Delta t\dot{u}^k & + & \frac{(\Delta t)^2}{2}\ddot{u}^k & - & \frac{(\Delta t)^3}{6}\dddot{u}^k & + & \mathcal{O}\big((\Delta t)^4\big) \\ u^{k-2} & = & u^k & - & 2\Delta t\dot{u}^k & + & 2\Delta t\ddot{u}^k & - & \frac{8(\Delta t)^3}{6}\dddot{u}^k & + & \mathcal{O}\big((\Delta t)^4\big) \\ u^{k-3} & = & u^k & - & 3\Delta t\dot{u}^k & + & \frac{9(\Delta t)^2}{2}\ddot{u}^k & - & \frac{27(\Delta t)^3}{6}\dddot{u}^k & + & \mathcal{O}\big((\Delta t)^4\big). \end{array} \tag{B.2}$$

Plugging these equations into (B.1), we need to find the set of coefficients $\{a, b, c, d\}$ such that all higher and lower (including the zeroth) orders of derivatives vanish except for the second. To do so, we solve the system of equations (given in matrix form)

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -1 & -2 & -3 \\ 0 & 1/2 & 2 & 9/2 \\ 0 & -1/6 & -8/6 & -27/6 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \tag{B.3}$$

which has the solution $\begin{bmatrix} 2 & -5 & 4 & -1 \end{bmatrix}^T$. Since the smallest local truncation error is $\mathcal{O}\big((\Delta t)^4\big)$ in u, the local truncation error in $\ddot{u}^k$ is $\frac{\mathcal{O}\left((\Delta t)^4\right)}{(\Delta t)^2} = \mathcal{O}\big((\Delta t)^2\big)$.

Similarly for approximating

$$\dot{u}^k \approx \frac{au^k + bu^{k-1} + cu^{k-2}}{\Delta t}, \tag{B.4}$$

using the same Taylor expansions as in (B.2), but then up to order $\mathcal{O}\big((\Delta t)^3\big)$, we get the following system of equations

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -2 \\ 0 & 1/2 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \tag{B.5}$$

which has solution $\begin{bmatrix} 3/2 & -2 & 1/2 \end{bmatrix}^T$. Note that this is also an approximation of $\mathcal{O}\big((\Delta t)^2\big)$.

So in conclusion we have

$$\begin{array}{rcccc} \ddot{u}^k & = & \frac{2u^k - 5u^{k-1} + 4u^{k-2} - u^{k-3}}{(\Delta t)^2} & + & \mathcal{O}\big((\Delta t)^2\big) \\ \dot{u}^k & = & \frac{3u^k - 4u^{k-1} + u^{k-2}}{2\Delta t} & + & \mathcal{O}\big((\Delta t)^2\big) \\ \ddot{u}^k & = & \frac{u^k - 2u^{k-1} + u^{k-2}}{(\Delta t)^2} & + & \mathcal{O}\big(\Delta t\big) \\ \dot{u}^k & = & \frac{u^k - u^{k-1}}{\Delta t} & + & \mathcal{O}\big(\Delta t\big), \end{array} \tag{B.6}$$

where the last two equations can be derived analogous to the first two.

