The Spectral Element Method for Seismic Wave Propagation

Theory, Implementation and Comparison to Finite Difference Methods

Diplomarbeit von Bernhard Schuberth

abgegeben am
7. November 2003

Betreuer:
Prof. Dr. Heiner Igel

Department für
Geo- und Umweltwissenschaften
Sektion Geophysik

Ludwig-Maximilians-Universität
München

Contents

Acknowledgements

First of all I want to thank my supervisor Professor Heiner Igel for his guidance, support and for proposing the interesting topic for this thesis. I am grateful that he whishes students to attend international meetings at an early stage, so I joined the EGS/AGU conference, April 2003 in Nice and a fantastic workshop in Smolenice, Slovakia in September. In addition, he set up the collaboration with Dimitri Komatitsch, so I was able to work with Dimitri in Pau, France, for a week in May 2003.

Many thousand thanks go to Dimitri Komatitsch, without whom this thesis would never be the way it is. Thanks for providing me with several SEM codes, your knowledge on the whole topic and the reviewing of several parts of this thesis. Especially in the last weeks of writing, your support and help were brilliant, when daily e-mails were common.
And many thanks for the nice time in Pau!

Next I want to thank Professor Helmut Gebrande, and Professor Hans-Peter Bunge representatives of the Geophysics Section of the Department of Earth and Environmental Sciences of the LMU Munich for supplying an office and working facilities, including the computer network. I also want to thank Hans-Peter Bunge for interesting discussions.

Furthermore I thank all my colleagues in the Geophyisics Institute for their help and open ears! Thousand thanks to you, Markus, whom I surely bothered most, but also had most fun with. Thanks to Tobias Metz, for the good co-operation and fruitful teamwork on the comparisons between Optimal FD Operators and the SEM. He supported many good ideas and programs for the evaluation of the codes.
A lots of thanks to Haijiang and Wiwit, not only for scientific discussions and help, but also for the nice conversations on life and religion. Many thanks also to Peter Danecek for helping when it was really needed.
Moreover I want to thank Michael Ewald, Gunnar Jahnke, Toni Kraft, Aji Sudarmaji, Guoquan Wang, Asher Flaws, Gilbert Brietzke and Melanie Reichardt of the Seismology group for all the small things they helped me with.

Thanks to Roman Leonhardt for the help on Linux and the computer network, to Michael Winklhofer for helpful discussions, reviewing and the script on FORTRAN and many thanks to Erika Vye for reviewing parts of the thesis and the helping hand, especially in english grammar. And, at last, many thanks to the people of the secretariate, all people of the other groups of the institute and the complete staff.

Many thanks to you, Oliver, for really teaching me geophysics, especially rheology and working with FEM. A thousand thanks to the crew of the Polarstern ARK XVIII/2 cruise, especially to Klaus, Bärbel, Christina, Dani, Veit and Tobi for the unforgettable and exciting time in the Artic and the knowledge what geophysics can be besides flips and flops.
Many thanks to all my friends in the near and far field, especially of the "Anglistenchor", for lots of amusement, sports and fun during the last years, to balance work and making life more favourable.

I am glad to owe to my Mother and Father all that I have achieved. You enabled me to learn to love life, live the way I want to and supported every step of mine. My greatest thanks and appreciation for all that you did for me!

No less admiration belongs to my brother Christian, the smartest person on earth and the perfect gentlemen. Thank you very much for all your clever advise, for being there when needed and for always leaving the bigger piece of cake to me.

The most lovely and loved one to mention is you, Conny. You make my world perfect!
Sorry for all that you had to endure during the last weeks. It is you, who takes the greatest part of help and support for my work.
Thank you for enjoying life with me!

Abbreviations and Symbols

This list comprises all symbols and abbreviations used, together with their definition and page reference. This reference denotes the page number where the symbol is used in its most significant context.
Typical mathematical conventions for variables, vectors, fields etc. are used:

- scalar functions and variables in italic letters, e.g.: $x, f(x)$
- vectors in small boldface letters, e.g.: $\mathbf{u}, \mathbf{f}$
- matrices in capital boldface letters, e.g.: $\mathbf{M}, \mathbf{K}$

Exceptions are made for special tensors, as for example the stress tensor $\boldsymbol{\sigma}$ of second order or the fourth order tensor of elastic coefficients $\mathbf{c}$, which are typically denoted with small letters in geophysical literature.

Symbol	**Definition**	**Page**
FDM	finite difference method	2
FEM	finite element method	2
GLL	Gauss-Lobatto-Legendre: a special integration quadrature of Gaussian type	15
ODE	ordinary differential equation	2
OPO	optimal operators, an improved FDM	2
PDE	partial differential equation	2
PML	Perfectly Matched Layer, special class of absorbing boundary conditions	28
SEM	spectral element method	2
C	Courant number used for the stability criterion	38

$f(x)$	1-D volume force at coordinate x 12
$\mathbf{f}$	global force vector 26
g	an arbitrary function 19
g^e	restriction of a function or interval to the element e 19
$\dot{g}$	differentiation with respect to time 14
$g_{(i),j} = \frac{\partial g_i(\mathbf{x})}{\partial x_j}$	a comma indicates a spatial derivative; here the derivative of the component i of any given vector $\mathbf{g}$ in the direction x_j 50
$\mathbf{J}$	the Jacobi-Matrix of the mapping function 19
$\mathcal{J}$	the determinant of the Jacobi-Matrix, called "Jacobian" 19
$^{\mathbf{i}}\mathbf{J}$	Jacobi matrix of the coordinate transformation from global to local coordinates, i.e. the inverse of the mapping function 25
K^e_{ij}	$N \cdot N$ matrix elements of the elemental stiffness matrix belonging to the spectral element e 25
$\mathbf{K}$	global stiffness matrix 26
ℓ	Lagrange polynomial 20
ℓ'_i	first derivative of Lagrange polynomial ℓ_i with respect to ξ ... 22
ℓ'_{ij}	the value of the first spatial derivative of ℓ_i at point ξ_j 25
$m(x)$	the mass of a infinitesimal Volume around x 12
M^e_{ij}	$N \cdot N$ matrix elements of the elemental mass matrix belonging to the spectral element e 24
$\mathbf{M}$	global mass matrix 26
$\mathbf{M^{-1}}$	the inverse matrix of the global mass matrix 27
n_a	overall number of anchor points of one element, used to define the shape of the elements 17
n_d	number of coordinate axises, i.e. the dimension of the considered problem 17
n_e	number of elements in the mesh 15
n_g	total number of global grid points 28

ngp	"number of grid points"; measure of the size of a model, independent from physical parameters . . . 78
npw	"number of propagated wavelength"; measure of distance in seismic modelling, independent from physical parameters . . . 78
N	degree of the Lagrange polynomials . . . 20
N_a	shape functions of the mapping, n_d-products of Lagrangian polynomials of degree 1 or 2 . . . 17
P_N	Legendre polynomial of degree N . . . 36
t	time coordinate . . . 12
dt	discrete time step of the time integration scheme . . . 27
$u(x)$	1-D displacement at coordinate x . . . 12
u_i	value of a function (here u) at local coordinate ξ, equivalent to $u(\xi_i)$ . . . 23
$\mathbf{u}$	global displacement vector . . . 26
$\mathbf{u_{t_i}}$	global displacement vector at time step i . . . 27
v	test function used to get the weak formulation of the elastodynamic equation . . . 14
V	volume . . . 12
x	spatial coordinate in 1-D, or first spatial coordinate in 3-D . . . 12
x_a^e	$a = 0, \ldots, N$, anchor points of element e to define its shape . . . 17
α	is the wave velocity in 1-D problems . . . 38
δ_{ij}	Kronecker symbol . . . 20
Δe	length of 1-D elements . . . 16
ε_{ij}	the elements of the strain tensor with $i, j = 1, \ldots, n_d$ . . . 50
ϵ	relative solution error, measure of accuracy of synthetic seismograms . . . 79
Γ	the boundary of the model domain Ω . . . 14
λ	wavelength . . . 78

Λ	SEM standard interval $[-1, 1]$ of local coordinates for every element ... 16
$\Lambda_{n_d} = \mathbf{\Lambda}$	the standard interval or reference square resp. cube for 2- or 3-D SEM simulations; $\mathbf{\Lambda} = \Lambda \otimes \Lambda \otimes \Lambda$ ... 54
μ	Lamé constante, elasticity coefficient ... 12
ω	weights associated with the Gauss-Lobatto-Legendre quadrature of integration ... 22
Ω	the domain of a model in 1-D ... 14
$\Omega_{n_d} = \mathbf{\Omega}$	the considered model domain with explicitly specified number of dimensions n_d; $\Omega_{n_d} = \Omega \otimes \Omega \otimes \cdots \otimes \Omega$, n_d times ... 50
φ_i^N	general denotation for Lagrangian interpolators, which are polynomials of degree N, satisfying $\varphi_i(\xi_j) = \delta_{ij}$ ... 30
ρ	density ... 12
σ_{ij}	elements of the stress tensor with $i, j = 1, \ldots, n_d$ ... 12
ξ	spatial coordinate used inside each element, i.e. the "local" coordinate ... 16
$(\xi, \eta, \zeta) = \boldsymbol{\xi}$	local coordinates in 3-D; $\boldsymbol{\xi} \in \mathbf{\Lambda}$ ... 54
$\hat{\boldsymbol{\xi}}$	the unit vector of local coordinates $\boldsymbol{\xi}$ ... 61
$\langle\rangle$	mean value ... 78
∇	Nabla operator; is equivalent to $\frac{\partial}{\partial x}$ in 1-D ... 14
$\otimes$	tensor product ... 50
$\partial_\xi = \frac{\partial}{\partial \xi}$	short form of a partial derivative in the direction given by the subsrcipt, roman number indices always denote a derivative in global $(x = 1, y = 2, z = 3)$ coordinates ... 62
$\triangle$	Laplace operator; is equivalent to $\frac{\partial^2}{\partial x^2}$ in 1-D ... 14
$\mathcal{F}_e : \Lambda \to \Omega^e$	Coordinate Transformation from local ξ to global x coordinates, i.e. "mapping function" ... 16

Introduction

Even at the beginning of the third millenium, a time ruled by physical development and technology, our knowledge of the Earths interior is still very limited. This is mostly due to the fact that direct access and measurement of properties is not possible. Yet, it is becoming increasingly necessary to learn more about the structure of and processes within the Earth. The cause of visible geological phenomenons is nowadays thought to originate from movements inside the Earth. The theory of plate tectonics developed in the 1960's has led to more effort in understanding the forces acting on the crust. Since then mantle convection was made responsible for those forces and the dynamics of the continents. Moreover, the movement of material in the fluid outer core is known to sustain and influence the geomagnetic field. Magnetic variations itself are of great interest not only to science, but also for economy. Magnetic storms for example, whose effects at the surface of the Earth depends on the intensity of Earth's magnetic field, may affect, even destroy electric power facilites, satellites and disturb radio communication.
Understanding the physics of earthquakes and gathering information on their impact on mankind and infrastructure can lead to a better limiting of their effects.

Seismology plays an important role for all of the above considerations. It is the only discipline that can provide pictures of the whole interior of the Earth and thus gives constraints for theoretical models in geodynamics and geomagnetics. The study of earthquakes over the last hundred years has also led to better risk mitigation by construction of appropriate buildings and more attention to this subject is paid by governments nowadays. In addition, the hazard assessment especially needed by insurance companies was improved.
The major tool for modern seismology is the computer technology, which developed in the last four decades and has strongly influenced not only seismology but the science as a whole. The possibility to compute highly sofisticated yet even analytically unsolvable equations for gigantic structures has taken our knowledge of the Earth an enormous step further. Weather prediction, circulation of the oceans, thermodynamic models of the Earth and tomography of the structure of our globe would be impossible without modern supercomputers.
Parallel to the development of computer hardware several different techniques for the solution of ordinary (partial) differential equations (ODE,PDE) emerged.

Based on the mathematics established by Gauß, Legendre, Taylor and many other great minds, numerical techniques like the finite difference methods (FDM) and finite element methods (FEM) evolved.

This work aims especially at understanding a further numerical approach, the so-called spectral element method (SEM) and to compare it with some of the methods mentioned above. The SEM, which is an extension to FEM, arose a lot of interest in the last years. For computational seismology and fluid dyanmics, it has several advantages over the other methods. So is the FEM, frequently used in engineering, not applicable to seismological studies because of lack in accuracy. Even though FDM have in many cases been applied successfully to simulate wave propagation, they are not tailored to certain classes of problems. Moreover, FD solutions suffer from suboptimal accuracy, too, albeit to a lesser degree than FEM simulations.
An improved version of the FD operators was presented in the last decade to obtain more accurate solutions. These are colloquially called "optimal operators" (OPO). The SEM is said to be very accurate and has also been successfully applied to various problems in computational seismology. The objective of this thesis is to benchmark the SEM against the OPO so as to decide which one of the methods may be more suitable for wave-field modelling.

Outline of the Thesis

This work is divided into two major parts. The first part deals with the theory and mathematical background of the SEM and is intended to provide the reader with the information needed to implement and apply the method, especially to simulations of wave propagation.
After a brief introduction to the history of SEM in Chapter 1, the basic ideas and concepts of the SEM are explained in full detail for one-dimensional problems (Chapter 2). Based on this theoretical framework, a computer programme was developed to apply the SEM for 1-D wave propagation as no such code had existed before. How the SEM formalism can be extended for three-dimensional applications is demonstrated in Chapter 3.
In the second part, the SEM is compared with different FD methods in terms of performance. It turns out that is necessary to develop a new definition of performance that combines both accuracy and speed, as previous benchmark tests were mostly concerned with accuracy alone. In Chapter 4, this definition is given together with the results of the comparison between SEM and OPO for the 1-D case. As 3-D simulations using OPO were not available at the time, the SEM is compared to common FDM for the 3-D case (Chapter 5). This was done using existing codes for both methods. A FORTRAN code for 3-D SEM simulations was

provided by Dimitri Komatitsch and the FD FORTRAN code was written by several students of the Geophysics institute of the Ludwig-Maximilians-University Munich.
In Chapter 6 the results of the investigation are summarized and discussed. Additional information, such as the FORTRAN 90/95 code for 1-D SEM simulations, is provided in the Appendix,

Part I

The Spectral Element Method: History, Theory and Practical Implementation

Chapter 1

The History of the Spectral Element Method

The first chapter deals with the history of the Spectral Element Method to provide the reader with a profound background to this numerical technique. To know about the provenience may be the key to understand the role of this method in today's research community.
The name of this method was derived from two previously developed numerical approaches, which are the pseudo-spectral methods and the Finite Element Method. The former one is known to be very accurate (it is exact up to the Nyquist frequency), whereas the latter has the advantage of being highly flexible when dealing with complex geometries.

The history of the FEM dates back to as early as the beginning of the 20th century with two very important publications by Ritz [1909] and Galerkin [1915]. The method resulting from the latter is also known as the method of weighted residuals (Jung & Langer [2001]). The first work on modern FEM was published in 1956 by Turner *et al.* [1956], and Clough [1960] was the first to use the name "finite element" (see Zienkiewicz & Taylor [2000]). The interest in the method increased in the years after publication of the first edition of Zienkiewicz & Taylor [2000] in 1967 (see Jung & Langer [2001]) and it is since the most frequently used technique to solve partial differential equations in engineering.
In geophysics it was never coequal to the Finite Difference Method, which is the most commonly used approach to simulate seismic wave propagation (Kelly & Marfurt [1990]). Although the FEM has advantages for the definition of numerical models, as the meshes created can be adapted very flexibly to the geological structures, it suffers from low accuracy.
The pseudo-spectral method was first proposed by Kreiss & Oliger [1972] (see Fornberg [1987]). The advantage of this technique is its high accuracy and much less memory requirements. On the other hand, pseudo-spectral

methods are more difficult to implement in complex geometries, because the position of grid points is fix as given by the choice of basis function. Moreover, parallelisation is difficult because of globally defined operators. They can be seen as a limiting case of FDM with increasing order. The PSM, the FDM and the FEM all had their first appearance in seismic wave modelling in the early 1970's (KELLY & MARFURT [1990]).

The SEM itself was first published by PATERA [1984] in the context of fluid dynamics. The idea which led to its development was to combine the advantages of the PSM with those of the FEM. That is, the accuracy and rapid convergence of the former and the geometrical flexibility of the latter. The name was deduced from the fact that the SEM has the same exponential convergence behaviour as the PSM when the order of interpolating polynomials N tends to infinity. PATERA [1984] used Chebychev polynomials to define the interpolating functions, because he thought them to have good approximating properties. Later on, the SEM was further developed in computational fluid dynamics by MADAY & PATERA [1989] who introduced another type of interpolating functions, the Lagrange polynomials, which led to the diagonal structure of the mass matrix when used in combination with the Gauss-Lobatto-Legendre quadrature. PRIOLO & SERIANI [1991] were the first to use the SEM for simulation of wave propagation using Chebychev polynomials. In their work they discussed the accuracy of SEM compared to FEM and the analytical solution for one-dimensional wave equation. In the following years they extended the Chebychev SEM to 2- and 3-D simulations for different geological applications which gave motivation to the publications by SERIANI & PRIOLO [1994]; PRIOLO *et al.* [1994]; PADOVANI *et al.* [1994]; PRIOLO [1999] and SERIANI [1998]. The latter presented a version of this method used on parallel architectures.
In the second half of the 1990's KOMATITSCH [1997] and KOMATITSCH & VILOTTE [1998] developed the SEM with Lagrange polynomials for large scale 3-D wavefield modelling. It was the first time that an exactly diagonal mass matrix could be obtained in seismological applications, which leads to a very simple explicit time scheme and effective parallel implementation. This new evolution in computational seismology gave rise to great interest in this method and to further progression. A detailed theoretical introduction of this version of the SEM for 3-D problems in seismology was given by KOMATITSCH & TROMP [1999].
CHALJUB [2000] was the first to introduce the SEM for global Earth simulations. In the following years it was possible to model the full 3-D laterally heterogeneous Earth, incorporating topography, attenuation and anisotropy (KOMATITSCH & TROMP [2002a]) and later also the effects of the oceans, gravity and rotation of the globe (KOMATITSCH & TROMP [2002b]). Hybrid methods were described by CAPDEVILLE [2000] and CAPDEVILLE *et al.* [2003] who combined the SEM with modal solutions based on normal mode summation after expansion of the spherical harmonics. Moreover, on the methodical side, improvements of the SEM

were employed by Komatitsch *et al.* [2001], who extended it to use with triangular elements in 2-D and an SEM on non-conforming meshes was published by Chaljub [2000] and Chaljub *et al.* [2003]. In addition, the so-called "Perfectly Matched Layer" absorbing boundary conditions for the variational formulation of the elastodynamic equation was implemented in the SEM by Komatitsch & Tromp [2003].
Applications to geological settings were performed to a great extent. Strong ground motion simulations (Komatitsch *et al.* [2003a]) as well as the simulation of different earthquakes (Komatitsch & Tromp [2001a,b]; Komatitsch *et al.* [2003b]). At the time of the appearance of the present thesis new ideas emerge to use the SEM in the context of rupture dynamics as presented at the NMESD workshop, September 1-3, 2003 (Ampuero *et al.* [2003]). These latest trends show that an important tool is now available for numerical modelling of seismic waves, which has a great potential for the future, but is still not yet methodologically sound.

Chapter 2

The Basic Concepts of the Spectral Element Method in 1-D

In order to achieve a good insight into the spectral element method, the mathematical treatment of the 1-D case of the general elastodynamic equation is discussed, followed by details on the computational implementation.
An advantage of this modus operandi is that readers, who know about the theory of SEM beforehand, may skip Section 2.1 and concentrate on the practical aspects.
In Chapter 3, it will be explained how to extend the SEM to 3-D problems. As one will see, all features of SEM that are important for solving the 3-D wave equations can be learned by looking at 1-D. In this way, the reader gets to know all the information needed, while the occuring expressions and formulae remain as simple as possible. Then in 3-D, the 1-D formulae are simply expanded using similar expressions for the two newly added coordinate axes. Only the concept of tensorisation and more complex meshing algorithms will be new for the reader.

Some of the concepts of the method described below are fundamental to the FEM and the SEM itself is based on them. One should keep this in mind as most of it will be introduced from the SEM point of view. It will become clearer in the course of this text that the SEM is a high order finite element method.
Readers who want to obtain a better understanding of the FEM are referred to the standard literature on this topic as for example Hughes [1987], Zienkiewicz & Taylor [2000] or Schwarz [1984].
The description of the SEM in this thesis is mainly based on papers by Dimitri Komatitsch, Jeroen Tromp and Jean-Pierre Vilotte with co-workers (Komatitsch [1997]; Komatitsch & Tromp [1999, 2002a,b]; Komatitsch & Vilotte [1998]). Especially the use of Lagrange polynomials in SEM for wave propagation simulations was introduced by Komatitsch [1997]. Preceding work by Priolo & Seriani [1991]; Seriani & Priolo [1994] will also be mentioned as it describes the use of different approximating basis functions based on Cheby-

chev polynomials.

2.1 The Mathematical Formulation of the Spectral Element Method

Definition of the Mathematical Problem

When modelling seismic waves, the aim is to solve the equation of motion in terms of displacement or equivalently velocity or acceleration in a continuous medium. A general expression can be derived from elasticity theory as shown in AKI & RICHARDS [2002]. It relates the displacement of a certain volume inside a given medium (i.e. with certain elastic properties and certain density) with the stresses and external forces acting on that volume.
I will not discuss topics like attenuation or anisotropic media in this thesis, which surely are of much interest when modelling seismic waves in the Earth. But dealing with these would be far beyond the scope of this work. As shown in the publications KOMATITSCH & TROMP [2002a,b]; KOMATITSCH *et al.* [2000, 2003a] the SEM is able to incorporate all of the mentioned effects on wave propagation with high accuracy.
The general wave equation of a one-dimensional medium can be written as follows:

$$\rho(x)\,\frac{\partial^2\,u(x,t)}{\partial t^2}\;-\;\frac{\partial}{\partial x}\,\Big(\,\mu(x)\,\frac{\partial u(x,t)}{\partial x}\Big) = f_{source}(x,t) \tag{2.1}$$

Where x and t are the spatial and time coordinate respectively, $\rho(x)$ is the density, $u(x,t)$ is the displacement field, μ is an elasticity coefficient (Lamé constant) and f_{source} is an external source, exciting the medium. This particular SH-formulation of the wave equation is chosen to be consistent with the formulations used in METZ [2003]. The derivation of optimal FD operators is discussed there, which were compared to the SEM. Details on these simulations and results of the comparison are presented in Chapter 4.

Note that all three terms of Equation (2.1) are volume forces:

$$f_{res}(x,t) \;=\; \rho(x)\,\frac{\partial^2\,u(x,t)}{\partial t^2} \;=\frac{m(x)}{V}\cdot a(x,t) \tag{2.2}$$

$$f_{internal} \;=\; \frac{\partial}{\partial x}\,(\mu\,\frac{\partial u(x,t)}{\partial x}) = \frac{\partial}{\partial x}\,\sigma(x,t) \tag{2.3}$$

$$f_{external} \;=\; f_{source} \qquad \text{a force acting on the medium} \tag{2.4}$$

The second equals sign of Equation (2.3) is valid due to Hooke's Law for the 1-D case. The generalized form of Hooke's Law can be found in Chapter 3 (Eq. 3.3).

In the spectral element method the considered model domain Ω (here a line), in which the simulation shall be performed, is devided into several elements of the same type, but not essentially of the same length. The elements are nonoverlapping and only connected at one gridnode. That is, the calculations performed inside both of two neighbouring elements contribute to the same degree to the values of the physical parameters at the shared node. This will get clearer when we take a closer look at the "connectivity-matrix". This matrix contains the information, which node belongs to which element. But before we do so, we will derive the mathematical formulations for one single element.

In the following sections the steps that are needed to perform a simulation of wave propagation with the SEM are demonstrated. As in the FEM, we do not consider the wave equation in its differential form of Equation (2.1), but introduce the variational or so-called "weak" formulation, which uses the integrated form of the functions. Starting from this equation we will see how the model domain is meshed and how the meshing has to be taken care of in the weak formulation. Then we will focus on the elements. These are defined in such a way that all following considerations apply to all elements in a similar manner. This simplifies matters significantly as one has to derive the formulations only for one element and the others are treated in exactly the same way. On the element we define a rule for the interpolation of functions and introduce the corresponding interpolating polynomials. Afterwards, the discretized functions have to be integrated numerically, which will be done by the use of a special integration quadrature. This will lead us to two matrices, defined for each element separately: the elemental mass matrix and the elemental stiffness matrix. Then we have to put the elements back together corresponding to the meshing we did in the beginning. This means, that the contributions of the functions have to be assigned to the correct gridpoints of the global mesh. This process is called "assembly". At the end we will get a linear system of equations, which can be rewritten in a matrix formulation. The final matrix equation is usually referred to as the "global" system. It is then the basis for the solution of the DE in time as it can be discretized using an explicit FD scheme in the time domain.

2.1.1 The Weak Formulation of the Elastodynamic Equation in 1-D

To derive the weak formulation of Equation (2.1), let us begin with the one-dimensional wave equation for an inhomogeneous elastic medium (Eq. 2.1):

$$\rho \frac{\partial^2 u}{\partial t^2} - \frac{\partial}{\partial x}(\mu \frac{\partial u}{\partial x}) = f \tag{2.5}$$

u, f, ρ and μ are considered to depend on x, and u and f also on t, in the following without further explicit declaration.

The first step to get the weak formulation is now to multiply Equation (2.5) with a time-independent so-called test function $v(x)$ on both sides. This may be any arbitrary function of the set of functions that are, together with their first derivative, square integrable over Ω (i.e. a continuous and "well-behaved" function). Afterwards we integrate the equation over the spatial domain. Both steps do not change the solution of the equation. The vector notation $\nabla = \frac{\partial}{\partial x}$ is used from now on to state the expressions in a similar way that is needed for 3-D problems:

$$\int_\Omega v\ \rho\ \ddot{u}\ dx\ -\ \int_\Omega v\ \nabla(\ \mu\ \nabla u)\ dx = \int_\Omega v\ f\ dx \tag{2.6}$$

If we integrate the second term on the left side of Equation (2.6) by parts, using a general expression in terms of dimensions, and denote the boundary of Ω by Γ, we obtain:

$$\int_\Omega \rho\ v\ \ddot{u}\ d\Omega\ -\ \int_\Gamma v\ \mu\ \nabla u\ d\Gamma\ +\ \int_\Omega \nabla v\ \mu\ \nabla u\ d\Omega = \int_\Omega v\ f\ d\Omega \tag{2.7}$$

The general integration by parts, wich results in one integral over Γ and one over Ω can be derived by combination of the chain rule and the Gaussian divergence theorem (BRONSTEIN *et al.* [1999], page 662). Here in fact the boundary only consists of two points on either side of the one-dimensional model domain, but this notation is chosen to be consistent with the 3-D case. In 1-D the integral over Γ thus is the antiderivative of the integrand:

$$\int_\Gamma v\ \mu\ \nabla u\ d\Gamma = v\ \mu\ \nabla u\ \Big|_\Gamma \tag{2.8}$$

Then we introduce a zero stress condition on the boundary Γ (i.e. using a free surface boundary condition, which is also called "Neumann condition" in mathematical terms):

$$\mu\ \frac{\partial u}{\partial x}\ \Big|_\Gamma = \sigma_\Gamma = 0 \tag{2.9}$$

Thus the integral over Γ vanishes, which leads to the final form of the weak formulation which we will use in the following:

$$\int_\Omega v\ \ddot{u}\ dx + \int_\Omega \nabla v\ \mu\ \nabla u\ dx = \int_\Omega v\ f\ dx \tag{2.10}$$

The application of other boundary conditions is shown in Section 2.1.9.