C Matlab implementations of various finite element schemes

C.1 A simple example of the linear wave equation

```
clear all; close all; clc; tic;

n =1500;
dt = 1/n * 0.4;
T = 2;

t = 0:dt:(round(T/dt)-1)*dt;

Mesh = linspace(0,1,n+1)';
% Topology is straight forward. e_i has node i-1 and i

% Function that assembles Mass and Stiffnes Matrices
[M, S, gM, gS] = MassAndStiffnessFirstOrder(Mesh,n,1);

% a specific choice of BC at x=0. This is f(t). This is just an example,
% chosen to be sufficiently smooth.
u0t = zeros(size(t));
u0t(round(0.1*T/dt):round(0.3*T/dt)) = 0.5*(1 - cos(2*pi/(0.2*T) * ...
    (t(round(0.1*T/dt):round(0.3*T/dt)) - t(round(0.1*T/dt)))));

%Constructing the time-dependent right hand side vector g
g = sparse(zeros(n-1,round(T/dt)));
g(1,:) = -( [0 u0t(2:round(T/dt))] -2*u0t + [u0t(1:round(T/dt)-1) 0] )*(Mesh(2) - ...
    Mesh(1))/3 + u0t*1/(Mesh(2) - Mesh(1));

% Initial condition
u1 = zeros(n-1,1);

% First time derivative IC
ut1 = zeros(n-1,1);

% Solving next time step with first order derivative
u2 = u1 + dt.*ut1;

% Initializing time solver
U = zeros(n-1,round(T/dt));
U(:,1) = u1;
U(:,2) = u2;

% Time solver, backward difference scheme
U(:,3) = (dt^2 * S + M) \ (2*M*U(:,i-1) - M*U(:,i-2) + dt^2 * g(:,3));
for k = 4:round(T/dt)
    U(:,k) = (dt^2 * S + 2*M) \ (5*M*U(:,i-1) - 4*M*U(:,i-2) + M*U(:,i-3) + dt^2 * ...
        g(:,i));
end

% adding the BC's
V(1,:) = u0t';
V(2:n,:) = U;
V(n+1,:) = zeros(1,k);

toc;
```

C.2 The Westervelt equation with a second-order BDF time scheme

```
clear all; close all; clc; tic;

c = 1500;
rho = 1000;
f0 = 1e5;
P0 = 1e6;
beta = 10;

xsh = rho*c^3/(beta*P0*2*pi*f0);

L = xsh + 12*c/f0;
n = round(72*f0*L/c);
Mesh = linspace(0,L,n+1)';
% Topology is straight forward. e_i has node i-1 and i

dt = (L/n)/(c*4);
T = L/c;
k = round(T/dt)+1;
t = 0:dt:(k-1)*dt;

% Functions that assemble the Mass and Stiffness Matrices, as well as the
% nonlinearity Matrices.
[M, S, gM, gS] = MassAndStiffnessFirstOrder(Mesh,n,c);
[C1, C2, C3, gC1, gC2, gC3] = CMatricesFirstOrder( Mesh,n,rho,c,beta );

%Function that evaluates the BC's at x=0 at times t. This is f(t).
u0t = Pulse(t, P0, f0);

% Calculate the rhs-vector with a Function that evaluate the second order
% time derivative of the source function.
u0t_tt = Pulsett(t,P0,f0);
g = sparse(ones(1,k),1:k,-u0t_tt*gM - u0t*gS,n-1,k,k);

% Initial condition
u1 = zeros(n-1,1);

% First time derivative IC
ut1 = zeros(n-1,1);

% Solving next time step with first order derivative
u2 = u1 + dt.*ut1;

%Time solver, linear solver
a=tic;
W = zeros(n-1,k);
W(:,1) = u1;
W(:,2) = u2;
W(:,3) = (dt^2 * S + M) \ (2*M*W(:,2) - M*W(:,1) + dt^2 * g(:,3));

for i = 4:k
    W(:,i) = (dt^2 * S + 2*M) \ (5*M*W(:,i-1) - 4*M*W(:,i-2) + M*W(:,i-3) + dt^2 * ...
        g(:,i));
end

% Time solver, nonlinear solver
a=tic;
U = zeros(n-1,k);
U(:,1) = u1;
U(:,2) = u2;

% Calculate the third time point using a first order BDF scheme.
for i = 3
```

```
62      z = (dt^2 * S + M) \ (2*M*U(:,i-1) - M*U(:,i-2) + dt^2 * g(:,3));
63      zt = 3*P0*ones(size(z));
64
65      while(max(abs(z-zt)) > 1e-9*P0)
66          zt = z;
67
68          AA = sparse(1:n-1,1:n-1,[u0t(i); zt(1:n-2)],n-1,n-1,n-1);
69          AB = sparse(1:n-1,1:n-1,zt,n-1,n-1,n-1);
70          AC = sparse(1:n-1,1:n-1,[zt(2:n-1); 0],n-1,n-1,n-1);
71          N2 = AA*C1 + AB*C2 + AC*C3; clear AA AB AC;
72
73          BA = sparse(1:n-1,1:n-1,([u0t(i); zt(1:n-2)] - [u0t(i-1); ...
                U(1:n-2,i-1)])/dt,n-1,n-1,n-1);
74          BB = sparse(1:n-1,1:n-1,(zt - U(:,i-1))/dt,n-1,n-1,n-1);
75          BC = sparse(1:n-1,1:n-1,([zt(2:n-1); 0] - [U(2:n-1,i-1); 0])/dt,n-1,n-1,n-1);
76          N1 = BA*C1 + BB*C2 + BC*C3; clear BA BB BC;
77
78          z = (dt^2 * S + M - dt*N1 - N2) \ ((2*M -dt*N1 - 2*N2)*U(:,i-1) + (-M + ...
                N2)*U(:,i-2) + dt^2 * g(:,i));
79      end
80      U(:,i) = z;
81  end
82
83  % Solve all the time points with a second order BDF scheme.
84  for i = 4:k
85      z = (dt^2 * S + 2*M) \ (5*M*U(:,i-1) - 4*M*U(:,i-2) + M*U(:,i-3) + dt^2 * g(:,i));
86      zt = 3*P0*ones(size(z));
87
88      while(max(abs(z-zt)) > 1e-9*P0)
89          zt = z;
90
91          AA = sparse(1:n-1,1:n-1,[u0t(i); zt(1:n-2)],n-1,n-1,n-1);
92          AB = sparse(1:n-1,1:n-1,zt,n-1,n-1,n-1);
93          AC = sparse(1:n-1,1:n-1,[zt(2:n-1); 0],n-1,n-1,n-1);
94          N2 = AA*C1 + AB*C2 + AC*C3; clear AA AB AC;
95
96          BA = sparse(1:n-1,1:n-1,(3/2*[u0t(i); zt(1:n-2)] - 2*[u0t(i-1); ...
                U(1:n-2,i-1)] + 1/2*[u0t(i-2); U(1:n-2,i-2)])/dt,n-1,n-1,n-1);
97          BB = sparse(1:n-1,1:n-1,(3/2*zt -2*U(:,i-1) +1/2*U(:,i-2))/dt,n-1,n-1,n-1);
98          BC = sparse(1:n-1,1:n-1,(3/2*[zt(2:n-1); 0] - 2*[U(2:n-1,i-1); 0] + ...
                1/2*[U(2:n-1,i-2); 0])/dt,n-1,n-1,n-1);
99          N1 = BA*C1 + BB*C2 + BC*C3; clear BA BB BC;
100
101         z = (dt^2 * S + 2*M - 3/2*dt*N1 - 2*N2) \ ((5*M -2*dt*N1 - 5*N2)*U(:,i-1) ...
                + (-4*M +1/2*dt*N1 + 4*N2)*U(:,i-2) + (M-N2)*U(:,i-3) + dt^2 * g(:,i));
102     end
103     U(:,i) = z;
104 end
105
106 U = [u0t; U; zeros(1,k)]; %Nonlinear FEM solution
107 W = [u0t; W; zeros(1,k)]; %Linear FEM solution
108 Uanalytic = linearpropagation(f0,P0,c,Mesh,t); %Linear Analytic solution
109 toc;
```