This is the weak form of the one-dimensional wave equation. As will be shown in the next subsections, the discretization and decomposition of the model leads to

the previously mentioned linear system of equations. To do so, the $N+1$ collocation points $\xi_i, i = 0, \ldots, N$ of the Gauss-Lobatto-Legendre (GLL) quadrature of order N are considered. Their definition will be given in Section 2.2.2. The order N is typically chosen between 4 and 8.

2.1.2 Domain Decomposition and Mapping Functions

As mentioned earlier, the model domain has to be divided into n_e elements. This is done by taking potential inhomogeneities or discontinuities of the model into account. It may be necessary to adapt the elements to the velocity structure as to avoid oversampling in regions of high wave velocities. In two- or three-dimensional simulations the topography and its realistic implementation in the model are of great interest. Thus, certain rules exist how to create a well matching mesh and how to create the corresponding elements, which may differ significantly. This process is usually time consuming, but on the other hand makes up the advantage of methods using elements over FD methods.
The domain decomposition is now discussed in detail, introducing all needed concepts. In the next chapter, when the SEM for 3-D applications is explained, some examples of meshes and problems that may occur in this context will be given.
Figure 2.1 illustrates the idea of dividing the model domain into elements on the basis of a one-dimensional string. Each element domain is then transformed as shown for element number 1. This issue will be discussed in the forthcoming section. Element number 3 is here defined using three anchor nodes, to include this option for completeness, even if it does not change the shape of a 1-D element.
After dividing the model domain into subdomains Ω^e, the integrations of Equation (2.10) are performed independently on these subdomains. Like this we establish the weak form for every element separately:

$$\int_{\Omega^e} v\ \ddot{u}\ dx + \int_{\Omega^e} \nabla v\ \mu\ \nabla u\ dx = \int_{\Omega^e} v\ f\ dx \tag{2.11}$$

for $e = 1, \ldots, n_e$.

Mapping Functions - Transformation of Physical Coordinates onto a Standardized Interval

Via a coordinate transformation, each element is mapped onto the standard interval $\Lambda = [-1, 1]$ of the GLL integration quadrature[1]. Thus all further calculations can be done in the same way for all the elements. The transform function is usually called "mapping function". It transforms the "global" (or physical) coordinates x_k and x_{k+1} of the element number k into the so-called "local" coordinates

[1] most quadratures used in FEM make use of the more typical interval $\Lambda_{FEM} = [0, 1]$

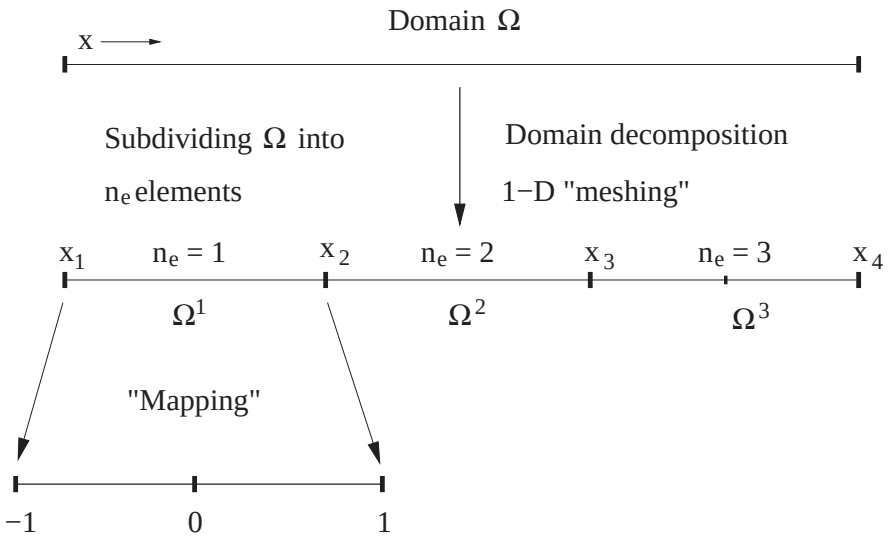


Figure 2.1: Simple domain decomposition and mapping. Here the model is devided into three elements. Element number 3 is show with $n_a = 3$ anchor nodes.

ξ with $\xi \in \Lambda$.
In the general case, where the elements have different lengths, the mapping function has to be defined separately for every element. In the case of elements having the same length, the transforming function is identical for all of them. It has the following form in 1-D:

$$\begin{aligned} \mathcal{F}_e : \quad & \Lambda \rightarrow \Omega^e \\ x^e(\xi) \;=\; \mathcal{F}_e(\xi) \;=\; & \Delta e \, \frac{\xi + 1}{2} + x_e \end{aligned} \tag{2.12}$$

Ω^e is the part of the model domain belonging to the e-th element, Δe is the length and x_e the left point of the element number e. The derivation of this formula is given in Equation (2.16).

Generally the mapping is done using shape functions N_a, which are defined on n_a anchor nodes x_a^e inside the element (the definition is given in Equation 2.13). For one-dimensional elements two anchor nodes per element are sufficient to define a linear structure, one node at each end. For two or three-dimensional elements more anchor points per coordinate axis may be considered to create curved elements. Usually $n_a = 3$ is chosen when the geometry is smooth as more nodes result in unnecessary accuracy of the meshing (KOMATITSCH & TROMP [1999]). Using two points per dimension for three-dimensional elements would result in $2^3 = 8$ anchor nodes, whereas $3^3 = 27$ nodes

would result from shape functions defined on three points per coordinate axis. Figures and definitions for 3-D elements can be found in Chapter 3, Section 3.1.2.

Figure 2.2 and 2.3 show the one-dimensional shape functions of the SEM which are Lagrange polynomials defined on the anchor nodes of the element. The quadratic shape functions on three points are shown here for completeness. Their use is only of relevant for modelling higher dimensional problems. The definition of the Lagrange polynomials is given in Equation (2.21) in the next subsection.

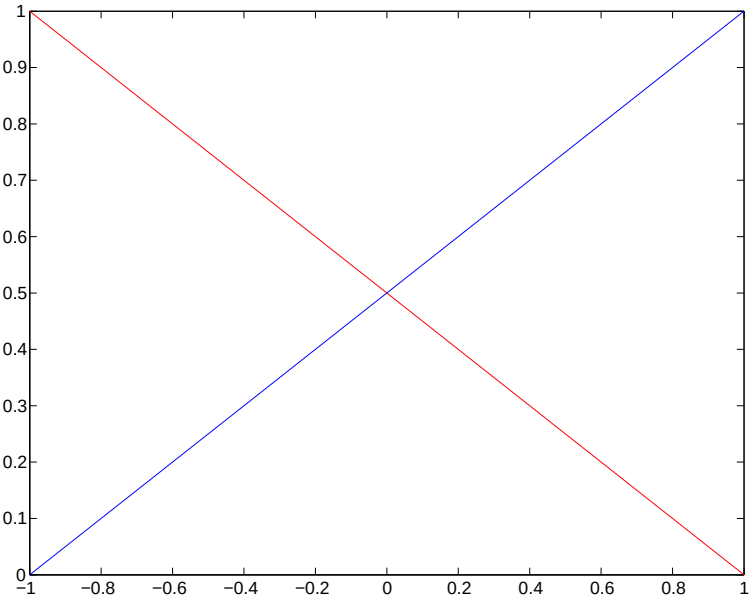


Figure 2.2: One-dimensional shape functions on two anchor nodes.

In general the shape functions are n_d-products of Lagrangian polynomials of degree 1 or 2, depending on the model (e.g. curved surfaces, interfaces etc.), with n_d being the number of dimensions of the considered problem. The general mapping function is of the form:

$$x^e(\xi) = \sum_{a=1}^{n_a} N_a(\xi)\ x_a^e \tag{2.13}$$

The Lagrange polynomials of degree $N = 1$ in Figure 2.2 can be written as:

$$\begin{aligned} \ell_0^1(\xi) &= \frac{\xi - (+1)}{-1 - (+1)} = -\frac{\xi - 1}{2} \\ \ell_1^1(\xi) &= \frac{\xi - (-1)}{1 - (-1)} = \frac{\xi + 1}{2} \end{aligned} \tag{2.14}$$

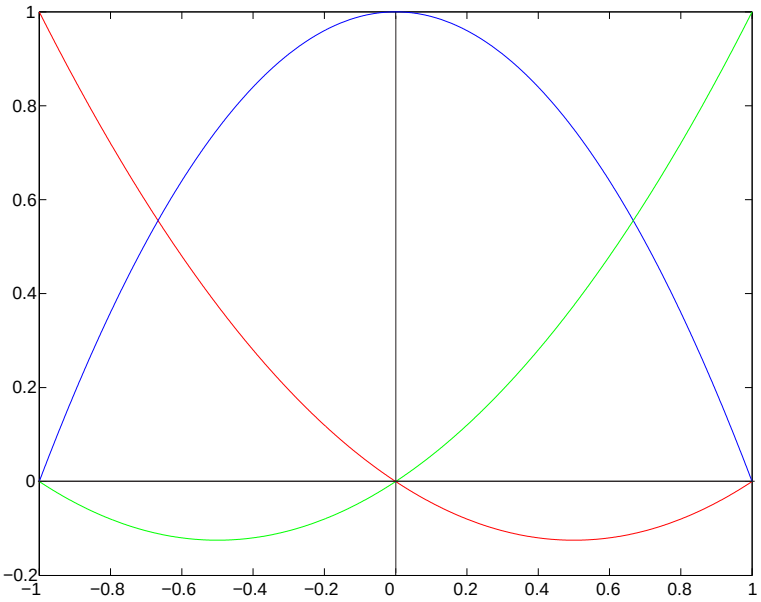


Figure 2.3: One-dimensional shape functions on three anchor nodes.

See the general definition of the Lagrange polynomials in Equation (2.21) for comparison. The shape functions in 1-D then simply are the Lagrange polynomials itself.

$$N_1 \;=\; \ell_0^1 \qquad N_2 \;=\; \ell_1^1 \tag{2.15}$$

Using these expressions we can derive the mapping function defined with two anchor nodes in 1-D, which was given earlier in Equation (2.12). Considering the element k with its corresponding anchor nodes x_k and x_{k+1} and introducing the notation $x_k = x_1^k$ and $x_{k+1} = x_2^k$, Equation (2.13) will lead to:

$$\begin{aligned}
x^k(\xi) &= \sum_{a=1}^{2} N_a(\xi)\; x_a^k \\
&= \ell_0^1\; x_1^k \;+\; \ell_1^1\; x_2^k \\
&= x_1^k \cdot (-\frac{\xi-1}{2}) \;+\; x_2^k \cdot (\frac{\xi+1}{2}) \\
&= \frac{x_k - x_k\xi + x_{k+1}\xi + x_{k+1}}{2} \\
&= \frac{x_k + (x_{k+1} - x_k)\xi + [x_k + (x_{k+1} - x_k)]}{2} \\
&= \frac{x_k + \Delta e_k\xi + x_k + \Delta e_k}{2}\,, \qquad \Delta e_k = x_{k+1} - x_k \\
&= \Delta e_k \frac{\xi+1}{2} + x_k
\end{aligned} \tag{2.16}$$

This is what we expected from Equation (2.12). So far we have discussed the coordinate transformation itself. Now an explaination on how this transformation affects the subsequent considerations is presented.

As we will see later, we have to integrate over the spatial coordinate. When doing so, we have to take the contribution of the transformation into account, which then occurs under the integral as the determinant of the transformation matrix of the mapping. This transformation matrix is usually called "Jacobi-Matrix" and denoted by $\mathbf{J}$. Its determinant is then referred to as "Jacobian" and denoted by $\mathcal{J}$. The term "matrix" is used here in the context of 1-D problems, although it is clear that in this case it consists of one matrix element only. In this way on can use the same expressions for the 3-D case. More information about the Jacobian and the Jacobi matrix can be found in the mathematical standard literature, for instance Bronstein *et al.* [1999].

The Jacobi-Matrix and -Determinant of the Mapping Function

One of the big advantages of the methods using elements is that one is able to apply the same mathematics to all elements in a similar way, but dealing with every element separately.
In order to achieve this we have to perform the above coordinate transformation $\mathcal{F}_e(\xi)$ for each element through which it is mapped onto the standard interval of the GLL integration quadrature $\Lambda = [-1, 1]$. It is known from mathematics that a coordinate transformation has to be taken into account when performing it during an integration. This is the case in Equation (2.11). An additional term appears under the integral, which is the Jacobian. In general, a coordinate transformation under an integral has to be performed in the following way (Bronstein *et al.* [1999]):

$$\int_{\Omega^e} g(x)\, dx = \int_{\Lambda} g^e(\xi)\, \frac{dx}{d\xi}\, d\xi = \int_{-1}^{1} g^e(\xi)\, \mathcal{J}^e\, d\xi \tag{2.17}$$

where g is an arbitrary function and the superscript denotes the restriction of g on the element number e.

The Jacobi-Matrix and its determinant are hereby generally defined as:

$$\begin{aligned} \mathbf{J}^e &= \frac{d\mathcal{F}_e(\xi)}{d\xi} \\ \mathcal{J}^e &= \det \mathbf{J}^{\mathrm{e}} \end{aligned} \tag{2.18}$$

In the present case of the one-dimensional SEM they are written as:

$$\mathbf{J}^e = \frac{dx^e}{d\xi}, \qquad \mathcal{J}^e = |\frac{dx^e}{d\xi}| \tag{2.19}$$

In practice, the elements of the Jacobi-Matrix are computed using the shape functions of the meshing:

$$\frac{dx(\xi)}{d\xi} = \sum_{a=1}^{n_a} \frac{dN_a(\xi)}{d\xi} \, x_a \tag{2.20}$$

In the case of 1-D elements having the same length Δe, the Jacobian becomes $\mathcal{J} = \frac{\Delta e}{2}$.

2.1.3 Interpolation of Functions on the Elements

The analytical functions of Equation (2.11) must now be approximated by discrete functions using an appropriate interpolation scheme on some discrete points. It turned out that it is very convenient to use Lagrange-Polynomials ℓ^N as interpolating functions, which are defined on the collocation points of the Gauss-Lobatto-Legendre quadrature for integration together with an interpolation scheme of Lagrangian type. Typically a degree $N = 4, \ldots, 8$ is chosen for the polynomials. The definition of the GLL quadrature and its corresponding collocation points is given in Sections 2.1.4 and 2.2.2.
Using the same points for interpolation and integration is the key feature of the SEM in connection with Lagrange functions. This will become clearer when we will talk about the mass-matrix as it will lead to the exact diagonal structure of this matrix. The Lagrange polynomials are defined in the following way:

$$\ell_i^N = \prod_{\substack{j=0 \\ j \neq i}}^{N} \frac{\xi - \xi_j}{\xi_i - \xi_j} \tag{2.21}$$

The superscript ^{N} will be omitted in the following as it stays the same during the calculations.

The Lagrange polynomials have an important characteristic for our considerations. Each polynomial ℓ_i is exactly 1 at the coordinate ξ_i and exactly 0 at all other nodes of the element. Between the collocation points the polynomial may have any value. But as we are only interested in the discrete points now, this does not change the following mathematical derivations. This property of the interpolating functions can be expressed by means of the Kronecker-delta δ_{ij}:

$$\ell_i(\xi_j) = \delta_{ij} \tag{2.22}$$

The functions we are interested in are then interpolated inside the elements using the Lagrange interpolation scheme:

$$u^e(\xi) \approx \sum_{i=0}^{N} u^e(\xi_i)\ \ell_i(\xi) \tag{2.23}$$

u^e is the function u restricted to the area $[x_e, x_{e+1}]$ of element e mapped onto the standard interval. Because of the property of the polynomials in Equation (2.22), the approximation of the function u^e inside the element e is exact on the collocation points and continuity of the functions across the borders of the elements is therefore assured. Equation 2.23 also serves for computing the value of a function at any point ξ inside the element, whose values are known at all GLL-points.

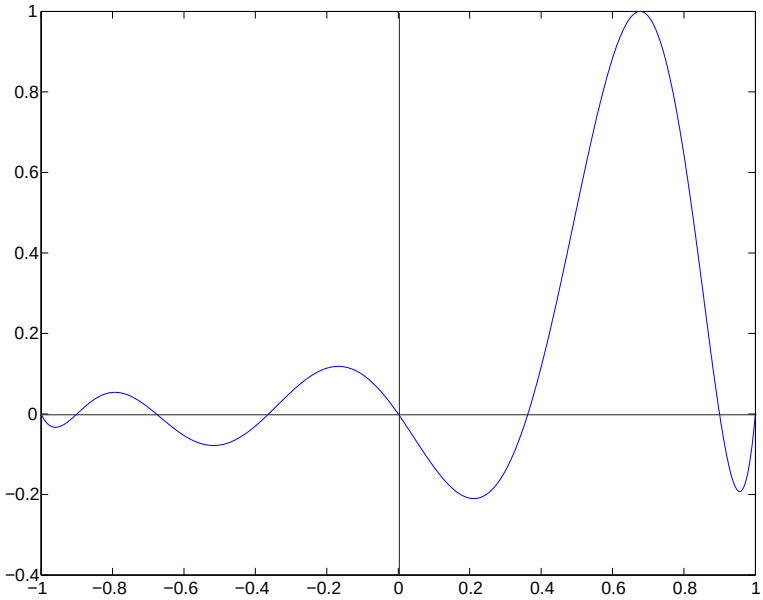


Figure 2.4: Lagrange polynomial ℓ_6 of order $N = 8$. The collocation points can be clearly distinguished as the points where the function takes the values 0 and 1.

As we can see on the left side of Equation (2.11), we need to interpolate the two derivatives of u^e and v^e which appear in the second term. To achieve this, we will have to calculate the derivations of the Lagrange polynomials on the collocation points of the element as the interpolation of a derivated function can be expressed as:

$$\nabla u^e(\xi) \approx \sum_{i=0}^{N} u^e(\xi_i)\ell_i^{'}(\xi) \tag{2.24}$$

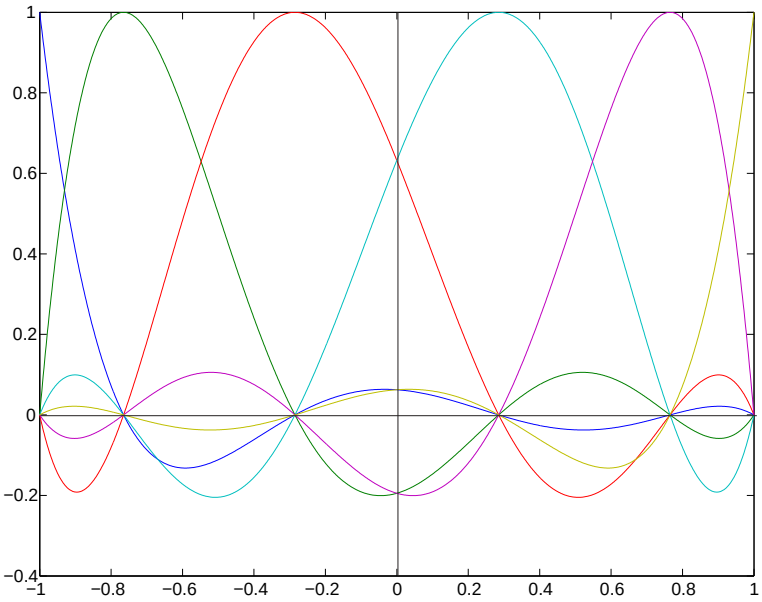


Figure 2.5: All six Lagrange polynomials of order $N = 5$.

How to obtain the derivatives ℓ_i' in practice will be explained in Section 2.2.2.

2.1.4 Integration of Functions over the Element Domain

The next step on the way to the matrix formulation of Equation (2.11), which we finally want to obtain, will now be presented in the following. As we are still treating each element separately, the integration has to be performed over the interval $[-1, 1]$. This is done using the so-called Gauss-Lobatto-Legendre quadrature of integration, i.e. transferring the integral into a finite weighted sum:

$$\int_\Lambda g(\xi)\, d\xi \approx \sum_{i=0}^{N} \omega_i\, g(\xi_i) \tag{2.25}$$

ω_i are the weights of the GLL quadrature. Their definition will be given in paragraph 2.2.2. Note that $g(\xi)$ is again an arbitrary function defined on the interval Λ. The collocation points $\xi_i,\ i = 0, \ldots, N$ are hereby the $N + 1$ roots of the first derivative of a Legendre polynomial[2].

The GLL quadrature is a special case of the Gauss quadrature, which has the same form as Equation (2.25), only the choice of collocation points is different (see BRONSTEIN *et al.* [1999], page 894 and 895). In the Gauss quadrature the

[2]Their calculation is shown together with the integration weights in Section 2.2.2.

boundary points -1 and 1 of the standard interval are not included. This leads to a numerical integration which is exact for polynomials up to degree $2N + 1$. In contrast to that the GLL quadrature is exact only for polynomials up to degree $2N - 1$. This may seem very disadvantageous, because we have to integrate polynomials of degree $2N$ (resulting from the product of the test function and the displacement). But only the GLL quadrature allows for a diagonal mass matrix as the Lagrange polynomials for the interpolation can then be defined on the same points.

2.1.5 The Elemental Mass Matrices and Their Diagonality

In this paragraph the procedure how to get a matrix formulation for Equation (2.11) is demonstrated. This formulation, which is valid for one single element, is obtained by successively applying all previously discussed steps to the integrals in this equation. After recombining the matrices of the elements we get a linear system of equations for the whole domain, the "global" system. This will then be the basis for the solution of the integration over the time-domain, which is done using a typical explicit FD scheme.
Starting with the first integral of Equation (2.11) we obtain for one element:

$$
\begin{aligned}
& & \downarrow & \quad \text{mapping, Jacobian is needed} \\
\int_{\Omega^e} \rho(x)\ v(x)\ \ddot{u}(x)\ dx & = & \int_{\Lambda} \rho^e(\xi)\ v^e(\xi)\ \ddot{u}^e(\xi)\ \mathcal{J}^e(\xi)\ d\xi & \\
\text{interpolation of } v \text{ and } \ddot{u} & \downarrow & & \\
& \approx & \int_{-1}^{1} \rho(\xi)\ [\sum_{i=0}^{N} v_i\ \ell_i(\xi)]\ [\sum_{j=0}^{N} \ddot{u}_j\ \ell_j(\xi)]\ \mathcal{J}(\xi)\ d\xi & \\
\text{GLL quadrature} & \downarrow & & \\
& \approx & \sum_{k=0}^{N} \{\rho(\xi_k)\ \omega_k\ [\sum_{i=0}^{N} v_i\ \ell_i(\xi_k)]\ [\sum_{j=0}^{N} \ddot{u}_j\ \ell_j(\xi_k)]\ \mathcal{J}(\xi_k)\} &
\end{aligned}
\tag{2.26}
$$

For simplification $v(\xi_i)$ and $\ddot{u}(\xi_j)$ are expressed by v_i and $\ddot{u}_j$ respectively. The same applies to ρ and $\mathcal{J}$ in the next equation. The superscript ^{e} points out that the functions have to be restricted to the element number e when being transformed from global to local coordinates. This is valid for all of the following considerations, but for clarity it will be droped from now on.

As there are no restrictions to the choice of v we can set $v_i = 1$ and by using Equation (2.22) we get:

$$\sum_{k=0}^{N}\{\rho(\xi_k)\ \omega_k\ [\sum_{i=0}^{N} v_i\ \ell_i(\xi_k)]\ [\sum_{j=0}^{N} \ddot{u}_j\ \ell_j(\xi_k)]\ \mathcal{J}(\xi_k)\} =$$

$$= \sum_{k=0}^{N}\{\rho_k\ \omega_k\ [\sum_{i=0}^{N} \delta_{ik}]\ [\sum_{j=0}^{N} \ddot{u}_j\ \delta_{jk}]\ \mathcal{J}_k\}$$

$$= \sum_{j=0}^{N}\{\ddot{u}_j\ [\sum_{i=0}^{N}\sum_{k=0}^{N} \rho_k\ \omega_k\ \mathcal{J}_k\ \delta_{ik}\ \delta_{jk}]\}$$

$$= \ddot{u}_j M^e_{ij} \tag{2.27}$$

Here we use the implicit sum convention over repeated indices for the last relation, which was originally introduced by Albert Einstein.
M^e_{ij} are now the $(N+1)\cdot(N+1)$ elements of the elemental mass matrix of the form:

$$M^e_{ij} = \rho_i\ \omega_i\ \mathcal{J}_i\ \delta_{ij} \tag{2.28}$$

Equation 2.28 follows directly from the definitions of the Kronecker-delta $\delta_{ik}\cdot\delta_{jk} = \delta_{ij}$ and tells us that only the diagonal elements of the elemental mass matrix and thus also of the global mass matrix are occupied. This is the key feature of the SEM in combination with the Lagrange polynomials. It simplifies and speeds up the inversion process of the mass matrix significantly as one only has to take the reciprocal value of the diagonal elements. In addition, computer memory requirements are reduced by a great amount as the values can be stored in a one-dimensional vector instead of a two dimensional matrix.

2.1.6 The Elemental Stiffness Matrices

The elemental stiffness matrices are derived in a similar way. But during their calculation we have to take a further term into account, which occurs because of the coordinate transformation of the Nabla-operator ∇ (i.e. in this case a one-dimensional derivation in space). Another point we have to keep in mind is that Equation 2.22 does not hold for the derivations of the Lagrange polynomials. Therefore not only the diagonal but all elements of the elemental stiffness matrix

are non-zero. Shown below is the derivation of the elemental stiffness matrix starting with the second integral of Equation (2.11). Note that in the first statement the Nabla operator ∇_x itself has to be transformed to local coordinates, afterwards denoted ∇_ξ. As we transform global to local coordinates in this case ($\Omega \rightarrow \Lambda$), we have to calculate the Jacobi-Matrix of the inverse transformation of the previously discussed mapping, which will be denoted by ${}^{\mathbf{i}}\mathbf{J}$:

$$\int_{\Omega^e} \nabla_x\, v(x)\, \mu(x)\, \nabla_x\, u(x)\, dx =$$

$$\begin{aligned}
&= \int_\Lambda \nabla_x\, v(\xi)\, \mu(\xi)\, \nabla_x\, u(\xi)\, \mathcal{J}(\xi)\, d\xi \\
&= \int_\Lambda \nabla_\xi\, v(\xi)\frac{\partial \xi}{\partial x}\, \mu(\xi)\, \nabla_\xi\, u(\xi)\frac{\partial \xi}{\partial x}\, \mathcal{J}(\xi)\, d\xi \\
&\approx \int_{-1}^{1} [\sum_{i=0}^{N} v_i\, \ell_i'(\xi)]\, \mu(\xi)\, [\sum_{j=0}^{N} u_j\, \ell_j'(\xi)] \left(\frac{\partial \xi}{\partial x}\right)^2\, \mathcal{J}(\xi)\, d\xi \\
&\approx \sum_{k=0}^{N} \{\omega_k\, [\sum_{i=0}^{N} v_i\, \ell_i'(\xi_k)]\, [\sum_{j=0}^{N} u_j\, \ell_j'(\xi_k)]\, \mu_k\, {}^{\mathbf{i}}\mathbf{J}^2(\xi_k)\, \mathcal{J}(\xi_k)\} \\
&= \sum_{j=0}^{N} \{u_j\, [\sum_{i=0}^{N}\sum_{k=0}^{N} \mu_k\, \omega_k\, {}^{\mathbf{i}}\mathbf{J}_k^2\, \mathcal{J}_k\, \ell_{ik}'\, \ell_{jk}']\} \\
&= u_j K_{ij}^e
\end{aligned} \tag{2.29}$$

Again we have used the sum convention in the last expression of Equation (2.29) and we introduced the notation $\ell_i'(\xi_k) = \ell_{ik}'$. Obviously, the elemental stiffness matrix is fully occupied since $\ell_{ij}' \neq \delta_{ij}$. It can be computed using:

$$K_{ij}^e = \sum_{i=0}^{N}\sum_{k=0}^{N} \mu_k\, \omega_k\, {}^{\mathbf{i}}\mathbf{J}_k^2\, \mathcal{J}_k\, \ell_i'(\xi_k)\ell_j'(\xi_k) \tag{2.30}$$

2.1.7 The Assembly of the Global Linear System

Having introduced all features of the SEM on the elemental level, we have to step back to physical coordinates again and add all the contributions of previously gained matrices together to obtain a global linear system, which we can then invert for the integration in the time domain. This step is called the "assembly" and can be performed in two different ways.