C.3 The Westervelt equation with ode15s time scheme

```
clear all; close all; clc; tic;

c = 1500;
rho = 1000;
f0 = 1e5;
P0 = 1e6;
beta = 10;

xsh = rho*c^3/(beta*P0*2*pi*f0);

L = xsh + 12*c/f0;
n = round(72*f0*L/c);
Mesh = linspace(0,L,n+1)';
% Topology is straight forward. e_i has node i-1 and i

dt = (L/n)/(c*4);
T = L/c;
k = round(T/dt)+1;
t = 0:dt:(k-1)*dt;

% BC at x=0
u0t = Pulse(t, P0, f0);
u0t_tt = Pulsett(t,P0,f0);

% Initial condition
u1 = zeros(n-1,1);

% First time derivative IC
ut1 = zeros(n-1,1);

% Solving next time step with first order derivative
u2 = u1 + dt.*ut1;

%Time solver, linear solver
W = zeros(n-1,k);
[M, S, gM, gS] = MassAndStiffnessFirstOrder(Mesh,n,c);
O = sparse(n-1,n-1);
I = speye(n-1);

A = [O M; I O];
B = [-S O; O I];

options ...
    =odeset('RelTol',1e-3,'MaxStep',1/(6*f0),'Mass',A,'MStateDependence','none','MassSingular','n
[Tode W] =ode15s(@(tslv,y) fun(tslv,y,n,B,P0,f0,gM,gS),t,[u1; ut1],options);
W = W';
W = W(1:n-1,:);

% Time solver, nonlinear solver
[M, S, gM, gS] = MassAndStiffnessFirstOrder(Mesh,n,c);
[C1, C2, C3, gC1, gC2, gC3] = CMatricesFirstOrder(Mesh,n,rho,c,beta);

I = speye(n-1);
I3 = spdiags(ones(n-1,3),-1:1,n-1,n-1);
O = sparse(n-1,n-1);
JPat = [I3 I3; O I];
MvPat = [O I3; I O];

options =odeset('RelTol',1e-3,'MaxStep',1/(6*f0),'Mass',@(tslv,y) MassNonl( ...
    tslv,y,n,M,C1,C2,C3,P0,f0 ),'MStateDependence','weak','MassSingular','no',...
     'MvPattern',MvPat,'JPattern',JPat,'BDF','on','MaxOrder',5);
```

```
[Tode U] =ode15s(@(tslv,y) fun_nonl(tslv,y,n,S,C1,C2,C3,P0,f0,gM,gS),t,[u1; ...
    ut1],options);
U = U';
U = U(1:n-1,:);

% Adding the boundary conditions
U = [u0t; U; zeros(1,k)]; %Nonlinear FEM solution
W = [u0t; W; zeros(1,k)]; %Linear FEM solution
Uanalytic = linearpropagation(f0,P0,c,Mesh,t); %Linear Analytic solution
toc;
```

```
function dy = fun(tslv,y,n,B,P0,f0,gM,gS)
    % Function that evaluates the linear rhs-vector in ode15s
    dy = zeros(2*n-2,1);
    dy = B*y;
    dy(1) = dy(1) -Pulsett(tslv, P0, f0)*gM -Pulse(tslv, P0, f0)*gS;
end
```

```
function out = fun_nonl(tslv,y,n,S,C1,C2,C3,P0,f0,gM,gS)
    % Function that evaluates the nonlinear rhs-vector in ode15s
    out = zeros(2*n-2,1);

    BA = sparse(1:n-1,1:n-1,[Pulset(tslv,P0,f0); y(n:2*n-3)],n-1,n-1,n-1);
    BB = sparse(1:n-1,1:n-1,y(n:2*n-2),n-1,n-1,n-1);
    BC = sparse(1:n-1,1:n-1,[y(n+1:2*n-2); 0],n-1,n-1,n-1);
    N1 = BA*C1 + BB*C2 + BC*C3;

    O = sparse(n-1,n-1);
    I = speye(n-1);

    B = [-S N1; O I];

    out = B*y;
    out(1) = out(1) -Pulsett(tslv, P0, f0)*gM -Pulse(tslv, P0, f0)*gS;
end
```

```
function [ out ] = MassNonl( tslv,y,n,M,C1,C2,C3,P0,f0 )
    % Function that evaluates the nonlinear mass matrix M(t,y) in ode15s
    AA = sparse(1:n-1,1:n-1,[Pulse(tslv,P0,f0); y(1:n-2)],n-1,n-1,n-1);
    AB = sparse(1:n-1,1:n-1,y(1:n-1),n-1,n-1,n-1);
    AC = sparse(1:n-1,1:n-1,[y(2:n-1); 0],n-1,n-1,n-1);

    N2 = AA*C1 + AB*C2 + AC*C3;

    O = sparse(n-1,n-1);
    I = speye(n-1);

    out = [O M-N2; I O];

end
```