The more complicated one is to calculate all elemental stiffness matrices, assemble them once during a simulation and then multiplying the global stiffness matrix with the displacement vector $\mathbf{u_{t_i}}$ in each time step t_i, as it is given in Equation (2.34). The problem using this procedure is to calculate the elemental stiffness matrices in 2- or 3-D. But as it is possible, even simple in 1-D, and making use of some standard FEM features, it will be introduced it in Section 2.2.6.

The second and more convenient way of assembling the global system is to compute the forces at each node of the global numerical grid separately by first calculating the forces on elemental level and afterwards summing the contributions on nodes shared by elements (assembly). This approach involves the storage of elastic properties, Jacobi-Matrices and corresponding Jacobians throughout the whole simulation (i.e. during the time loop), calculation of stresses and numerical integration at all grid nodes and for all time steps as well as the addition of external forces. It has therefore higher memory requirements. The concrete procedure of the calculation of forces is explained together with a descriptive illustration of the assembly process in Section 2.2.

They both need an appropriate information, which element (and its corresponding values at the collocation points) contributes to the values of the functions at the global nodes. This information consists of a transformation of the global numbering of the nodes to the local numbering of the interpolating points and elements and is stored in a matrix called "connectivity-matrix". All this is discussed when I demonstrate the practical implementation of the SEM (Sec. 2.2.6). When dealing with the theory it suffices to know that there is an appropriate operator called "assembly-Operator" $\boldsymbol{\mathcal{A}}$, which reassembles the elemental matrices to a globally defined expression (see KOMATITSCH & VILOTTE [1998]):

$$\mathbf{M} = \mathop{\boldsymbol{\mathcal{A}}}_{e=1}^{n_e}(\mathbf{M}^e) \tag{2.31}$$

In this way we can assemble all matrices ($\mathbf{M}^e$ and $\mathbf{K}^e$) and vectors ($\mathbf{u}^e$ and $\mathbf{f}^e$) defined for the elements separately to obtain a global matrix equation. After having performed all steps mentioned previously, one finally gets this global matrix equation derived from the weak formulation of the DE:

$$\begin{aligned} \mathbf{M\ddot{u}} + \mathbf{Ku} &= \mathbf{f} \\ \mathbf{M\ddot{u}} + \mathbf{f_{internal}} &= \mathbf{f_{external}} \end{aligned} \tag{2.32}$$

When calculating the complete forces at every node instead of computing them using the stiffness matrix, the global linear systems of equation reduces to:

$$\mathbf{M\ddot{u}} = \mathbf{f_{total}} \tag{2.33}$$

By applying a commonly used explicit FD scheme this equation then has to be inverted to solve the time-dependent problem.
In fact, to set up this equation correctly one would have to transpose both matrices **M** and **K**. But as they are symmetric, this can be neglected.

2.1.8 Integration of the Global Linear System in Time

In order to integrate Equation (2.32) in time, an explicit 3 point FD scheme is used in the present 1-D code. The solution for the displacement at the next time step $t_i + 1$ is obtained by inverting the mass matrix, which gives:

$$\mathbf{u_{t_{i+1}}} = dt^2 \left[\mathbf{M^{-1}} \left(\mathbf{f} - \mathbf{K u_{t_i}}\right)\right] + 2\mathbf{u_{t_i}} - \mathbf{u_{t_{i-1}}} \tag{2.34}$$

Both matrices **M** and **K** need not to be transponed as they are symmetric (see Eqs. 2.28 and 2.30).
The inversion of the mass matrix (resp. vector) is very easy as every element of the inverse vector can be derived by the reciprocal value of the corresponding element of the mass vector. The implementation in the code is very convenient when using Fortran as explained in Section 2.2.5. It is important to state here that the overall accuracy of the method is mainly governed by the accuracy of the time integration scheme, and not by the spatial discetization, especially when simulating a great number of time steps.

Further Time Integration Schemes

One important group of generalized time integration schemes are the so-called "Newmark-type" schemes. The general form is for example explained in Komatitsch [1997] and Hughes [1987]. Latter also discusses stability issues. The general scheme is written using two different coefficients and its behaviour stronlgy depends on the choice of these. Using one coefficient equal to zero and the other one exactly 0.5 leads to an explicit second order scheme, which is equivalent to a second order finite difference scheme (Komatitsch [1997]). One special property of this explicit Newmark scheme is the conservation of total angular momentum. In addition, they provide higher accuracy of the discrete integrations. Thus these are mostly prefered. It can be written in acceleration or in velocity formulation. Either one of them can also be rewritten in an iterative predictor-corrector formulation.
Two formulations beside the simpler explicit FD scheme given above were used in this study for comparison of 1-D simulations. Both use an acceleration formulation and differ in number of iterative steps of the predictor-corrector algorithm. Small excerpts of the codes, illustrating the implementation of these time integration schemes, is given in the Appendix A.2.

2.1.9 Boundary Conditions

The last point that will be discussed in the context of the theory of SEM is how to implement several different boundary conditions.
We have seen during the derivation of the weak formulation that free surface boundary conditions (the Neumann conditions) are naturally included in methods using an elemental approach. This is another big advantage of the SEM, compared to FD methods, where the free surface conditions need great effort to be implemented. It is, together with the better approximation of curved topography, the reason why the SEM is better suited for the simulation of surface waves.
It shows up that the implementation of other boundary conditions such as rigid (Dirichlet conditions), periodic or absorbing boundaries is also not very difficult. Only the latter one poses problems in real simulations. But only very recently, the so-called "Perfectly Matched Layer" (PML) absorbing conditions were introduced in the context of the weak formulation by KOMATITSCH & TROMP [2003] which show very high efficiency. The PML were in the first part introduced for FD simulations of wave propagation in heterogeneous media using the differential form of the wave equation in the velocity-stress formulation by COLLINO & TSOGKA [2001].

Rigid Boundary Condition

The simplest boundary condition beside the free surface condition is a rigid boundary. This condition for rigid boundaries implies that the displacement at the boundary Γ is 0 for all times:

$$\mathbf{u}_\Gamma(t) = 0 \ , \qquad \forall \ t \tag{2.35}$$

The rigid behaviour of the boundaries can easily be obtained in the simulations by just omitting the lines and rows of the matrices and vectors of the global system belonging to the boundary nodes. This is often referred to as the "condensation" of the matrices. Like this we implicitly specify that the values of $\mathbf{u}$ are 0 for all times. For the vectors in the 1-D case that means only considering the global nodes from 2 to $(n_g - 1)$, n_g being the total number of grid points in the global mesh.

$$\begin{aligned} \mathbf{u}_{\mathbf{t_{i+1}}}(2,\ldots,n_g-1) \ = \ \Delta \mathrm{t}^2 \cdot \mathbf{M}^{-1}(2,\ldots,\mathrm{n_g}-1) \ \ \mathbf{f}_{\mathrm{tot}}(2,\ldots,\mathrm{n_g}-1) \ + \\ + \ 2\mathbf{u}_{\mathbf{t_i}}(2,\ldots,n_g-1) \ - \ \mathbf{u}_{\mathbf{t_{i-1}}}(2,\ldots,n_g-1) \end{aligned} \tag{2.36}$$

Periodic Boundary Condition

Periodic boundary conditions can also easily be implemented in a SEM code. The only change compared to the zero stress condition is that in this case the first and the last node of a one-dimensional string are now the same. This means that the first and the last spectral element contribute to the values of displacement at this node. The whole string can be seen as a ring now. In terms of the global system of equations we have to add these contributions to the elements of the mass- and stiffness matrix, or when using forces instead, the force vector, corresponding to the node where the "ring" is welded together. In other words, we have to assemble the last and the first element in exactly the same way as all other elements are.

$$\begin{aligned} \mathbf{M}_{1,1}^{periodic} &= \mathbf{M}_{1,1} + \mathbf{M}_{n_g,n_g} & \mathbf{f}_1^{periodic} &= \mathbf{f}_1 + \mathbf{f}_{n_g} \\ \mathbf{M}_{n_g,n_g}^{periodic} &= \mathbf{M}_{n_g,n_g} + \mathbf{M}_{1,1} & \mathbf{f}_{n_g}^{periodic} &= \mathbf{f}_{n_g} + \mathbf{f}_1 \end{aligned} \tag{2.37}$$

In the case of the stiffness matrix it would be a bit more complicated as one would have to expand the stiffness matrix with $2N$ columns to account for the elastic properties of both, the first and the last element, to the value of displacement at node $1 = n_g$.

Absorbing Boundary Condition

Absorbing boundaries in seismic wave simulations still pose a big problem. Nevertheless, the PML absorbing boundaries seem to be very efficient and may be sufficient for most of the problems in seismology. The first conditions for absorbing boundaries in elastic wave equations were introduced by Clayton & Engquist [1977].
Komatitsch *et al.* [1999] and Komatitsch & Tromp [1999] use a first order approximation based on a formulation by Stacey [1988] for the absorbing boundaries. For one-dimensional wave propagation it is rather simple, relating traction to velocity in the following way:

$$\sigma_{\Gamma} \;=\; \rho_{\Gamma}\; \alpha\; \dot{u} \tag{2.38}$$

α is the wave speed.
As it only applies to the two corner nodes the absorbing conditions are implemented by:

$$\begin{aligned} \sigma(1) &= \rho(1)\; \alpha(1)\; \dot{u}(1) \\ \sigma(n_g) &= \rho(n_g)\; \alpha(n_g)\; \dot{u}(n_g) \end{aligned} \tag{2.39}$$

The stresses obtained are simply added to the stresses acting on the nodes due to internal forces. See Section 2.2.8 for an explanation on how these stresses and internal forces are computed.

In the last paragraphs it became clear that most boundary conditions can be implemented using the calculation of forces at the nodes. This is another reason why this approach is the preferable one. Using the stiffness matrix would result in many more complications.

2.1.10 The SEM with Chebychev polynomials

In the last section of the theoretical introduction to the simulation of 1-D problems, the implementations of Chebychev polynomials, used in the early stages of the SEM, will be explained in short. This is done for several purposes: on one hand it is considered very important for a complete overview of the method. On the other hand it shows the improvement which the method experienced due to the usage of Lagrange polynomials and in addition it is meant to honour the work of the people first using this method, like PATERA [1984]; MADAY & PATERA [1989]; PRIOLO & SERIANI [1991]; SERIANI & PRIOLO [1994]. A short treatise of the history of SEM can be found in Chapter 1.

Most of the concepts for the SEM with Chebychev polynomials are the same as with Lagrange polynomials as there are the domain decomposition, the mapping and the calculations of derivatives under an integral using the Jacobi-Matrix. The functions are also interpolated by a piecewise polynomial approximation using an Lagrangian interpolation on each element as in PATERA [1984] (compare also to Eq. 2.23):

$$u^e(\xi) \approx \sum_{i=0}^{N} u^e(\xi_i)\ \varphi_i^N(\xi) \tag{2.40}$$

φ_i^N are Lagrangian interpolators (polynomials of degree N) defined on $[-1, 1]$, satisfying the relation $\varphi_i(\xi_j) = \delta_{ij}$. In contrast to the Lagrange polynomials the calculation of these interpolating functions is more complex. The polynomials φ_i^N are defined using Chebychev polynomials T_k up to order N:

$$\varphi_i^N(\xi) = \frac{2}{N} \sum_{k=0}^{N} \frac{1}{\bar{c}_i\ \bar{c}_k} \cdot T_k(\xi_i) T_k(\xi) \tag{2.41}$$

with

$$\bar{c}_i = \begin{cases} 1 & \text{for } i \neq 0, N \\ 2 & \text{for } i = 0, N \end{cases}$$

The Chebychev polynomials are defined in the following way (see Bronstein *et al.* [1999], page 918):

$$T_k(x) = \cos(k \arccos x) \tag{2.42}$$

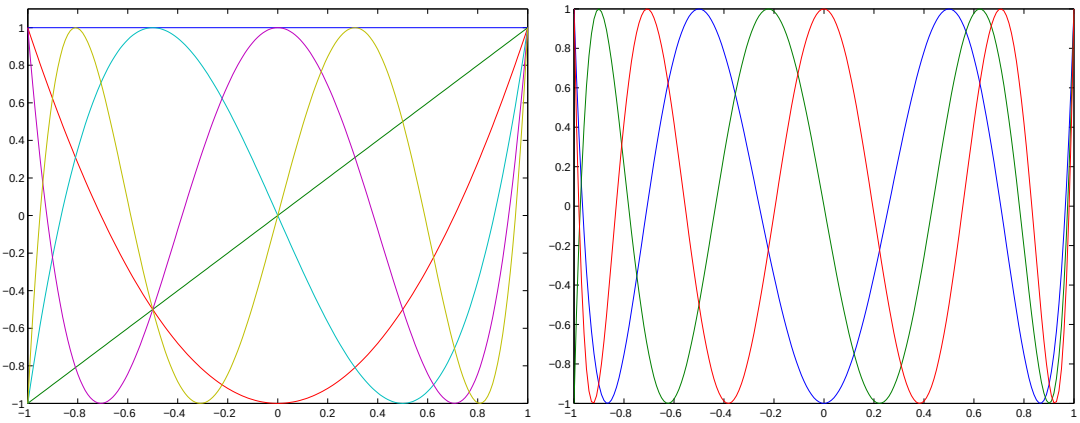
Another frequently used definition is (see Patera [1984]):

$$T_k(cos\theta) = \cos k\theta \tag{2.43}$$

The Chebychev polynomials can also be computed using a recursion formula (see Priolo & Seriani [1991]):

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x) \tag{2.44}$$

with $T_0(x) = 1$ and $T_1(x) = x$.



(a) Chebychev Polynomials of order $N = 0, \ldots, 5$.

(b) Chebychev Polynomials of order $N = 6, 7, 8$.

Figure 2.6: Chebychev Polynomials up to degree $N = 8$ on the interval $[-1, 1]$.

The interpolating points ξ_i are chosen as the Chebychev Gauss-Lobatto quadrature (CGL) points which can be computed using:

$$\xi_i = \cos\left(\frac{\pi\ i}{N}\right) \qquad \text{for } j = 0, \ldots, N \tag{2.45}$$

The resulting interpolating polynomials φ_i look very similar to the Lagrange polynomials. This is because the collocation points of the different quadratures do not lie very far from each other as shown in Table 2.1. One of the differences is that the polynomials φ_i can have values higher than 1 in the interval $[-1, 1]$. This is shown in Figure 2.7(a), where the violett and green polynomial exceed

GLL points	CGL points
± 0.285231531	± 0.309016994
± 0.765055299	± 0.809016994
± 1	± 1

Table 2.1: Comparison of the collocation nodes of the Gauss-Lobatto-Legendre quadrature (left) and the Chebychev Gauss-Lobatto quadrature (right) for order $N = 5$.

the value of 1 in the vicinity of the collocation point $\xi_i = 0.809016994$. Also shown are the Lagrange polynomials for comparison. In Figure 2.7(d) only one polynomial of each kind is plotted to illustrate the small but obvious difference of the graphs.
The major difference of the Chebychev-SEM compared to the Lagrange-SEM is that in deriving the elemental matrices, the integrations occuring in the variational formulation are not evaluated by a discrete numerical quadrature. This leads to elemental mass matrices that are fully occupied (i.e. all matrix elements are non-zero). In this case the elemental mass matrices for a homogenous model and elements of same length get the following form (PRIOLO & SERIANI [1991]):

$$\mathbf{M}^e_{ij} = \rho \, \frac{\Delta e}{2} \, \frac{4}{N^2 \, \bar{c}_i \, \bar{c}_j} \sum_{k,l=0}^{N} \Big(\frac{1}{\bar{c}_k \, \bar{c}_l} \cdot T_k(\xi_i) T_l(\xi_j) \int_{-1}^{1} T_k(\xi) T_l(\xi) \, d\xi \Big) \tag{2.46}$$

The value $\frac{\Delta e}{2} = \mathcal{J}$ is the Jacobian for elements of same length (see page 20). The evaluation of the integral over both Chebychev polynomials $T_k(\xi)T_l(\xi)$ leads to:

$$\int_{-1}^{1} T_k(\xi) T_l(\xi) \, d\xi = \begin{cases} 0 & \text{for } k+l \text{ odd} \\ \frac{1}{1-(k+l)^2} + \frac{1}{1-(k-l)^2} & \text{for } k+l \text{ even} \end{cases} \tag{2.47}$$

For the same model, the elemental stiffness matrices will look like:

$$\mathbf{K}^e_{ij} = \frac{\Delta e}{2} \, \frac{4}{\Delta e^2} \, \frac{4}{N^2 \, \bar{c}_i \, \bar{c}_j} \sum_{k,l=0}^{N} \Big(\frac{1}{\bar{c}_k \, \bar{c}_l} \cdot T_k(\xi_i) T_l(\xi_j) \int_{-1}^{1} T_k'(\xi) T_l'(\xi) \, d\xi \Big) \tag{2.48}$$

The additional term $\frac{4}{\Delta e^2}$ represents the Jacobi-Matrix of the inverse mapping for this case. Evaluating the integral over the two derivatives $T'_k(\xi)T'_l(\xi)$ yields:

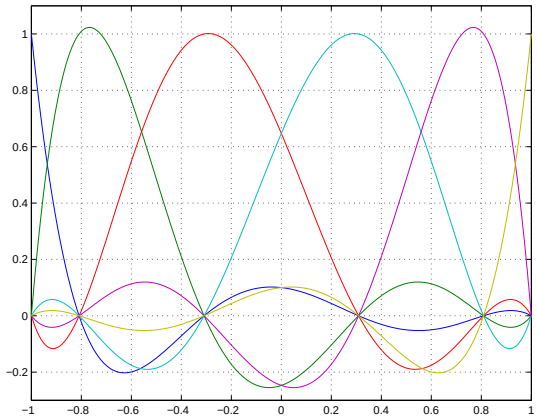
$$\int_{-1}^{1} T_k'(\xi) T_l'(\xi) \, d\xi = \begin{cases} 0 & \text{for } k+l \text{ odd} \\ \frac{k\,l}{2}(H_{|(k-l)/2|} - H_{|(k+l)/2|}) & \text{for } k+l \text{ even} \end{cases} \tag{2.49}$$

with

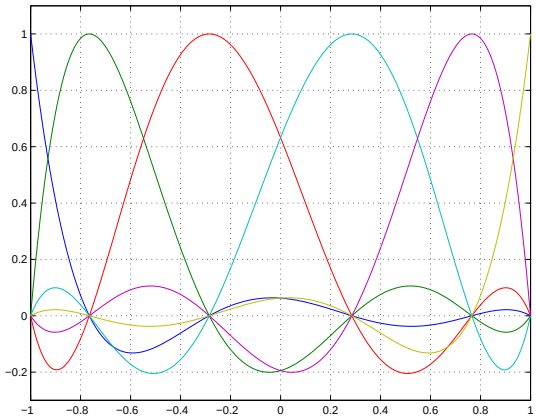
$$H_n = \begin{cases} 0 & \text{for } n = 0 \\ -4 \sum_{r=1}^{n} \frac{1}{2r-1} & \text{for } n \geq 1 \end{cases} \tag{2.50}$$

The drawback of the Chebychev SEM is the non-diagonal structure of the global mass matrix. Its inversion is therefore no longer trivial and the numerical scheme for integration in time gets more complicated. The explicit FD scheme in Equation (2.34) can no longer be used. Priolo & Seriani [1991] use and Newmark-type central difference scheme, for 1-D simulations. This is an implicit two-step scheme, conditionally stable and second order accurate.

Following this theoretical introduction the focus in the next section will be on the implementation of these mathematical constructs into the program code. Furthermore some more definitions, like those of the connectivity-matrix and the assembly process will be given, which were omitted so far. In these explanations the same order is used, in which the several stages have to be carried out in the program. Thus it is possible to successively understand the flowpath of the code. Several topics concerning programming as for example the initialisation, reading of parameters into the program or the construction of certain source signals are left out in this thesis as they are not relevant for understanding the SEM. Neither will the FD scheme for the integration in the time domain be explained in more detail. Information on these topics can be found in the standard literature on numerical methods for differential equations.
The parameter file of the 1-D Fortran code is given in appendix A.2, where it is pointed out which parameters can be changed in the program and why this may be of relevance.

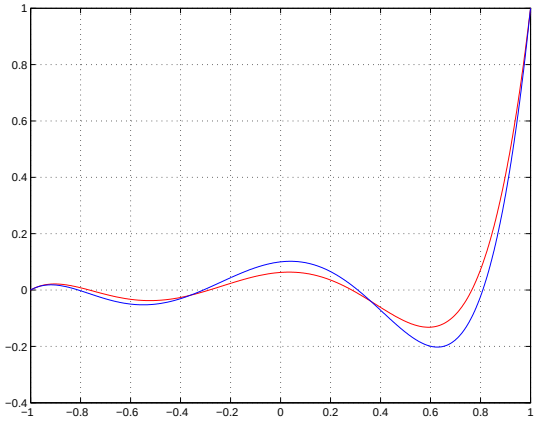


(a) Lagrangian interpolators of order $N = 5$ based on Chebychev polynomials.



(b) Lagrange polynomials of degree $N = 5$.

(c) Difference between corresponding Lagrange-Chebychev interpolators and Lagrange polynomials (both of order $N = 5$).



(d) Lagrange polynomial ℓ_6^5 (red) and Lagrange-Chebychev interpolator φ_6^5 (blue).

Figure 2.7: Comparison of interpolating polynomials: (a) All six Lagrangian interpolators of order $N = 5$ based on Chebychev polynomials. (b) Lagrange polynomials of degree $N = 5$. (c) Difference between corresponding Lagrange-Chebychev interpolators and Lagrange polynomials (both of order $N = 5$). (d) Small difference between the Lagrange polynomial ℓ_6^5 (red) and the Lagrangian interpolator derived with Chebychev polynomials. φ_6^5 (blue).

2.2 The Implementation of the Theoretical Formulations in the Program Code

2.2.1 Structure of the Program

- Initialization
- Reading of input files:
  - Parameter file (for details see description in Appendix A.2, page 137)
  - meshfile (if irregular mesh is used)
  - density and *mu* data files
- Preparations for the calculation of the stiffness and mass matrix
  - Calculation of collocation points and integration weights
  - Calculation of the first derivative of the Lagrange polynomials at the collocation points
- Calculation of shape functions
- Calculation of Jacobi-Matrices and Jacobians
- Generation of a non-equidistant x-vector for run-time plotting of the displacement field (Gauss-Lobatto-Legendre (GLL) points of each element are not evenly distributed inside the interval [-1,1])
- Calculation of the time step dt depending on the stability criterion
- Generation of the source signal (e.g. delta-peak or ricker-wavelet) or reading of initial displacement field data
- Calculation of the elemental stiffness matrices (skipped, if calculation of forces in time loop is preferred)
- Calculation of the elemental mass matrices
- Generation of the "connectivity-matrix" for the 1-D case
- Assembly of the global matrices $\mathbf{M}$ and if used $\mathbf{K}$
- setup of receivers
- Inversion of the mass-matrix
- Time-Loop: Integration in the time domain (explicit scheme)

  - Calculation of forces (if used)
  - implementation of boundary conditions
  - Calculation of displacement $\mathbf{u}_{t_{i+1}}$ at the next time step

- writing seismograms to output files

2.2.2 Preparations for the Calculation of the Stiffness and Mass Matrix

These Preparations comprise mainly the calculation of the appropriate collocation points ξ_i in the interval $[-1, 1]$ belonging to the associated order N and, based on this, the computation of the derivation of the Lagrange polynomials at the above gridnodes. Moreover the weights ω_i needed for the GLL quadrature are determined.

Calculation of the Collocation Points and Quadrature Weights

The collocation points of the GLL quadrature, used for the numerical integration and also for the Lagrange type interpolation, are defined as the roots of the first derivative of a Legendre polynomial P_N of degree N (ABRAMOWITZ & STEGUN [1984]). The Definition of the Legendre polynomials can be found in BRONSTEIN *et al.* [1999], page 507:

$$P_N(x) = \frac{1}{2^N N!} \frac{d^N}{d\xi^N} (\xi^2 - 1)^N \tag{2.51}$$

The weights of the Gauss-Lobatto-Legendre quadrature are obtained by (ABRAMOWITZ & STEGUN [1984]):

$$\begin{aligned} \omega_i &= \frac{2}{N(N+1)[P_N(\xi_i)]^2} \qquad (\xi_i \neq \pm 1) \\ \omega_i &= \frac{2}{N(N+1)} \qquad\qquad (\xi_i = \pm 1) \end{aligned} \tag{2.52}$$

In the program code the Legendre polynomials needed for the calculation of the collocation points and these points themselves are computed by the use of special FORTRAN subroutines published by FUNARO [1993]. Short descriptions of the underlying mathematics together with the subroutine code can be found there. The calculation of the Legendre polynomials in these subroutines is done using recursion formulae, which can also be obtained from BRONSTEIN *et al.* [1999],

where they have the following form:

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ P_n(x) &= \frac{1}{n}[(2n-1)\ x\ P_{n-1}(x) - (n-1)P_{n-2}(x)] \qquad n \geq 2 \end{aligned} \tag{2.53}$$

$$\begin{aligned} P_0^{'}(x) &= 0 \\ P_1^{'}(x) &= 1 \\ P_n^{'}(x) &= \frac{1}{n}[(2n-1)(x\ P_{n-1}'(x) + P_{n-1}(x)) - (n-1)P_{n-2}'(x)] \qquad n \geq 2 \end{aligned} \tag{2.54}$$

$$\begin{aligned} P_0''(x) &= 0 \\ P_1''(x) &= 0 \\ P_n''(x) &= \frac{1}{n}[(2n-1)(x\ P_{n-1}''(x) + 2P_{n-1}'(x)) - (n-1)P_{n-2}''(x)] \qquad n \geq 2 \end{aligned} \tag{2.55}$$

Calculation of the First Derivative of the Lagrange Polynomials at the Collocation Points

The calculation of the first derivatives of the Lagrange polynomials is also done with another subroutine of Funaro [1993], which uses the results of the above subroutine for Legendre polynomials. In this routine the derivates of all $N+1$ Lagrange polynomials ℓ_i are only calculated at the points of interpolation ξ_i, which then are stored in a matrix in such a way that in each column i are all the values of the derivative of one particular polynomial i at all nodes. This means that the number of the columns is equivalent to the index of the polynomial. The derivatives are obtained by applying the following formula (see Funaro [1993]):

$$\ell^{'}(\xi_i) = \sum_{j=1}^{N} \tilde{d}_{ij}^{\,(1)} \ell(\xi_j) \tag{2.56}$$

with

$$\tilde{d}_{ij}^{\,(1)} = \begin{cases} -\frac{1}{4}N(N+1) & \text{if } i = j = 0 \\ \frac{P_N(\xi_i)}{P_N(\xi_j)} \frac{1}{\xi_i - \xi_j} & \text{if } 0 \leq i \leq N,\ 0 \leq j \leq N,\ i \neq j \\ 0 & \text{if } 1 \leq i = j \leq N-1 \\ \frac{1}{4}N(N+1) & \text{if } i = j = N \end{cases} \tag{2.57}$$

Now we have accomplished all necessary preparations we need on the one hand to construct an adequate x-vector, which we will use to plot the simulated displacements during run-time. On the other hand we make use of these calculations when generating the elemental stiffness and mass matrices.

2.2.3 Implementation of the Domain Decomposition and Definition of the Mesh in 1-D

To allow for the fact that the integration points are not evenly spaced inside each element, one will have to generate an appropriate x-vector, i.e. the distances $\xi_{i+1} - \xi_i$ repeat every N elements of the vector (n_e times). This means appending element after element with all its GLL nodes. Figure 2.8 shows an example of such a vector with $N = 8$ and $n_e = 3$:

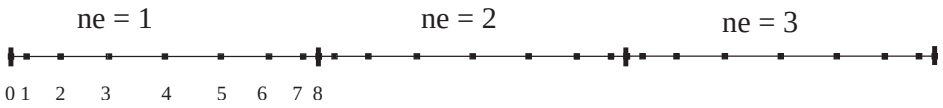


Figure 2.8: Illustration of a x-vector with non equidistant collocation points of the Gauss-Lobatto-Legendre quadratur for $N = 8$. Three elements are shown here.

2.2.4 Calculation of Time Step dt using the Stability Criterion

Corresponding to CAPDEVILLE [2000] the stability criterion of the SEM for the one-dimensional wave propagation in a homogeneous medium is:

$$dt \leq C\frac{dx_{min}}{v_{max}} = C\frac{dx_{min}}{\alpha} \qquad \alpha \text{ is the wave velocity in 1-D} \tag{2.58}$$

Here dx_{min} is the smallest distance (physical coordinates) between two points of the mesh. When using elements with different lengths this corresponds to the difference between nodes $i = 0$ and $i = 1$ of the smallest element as the distances between collocation points of one element get smaller at the border. C is the Courant number, which was determined empirically. CAPDEVILLE [2000] gives a value of around 0.84 for 1-D. This value could be verified with the code of the present work.

2.2.5 Getting the Elemental Stiffness and Mass Matrices

We have seen that one obtains two matrices $\mathbf{K}^e$ und $\mathbf{M}^e$ of the form $(N+1, N+1)$ for all n_e elements. These matrices contain all the contributions of the collo-

cation points to the corresponding global matrices. The derivatives $\ell_i'(\xi_j)$ of the Lagrange interpolators needed for the calculation of the elemental stiffness matrix have already been computed in Section 2.2.2. The elements K_{ij}^e of the elemental stiffness matrices then are the sum over the integrating points of two derivatives i und j multiplied by the corresponding weights an the inverse of the Jacobi-Matrix elements (see Eq. 2.30).

It was shown in the previous section (Sec. 2.1) that the property of the Lagrange polynomials of Equation (2.22) leads to the very important diagonal structure of the elemental mass matrices. Therefore the inversion of the global mass matrix, which is thus also diagonal, is now very easy. Corresponding to Equation (2.28), the diagonal elements M_{ii}^e of the elemental mass matrix contain only the weight ω_i of the i-th collocation point multiplied by the associated Jacobian $\mathcal{J}_i$ and density ρ_i. As mentioned earlier only the non-zero elements of the mass matrix are stored in a vector, which affects the mathematical matrix multiplication. This multiplication is easily done in Fortran, which performs an elementwise multiplication when using the usual $*$-operator (in contrast to matlab). The result of this $*$-operation is therefore a vector itself. An equivalent vector would be obtained by a real matrix - vector multiplication (in the mathematical sense) using a matrix in which only the diagonal elements are $\neq 0$. This is exactly the case with the mass matrix $\mathbf{M}$ and similarily $\mathbf{M^{-1}}$. Doing so leads to much lower memory requirements of the order of 40 to 50 % in 1-D simulations, which may seem uninteresting in this case but is absolutely inevitable for 3-D calculations.

2.2.6 Generation of the Connectivity Matrix in the 1-D Case for the Assembly of the Global System

In this section the generation and use of the connectivity-matrix is demonstrated, which is the key feature for the assembly of global variables for the simple 1-D case.

The Connectivity-Matrix

The connectivity-matrix contains all information of the 1-D grid concerning the association of the global nodes to the elements. This can be regarded as a kind of transformation of the element and local numbering to a complete global numbering of the points of the grid. The connectivity-matrix is structured in a way, that the index of columns equals the index of the associated element and the corresponding rows are filled with the global number of the collocation points. In the global numbering the first node is denoted by 1 (contrary to the local numbering of the GLL points inside the elements, which starts from 0). In the

easiest case of $N = 1$ the connectivity-matrix will look like:

$$\mathbf{C} = \begin{pmatrix} 1 & 2 & \dots & n_g - 1 \\ 2 & 3 & \dots & n_g \end{pmatrix} \tag{2.59}$$

n_g is the overall number of global points in the mesh: $n_g = N \cdot n_e + 1$

When the overall number of Spectral Elements is kept the same but a higher order of the interpolating functions is used, the number of columns remains the same while the number of rows becomes $N+1$. Using for example $N = 5$, $\mathbf{C}$ will become:

$$\mathbf{C} = \begin{pmatrix} 1 & 6 & [11 = (3-1)N+1 & \dots & [(n_e-1)N+1] \\ 2 & 7 & = (j-1)N+1] & & [(n_e-1)N+2] \\ 3 & 8 & & & \\ 4 & 9 & \vdots & & \\ 5 & 10 & & & \\ 6 & [11 = 2N+1] & \dots & \dots & [(n_e-1)N+N+1] \end{pmatrix} \tag{2.60}$$

2.2.7 Assembly of the Global Linear System using the Stiffness Matrix

After the connectivity-matrix is generated the information stored in its elements can be used to assemble the elemental vectors and matrices. This is done by assigning the elements of the elemental matrices (resp. vectors) to the elements of the global fields in a convenient manner. The correlation can either be expressed in a program assignment (see Eq. 2.61) or in words as done in SCHWARZ [1984]:

> *„Ganz allgemein beschreibt sich dieser Kompilationsprozess wie folgt: Steht in der Liste der Nummern der Knotenvariablen an der Position j die Nummer l und an der Position k die Nummer m, so ist beispielsweise der Wert des Elementes $k^{(e)}_{jk}$ der Elementsteifigkeitsmatrix $\mathbf{K_e}$ zum Element k_{lm} der Gesamtsteifigkeitsmatrix $\mathbf{K}$ zu addieren. Ferner ist die Komponente $b^{(e)}_j$ des Elementvektors $\mathbf{b_e}$ zur Komponente b_l des Gesamtvektors $\mathbf{b}$ zu addieren.“*

*"In general this compilation can be described as follows:
If the list of node-variable numbers contains the number l at position j and the number m at position k then for example the value of the element $k_{jk}^{(e)}$ of the elemental stiffness matrix $\mathbf{K_e}$ has to be added to the element k_{lm} of the global stiffness matrix $\mathbf{K}$. Furthermore, the component $b_j^{(e)}$ of the elemental vector $\mathbf{b_e}$ has to be added to the component b_l of the global vector $\mathbf{b}$."*

In the program code this relationship will have the following form which should be read as an program assignement rather than a correct mathematical statement:

$$\begin{aligned} \mathbf{K}(C_{ki}, C_{ji}) &= \mathbf{K}(C_{ki}, C_{ji}) + K_{kj}^{(i)} \\ \mathbf{M}(C_{ji}) &= \mathbf{M}(C_{ji}) + M_j^{(i)} \end{aligned} \tag{2.61}$$

i is the index of element numbering, j and k are indices of the order N.

The first relation appears in three loops of indices i, j, k with $i = 1, \ldots, n_e$ and $j, k = 0, \ldots, N$. For the second one only two loops are needed with i and j being the same as above.
A possible way of expressing this relation in mathematically correct form is shown below:

$$\mathbf{K}(C_{ki}, C_{ji}) = K_{kj}^{(i)} + \sum_{l=1}^{n_e} \sum_{m=0}^{N} \sum_{n=0}^{N} K_{mn}^{(l)} \; \delta_{C_{ki}C_{ml}} \; \delta_{C_{ji}C_{nl}} \tag{2.62}$$

$$\mathbf{M}(C_{ji}) = M_j^{(i)} + \sum_{l=1}^{n_e} \sum_{k=0}^{N} M_k^{(l)} \; \delta_{C_{ji}C_{kl}} \tag{2.63}$$

$$\forall \; i = 0, \ldots, N; \; j = 1, \ldots, n_e$$

In Sections 2.1.5 and 2.1.6 it was pointed out that the elemental matrices $\mathbf{M}^e$ and $\mathbf{K}^e$ have the form $(N+1, N+1)$, where only the diagonal elements of $\mathbf{M}^e$ are non-zero. The process of assembling the global fields now leads to a structure of (n_g, n_g) matrices. In the case of the mass matrix being stored in a vector, the second relation of Equation (2.61) applies. In order to better understand this

important procedure it is illustrated in pictorial form for a simple case of $N = 2$ and $n_e = 3$ in Equation (2.64):

$$\mathbf{M} = \begin{pmatrix} \star \\ \star \\ \star \\ \\ \\ \end{pmatrix} + \begin{pmatrix} \\ \\ \circ \\ \circ \\ \circ \end{pmatrix} + \begin{pmatrix} \\ \\ \\ \diamond \\ \diamond \\ \diamond \end{pmatrix} = \begin{pmatrix} \star \\ \star \\ \star + \circ \\ \circ \\ \circ + \diamond \\ \diamond \\ \diamond \end{pmatrix} \tag{2.64}$$

The elemental vectors appear to be expanded virtually to the length n_g of the global ones. From this it can clearly be seen that the elements $\mathbf{M}(3)$ and $\mathbf{M}(5)$ of the global mass "vector" contain the sum of the values of the corresponding spectral elements. These matrix elements thus comprise the contributions of two spectral elements as they are associated with the nodes at the border of these elements (i.e. these nodes are shared between two elements).
The assembly of the global stiffness matrix $\mathbf{K}$ for the same N and n_e as above then gives:

$$\mathbf{K} = \begin{pmatrix} K^{(1)}_{1,1} & K^{(1)}_{1,2} & K^{(1)}_{1,3} & & & & \\ K^{(1)}_{2,1} & K^{(1)}_{2,2} & K^{(1)}_{2,3} & & & \mathbf{0} & \\ K^{(1)}_{3,1} & K^{(1)}_{3,2} & \left[K^{(1)}_{3,3} + K^{(2)}_{1,1}\right] & K^{(2)}_{1,2} & K^{(2)}_{1,3} & & \\ & & K^{(2)}_{2,1} & K^{(2)}_{2,2} & K^{(2)}_{2,3} & & \\ & & K^{(2)}_{3,1} & K^{(2)}_{3,2} & \left[K^{(2)}_{3,3} + K^{(3)}_{1,1}\right] & K^{(3)}_{1,2} & K^{(3)}_{1,3} \\ & \mathbf{0} & & & K^{(3)}_{2,1} & K^{(3)}_{2,2} & K^{(3)}_{2,3} \\ & & & & K^{(3)}_{3,1} & K^{(3)}_{3,2} & K^{(3)}_{3,3} \end{pmatrix} \tag{2.65}$$

The stiffness matrix therefore has a certain bandwith, which is $2 \cdot N + 1$, with some additional elements being zero within this band. A very good introduction to the assembly and connectivity is presented in the first chapter of the book on FEM by ZIENKIEWICZ & TAYLOR [2000].

The Stiffness Matrix in practical use

We can see from Equation (2.65) that only a small band around the diagonal of the global stiffness matrix is occupied and that most of the matrix elements are 0. Therefore it would be very awkward to store all of those latter elements in the computer memory which are not of any use in the computations as usually the number of elements by far exceeds the order N of interpolating polynomials. In addition, most of the computations concerning Equation (2.32) would include multiplications with 0, which is a nonacceptable waste of CPU resources. In

order to circumvent this problem we have to find a way of reducing the memory requirements of the stiffness matrix whithout changing the calculations.
One way of reducing the memory is to compress the rows of the stiffness matrix. From Equation (2.32) we see that every row of the stiffness matrix is multiplied with the column vector $\mathbf{u}$. f_i is hereby the i-th element of the vector of internal forces $\mathbf{f_{internal}}$ (the same applies to Eq. 2.68).

$$f_i = \sum_{j=1}^{n_g} K_{ij}\ u_j\ , \qquad \forall\ i = 1, \ldots, n_g \tag{2.66}$$

i is the index of rows, j is the index of columns.

Thus every line of the stiffness matrix is independent from the others and can be seen as a single row vector.
As the elements containing zero do not change the result of this vector multiplication we can omit most of them in every row and reduce those to the size $2 \cdot N + 1$, which is the bandwith of the stiffness matrix. Only a few zeros remain in the truncated matrix.

$$\mathbf{K}= \begin{pmatrix} \ddots & & & & & & & & \\ & K_{1,1}^{(i-1)} & K_{1,2}^{(i-1)} & K_{1,3}^{(i-1)} & & & & & \\ & K_{2,1}^{(i-1)} & K_{2,2}^{(i-1)} & K_{2,3}^{(i-1)} & & & \mathbf{0} & & \\ & K_{3,1}^{(i-1)} & K_{3,2}^{(i-1)} & [K_{3,3}^{(i-1)} + K_{1,1}^{(i)}] & K_{1,2}^{(i)} & K_{1,3}^{(i)} & & & \\ & & \longleftarrow & K_{2,1}^{(i)} & K_{2,2}^{(i)} & K_{2,3}^{(i)} & & & \\ & & \longleftarrow & K_{3,1}^{(i)} & K_{3,2}^{(i)} & [K_{3,3}^{(i)} + K_{1,1}^{(i+1)}] & K_{1,2}^{(i+1)} & K_{1,3}^{(i+1)} & \\ & & \mathbf{0} & & \longleftarrow & K_{2,1}^{(i+1)} & K_{2,2}^{(i+1)} & K_{2,3}^{(i+1)} & \\ & & & & \longleftarrow & K_{3,1}^{(i+1)} & K_{3,2}^{(i+1)} & K_{3,3}^{(i+1)} & \\ & & & & & & & & \ddots \end{pmatrix}$$

$$\Big\| \bigvee$$

$$\mathbf{K} = \begin{pmatrix} \vdots & & & & \\ K_{1,1}^{(i-1)} & K_{1,2}^{(i-1)} & K_{1,3}^{(i-1)} & 0 & 0 \\ 0 & K_{2,1}^{(i-1)} & K_{2,2}^{(i-1)} & K_{2,3}^{(i-1)} & 0 \\ K_{3,1}^{(i-1)} & K_{3,2}^{(i-1)} & [K_{3,3}^{(i-1)} + K_{1,1}^{(i)}] & K_{1,2}^{(i)} & K_{1,3}^{(i)} \\ 0 & K_{2,1}^{(i)} & K_{2,2}^{(i)} & K_{2,3}^{(i)} & 0 \\ K_{3,1}^{(i)} & K_{3,2}^{(i)} & [K_{3,3}^{(i)} + K_{1,1}^{(i+1)}] & K_{1,2}^{(i+1)} & K_{1,3}^{(i+1)} \\ 0 & K_{2,1}^{(i+1)} & K_{2,2}^{(i+1)} & K_{2,3}^{(i+1)} & 0 \\ 0 & 0 & K_{3,1}^{(i+1)} & K_{3,2}^{(i+1)} & K_{3,3}^{(i+1)} \\ \vdots & & & & \end{pmatrix} \tag{2.67}$$

The last thing we have to do now is to perform the multiplication with the displacement vector in a correct manner, i.e. multiplying the rows of the new stiffness matrix with those elements of $\mathbf{u}$ given by the full matrix-vector multiplication of Equation (2.32).

$$f_i = \sum_{j=-N}^{N} K_{i(j+N)}\ u_{(i+j)}\ , \qquad \forall\ i = 1, \ldots, n_g \tag{2.68}$$

2.2.8 Assembly Process by Calculation of total Forces instead of using the Stiffness Matrix

As explained in Section 2.1.7, the easier way to obtain the global matrix equation is to compute the forces acting on every node of the elements and later summing these contributions on nodes that belong to more than one element. This latter stage is the same as the assembly of the vector containing the global mass matrix elements, which was discussed in the previous section.

At first only the interior forces of the medium are considered, omitting sources, which will later be added to the total forces acting on the grid points.
According to the weak formulation for one element (Eq. 2.11) the internal forces inside one element are computed by the second integral of this equation. Previously, this led to the stiffness matrix. As we want to obtain forces, we start to derive the stresses acting in the element. Therefore we have to get the strain at all the collocation points of one element at first:

$$\begin{aligned} \frac{\partial u_i}{\partial x} &= \frac{\partial u_i}{\partial \xi}\ \frac{\partial \xi_i}{\partial x} \\ &= \sum_j u_j\ \ell_j'(\xi_i)\ \cdot\ \mathbf{J_i^{-1}} \end{aligned} \tag{2.69}$$

Here we have to account for the mapping again (the inverse mapping is applied) by multiplying with the Jacobi-Matrix of the transformation from global to local coordinates. In addition, the interpolation is done using the well-known Lagrange interpolation scheme. The values of the displacement at the points of interpolation are derived using the connectivity-matrix. The displacement u at point ξ_i of element j is obtained using:

$$u^j(\xi_i) = \mathbf{u}(C_{ij}) \tag{2.70}$$

Now, by applying Hooke's Law, we can get the stress at one node:

$$\sigma_i \;=\; \mu_i \, \frac{\partial u_i}{\partial x} \tag{2.71}$$

To finally determine the internal force we have to insert the stress into the integral, interpolate the test function v, again apply the inverse mapping, and account for this by multiplication with $\mathbf{J}_k^{-1}$. Then, integrating numerically by means of the GLL integration quadraturei leads to:

$$\begin{aligned}
\mathbf{f}_{int}^e &= \int_{\Omega^e} \nabla v \; \sigma \\
&= \int_{-1}^{1} \nabla v \; \sigma \; \mathcal{J} \; \partial\xi \\
&= \sum_k [\sum_j \ell_j'(\xi_k)] \; {}^{\mathbf{i}}\mathbf{J}_k \; \sigma_k \; \mathcal{J}_k \\
&= \sum_j \sum_k \ell_j'(\xi_k) \; {}^{\mathbf{i}}\mathbf{J}_k \; \sigma_k \; \mathcal{J}_k
\end{aligned} \tag{2.72}$$

And, as the interpolation scheme for the force at one node j gives

$$f(\xi_j) = f_j = \sum_i f_i \delta_{ij} = \sum_i f_i \; \ell_i(\xi_j) \; , \tag{2.73}$$

we receive the expression for the forces at each node of one element as:

$$f_{int}(\xi_j) = \sum_k \ell_j'(\xi_k) \; {}^{\mathbf{i}}\mathbf{J}_k \; \sigma_k \; \mathcal{J}_k, \qquad\qquad j = 0, \ldots, N \tag{2.74}$$

The stresses are computed separately, despite of being evaluated under the sum over k in Equation (2.74). Thus the loops used to calculate the sums over the $N + 1$ collocation points are not nested.

External Forces: Contributions of the Sources

For the calculation of the global force vector, we already determined the internal forces. Now we just have to add the forces of external excitations of the medium. We only have to identify the GLL-points on which we impose those sources. To do so, we test which elements contain sources and look for the closest collocation point. Then without further normalisation we can add the force associated to the source location to the internal forces at this collocation point. At the end we have the total forces at all nodes of each element.

$$f^j_{total}(\xi_i) = f^j_{int}(\xi_i) + f^j_{source}(\xi_i)\ \delta_{s_e j}\ \delta_{si}\ , \qquad \forall\ i = 0, \ldots, N;\ j = 1, \ldots, n_e \quad (2.75)$$

The two Kronecker-deltas symbolize the fact that only elements containing sources (i.e element j is an "source-element", $j \in s_e$) and only those GLL-points closest to the source location ($i = s$, where s denotes the GLL-point corresponding to the source) have to be considered. It is obvious, that this is not preferable as we take a loss of accuracy in locating the source. Another possibilty of implementing sources is to use stresses applied on the medium. These stresses only have to be added to the internal stresses of the medium. The resulting stresses then are treated as shown in Equation (2.72). This is the most convenient way for 3-D applications, as will be shown in Section 3.1.8.

Construction of the Global Force Vector f

The elements of the global force vector can then be maintained by summing the forces computed on the elements according to the information stored in the connectivity-matrix. We get:

$$\mathbf{f}(C_{ij}) = f^j(\xi_i) + \sum_{l=1}^{n_e} \sum_{k=0}^{N} f^l(\xi_k)\ \delta_{C_{ij} C_{kl}}\ , \qquad \forall\ i = 0, \ldots, N;\ j = 1, \ldots, n_e \quad (2.76)$$

This means that only those forces are summed where $C_{ij} = C_{kl}$. After having assembled all variables and fields needed in the global linear system we can solve Equation (2.33), the reduced version of Equation (2.32), for the displacement $\mathbf{u}$:

$$\mathbf{M\ddot{u}} = \mathbf{f_{total}}$$

$$\mathbf{u_{t_{i+1}}} = dt^2\ (\mathbf{M^{-1}}\ \mathbf{f_{total}}) + 2\mathbf{u_{t_i}} - \mathbf{u_{t_{i-1}}} \quad (2.77)$$

The calculation of $\mathbf{u}$ in each time step thus comprises two steps, in contrast to the formulation using the stiffness matrix. First calculating the forces and afterwards multiplying by the mass matrix. Although the latter operation only consist of

one multiplication per time step[3], the first step is computationally expensive. All operations together take more CPU time than the calculations using the stiffness matrix. Thus, for 1-D applications the latter one is preferable, but as said before, for 2- or 3-D simulations it is no longer convenient.

[3]This is due to the diagonal mass matrix

Chapter 3

The Spectral Element Method for 3-D Seismic Wave Propagation

In Chapter 2 the reader was familiarized with the ideas of the SEM by introducing the theory and concepts of the method for one-dimensional wave propagation problems. As mentioned earlier, these considerations for 1-D can more or less be translated one-by-one to 3-D applications. Nevertheless, it is clear that we need some extensions to compensate for the additional dimensions. In this context the tensorisation of the interpolation as well as more complex meshing algorithms have to be mentioned. All this will be explained in this chapter by taking the same approach as in the 1-D case, but in a shorter and condensed way. Topics that do not change from 1-D to 3-D will also be repeated as some readers may want to read only this chapter about 3-D simulations.
In Section 3.1.2 some examples of meshes for 2- and 3-D models are shown, which were published in the last 10 years. In this way the reader can get an idea of the present state of the art. The SEM for 2-D simulations is self explanatory within this chapter, but the most important issues will additionally be presented for completeness and sometimes it will be more convenient to show figures of two-dimensional elements instead of three-dimensional ones.

Again, the content of the current chapter is mainly based on publications by Komatitsch [1997]; Komatitsch & Tromp [1999, 2002a,b]; Komatitsch & Vilotte [1998], but some others will be mentioned in addition as for example Komatitsch *et al.* [2001]. The latter discusses a special kind of spectral elements, with triangular shape. In the theory of the SEM for multidimensional problems we will recognise, that the method is originally designed for hexagonal elements.

3.1 The Mathematical Formulation of the Spectral Element Method in 3-D

Definition of the Mathematical Problem

As in the previous chapter we start with the definition of the mathematical problem we want to solve. Therefore we state the elastodynamic equation for three-dimensional wave propagation (after UDIAS [1999]):

$$\rho(\mathbf{x}) \cdot \frac{\partial^2}{\partial t^2} \mathbf{u}(\mathbf{x},t) \; - \; \nabla \cdot \; \boldsymbol{\sigma}(\mathbf{x},t) = \mathbf{f}(\mathbf{x},t) \tag{3.1}$$

Attenuation and anisotropy are not included here, but they both pose no problem to the SEM as stated in Chapter 2.
The individual terms are all volume forces as described in Equation (2.2). This formula can also be expressed in a shorter way using the sum convention over repeated indices (see UDIAS [1999] or AKI & RICHARDS [2002]):

$$\begin{aligned} \rho \; \frac{\partial^2 u_i}{\partial t^2} \; - \; \frac{\partial \sigma_{ij}}{\partial x_j} &= \; f_i \\ \rho \; \ddot{u}_i \; - \; \sigma_{ij,\,j} &= \; f_i \qquad\qquad \forall \; i = 1,2,3 \end{aligned} \tag{3.2}$$

Equation (3.2) is therefore written separately for all components of $\mathbf{u}$. The comma denotes a spatial derivative in the direction given by the following index.
The components of the stress tensor σ_{ij} can be computed using the general form of Hooke's Law:

$$\begin{aligned} \sigma_{ij} &= \; c_{ijkl} \; \varepsilon_{kl} \\ &= \; c_{ijkl} \; u_{k,\,l} \end{aligned} \tag{3.3}$$

In the same way as in Chapter 2, we will derive the variational form of the wave equation with three spatial coordinates. From that we can formulate the SEM by first dividing the Model Domain Ω_{n_d}[1] into hexagonal elements (meshing), then mapping them onto a standard interval, afterwards determining the matrices on elemental level and later for the global system (in the course of the assembly).

3.1.1 The Weak Formulation of the Elastodynamic Equation in 3-D

As we need the integrated form of the wave equation we start by multiplying Equation (3.1) with a time-independent test function $\mathbf{v}(\mathbf{x})$, which is now also a

[1] n_d is the number of dimensions and Ω_{n_d} is here defined as the n_d tensor product of one-dimensional domains Ω: $\Omega_{n_d} = \Omega \otimes \Omega \otimes \Omega$ for $n_d = 3$; $\otimes$ is the tensor product, for details see BRONSTEIN *et al.* [1999], page 263

vector. In the present formulation of Equation (3.1) the following considerations hold for any elastic medium including inhomogeneity and anisotropy. When attenuation shall be included the stresses have to be computed in a different way (see Komatitsch & Tromp [1999]). After multiplication with $\mathbf{v}$ we integrate over the whole domain and get:

$$\int_{\Omega_{n_d}} \mathbf{v}\ \rho\ \ddot{\mathbf{u}}\ d\mathbf{x}\ -\ \int_{\Omega_{n_d}} \mathbf{v}\ \nabla\cdot\boldsymbol{\sigma}\ d\mathbf{x} = \int_{\Omega_{n_d}} \mathbf{v}\ \mathbf{f}\ d\mathbf{x} \tag{3.4}$$

$\mathbf{u}$, $\mathbf{f}$, and $\boldsymbol{\sigma}$ are considered to depend on $\mathbf{x}$ and t in the following without further explicit declaration. $\mathbf{v}$ is implicitly considered to depent on $\mathbf{x}$ and ρ only on t. In addition Ω_{n_d} will be denoted $\boldsymbol{\Omega}$ from now on throughout the whole chapter.

If we now integrate the second term on the left side of Equation (2.6) by parts and denote the boundary of $\boldsymbol{\Omega}$ by Γ, we obtain:

$$\int_{\boldsymbol{\Omega}} \rho\ \mathbf{v}\ \ddot{\mathbf{u}}\ d\mathbf{x}\ -\ \int_{\Gamma} \mathbf{v}\ \nabla\cdot\boldsymbol{\sigma}\ d\mathbf{x}\ +\ \int_{\boldsymbol{\Omega}} \nabla\mathbf{v}\ \boldsymbol{\sigma}\ d\mathbf{x} = \int_{\boldsymbol{\Omega}} \mathbf{v}\ \mathbf{f}\ d\mathbf{x} \tag{3.5}$$

Now we apply the same stress conditions as we used for 1-D problems, namely the free surface or Neumann-conditions with stresses being zero at the boundaries:

$$\boldsymbol{\sigma}_{\Gamma} = 0 \tag{3.6}$$

Thus the integral over Γ vanishes, which leads to the final form of the weak formulation which we will use in the following:

$$\int_{\boldsymbol{\Omega}} \mathbf{v}\ \ddot{\mathbf{u}}\ d\mathbf{x} + \int_{\boldsymbol{\Omega}} \nabla\mathbf{v}\ \boldsymbol{\sigma}\ d\mathbf{x} = \int_{\boldsymbol{\Omega}} \mathbf{v}\ \mathbf{f}\ d\mathbf{x} \tag{3.7}$$

The application of other boundary conditions was shown for the 1-D case in Section 2.1.9, where also some references to papers on this topic were given. Details on boundary conditions for the 3-D case will not be presented in the course of this thesis.