C.4 Custom function definitions called in the implementations

```
function [M, S, gM, gS] = MassAndStiffnessFirstOrder(Mesh,n,c)
    %function that assembles the Mass and Stiffness matrices for first
    %order elements.

    dMesh = diff(Mesh);
    S = spdiags([-1./dMesh(2:n) 1./dMesh(1:n-1)+1./dMesh(2:n) ...
        -1./dMesh(1:n-1)],-1:1,n-1,n-1);
    M = spdiags([dMesh(2:n)  2*(dMesh(1:n-1)+dMesh(2:n))  ...
        dMesh(1:n-1)]/(6*c^2),-1:1,n-1,n-1);

    %values that move to the rhs of the system (as knowns) since they
    %multiply with the known BC at x=0.
    gS = -1./dMesh(1);
    gM = dMesh(1)/6/c^2;
end
```

```
function [ C1, C2, C3, gC1, gC2, gC3 ] = CMatricesFirstOrder( Mesh,n,rho,c,beta )
% Function that build the nonlinear matrices using element matrices.

C1EI = cell(n);
C2EI = cell(n);
C3EI = cell(n);
for i=1:n
    C1EI{i} = 2*beta/(rho*c^4) * 1/12 * [0 1; 0 1]*(Mesh(i+1)-Mesh(i));
    C2EI{i} = 2*beta/(rho*c^4) * 1/12 * [3 1; 1 3]*(Mesh(i+1)-Mesh(i));
    C3EI{i} = 2*beta/(rho*c^4) * 1/12 * [1 0; 1 0]*(Mesh(i+1)-Mesh(i));
end
C1EI{1}(1,2) = 0;
C3EI{n}(2,1) = 0;

C1 = spalloc(n+1,n+1,3*(n+1));
C2 = spalloc(n+1,n+1,3*(n+1));
C3 = spalloc(n+1,n+1,3*(n+1));
for i=1:n
   C1(i,i) = C1(i,i) + C1EI{i}(1,1);
   C1(i,i+1) = C1(i,i+1) + C1EI{i}(1,2);
   C1(i+1,i) = C1(i+1,i) + C1EI{i}(2,1);
   C1(i+1,i+1) = C1(i+1,i+1) + C1EI{i}(2,2);

   C2(i,i) = C2(i,i) + C2EI{i}(1,1);
   C2(i,i+1) = C2(i,i+1) + C2EI{i}(1,2);
   C2(i+1,i) = C2(i+1,i) + C2EI{i}(2,1);
   C2(i+1,i+1) = C2(i+1,i+1) + C2EI{i}(2,2);

   C3(i,i) = C3(i,i) + C3EI{i}(1,1);
   C3(i,i+1) = C3(i,i+1) + C3EI{i}(1,2);
   C3(i+1,i) = C3(i+1,i) + C3EI{i}(2,1);
   C3(i+1,i+1) = C3(i+1,i+1) + C3EI{i}(2,2);
end

gC1 = C1(2,1);
gC2 = C2(2,1);
gC3 = C3(2,1);

C1 = C1(2:n,2:n);
C2 = C2(2:n,2:n);
C3 = C3(2:n,2:n);


end
```

```
function [ s ] = Pulse( t, P0, f0 )
% Function that evaluates the source function.
tw = 3/f0;
td = 6/f0;
s = P0.*exp(-((t-td)./(tw/2)).^2).*sin(2.*pi.*f0.*(t - td));

% The step function, setting all values greater than Tend identically to
% zero.
i=find(t>12/f0,1,'first');
s(i:end) = 0;
end
```

```
function [ s ] = Pulsett( t, P0, f0 )
% Function that evaluates the second time derivative of the source
% function.
tw = 3./f0;
td = 6./f0;
s = ...
    -8.*P0.*exp(-4.*(t-td).^2./tw.^2).*sin(2.*pi.*f0.*(t-td))./tw.^2 + ...
    64.*P0.*(t-td).^2.*exp(-4.*(t-td).^2./tw.^2).*sin(2.*pi.*f0.*(t-td))./tw.^4 - ...
    32.*P0.*(t-td).*exp(-4.*(t-td).^2./tw.^2).*...
        cos(2.*pi.*f0.*(t-td)).*pi.*f0./tw.^2 - ...
    4.*P0.*exp(-4.*(t-td).^2./tw.^2).*sin(2.*pi.*f0.*(t-td)).*pi.^2.*f0.^2;

i=find(t>12./f0,1,'first');
s(i:end) = 0;
end
```

```
function s = linearpropagation(f0,P0,c,x,t)
    % Function that solve the linear wave equations analytically.
    [T,X] = meshgrid(t,x);

    tw = 3/f0;
    td = 6/f0;
    s = P0.*exp(-(((T-X/c)-td)./(tw/2)).^2).*sin(2.*pi.*f0.*((T-X/c) - td));
end
```