Again we have the basis for the SEM with which we can start by first meshing the model. The domain decomposition is shown in the next subsection.

3.1.2 Domain Decomposition and Mapping Functions

As mentioned earlier, the model domain has to be divided into n_e elements. The idea behind methods using elements is to approximate the model as good as possible by first defining several subregions. These subregions may be for example a sediment basin overlaying some bedrock, or in a whole Earth model there may be the regions crust, mantle, outer core and inner core (see Komatitsch & Tromp [2002a]). This leads to interfaces inside the model that

exactly represent the interfaces in the geological model. When now discretising each subregion, the interfaces as well as the free surface (with topography) or artifical boundaries (where absorbing boundary conditions shall be applied) are represented by the faces of elements, which can be curved. How to obtain these curvatures and why the SEM almost exclusively uses hexagonal elements will be explained in the following.

Figure 3.1 illustrates the idea of meshing a 2-D structure with curved interfaces. In the middle layer a special kind of size doubling is used to ensure a conforming mesh. This property is important to obtain a diagonal mass matrix in 2- and 3-D SEM simulations. This doubling algorithm was used by KOMATITSCH & TROMP [2002a,b] to mesh the entire globe.
Figure 3.3 shows a simple example of 3-D meshing of a cube divided into several differently shaped hexaedral elements. Figures 3.4 and 3.5 show several examples of meshing geological structures.

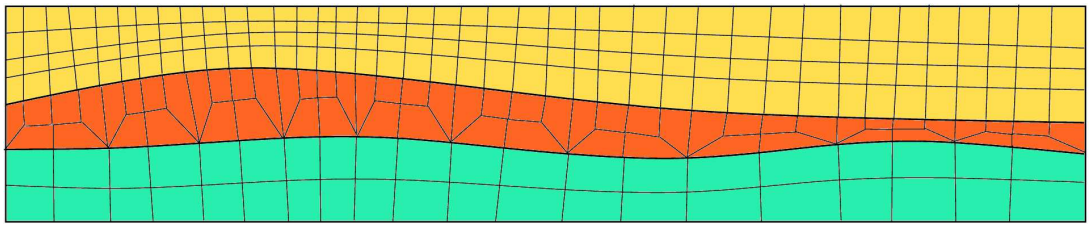


Figure 3.1: Two dimensional mesh of curved structures using size doubling in the middle layer.

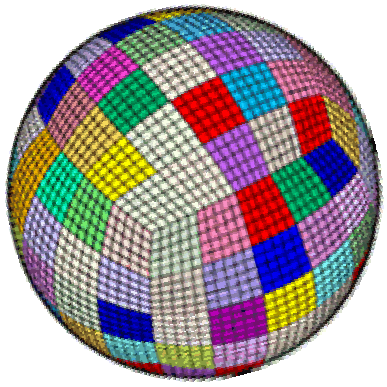


Figure 3.2: A "cubed sphere" mesh of the globe. (Taken from KOMATITSCH & TROMP [2002a]).

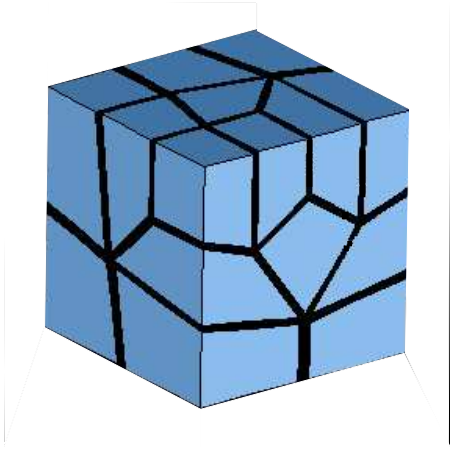


Figure 3.3: Illustration of a three-dimensional meshing of a cube. (Taken from MÜLLER-HANNEMANN [2000]).

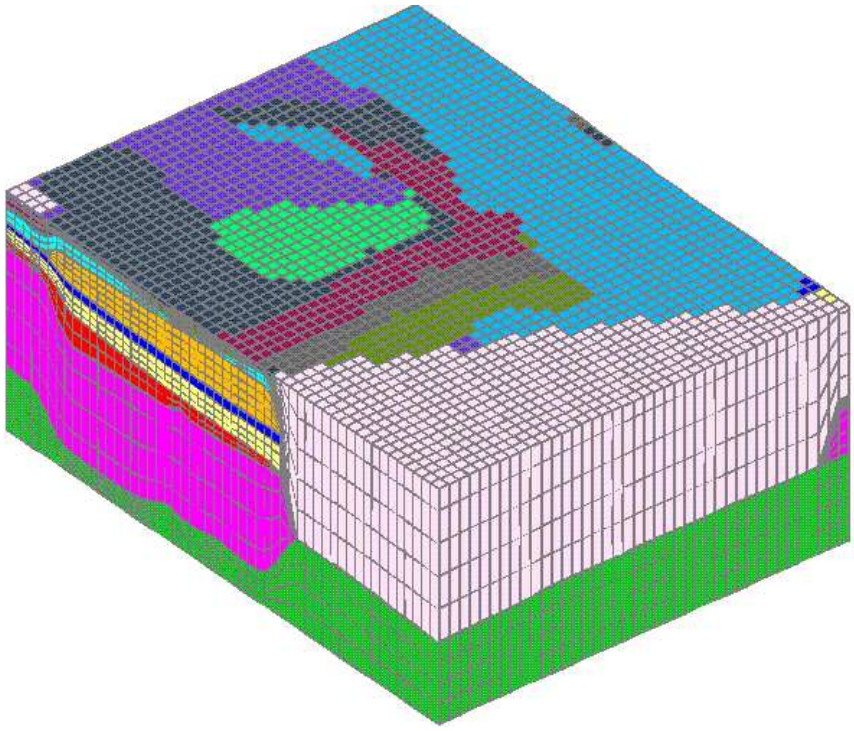


Figure 3.4: Meshing of geological structures in three-dimensional models based on hexaedra. (Taken from Gable & Cherry [2000]).

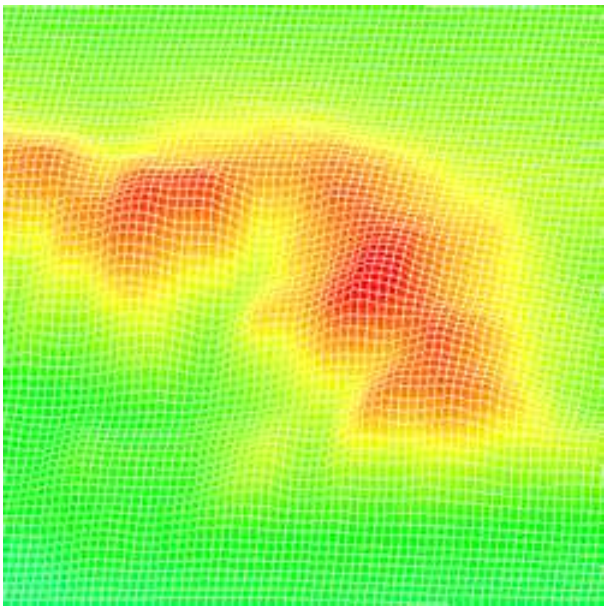


Figure 3.5: Approximation of curved topography by hexaedrons. (Taken from Komatitsch *et al.* [2003a]).

After dividing the model domain into subdomains $\mathbf{\Omega}^e$, the integrations of Equation (3.7) can be performed independently on each of these subdomains. Thus we can state the weak formulation separately for all n_e elements.

$$\int_{\mathbf{\Omega}^e} v\ \ddot{u}\ dx + \int_{\mathbf{\Omega}^e} \nabla v\ \mu\ \nabla u\ dx = \int_{\mathbf{\Omega}^e} v\ f\ dx \tag{3.8}$$

for $e = 1, ..., n_e$.

Mapping Functions - Transformation of Physical Coordinates onto a Standard Interval

In Chapter 2 we proceeded by defining a coordinate transformation for each element in such way that all elements had locally defined coordinates $\xi : \xi \in [-1, 1] = \Lambda$. This mapping allowed us to handle all subdomains in the same manner, which simplified calculations a lot.
Doing the same in 3-D is also convenient and straight forward. Each element domain $\mathbf{\Omega}^e$ is mapped onto a three-dimensional reference cube $\Lambda_{n_d} = \mathbf{\Lambda} = \Lambda \otimes \Lambda \otimes \Lambda$, where Λ is again the one-dimensional standard interval $[-1, 1]$. In the most general case the mapping functions $\boldsymbol{\mathcal{F}}_e :\ \mathbf{\Lambda} \rightarrow \mathbf{\Omega}_e$ of the elements are defined by:

$$\begin{aligned} \boldsymbol{\mathcal{F}}_e(\boldsymbol{\xi}) \ &= \ \mathbf{x}^e(\boldsymbol{\xi}) = \sum_{a=1}^{n_a} N_a(\boldsymbol{\xi})\ \mathbf{x}_a^e \\ \mathbf{x}^e(\xi, \eta, \zeta) &= \sum_{a=1}^{n_a} N_a(\xi, \eta, \zeta)\ \mathbf{x}_a^e \end{aligned} \tag{3.9}$$

$\mathbf{x}_a^e$ are the n_a anchor points of the e-th element. Depending on the choice of the order N of Lagrange polynomials, a 3-D element can either contain 8 nodes ($N = 1$) at each of its corners or 27 nodes ($N = 2$) (see Fig. 3.8). The first have straight edges and faces, whereas the latter can have curved ones.
Sometimes, when using curved elements, it is possible to omit the anchor nodes in the middle of faces (white quadrangles in Fig. 3.8) and the middle of the volume (white triangle in Fig. 3.8).
Figure 3.6 illustrates the mapping considering as an example one straight and one curved 2-D element, which are both transferred to the reference square $[-1, 1] \otimes [-1, 1]$, only the number of anchor points is differing.
The shape functions N_a are n_d-products of Lagrangian polynomials of degree either $N = 1$ for straight edged elements or $N = 2$ for curved elements. These polynomials were shown in Figure 2.2 and 2.3.
The shape functions for a 2-D four node element based on Lagrange polynomials

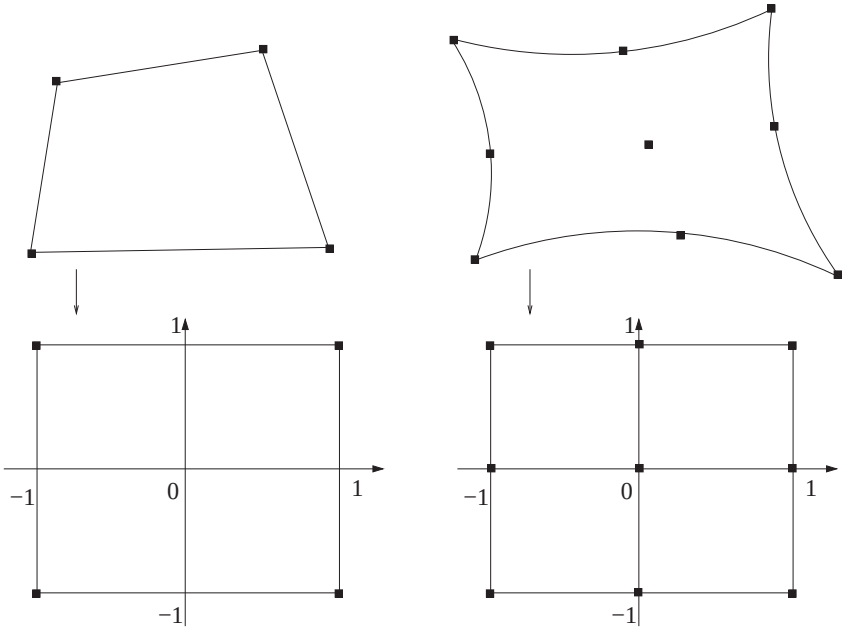


Figure 3.6: Mapping of 2-D elements on the reference square $\Omega_2 = [-1,1]\otimes[-1,1]$. Left: straight element with four anchor nodes. Right: curved element with nine anchor nodes.

of degree $N = 1$ are as follows:

$$\begin{aligned} N_1(\xi,\eta) &= \ell_0^1(\xi)\ \ell_0^1(\eta) & N_2(\xi,\eta) &= \ell_1^1(\xi)\ \ell_0^1(\eta) \\ N_3(\xi,\eta) &= \ell_0^1(\xi)\ \ell_1^1(\eta) & N_4(\xi,\eta) &= \ell_1^1(\xi)\ \ell_1^1(\eta) \end{aligned} \tag{3.10}$$

For a curved 9 node 2-D element the shape functions are (Lagrange polynomials have degree $N = 2$ here):

$$\begin{aligned} N_1(\xi,\eta) &= \ell_0^2(\xi)\ \ell_0^2(\eta) & N_2(\xi,\eta) &= \ell_0^2(\xi)\ \ell_1^2(\eta) \\ N_3(\xi,\eta) &= \ell_1^2(\xi)\ \ell_0^2(\eta) & N_4(\xi,\eta) &= \ell_1^2(\xi)\ \ell_1^2(\eta) \\ N_5(\xi,\eta) &= \ell_1^2(\xi)\ \ell_2^2(\eta) & N_6(\xi,\eta) &= \ell_2^2(\xi)\ \ell_1^2(\eta) \\ N_7(\xi,\eta) &= \ell_0^2(\xi)\ \ell_2^2(\eta) & N_8(\xi,\eta) &= \ell_2^2(\xi)\ \ell_0^2(\eta) \\ N_9(\xi,\eta) &= \ell_2^2(\xi)\ \ell_2^2(\eta) \end{aligned} \tag{3.11}$$

The shape functions of three-dimensional elements then are triple products of the corresponding Lagrange polynomials.

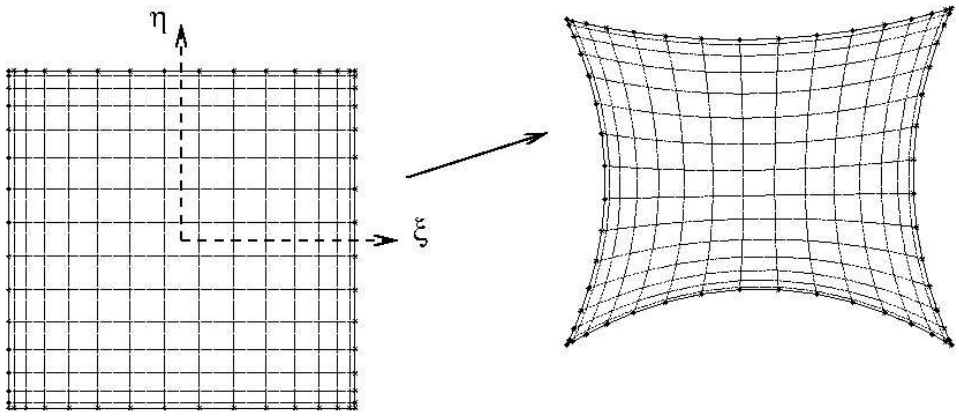


Figure 3.7: Mapping of a 2-D elements from the reference square. The element is shown with the Gauss-Lobatto-Legendre collocation points of integration for the order $N = 8$. (This figure is taken from KOMATITSCH [1997], page 50).

For eight node elements:

$$\begin{aligned}
N_1(\xi,\eta,\zeta) &= \ell_0^1(\xi)\,\ell_0^1(\eta)\,\ell_0^1(\zeta) & N_2(\xi,\eta,\zeta) &= \ell_1^1(\xi)\,\ell_0^1(\eta)\,\ell_0^1(\zeta)\\
N_3(\xi,\eta,\zeta) &= \ell_0^1(\xi)\,\ell_1^1(\eta)\,\ell_0^1(\zeta) & N_4(\xi,\eta,\zeta) &= \ell_0^1(\xi)\,\ell_0^1(\eta)\,\ell_1^1(\zeta)\\
N_5(\xi,\eta,\zeta) &= \ell_1^1(\xi)\,\ell_1^1(\eta)\,\ell_0^1(\zeta) & N_6(\xi,\eta,\zeta) &= \ell_0^1(\xi)\,\ell_1^1(\eta)\,\ell_1^1(\zeta)\\
N_7(\xi,\eta,\zeta) &= \ell_1^1(\xi)\,\ell_0^1(\eta)\,\ell_1^1(\zeta) & N_8(\xi,\eta,\zeta) &= \ell_1^1(\xi)\,\ell_1^1(\eta)\,\ell_1^1(\zeta)
\end{aligned} \tag{3.12}$$

For 27 node elements:

$$\begin{aligned}
N_1(\xi,\eta,\zeta) &= \ell_0^2(\xi)\,\ell_0^2(\eta)\,\ell_0^2(\zeta) & N_2(\xi,\eta,\zeta) &= \ell_1^2(\xi)\,\ell_0^2(\eta)\,\ell_0^2(\zeta)\\
N_3(\xi,\eta,\zeta) &= \ell_0^2(\xi)\,\ell_1^2(\eta)\,\ell_0^2(\zeta) & N_4(\xi,\eta,\zeta) &= \ell_0^2(\xi)\,\ell_0^2(\eta)\,\ell_1^2(\zeta)\\
&\vdots & &\vdots\\
N_{26}(\xi,\eta,\zeta) &= \ell_1^2(\xi)\,\ell_2^2(\eta)\,\ell_2^2(\zeta) & N_{27}(\xi,\eta,\zeta) &= \ell_2^2(\xi)\,\ell_2^2(\eta)\,\ell_2^2(\zeta)
\end{aligned} \tag{3.13}$$

Figure 3.1.2 shows two shape functions of a two-dimensional 9 node element with curved faces.
With the general definition of the shape functions (Eq. 3.9), we can now easily obtain the Jacobi matrix for the 3-D case. It will be needed together with its determinant, the Jacobian, when we will derive the elemental matrices.

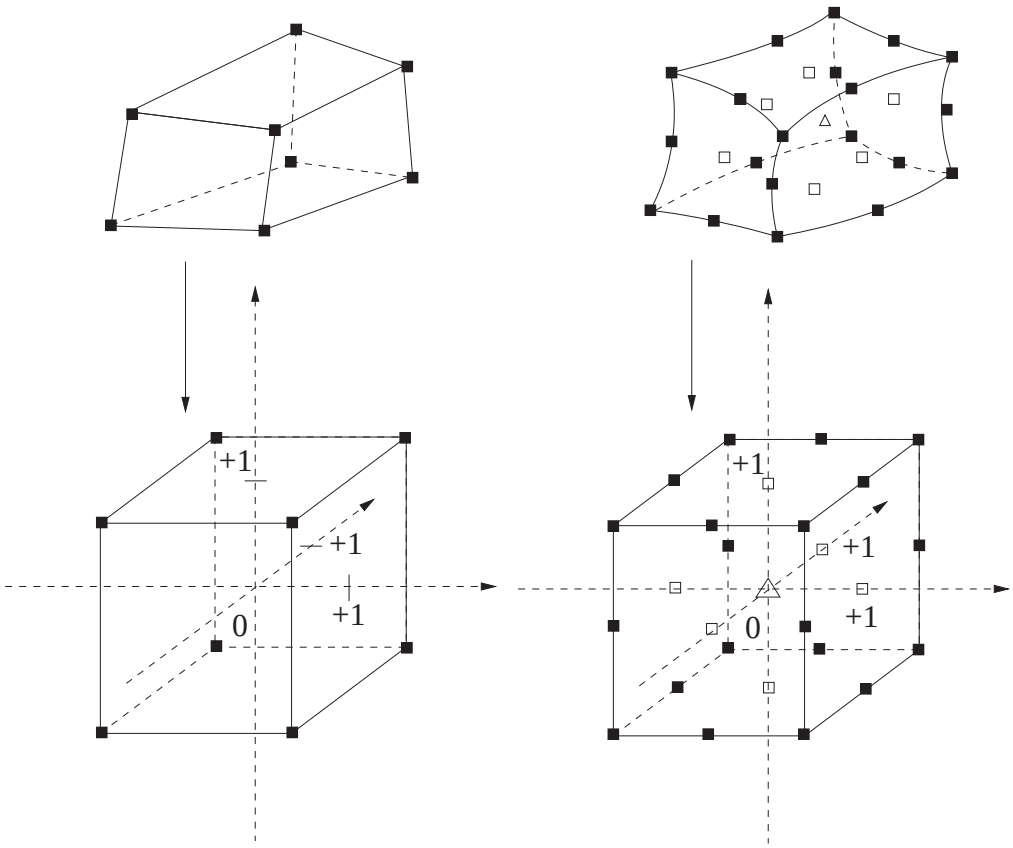
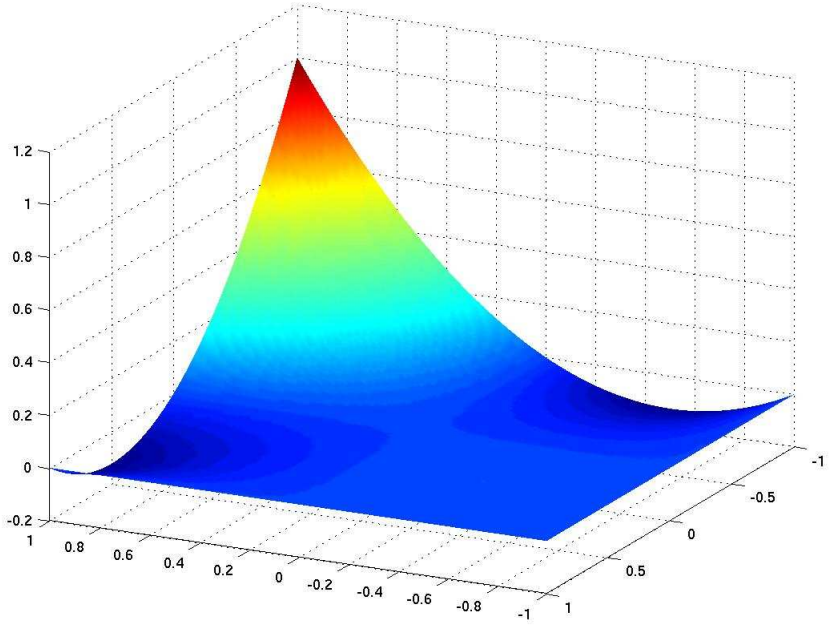
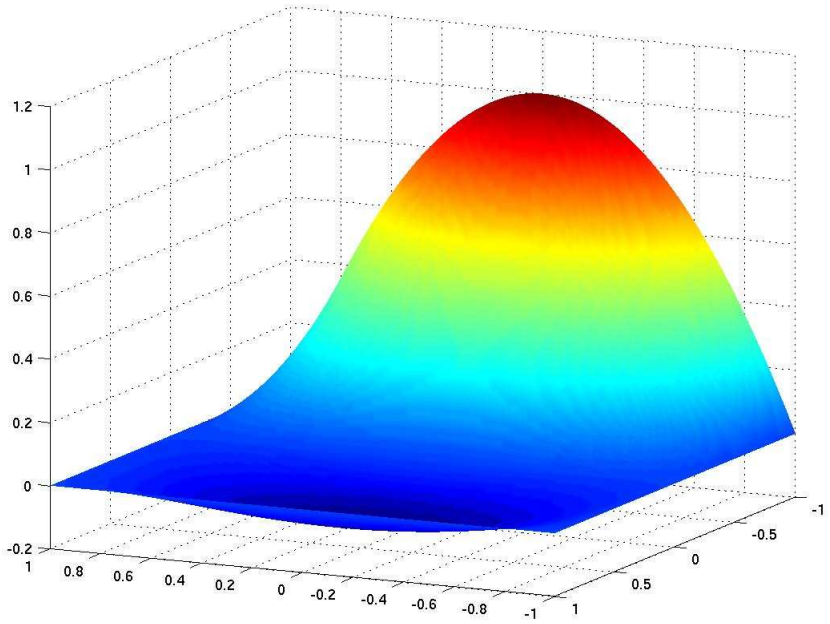


Figure 3.8: Mapping of 3-D elements on the reference cube $\Omega_3 = [-1,1] \otimes [-1,1] \otimes [-1,1]$. Left: 8 node element with straight edges and faces. Right: 27 node element with curved edges and faces.



(a) Shape function N_7.



(b) Shape function N_2.

Figure 3.9: Shape functions of two-dimensional curved elements based on 9 anchor nodes.

The Jacobi-Matrix and the Jacobian for 3-D elements

For the definition of a proper mesh, the Jacobi-Matrix and the Jacobian have to fullfill certain restriction: the unit vector being normal to the surface of the element has to be positive outwards and the Jacobian must never vanish.
On the basis of Equation (3.9) the elements of the Jacobi matrix, which now is a 3×3 matrix, are computed using the shape functions of the mapping:

$$\begin{aligned} \mathbf{J}^e &= \frac{d\boldsymbol{\mathcal{F}}_e(\boldsymbol{\xi})}{d\boldsymbol{\xi}} \\ &= \frac{d\mathbf{x}^e(\boldsymbol{\xi})}{d\boldsymbol{\xi}} \\ &= \sum_{a=1}^{n_a} \frac{dN_a(\boldsymbol{\xi})}{d\boldsymbol{\xi}}\, \mathbf{x}_a^e \end{aligned} \tag{3.14}$$

Writing the full matrix:

$$\mathbf{J} \;=\; \begin{pmatrix} \frac{\partial x(\boldsymbol{\xi})}{\partial \xi} & \frac{\partial x(\boldsymbol{\xi})}{\partial \eta} & \frac{\partial x(\boldsymbol{\xi})}{\partial \zeta} \\ \frac{\partial y(\boldsymbol{\xi})}{\partial \xi} & \frac{\partial y(\boldsymbol{\xi})}{\partial \eta} & \frac{\partial y(\boldsymbol{\xi})}{\partial \zeta} \\ \frac{\partial z(\boldsymbol{\xi})}{\partial \xi} & \frac{\partial z(\boldsymbol{\xi})}{\partial \eta} & \frac{\partial z(\boldsymbol{\xi})}{\partial \zeta} \end{pmatrix} \tag{3.15}$$

Again the superscript e, denoting the number of the considered element, is omitted here.

Thus all we need to do is to calculate the derivatives of the shape functions of Equations (3.10),(3.11),(3.12) and (3.13). This calculation means only taking the derivative of order $N = 1$ (see also Eq. 2.14) or $N = 2$ Lagrange polynomials, which is very simple. These polynomials and their derivatives are:

$N = 1$:

$$\ell_0^1(\xi) = \frac{\xi - (+1)}{-1 - (+1)} = -\frac{\xi - 1}{2} \qquad \ell_0^{1\,\prime} = -\frac{1}{2}$$

$$\ell_1^1(\xi) = \frac{\xi - (-1)}{1 - (-1)} = \frac{\xi + 1}{2} \qquad \ell_1^{1\,\prime} = \frac{1}{2} \tag{3.16}$$

(3.17)

$N = 2$:

$$\ell_0^2(\xi) = \frac{\xi - (+1)}{-1 - (+1)} \cdot \frac{\xi - (0)}{-1 - (0)} = \frac{\xi^2 - \xi}{2} \qquad \ell_0^{2\,\prime} = \xi - \frac{1}{2}$$

$$\ell_1^2(\xi) = \frac{\xi - (-1)}{0 - (-1)} \cdot \frac{\xi - (+1)}{0 - (+1)} = 1 - \xi^2 \qquad \ell_1^{2\,\prime} = -2\xi$$

$$\ell_2^2(\xi) = \frac{\xi - (-1)}{1 - (-1)} \cdot \frac{\xi - (0)}{1 - (0)} = \frac{\xi^2 + \xi}{2} \qquad \ell_2^{2\,\prime} = \xi + \frac{1}{2} \tag{3.18}$$

The Jacobian then gets the following form for 3-D elements:

$$\mathcal{J} = \mathbf{detJ} = \left| \frac{\partial(x, y, z)}{\partial(\xi, \eta, \zeta)} \right| = \begin{vmatrix} \frac{\partial x(\boldsymbol{\xi})}{\partial \xi} & \frac{\partial x(\boldsymbol{\xi})}{\partial \eta} & \frac{\partial x(\boldsymbol{\xi})}{\partial \zeta} \\ \frac{\partial y(\boldsymbol{\xi})}{\partial \xi} & \frac{\partial y(\boldsymbol{\xi})}{\partial \eta} & \frac{\partial y(\boldsymbol{\xi})}{\partial \zeta} \\ \frac{\partial z(\boldsymbol{\xi})}{\partial \xi} & \frac{\partial z(\boldsymbol{\xi})}{\partial \eta} & \frac{\partial z(\boldsymbol{\xi})}{\partial \zeta} \end{vmatrix} \tag{3.19}$$

It appears that the Jacobian describes the change in volume of the element when transforming it from global to local coordinates (see ZIENKIEWICZ & TAYLOR [2000], page 209):

$$dx\ dy\ dz \ = \ \mathcal{J}\ d\xi\ d\eta\ d\zeta \tag{3.20}$$

3.1.3 Interpolation of the Functions on the Elements

The interpolation of the continous functions on the volume of the element is done similar to the 1-D case (Eq. 2.23), using a Lagrangian interpolation scheme. We choose the interpolating points, on which also the Lagrange polynomials are defined (see Eq. 2.21), to be the same as the collocation points of the GLL quadrature of integration. This will lead to a diagonal mass matrix in almost the same manner as in the 1-D case. The derivation of the mass matrix follows in Section 3.1.5.

In the case of three-dimensional interpolation we obtain Lagrangian interpolants $\mathcal{L}$ defined on the reference cube, which are triple products of one-dimensional Lagrange polynomials as they define a tensorial basis $\ell_i \otimes \ell_j \otimes \ell_k$ of the space in which the functions to be interpolated are defined (remember: the order N of the interpolating polynomials is typically chosen between 4 and 8). This characteristic is also known as the "tensorisation" of the interpolation scheme. It means that all coordinate directions are interpolated independently from the others. This is only true for hexaedral elements but not for tetrahedrons. That is why the SEM is mainly used together with elements that can be mapped on a cube. The interpolating three-dimensional polynomials are also said to have the degree N when

$\mathcal{L}^N_{ijk} = \ell^N_i \otimes \ell^N_j \otimes \ell^N_k$ For a scalar function g the three-dimensional interpolation then yields:

$$\begin{aligned} g(\boldsymbol{\xi}) &\approx \sum_{i,j,k=0}^{N} g(\xi_i, \eta_j, \zeta_k)\ \mathcal{L}_{ijk}(\boldsymbol{\xi}) \\ g(\boldsymbol{\xi}) &\approx \sum_{i,j,k=0}^{N} g(\xi_i, \eta_j, \zeta_k)\ \ell_i(\xi) \otimes \ell_j(\eta) \otimes \ell_k(\zeta) \\ &= \sum_{i,j,k=0}^{N} g(\xi_i, \eta_j, \zeta_k)\ \ \ell_i(\xi)\ \ell_j(\eta)\ \ell_k(\zeta) \\ &= \sum_{i,j,k=0}^{N} g_{ijk}\ \ell_i(\xi)\ \ell_j(\eta)\ \ell_k(\zeta) \end{aligned} \tag{3.21}$$

The superscript ^{N} of the polynomial degree of ℓ and $\mathcal{L}$ is again omitted as in most parts of the thesis. It follows directly from $\ell_i(\xi_j) = \delta_{ij}$ (Eq. 2.22) that:

$$\mathcal{L}_{ijk}(\xi_l, \eta_m, \zeta_n) = \delta_{il}\ \delta_{jm}\ \delta_{kn} \tag{3.22}$$

For a three-component vector the interpolation is of the form:

$$\begin{aligned} \mathbf{u}^e(\boldsymbol{\xi}) &\approx \sum_{l=1}^{3} \hat{\boldsymbol{\xi}}_l \sum_{i,j,k=0}^{N} \mathbf{u}^e(\xi_i, \eta_j, \zeta_k)\ \ \ell_i(\xi)\ \ell_j(\eta)\ \ell_k(\zeta) \\ &= \sum_{l=1}^{3} \hat{\boldsymbol{\xi}}_l \sum_{i,j,k=0}^{N} \mathbf{u}^e_{ijk}\ \ell_i(\xi)\ \ell_j(\eta)\ \ell_k(\zeta) \end{aligned} \tag{3.23}$$

The interpolation scheme is again exact on all the collocation points $\boldsymbol{\xi}_{ijk} = (\xi_i, \eta_j, \zeta_k)$ as the Lagrange polynomials assume the value 1 at these points (see Eq. 2.22) and thus also $\mathcal{L}$ (Eq. 3.22). For a more compact notation we define $\mathbf{u}^e_{ijk} = \mathbf{u}^e(\xi_i, \eta_j, \zeta_k)$. And the same that was mentioned for the 1-D case, is valid here: this interpolation can also be used in the code to calculate the values of a function at any arbitrary point inside the element. It is for example used to calculate the displacement at some receiver, which is most unlikely to be exact on one of the collocation points.

The interpolation of partial derivations of a function is performed by computing the derivative of one or more of the Lagrange polynomials:

$$\begin{aligned}
\frac{\partial \mathbf{u}^e(\boldsymbol{\xi})}{\partial \xi} &= \partial_\xi \mathbf{u}^e(\xi,\eta,\zeta) \approx \sum_{i,j,k=0}^{N} \mathbf{u}^e_{ijk}\ \ell_i'(\xi)\ \ell_j(\eta)\ \ell_k(\zeta) \\
\frac{\partial \mathbf{u}^e(\boldsymbol{\xi})}{\partial \eta} &= \partial_\eta \mathbf{u}^e(\xi,\eta,\zeta) \approx \sum_{i,j,k=0}^{N} \mathbf{u}^e_{ijk}\ \ell_i(\xi)\ \ell_j'(\eta)\ \ell_k(\zeta) \\
\frac{\partial \mathbf{u}^e(\boldsymbol{\xi})}{\partial \zeta} &= \partial_\zeta \mathbf{u}^e(\xi,\eta,\zeta) \approx \sum_{i,j,k=0}^{N} \mathbf{u}^e_{ijk}\ \ell_i(\xi)\ \ell_j(\eta)\ \ell_k'(\zeta)
\end{aligned} \tag{3.24}$$

The way how to obtain these derivations ℓ_i' in practice was explained in Section 2.2.2.
The gradient in global $\mathbf{x}$ coordinates of a function g then can be maintained by:

$$\begin{aligned}
\nabla_{\mathbf{x}}\, g(\mathbf{x}(\boldsymbol{\xi})) &\approx \sum_{l=1}^{3} \hat{\mathbf{x}}_l\ \partial_l g(\mathbf{x}(\xi,\eta,\zeta)) \\
&= \sum_{l=1}^{3} \hat{\mathbf{x}}_l \sum_{i,j,k=0}^{N} g_{ijk}\ [\ell_i'(\xi)\ \ell_j(\eta)\ \ell_k(\zeta)\ \partial_l\xi\ + \\
&\quad + \ell_i(\xi)\ \ell_j'(\eta)\ \ell_k(\zeta)\ \partial_l\eta + \ell_i(\xi)\ \ell_j(\eta)\ \ell_k'(\zeta)\ \partial_l\zeta]
\end{aligned} \tag{3.25}$$

(see for example KOMATITSCH & TROMP [1999]).

Here we have to use the Jacobi matrix ($\partial_l \boldsymbol{\xi} = \frac{\partial \boldsymbol{\xi}}{\partial \mathbf{x}_l}$) of the inverse mapping function to account for the coordinate transformation during derivation. This will also be used when calculating the forces acting on the gridnodes.
When evaluating Equation (3.25) at any of the GLL points $\mathbf{x}(\xi_\alpha,\eta_\beta,\zeta_\gamma)$ it reduces to:

$$\begin{aligned}
\nabla_{\mathbf{x}}\, g(\mathbf{x}(\xi_\alpha,\eta_\beta,\zeta_\gamma)) \approx \sum_{l=1}^{3} \hat{\mathbf{x}}_l\ \Big[&\sum_{i=0}^{N} g_{i\beta\gamma}\ \ell_i'(\xi_\alpha)\ \partial_l\ \xi\ (\xi_\alpha,\eta_\beta,\zeta_\gamma)\ + \\
&+ \sum_{j=0}^{N} g_{\alpha j\gamma}\ \ell_j'(\eta_\beta)\ \partial_l\ \eta\ (\xi_\alpha,\eta_\beta,\zeta_\gamma)\ + \\
&+ \sum_{k=0}^{N} g_{\alpha\beta k}\ \ell_k'(\zeta_\gamma)\ \partial_l\ \zeta\ (\xi_\alpha,\eta_\beta,\zeta_\gamma)\ \Big]
\end{aligned} \tag{3.26}$$

We can see that for this calculaton we need the nine elements of the Jacobi-Matrix $\frac{\partial \boldsymbol{\xi}_i}{\partial x_j},\ i,j = 1,2,3$ with $\boldsymbol{\xi}_1 = \xi, \boldsymbol{\xi}_2 = \eta, \boldsymbol{\xi}_3 = \zeta$ of the inverse mapping. These interpolations will then be used for the derivations of the elemental mass matrix $\mathbf{M}^e$ and the forces $\mathbf{f}^e$ at the sampling points of the element.

3.1.4 Integration of Functions over the Volume of the Element

The next step on the way to the matrix formulation for three-dimensional simulations is to define a numerical integration scheme to translate the integrals of Equation (3.8) into finite sums. In Chapter 2 we have introduced the Gauss-Lobatto-Legendre quadrature of integration, which is defined on those collocation points which we chose for the interpolation. This combination of choice of interpolating points together with the GLL integration quadrature led to the exact diagonal shape of the mass matrix in 1-D. In the next section we will see how the diagonality is also obtained in the 3-D case.
The GLL integration rule for the 3-D case reads as follows:

$$\begin{aligned}\int_{\boldsymbol{\Lambda}} g(\boldsymbol{\xi}) d\boldsymbol{\xi} &= \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} g(\xi,\eta,\zeta)\ d\xi\, d\eta\, d\zeta \\ &\approx \sum_{i,j,k=0}^{N} \omega_i\ \omega_j\ \omega_k\ g(\xi_i,\eta_j,\zeta_k) \\ &= \sum_{i,j,k=0}^{N} \omega_i\ \omega_j\ \omega_k\ g_{ijk} \end{aligned} \tag{3.27}$$

ω_i are the weights of the GLL quadrature, which we already know from the 1-D case. $g(\boldsymbol{\xi})$ is again an arbitrary function defined on the interval $\boldsymbol{\Lambda}$.
The GLL quadrature is exact for polynomials up to degree $2N-1$ (see Komatitsch & Tromp [1999] or Zienkiewicz & Taylor [2000],page 219). It has been shown that even for a simple homogeneous undeformed element, the integration involves a product of two polynomials of order N. One results from the interpolation of the displacement, the other one results from the test function. Therefore, the integration of a polynomial of order $2N$ can never be exact. In addition, introducing heterogeneities and deformed elements leads to even higher errors in the solution. But in the SEM this price is paid for the sake of an exactly diagonal mass matrix.

3.1.5 The Elemental Mass Matrices and Their Diagonality

We have seen before that in the SEM one pays a high price in terms of accuracy in the integration to obtain a mass matrix which is exactly diagonal. As in the previous chapter, all discussed features are successively applied to the first integral of Equation (3.8) to derive the expression for the elements of the elemental mass matrix. In the following only one component of the vectors (e.g. $u = u_x$ instead of $\mathbf{u}$) that appear under the integrals is considered. This is possible, because the multiplication of the test function $\mathbf{v}$ with $\mathbf{u}$ results in:

$$\mathbf{v} \cdot \mathbf{u} = \sum_{i,j=1}^{3} v_i \; u_j \; \hat{\mathbf{x}}_i \cdot \hat{\mathbf{x}}_j = \sum_{i,j=1}^{3} v_i \; u_j \delta_{ij} = \sum_{i=1}^{3} v_i \; u_i \tag{3.28}$$

Therefore, we can treat each component of the displacement separately as we can choose the test function for every component u_i to be $v = (\delta_{1i}, \delta_{2i}, \delta_{3i})$.
The equations are more concise this way as we can get rid of the first sum in Equation (3.23), which would appear twice in the calculation of the mass matrix. The first one in conjunction with the interpolation of the displacement, and the second one for the interpolation of the test function. In addition, the number of indices used would get too high. The other components then can be determined likewise.
Considering the integral over the volume of the element ($\mathbf{\Omega}^e$) gives:

$$\begin{aligned}
&\qquad\qquad\qquad\qquad \downarrow \text{ mapping, Jacobian is needed} \\
&\int_{\mathbf{\Omega}^e} \rho(\mathbf{x}) \; v(\mathbf{x}) \; \ddot{u}(\mathbf{x}) \; d\mathbf{x} = \int_{\mathbf{\Lambda}} \rho(\boldsymbol{\xi}) \; v(\boldsymbol{\xi}) \; \ddot{u}(\boldsymbol{\xi}) \; \mathcal{J}(\boldsymbol{\xi}) \; d\boldsymbol{\xi} \\
&\text{interpolation of } v \text{ and } \ddot{u} \downarrow \\
&\qquad\qquad\qquad \approx \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \rho(\xi,\eta,\zeta) \; \Big[\sum_{i,j,k=0}^{N} v_{ijk} \; \ell_i(\xi)\ell_j(\eta)\ell_k(\zeta) \Big] \\
&\qquad\qquad\qquad \Big[\sum_{l,m,n=0}^{N} \ddot{u}_{lmn} \; \ell_l(\xi)\ell_m(\eta)\ell_n(\zeta) \Big] \; \mathcal{J}(\xi,\eta,\gamma) \; d\xi \, d\eta \, d\gamma \\
&\qquad\qquad\text{GLL quadrature} \downarrow \\
&\qquad\qquad\qquad \approx \sum_{r,s,t=0}^{N} \Big\{ \rho_{rst} \; \omega_r \; \omega_s \; \omega_t \; \Big[\sum_{i,j,k=0}^{N} v_{ijk} \; \ell_i(\xi_r)\ell_j(\eta_s)\ell_k(\zeta_t) \Big] \\
&\qquad\qquad\qquad \Big[\sum_{l,m,n=0}^{N} \ddot{u}_{lmn} \; \ell_l(\xi_r)\ell_m(\eta_s)\ell_n(\zeta_t) \Big] \; \mathcal{J}_{rst} \Big\}
\end{aligned} \tag{3.29}$$

For simplification, all functions as for example $u(\xi_l, \eta_m, \zeta_n)$ are again expressed by using subscripts u_{lmn} directly.
Utilizing similar factorizations as in Chapter 2 we obtain:

$$\begin{aligned}
&\sum_{r,s,t=0}^{N} \Big\{ \rho_{rst}\ \omega_r\ \omega_s\ \omega_t \Big[\sum_{i,j,k=0}^{N} v_{ijk}\ \ell_i(\xi_r)\ell_j(\eta_s)\ell_k(\zeta_t) \Big] \\
&\qquad \Big[\sum_{l,m,n=0}^{N} \ddot{u}_{lmn}\ \ell_l(\xi_r)\ell_m(\eta_s)\ell_n(\zeta_t) \Big]\ \mathcal{J}_{rst} \Big\} = \\
&\qquad\qquad = \sum_{r,s,t=0}^{N} \Big\{ \rho_{rst}\ \omega_r\ \omega_s\ \omega_t\ \mathcal{J}_{rst} \Big[\sum_{i,j,k=0}^{N} \delta_{ir}\ \delta_{js}\ \delta_{kt} \Big] \\
&\qquad\qquad\quad \Big[\sum_{l,m,n=0}^{N} \ddot{u}_{lmn}\ \delta_{lr}\ \delta_{ms}\ \delta_{nt} \Big] \Big\} \\
&\qquad\qquad\qquad = \sum_{l,m,n=0}^{N} \Big\{ \ddot{u}_{lmn} \\
&\qquad\qquad\qquad\quad \Big[\sum_{i,j,k=0}^{N} \sum_{r,s,t=0}^{N} \rho_{rst}\ \omega_{rst}\ \mathcal{J}_{rst}\ \delta_{ir}\ \delta_{js}\ \delta_{kt}\ \delta_{lr}\ \delta_{ms}\ \delta_{nt} \Big] \Big\}
\end{aligned} \tag{3.30}$$

The weights of the integration quadrature in each direction $\omega_r\ \omega_s\ \omega_t$ were combined to ω_{rst}. As there are no restrictions to the choice of v we have set $v_{ijk} = 1$ and by using Equation (2.22) we have evaluated the term on the left side and obtained the Kronecker-delta distributions. By further arrangement of the sums we get:

$$\begin{aligned}
&\sum_{l,m,n=0}^{N} \Big\{ \ddot{u}_{lmn} \Big[\sum_{i,j,k=0}^{N} \sum_{r,s,t=0}^{N} \rho_{rst}\ \omega_{rst}\ \mathcal{J}_{rst}\ \delta_{ir}\ \delta_{js}\ \delta_{kt}\ \delta_{lr}\ \delta_{ms}\ \delta_{nt} \Big] \Big\} = \\
&\qquad\qquad = \sum_{l,m,n=0}^{N} \ddot{u}_{lmn} \sum_{i,j,k=0}^{N} \rho_{ijk}\ \omega_{ijk}\ \mathcal{J}_{ijk}\ \delta_{li}\ \delta_{mj}\ \delta_{nk} \\
&\qquad\qquad = \sum_{l,m,n=0}^{N} \ddot{u}_{lmn} \sum_{i,j,k=0}^{N} \rho_{ijk}\ \omega_{ijk}\ \mathcal{J}_{ijk} \delta_{(lmn)(ijk)} \\
&\qquad\qquad = \sum_{\alpha=1}^{(N+1)^3} \ddot{u}_{\alpha} \sum_{\beta=1}^{(N+1)^3} \rho_{\beta}\ \omega_{\beta}\ \mathcal{J}_{\beta} \delta_{\alpha\beta} \\
&\qquad\qquad = \ddot{u}_{\alpha} M^{e}_{\alpha\beta}
\end{aligned} \tag{3.31}$$

Here we used $\alpha = lmn$ and $\beta = ijk$, with $\alpha, \beta = 1, \ldots, (N+1)^3$, since all Roman indices are in the range of $0, \ldots, N$. $\delta_{(lmn)(ijk)} = \delta_{li}\delta_{mj}\delta_{nk}$ was used to get the appropriate expression for M as a matrix. The superscript ^{e} of the mass matrix only indicates that this is the elemental mass matrix, whereas $u = u^e$ was not explicitly specified. On the last line of Equation (3.31) we have used the implicit sum convention over repeated indices.
$M^e_{\alpha\beta}$ the $(N+1) \cdot (N+1)$ elements of the elemental mass matrix can therefore be maintained by a product of the density ρ, the Jacobian and the weights at a given point (ξ_i, η_j, ζ_k):

$$\begin{aligned} M^e_{\alpha\beta} &= \rho_\beta \; \omega_\beta \; \mathcal{J}_\beta \; \delta_{\alpha\beta} = \rho_{ijk} \; \omega_{ijk} \; \mathcal{J}_{ijk} \\ \forall \; i,j,k &= 0, \ldots, N \text{ and } \;\; \alpha, \beta \text{ as above} \end{aligned} \tag{3.32}$$

A unique transformation between α, β and ijk can be written as:

$$\alpha = \sum_{i,j,k=0}^{N} (i+1) \cdot (j+1) \cdot (k+1) \tag{3.33}$$

Again the mass matrix is exactly diagonal, which is the desired characteristic for the SEM as the inversion of the mass matrix is therefore trivial. And computer memory can be saved to a great extent as was stated earlier, when storing only the diagonal elements of the mass matrix in a vector.

3.1.6 Calculation of Forces in 3-D

Unlike in Chapter 2, the derivation of the stiffness matrix for 3-D applications will not be described in detail, since it is not used in practice in the SEM. As discussed in Section 2.2.8 it is much easier to calculate step by step the forces acting on the nodes of an element separately. To do so we will first have to determine the nine elements of the strain tensor at a given point $\boldsymbol{\xi}_{\alpha\beta\gamma} = (\xi_\alpha, \eta_\beta, \zeta_\gamma)$, i.e. the displacement gradient $\nabla \mathbf{u}$ (see Eq. 3.26):

$$\begin{aligned} \partial_i \; u_{(j)} \left(\mathbf{x}(\boldsymbol{\xi}_{\alpha\beta\gamma}) \right) \approx & \Big[\sum_{l=0}^{N} \; u_{(j)\, l\beta\gamma} \; \ell'_l(\xi_\alpha) \; \Big] \; \partial_i \xi(\xi_\alpha, \eta_\beta, \zeta_\gamma) \\ & + \Big[\sum_{m=0}^{N} \; u_{(j)\, \alpha m\gamma} \; \ell'_m(\eta_\beta) \; \Big] \; \partial_i \eta(\xi_\alpha, \eta_\beta, \zeta_\gamma) \\ & + \Big[\sum_{n=0}^{N} \; u_{(j)\, \alpha\beta n} \; \ell'_n(\zeta_\gamma) \; \Big] \; \partial_i \zeta(\xi_\alpha, \eta_\beta, \zeta_\gamma) \end{aligned} \tag{3.34}$$

$u_{(j)\, \alpha\beta\gamma}$ is the j-th component of the displacement vector $\mathbf{u}$ at the interpolating point $(\xi_\alpha, \eta_\beta, \zeta_\gamma)$.

The next step is the calculation of the stress tensor by applying the generalised Hooke's Law (Eq. 3.3):

$$\boldsymbol{\sigma}(\mathbf{x}(\boldsymbol{\xi}_{\alpha\beta\gamma})) = \mathbf{c}\,(\mathbf{x}(\boldsymbol{\xi}_{\alpha\beta\gamma})) \otimes \nabla\mathbf{u}\,(\mathbf{x}(\boldsymbol{\xi}_{\alpha\beta\gamma}))$$

The index notation of this expression is written in Equation (3.3).
The evaluation of the integrand $\nabla\mathbf{v}\cdot\boldsymbol{\sigma}$ of the second integral of Equation (3.8) at one node yields ($\mathbf{v}$ has three components and $\boldsymbol{\sigma}$ is a 3×3 tensor):

$$\begin{aligned}\nabla\mathbf{v}\cdot\boldsymbol{\sigma} &= \sum_{i,j=1}^{3}\sigma_{ij}\partial_j v_i \approx \sum_{i,k=1}^{3}\Big(\sum_{j=1}^{3}\sigma_{ij}\partial_j\boldsymbol{\xi}_k\Big)\frac{\partial v_i}{\partial\boldsymbol{\xi}_k}\\ &= \sum_{i,k=1}^{3} B_{ik}\;\frac{\partial v_i}{\partial\boldsymbol{\xi}_k}\end{aligned} \tag{3.35}$$

with

$$B_{ik} = \sum_{j=1}^{3}\sigma_{ij}\partial_j\boldsymbol{\xi}_k \tag{3.36}$$

Now we can apply the interpolation scheme for $\frac{\partial v_i}{\partial\boldsymbol{\xi}_k}$ and introduce the GLL integration quadrature (i.e. evaluating the interpolation of $\frac{\partial v_i}{\partial\boldsymbol{\xi}_k}$ at the points of the GLL quadrature; remember Equation (3.26)) and obtain:

$$\begin{aligned}\int_{\boldsymbol{\Omega}^e}\nabla\mathbf{v}\cdot\boldsymbol{\sigma}\;d\mathbf{x} &\approx \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1}\Big[\sum_{i,k=1}^{3}B_{ik}\;\frac{\partial v_i}{\partial\boldsymbol{\xi}_k}\;\Big]\;\mathcal{J}\;d\xi\,d\eta\,d\zeta\\ &\approx \sum_{i=1}^{3}v_{(i)\,\alpha\beta\gamma}\;\Big[\;\omega_\beta\omega_\gamma\sum_{l=0}^{N}\omega_l\;\mathcal{J}_{l\beta\gamma}\;B_{(i1)\,l\beta\gamma}\;\ell'_\alpha(\xi_l)\\ &\quad+\omega_\alpha\omega_\gamma\sum_{m=0}^{N}\omega_m\;\mathcal{J}_{\alpha m\gamma}\;B_{(i2)\,\alpha m\gamma}\;\ell'_\beta(\xi_m)\\ &\quad+\omega_\alpha\omega_\beta\sum_{n=0}^{N}\omega_n\;\mathcal{J}_{\alpha\beta n}\;B_{(i2)\,\alpha\beta n}\;\ell'_\gamma(\xi_n)\Big]\end{aligned} \tag{3.37}$$

After all these calculations on the elemental level we can now focus on the global system of equations for numerical integration of the weak form of the wave Equation (Eq. 3.7).

3.1.7 The Assembly of the Global Linear System

The last step to complete the numerical spatial integration of the SEM is to combine the expressions for the elements to the often addressed global linear system of equations. The process of assembling the elements was introduced in Section 2.2.6 of the previous chapter. The idea of transforming a global to a local numbering and vice versa was demonstrated there and is also applied in 3-D. Nevertheless, it is clear that the additional dimensions significantly increase the difficulties encountered during this operation. In 1-D only one node was shared between two elements, which led to a rather simple connectivity-matrix. In 2-D the elements share all of their nodes on the edges and in 3-D the number of shared nodes far exceeds the number of nodes lying inside one element and being unaffected by other elements. For an order $N = 4$, we have elements containing $(N + 1)^3 = 125$ collocation points, of which 98 belong to more than one element, and only 27 being separated from others. Four of those nodes, which are lying on the corners of the cube, belong to eight elements, 40 nodes are on the edges (without those on the corners) and belong to four elements and additional 54 nodes belong to two elements as they are lying on the faces of the elements. This example clearly demonstrates the complexity one has to deal with when reassembling the elements of the mesh.

Figure 3.10 shows four quadratic elements attached at four edges. The black circles indicate the shared nodes.

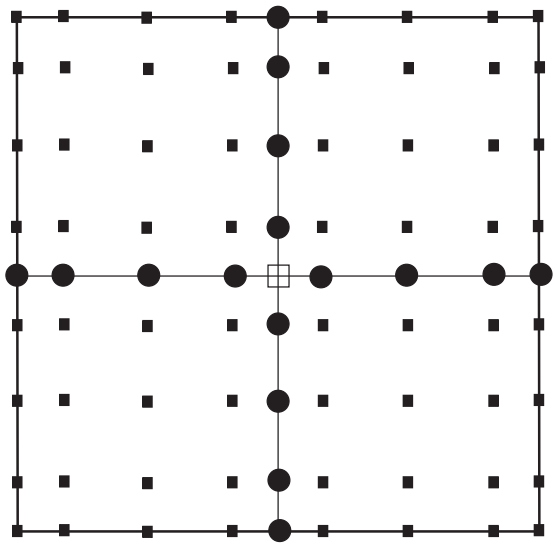


Figure 3.10: Illustration of the connection of 2-D elements at four of their edges. The node in the middle, displayed as a white square, is shared by all four of them.

Concerning the practical implementation of the assembly for three-dimensional

meshes further detail will not be given as the complexity of this topic lies far outside the frame of this thesis and is only of technical importance. A solution for this problem may either be found in publications on classical FEM (Zienkiewicz & Taylor [2000]; Hughes [1987]) or in the SEM code for global seismic wave propagation written by Dimitri Komatitsch and co-workers, where information about the connectivity is obtained by some special sorting routines. The code is available for noncommercial science from `http://www.gps.caltech.edu/~jtromp/research/downloads.html` (CalTech [2002]).