References

[1] Lauterborn, W., Kurz, T. and Akhatov, I., 2007. *Nonlinear Acoustics in Fluids* in *Handbook of Acoustics*, Chap. 8, p. 257-297, Springer.

[2] Hamilton, M. F. and Morfey, C.L., 1998. *Model Equations* in *Nonlinear Acoustics* edited by Hamilton, M. F. and Blackstock, D. T., Chap. 3, p. 41-63, Academic Press.

[3] Blackstock, D. T., Hamilton, M. F. and Pierce, A.D., 1998. *Progressive Waves in Lossless and Lossy fluids* in *Nonlinear Acoustics* edited by Hamilton, M. F. and Blackstock, D. T., Chap. 4, p. 66-147, Academic Press.

[4] Huijssen, J., 2008. *Modeling of Nonlinear Medical Diagnostic Ultrasound*, PrintPartners Ipskamp B.V., Enschede

[5] Blackstock, D. T., 1966. *Connection between the Fay and Fubini Solutions for Plane Sound Waves of Finite Amplitude*, The journal of the Acoustical Society of America, Vol. 39, No. 6, p. 1019-1026

[6] Karamalis, A., Wolfgang, W. and Navab, 2010. *Fast Ultrasound Image Simulation using the Westervelt Equation*, Medical Image Computing and Computer Assisted Intervention

[7] Unknown Author, 2013. *Nonlinear Acoustics: Modeling of the 1D Westervelt Equation*, COMSOL Multiphysics Model guide, Model ID: 12783

[8] Lahaye, D. and Vermolen, F., 2011. *Building Virtual Models in Engineering: An Introduction to Finite Elements*, Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft Institute for Applied Mathematics

[9] Holmes, M. J., Parker, N. G. and Povey, M. J. W., 2011. *Temperature dependence of bulk viscosity in water using acoustic spectroscopy*, Journal of Physics: Conference Series

[10] Hoffelner, J. and Kaltenbacher M., 2001. *Finite Element Simulation of Nonlinear Wave Propagation in Thermoviscous Fluids Including Dissipation*, IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control, Vol. 48, No. 3, p. 779-786

[11] McHugh, P., 2007. *Non-Linear Finite Element Analysis: Finite Element Solution Schemes I & II*, National University of Ireland, Galway, National Centre for Biomedical Engineering Science, Department of Mechanical and Biomedical Engineering.

[12] Budd, C. J., Weizhang, H., Russel, R. D., 2009. *Adaptivity with moving grids* in *Acta Numerica (2009)*, Vol. 18, p. 1-131, Cambridge University Press.

[13] Unknown Author, 2013. *ode15s* in *Matlab R2013b documentation*, http://www.mathworks.nl/help/matlab/ref/ode15s.html, The MathWorks, Inc.

[14] Nguyen, H., 2008. *p-Adaptive and Automatic hp-Adaptive Finite Element Methods*, Center for Computational Mathematics, Department of Mathematics, University of California, San Diego.

[15] Uthman, Z., Askes, H., 2005. *R-adaptive and h-adaptive mesh optimisation based on error assessment*, 13^{th} ACME conference: Department of Civil and Structural Engineering, University of Sheffield.

[16] Wismer, M., G., 2006. *Finite element analysis of broadband acoustic pulses through inhomogeneous media with power law attenuation*, Acoustical Society of America, Vol. 120, No 6, P. 3493-3502.