3.1.8 Implementation of sources in 3-D

The implementation of sources in the SEM is very convenient. We have already seen in Section 2.2.8 that we can simply add external stresses to the stresses derived from the displacement gradient using Hooke's Law. In most cases of earthquake studies, the source mechanism is described using the moment tensor. When simulating real events, we can use these moment tensors. A detailed mathematical formulation is given in Komatitsch & Tromp [1999].
Very recently seismologists also have made efforts to implement rupture mechanisms into SEM (Ampuero *et al.* [2003]). Simulation of dynamic rupture is probably one of the most important issues in seismology in the coming years.

Part II

Results of the Simulations - Evaluation and Comparisons

Chapter 4

Evaluation of the SEM in 1D

One of the aims of this work was to gain knowledge on the performance and accuracy of the spectral element method compared to standard and improved finite difference methods. Optimal FD operators (Geller & Takeuchi [1995]) are of the latter kind and were investigated in the course of an other diploma thesis at our institute (Metz [2003]). Therefore, the main interest was to compare these two methods in case of 1-D simulations to find out about advantages and disadvantages of either kind. It would have taken much longer to set up appropriate codes for 3-D modelling, than is given for a diploma thesis. "Optimal operators" are finite difference operators, that are especially designed to minimize the error of the numerical differentiation. Further details on this method can be found in Metz [2003] and references therein. The OPO used in this study are optimized 3-point FD operators.
Because the SEM allows for several different time integration schemes, a comparison between those was performed before comparing the best[1] one with the OPO. This included also a test which order of interpolating polynomials leads to the best results.

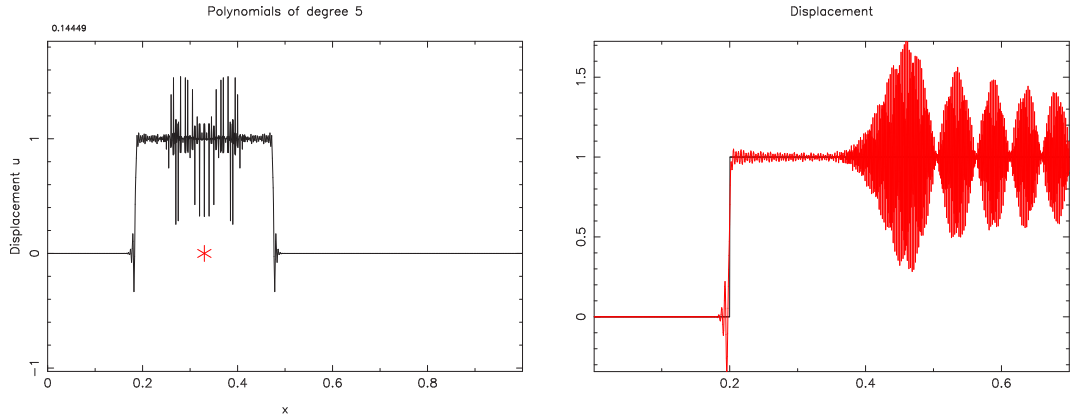
As the comparisons presented in Section 4.2 are based on seismograms of 1-D simulations, some examples will be given in Section 4.1 first. In addition, several snapshots of simulations with different boundary conditions are shown. The serial SEM code used for these simulations is printed in the Appendix A.2. The code was compiled on `Linux 2.4.18` using the "intel fortran compiler" version 6.0 by intel ®. The different simulations, including those with OPO, were performed on an AMD Athlon XP 1800+ Processor. Attention was paid to run the simulations used for comparison[2] on exactly the same machine.

[1] in terms of performance (see descriptions later on)
[2] especially the CPU benchmarking in Section 4.2

4.1 Snapshots and Seismograms of 1-D SEM Simulations

In the following, some examples of snapshots and seismograms of 1-D simulations are presented. All these simulations were performed with the program given in the Appendix A.2. The order of Lagrange polynomials was $N = 5$ and a total of $n_e = 200$ elements was used. The Length of the model was 1 meter.

Figure 4.1 shows a snapshot and one seismogram obtained in a homogeneous model using a point source acting only one time step. The source time function is therefore a delta-peak and the medium is excited at all frequencies from 0 up to the Nyquist frequency. This is the frequency given by half the sampling frequency, $\nu_N = \frac{1}{2dt}$.
The analytical solution for the displacement in a homogeneous medium can be expressed by the Heaviside function with arrival time $t_a = \frac{d}{\alpha}$, with d being the distance between the source and the receiver. The wave velocity α was $1\frac{\mathrm{m}}{\mathrm{s}}$ and $d = +0.2$ m. In Figure 4.1(b) the analytical solution is given together with the recorded synthetic seismogram.



(a) Snapshot of the displacement field. The red star marks the location of the source.

(b) Displacement seismogram at a distance of 0.2 m away from the source. The analytical solution is plotted in black.

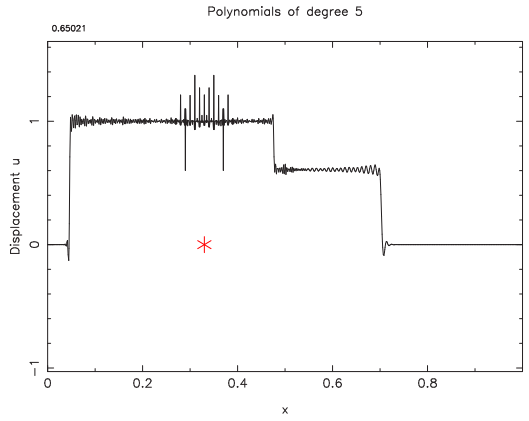
Figure 4.1: Simulation of a point source acting in a homogeneous model. The source time function was a delta-peak in time.

In Figure 4.2 a similar simulation is shown with a "two layer model", meaning that at a certain coordinate $x = 0.55$ m the velocity inside the medium is abruptly changing from $\alpha = 1\frac{\mathrm{m}}{\mathrm{s}}$ for $x > 0.55$ m to $\alpha = 0.436\frac{\mathrm{m}}{\mathrm{s}}$ for $x < 0.55$ m. This change

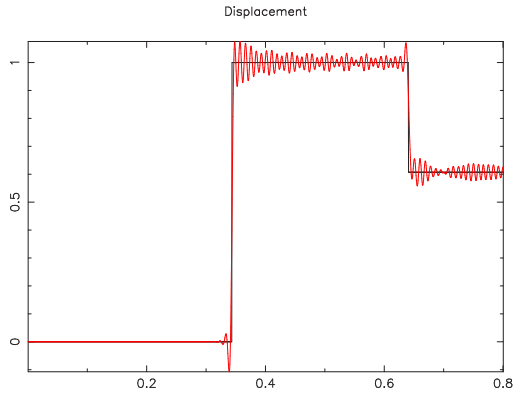
in model parameters lies exactly on the boundary of two adjacent elements. A reflected phase can clearly be seen at $x = 0.5$ m in Figure 4.2(a). The transmitted phase is at around $x = 0.6$ m. The reflected phase can also be seen at $t = 0.52$s in Figure 4.2(b). The receiver was placed at $x = 0.53$ m.
When simulations are done using a delta-peak source time function, one is able to filter the seismograms with several differently shaped filters. This is usually done by convolution of the seismogram with the source time function wanted. Possible wavelets that are used for this convolution are ricker wavelets. These are either the first or second derivative of a Gaussian shaped source time function.

Figure 4.3 shows snapshots of simulations using different boundary conditions. These simulations were convolved with a first order ricker wavelet. The Gaussian shaped response is due to the fact that the convolution with a Heaviside function[3] acts like an integration of the source signal. For a free surface boundary condition the phase of the reflected wave is the same as that of the incident wave. When using rigid boundaries the phase is changed by $\pi = 180°$. When using absorbing boundaries, some energy is still reflected back into the model as can be seen in Figure 4.3(d).

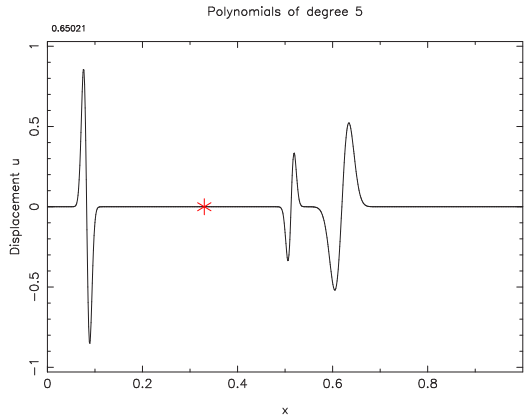
[3]The response of the medium to excitation with all frequencies, i.e. the Green's function



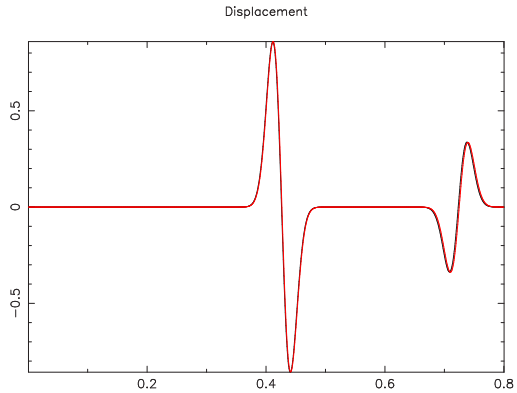
(a) Snapshot of the displacement field. The red star marks the location of the source.



(b) Displacement seismogram at a distance of 0.2 m away from the source. The analytical solution is plotted in black.

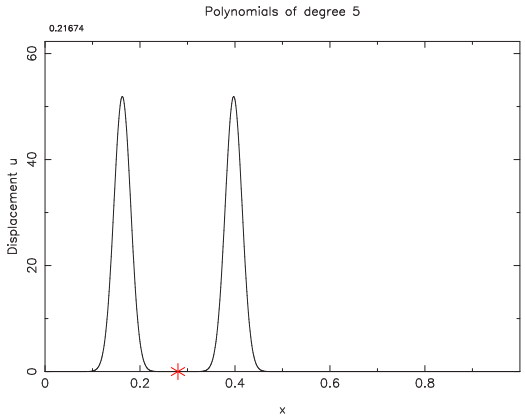


(c) Snapshot of the displacement field filtered with a ricker wavelet using a dominant frequency of 12 Hz.

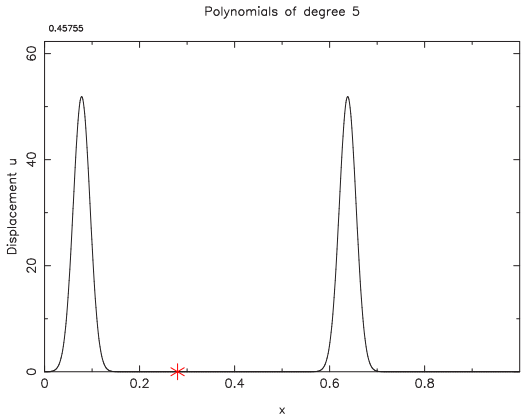


(d) Filtered displacement seismogram at a distance of 0.2 m away from the source (12 Hz). The analytical solution is plotted in black.

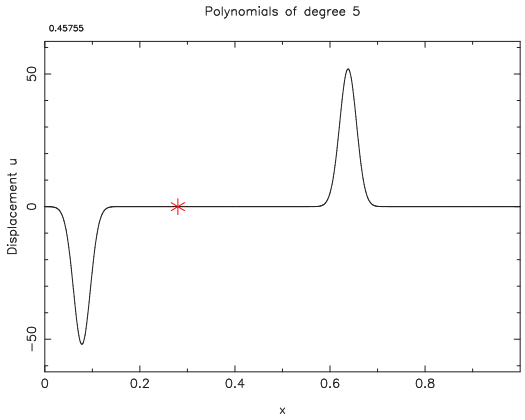
Figure 4.2: Simulation of a point source acting in a "two layer model". Wave velocity α changed at $x = 0.55$ m from $\alpha = 1\frac{\text{m}}{\text{s}}$ for $x > 0.55$ m to $\alpha = 0.436\frac{\text{m}}{\text{s}}$ for $x < 0.55$ m. The source time function was a delta-peak in time.



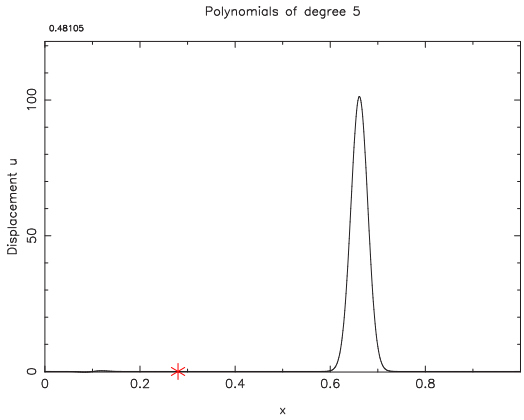
(a) Before reaching the boundary (same for all simulations).



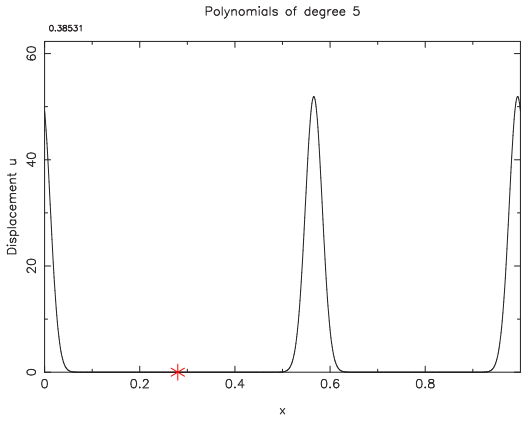
(b) After reflection at the free surface boundary.



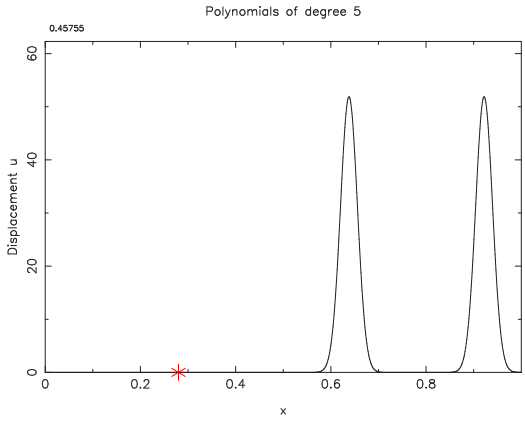
(c) After reflection at the rigid boundary.



(d) After reflection at the absorbing boundary.



(e) No reflection, when periodic conditions are applied.



(f) Wavelet appears on the right after circling the periodic boundary.

Figure 4.3: Examples of snapshots showing 1-D simulations with different boundary conditions. The red star marks the location of the source. Note that not the whole energy of the wavelet is suppressed by the absorbing boundaries (d). Theoretical background to boundary conditions in SEM are given in Section 2.1.9.

4.2 Description of the Method Used to Compare the performance of SEM and Different FD methods

As demonstrated in the previous section, there is a variety of ways to present the output of a numerical simulation. Therefore, it is necessary to define a certain model setup and output format for comparison between different simulations. In this study the aim was to benchmark the SEM against several FD methods, especially the mentioned OPO. Thus, the setup chosen for this benchmark is explained in more detail.

Altogether, the performance of a method can be divided into 3 major subjects: the accuracy, the time needed to compute a certain part of the simulation, called CPU time, and in addition the requirements in computer memory. For 1-D simulations the latter can be neglected as the memory available in modern computers is sufficient to simulate vibration of strings or the strain of a one-dimensional bar exposed to some load. Nevertheless, we did a rough comparison of memory requirements for same model sizes between the OPO and the SEM. It showed that the SEM probably needs 1.2 to 1.5 times the memory of OPO simulations. The storage of the Jacobi-Matrices and Jacobians makes for most of the additional memory. It would be very interesting to compare the memory demands for 3-D simulations to achieve the same accuracy. The memory needed for storage of the Jacobians is growing exponentially with additional dimensions.

In order to make comparisons easier, a way to combine the other two criterions for the perfomance of a method is desirable. This was achieved by a method that consists of two steps, which will be explained in the following.

An objective measure of the overall performance of a numerical method must be independent from physical parameters, and is here chosen to be the time it takes (in CPU time) to achieve a certain accuracy ϵ for a given model. This will be referred to as the CPU cost in the following. In other words, a certain accuracy shall be obtained for a given smallest wavelength, that shall be propagated for a given distance d (expressed in number of propagated wavelengths=npw).

$$npw = \frac{d}{\lambda} = \frac{\frac{x}{\langle dx \rangle}}{\frac{\lambda}{\langle dx \rangle}} = \frac{ngp}{ppw}, \tag{4.1}$$

with $\langle dx \rangle$ the mean grid spacing, λ the wavelength and x the propagated distance. ngp is the number of grid points. The denominator of the last term appears to

be a good measure of the spatial resolution and is expressed as the grid points per minimum wavelength:

$$ppw = \frac{\lambda_{min}}{\langle dx \rangle} \tag{4.2}$$

Thus, the error ϵ depends on ppw and npw. The measure of the accuracy used here, is the so-called relative solution error, which can be obtained in the following way:

$$\epsilon(ppw, npw) = \frac{\int_{-\infty}^{\infty} (u_{analytical} - u_{synthetic})^2 \, dt}{\int_{-\infty}^{\infty} u_{analytical}^2 \, dt} \tag{4.3}$$

On the one hand the CPU cost depends on the spatial resolution. On the other hand, a higher spatial resolution of a given model leads to higher accuracy, but also to more gridnodes in the model. This in turn increases the CPU time per time step (cpt). We can therefore define the additional CPU time it takes for every additional gridpoint, which will be called "effort factor" here, and is denoted by ef:

$$ef = \frac{\Delta cpt}{\Delta ngp} \tag{4.4}$$

as the cpt per model size is $\frac{cpt}{ngp}$. Finally, the overall CPU cost per given model size $ngp = npw \cdot ppw$ can be expressed by:

$$CPUcost = npw \cdot ppw \cdot ef \tag{4.5}$$

To calculate the CPU cost in practice we have to perform two steps. In the first step we calculate the error ϵ as a function of ppw and npw ($\epsilon = \epsilon(ppw, npw)$). This dependence can be plotted as a surface as done in Figure 4.4(a). Then the values are rearranged to obtain the ppw needed to limit the error to a certain value in distance npw: $ppw = ppw(\epsilon, npw)$. Figure 4.4(b) illustrates this case.
In a second step, the mean CPU time per time step is calculated for several models of different size. From Figure 4.5 one can see that this leads to a more or less linear relation. The effort factor ef is obtained as the slope of the graph given by cpt against ngp.
The results of these calculations can be found in the next section.

For the first step, a model was set up for both methods with 2341 grid nodes, of which 1200 belonged to the actual model. The rest served for simulating an infinitely long medium to prevent reflections from the artificial boundaries. To obtain this number of points in the SEM an appropriate number of elements, depending on the order of polynomials, was used. A value of 0.82 for the Courant

number was chosen in all SEM simulations, which is the highest number that can be used to achieve stable solutions. Three items have to be emphasized at this point:

- The distance between grid points varies in the SEM. Therefore all that is said about any relation between number of gridpoints and length corresponds to the mean distance.
- Boundary effects did not play a role as the time that was simulated was chosen to be shorter than it takes the reflected waves to arrive at the receiver being nearest to the boundary.
- The Courant value of 0.82 in SEM leads to an effective Courant value of around 0.5 for order N=5 spectral elements. This is due to the fact that the stability criterion uses the minimum grid distance, whereas the mean grid spacing was used for the effective Courant number.

The source was located one third of the overall length away from the left end, exactly on the boundary of two elements. Thus, the index of the corresponding grid node was slightly different for different polynomial degrees. The source location marks the left end of the receiver array. A total of 240 receivers were located in the model, spread over the 1200 points mentioned. The mean distance between the receivers was thus 5 times the mean grid spacing $\langle dx \rangle$. The first receiver was located $5\langle dx \rangle$ away from the source, the last at $240 \cdot 5\langle dx \rangle$.
The source acting on the medium was a point source with a delta-peak in time to enable convolution of different source time functions after the simulation was completed. The evaluation was carried out with several "Matlab" routines.

This special setup allowed for a maximum of 30 propageted wavelengths (npw), with 40 points per wavelength. Frequencies were chosen such that the simulations could be evaluated in the range of 5 up to 40 ppw.
Two different kinds of models were simulated. One homogeneous model and one "two layer model" where the wave velocity was changing in the middle of the receiver array, again, the interface lying exactly on a boundary of elements in the SEM. The ratio of velocities was chosen to be $\frac{\alpha_1}{\alpha_2} = 0.7$, with α_1 being the wave velocity in the left part of the model.
For both models an analytical solution could be obtained at every receiver. Thus it was possible to calculate the relative solution error ϵ of the simulations. In practice, the value of ϵ is multiplied by 100 to obtain percentages.

It is worth mentioning that the values of *ppw* used for the preliminary results of this study were calculated using the dominant frequency ν_0 of the convolved ricker wavelet, in contrast to many publications, in which a value of $\nu_{max} = 2.5\nu_0$ is used. At approximately this frequency the radiated energy drops below 5 % of

the total energy of the wavelet (Komatitsch & Vilotte [1998]). Therefore, the error ϵ may be significantly higher than expected for a value of associated ppw. To avoid confusion and to be consistent with earlier publications, the value of *ppw* defined by the minimum wavelength, and thus calculated using the maximum frequency, was used for the comparisons.

4.3 Results of the Comparison

In the first part of this section the CPU benchmark of the SEM in displacement formulation[4] together with the Optimal 3-point FD Operators is shown. The results of the CPU benchmark of all further SEM types are summed in Table 4.1. In a second part, comparisons between different time scheme implemenations for the SEM are persented, followed by comparisons of SEM using different polynomial orders. The last part is dealing with the comparisons of SEM to OPO.

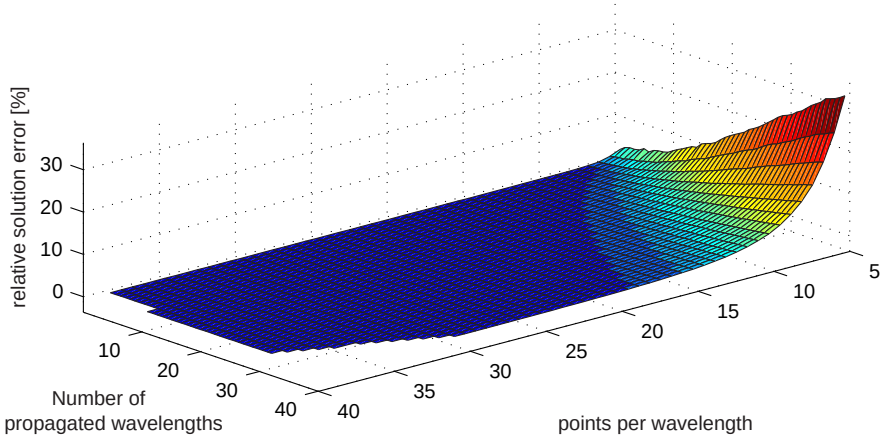
4.3.1 Benchmark of the CPU time per time step

The CPU benchmark consisted of 40 to 50 different simulations using model sizes between 401 and 20001 grid points. Each simulation was done ten times and ran for 2000 time steps. The value of CPU time per time step is therefore the mean of 20000 values to satisfy statistical requirements. All tests were carried out on the same machine as mentioned at the beginning of this chapter.

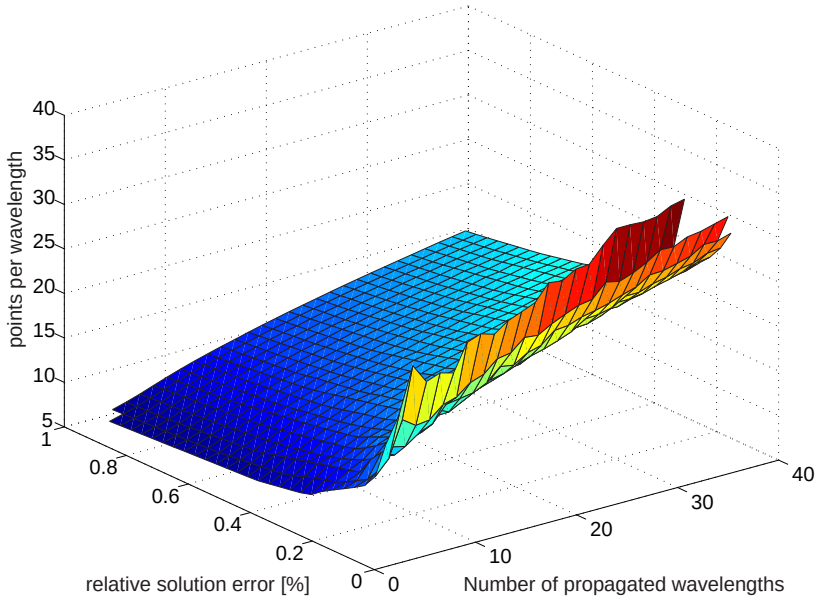
In Figure 4.5 the results of this benchmark for the OPO together with the SEM based on displacement formulation and calculation of forces are shown . It is clearly visible that the OPO need the fewest time, which may be due to the lower number of points per operator.
Usual FD Taylor Operators

From Table 4.1 one can see that using the displacement formulation for the SEM leads to the fewest CPU time needed for one time step only considering different time schemes of the SEM. Nevertheless the difference is not too great. The CPU time per time step increases with increasing order of spectral elements, but surprisingly only for values greater than $N = 5$. Only for the stiffness matrix formulation the effort factor of order 4 is less than that of order 5. Despite this, usual FD Taylor Operators appear to be less time consuming than the OPO, even the 5-point FD operators.
The effort factors gained from the CPU benchmark are used in the next section for calculating the corresponding cost.

[4]formulation that is used throughout Chapter 2



(a) Relative solution error plotted against npw and ppw.



(b) Inverted surface showing ppw plotted against ϵ and npw.

Figure 4.4: Surfaces showing the relative solution error ϵ and ppw as examples of the results used for the comparisons. A homogeneous model was used and wave propagation was simulated with SEM using degree 8 polynomials. Note that the dominant frequency ν_0 itself was used for the calculation of ppw here. For the comparisons the value was changed to be the maximum frequency of the corresponding ricker wavelet, which is given by $\nu_{max} = 2.5\nu_0$.

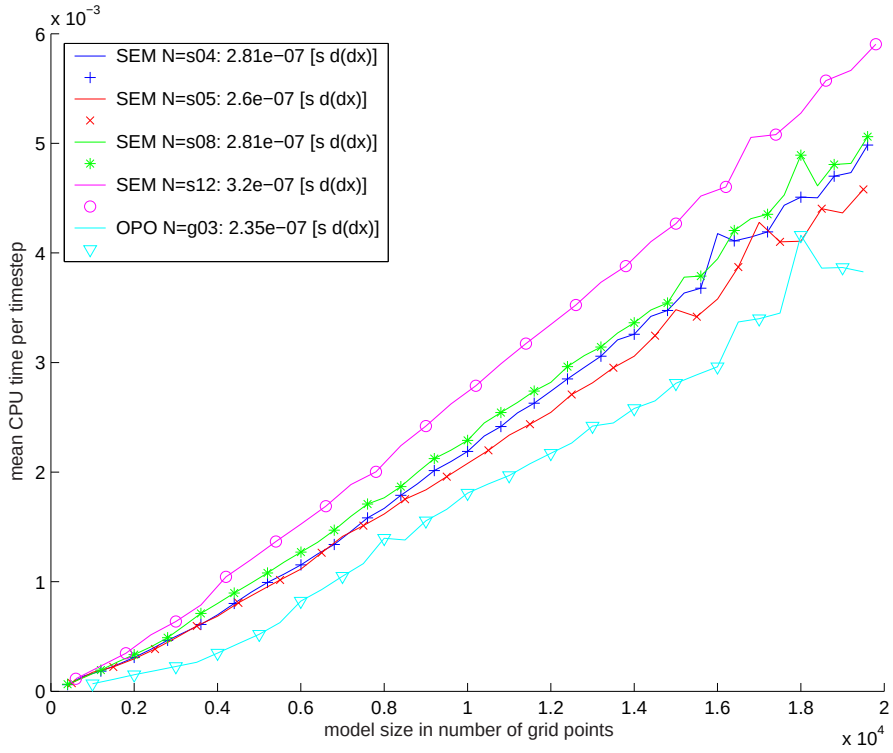


Figure 4.5: Benchmark of CPU time per time step for different model sizes performed for four different SEM's and the OPO. The values shown in the legend give the corresponding effort factor ef, determined by the slope of each graph.

Method	Order of Operator N	effort factor $ef \quad 10^{-7}$
SEM	4	2.81
	5	2.60
	8	2.81
	12	3.20
OPO	3	2.35
Tayler Op.	3	1.2
Taylor Op.	5	1.3
SEM smf	4	2.48
	5	2.74
	8	3.59
	12	4.14
SEM acf	4	2.96
	5	2.89
	8	3.08
	12	3.53
SEM pcf	4	5.29
	5	5.07
	8	5.55
	12	6.19

Table 4.1: Summary of the results from the CPU benchmark. The abbreviations are: smf="stiffness matrix formulation", acf="acceleration formulation", pcf="predictor-corrector formulation". Markers are printed every second sampling point.

4.3.2 Comparison of Spectral Element Methods Using Different Time Integration Schemes

In the following several types of time integration schemes used in the SEM are compared to find out which one is performing best. The implemented time schemes are the explicit FD 3-point displacement formulation, an explicit Newmark scheme in acceleration formulation using no iterations and a predictor-corrector Newmark scheme with one iteration. For details on these time schemes see Section 2.1.8 and Komatitsch [1997]; Komatitsch & Vilotte [1998]. For general introduction into this topic see for example Hughes [1987].
All simulations were done for a homogeneous medium. Results of simulations of heterogeneous media differ slightly but do not change the outcome of this comparison significantly.
The performance is defined in terms of CPU cost as explained above. Nevertheless it is important to know about the accuracy itself. Therefore two plots are shown for each comparison. The first one gives the difference in relative solution error ϵ plotted against points per wavelength and numbers of propagated wavelengths. From these plots one can clearly distinguish the areas in which one or the other method is more accurate. Colour scales are chosen in such a way that red colours denote positve and blue colours denote negative differences with white colours giving zero. Positive difference means here that the method mentioned first is less accurate than the second method and vice versa. Sometimes differences appear to be either positive or negative without changing signs indicating that one method dominates for all values of *ppw* and *npw*.
The second plot of each comparison illustrates the relative CPU cost of the methods. Here the surfaces, derived like in Section 4.2 by inverting the accuracy surface and multiplying by the effort factor, are divided point by point. Thus, values higher than one show areas where the method mentioned first is more costly than the second, whereas values smaller than one indicate areas where the first method is less accurate. For better distinction the colour scales are defined that white colours indicate the value 1, red are values higher and blue values smaller than 1. The surfaces are only plotted for values of the error ϵ in the range 0 to 1 as higher errors are typically unacceptable.
All comparisons in this subsection are based on SEM simulations with order $N = 5$ Lagrange polynomials. The difference in performance of different orders will be investigated in the next subsection.
The abbreviations used in the following are:

DF displacement formulation

ACF acceleration formulation

PCF predictor-corrector formulation

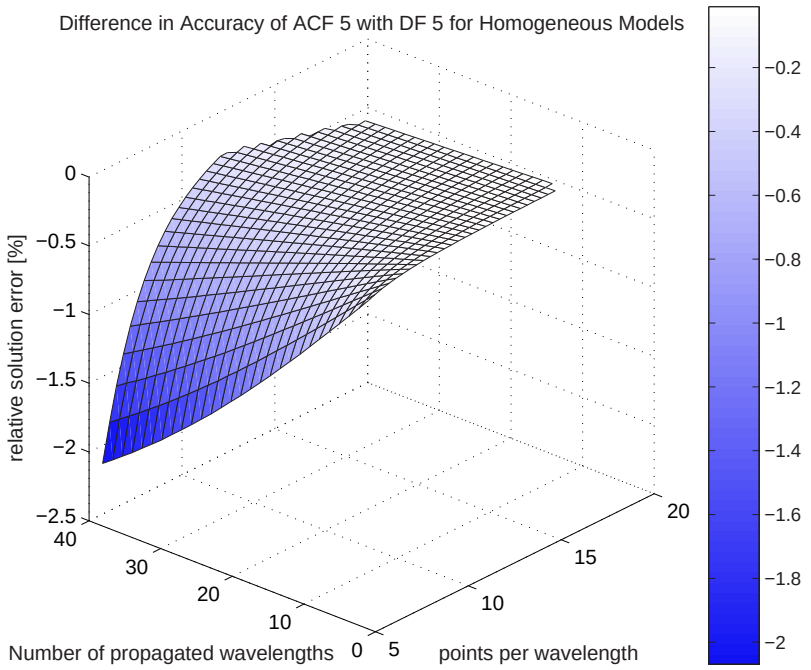


Figure 4.6: ACF - DF: Difference in accuracy of the acceleration and displacement formulation of the SEM. Using an acceleration method leads to higher accuracy for all values of ppw and npw. This can be seen from the values of the surface, which are all less than zero as given by the colour scale.

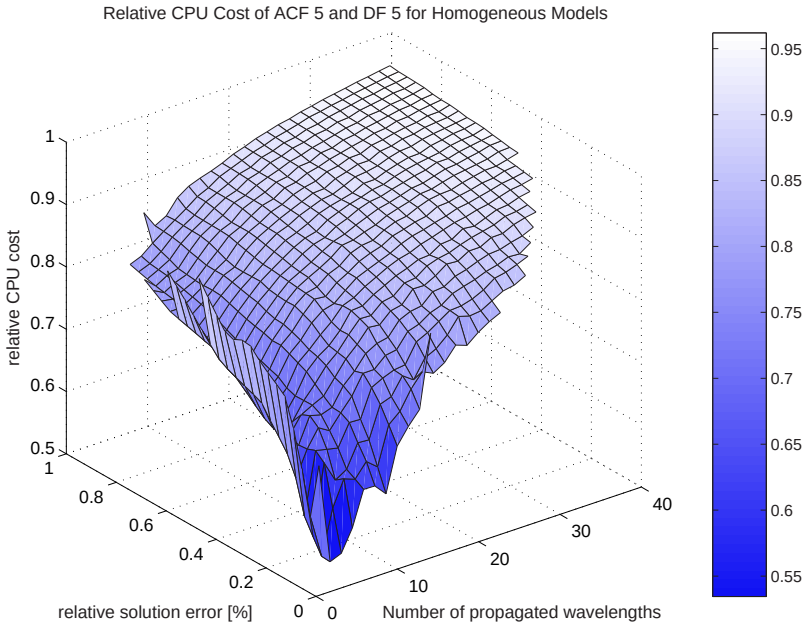


Figure 4.7: ACF - DF: Relative CPU cost of the acceleration and displacement formulation of the SEM. The acceleration formulation shows less CPU cost for all values of npw and ϵ as the values of the cost surface are all smaller than 1. Particularly for smaller errors and small numbers of propagated wavelengths it performs better.

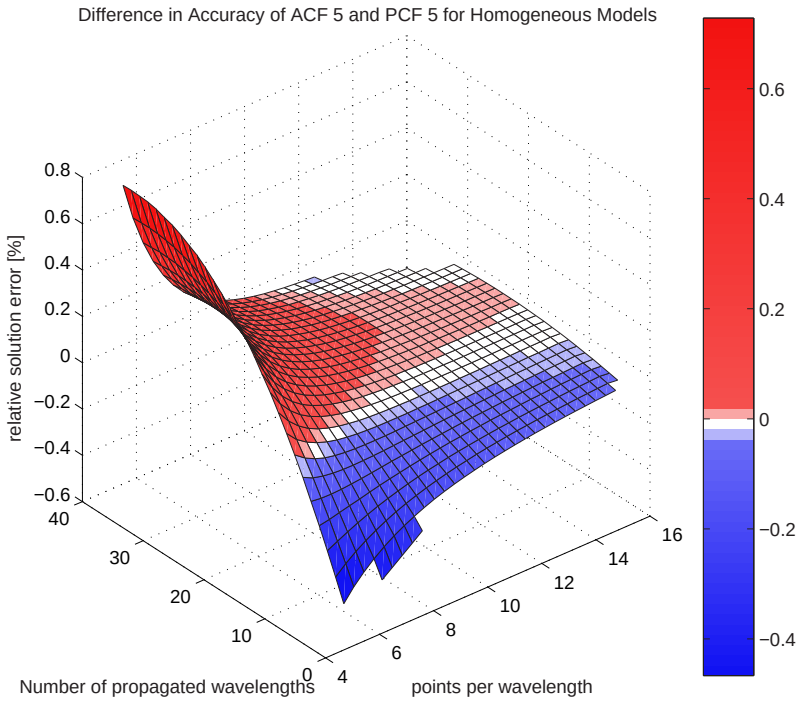


Figure 4.8: ACF - PCF: Difference in accuracy of the acceleration and predictor-corrector formulation of the SEM. Here the acceleration formulation is only better for a smaller number of propagated wavelengths, indicating that the predictor-corrector formulation is more accurate for greater distances. The difference is getting less with increasing number of ppw. At values of $ppw = 10 - 12$ finally, the ACF is getting better than the PCF again for great numbers of npw.

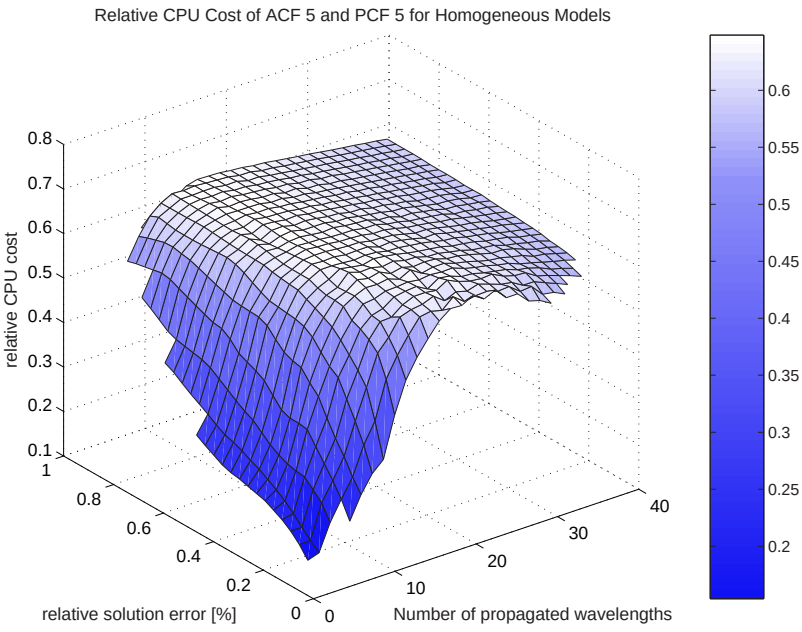


Figure 4.9: ACF - PCF: Relative CPU cost of the acceleration and predictor-corrector formulation of the SEM. This comparison indicates that the ACF performs much better in terms of CPU cost. The whole surface lies below 0.65.

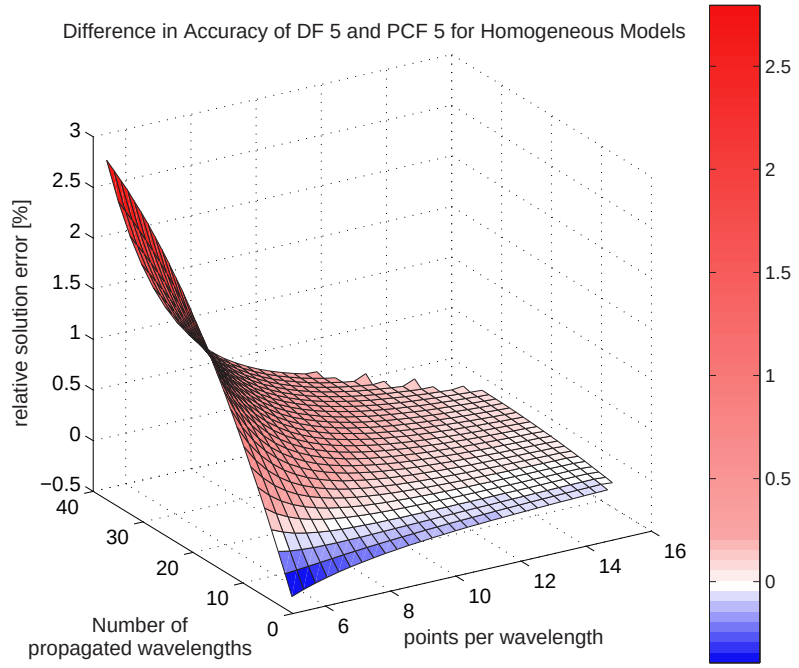


Figure 4.10: DF - PCF: Difference in accuracy of the displacement and predictor-corrector formulation of the SEM. The behaviour of this surface is relatively similar to Figure 4.8, showing higher errors for DF than for the PCF for most areas. Only within a small band of the first few *npw* the DF is more accurate. As expected the difference in accuracy is decreasing with increasing *ppw*.

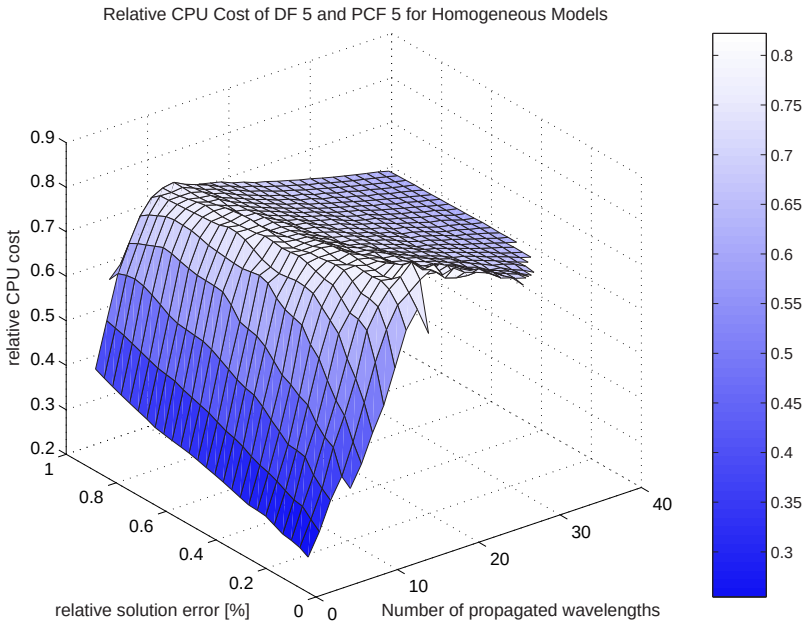
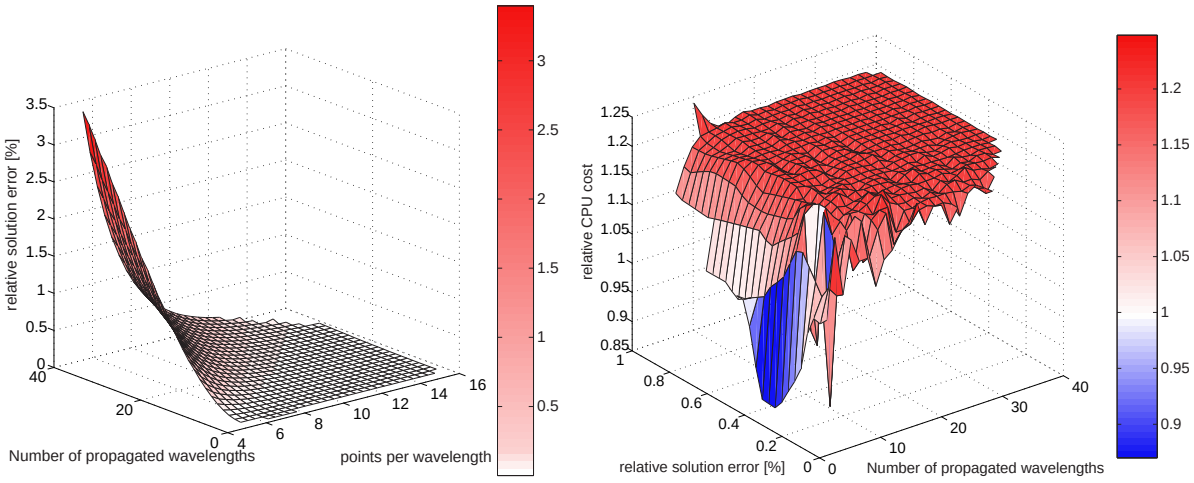


Figure 4.11: DF - PCF: Relative CPU cost of the displacement and predictor-corrector formulation of the SEM. This figure shows that the PCF is again more time consuming as the values of the surface are everywhere smaller than 0.82.

Combining the results of the three upper comparisons, it can clearly be seen that the ACF exhibits the best performance, although it is not the most accurate one for all tested numbers of *npw* and *ppw*.

4.3.3 Comparisons of Accuracy and CPU Cost for different orders of Spectral Elements

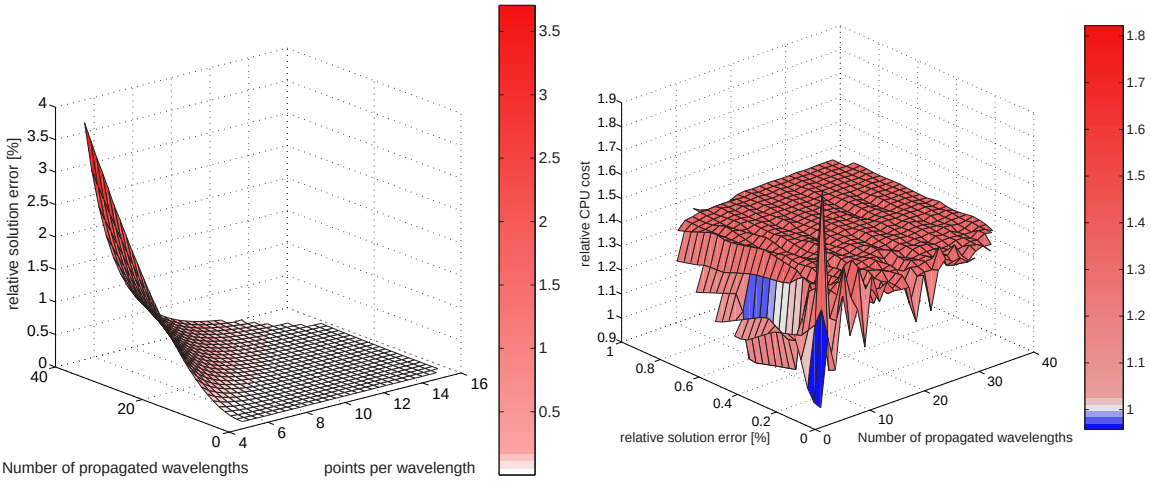
Following the comparsions of different time integration schemes, the performance of the SEM with different order of interpolating functions will be investigated. All results in this subsection were obtained from simulations using the acceleration formulation which showed the best performance as demonstrated above. Three different orders for the Lagrange polynomials were tested: $N = 4, 5,$ and 8. It is important to keep the value of the Courant number of 0.82 in mind, which was used for all orders. This is leading to different time steps dt, but is considered here to get the maximum performance of each SEM type.



(a) Difference in accuracy of SEM with $N = 4$ and $N = 5$.

(b) Relative CPU cost of SEM with $N = 4$ and $N = 5$.

Figure 4.12: Comparison of accuracy and CPU cost for SEM with order $N = 4$ and $N = 5$ of the interpolating Lagrange polynomials. Using a higher order is expected to be more accurate. Therefore, the differential surface is lying above 0 for all values of *npw* and *ppw*. In addition, order $N = 5$ is less expensive in CPU cost as almost all of the surface showing the relative cost is lying above 1.



(a) Difference in accuracy of SEM with $N = 5$ and $N = 8$.

(b) Relative CPU cost of SEM with $N = 5$ and $N = 8$.

Figure 4.13: Comparison of accuracy and CPU cost for SEM using Lagrange polynomials of order $N = 5$ and $N = 8$. Again, the higher polynomial order leads to higher accuracy for the whole area. Despite the fact that the effort factor is larger for order 8, the gained accuracy compensates for this and the overall CPU cost of order 8 is less than for order 5. Only at very few *npw*s the SEM with $N = 5$ seems to be more effective. Thus using order of 8 leads to a maximum in performance.

4.3.4 Comparison of Accuracy and CPU Cost of the Spectral Element Method with Optimal Operators

The final and most interesting part of the comparisons deals with the question if the SEM is more accurate and in addition less expensive in CPU cost than the Optimal FD Operators. The setup chosen for this comparison was described in Section 4.2. Simulations were performed for homogeneous models as well as for a "two layer model". As demonstrated in Section 4.3.2 the best performing SEM type is the one based on an acceleration formulation of the time integration. Therefore, the comparisons shown here were done using this type of method. We have seen that the most accurate and best performing SEM was the one using polynomials of degree eight. Higher orders would surely perform better, but they make less sense for real applications, as the value for the time step dt, calculated using the stability criterion, depends on the minimum grid spacing. Thus choosing high orders leads to very low values of dt and therefore needs a lot more time steps to calculate the same physical time.
As mentioned earlier, the effective Courant number of the SEM using $N = 5$ was

around 0.5, which was also used for the OPO. These both methods thus have comparable time steps dt. To account for this, the effort factor of the better performing eight order SEM is corrected by the ratio $\frac{dt_{OPO}}{dt_{SEM8}}$ for the comparisons. The effort factor therefore increases from $3.08 \cdot 10^{-7}$ to $4.52 \cdot 10^{-7}$.

Homogeneous Models

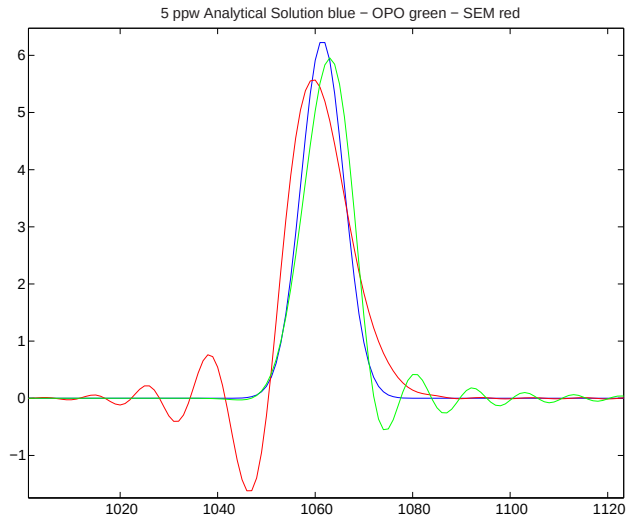


Figure 4.14: Filtered seismograms of station 100 for OPO (green) and SEM (red) for homogeneous models (5 *ppw*). The analytical solution is plotted in blue.

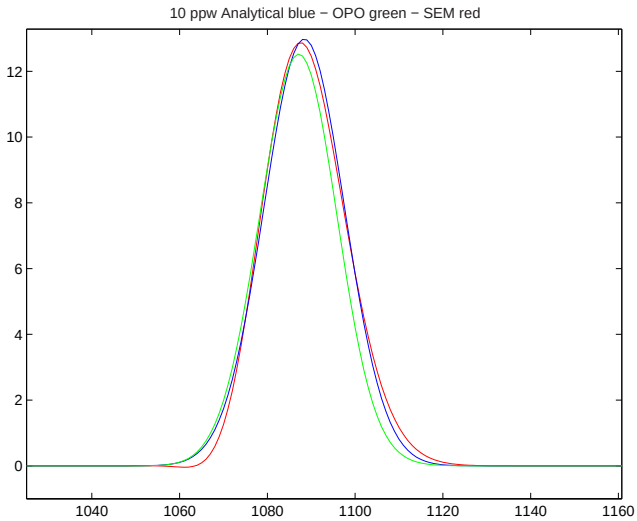
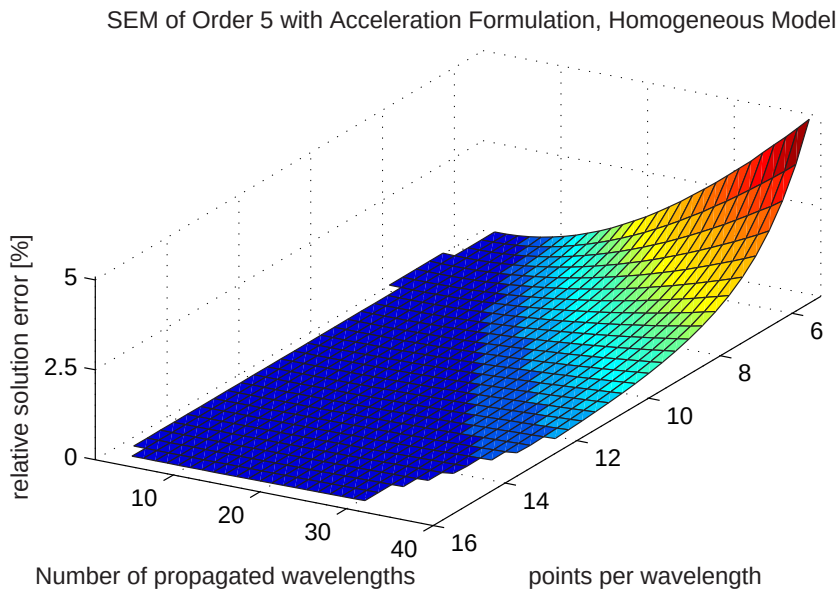
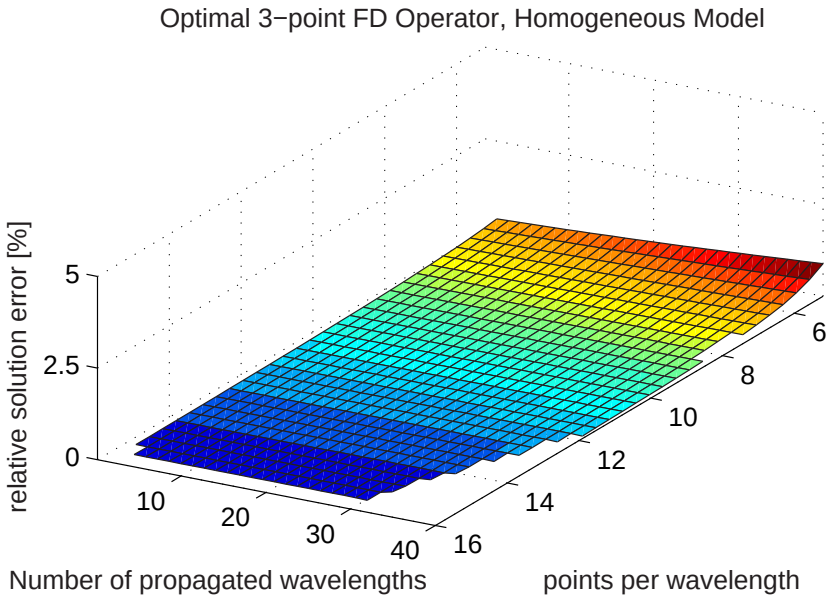


Figure 4.15: Filtered seismograms of station 100 for OPO (green) and SEM (red) for homogeneous models (10 *ppw*). The analytical solution is plotted in blue.

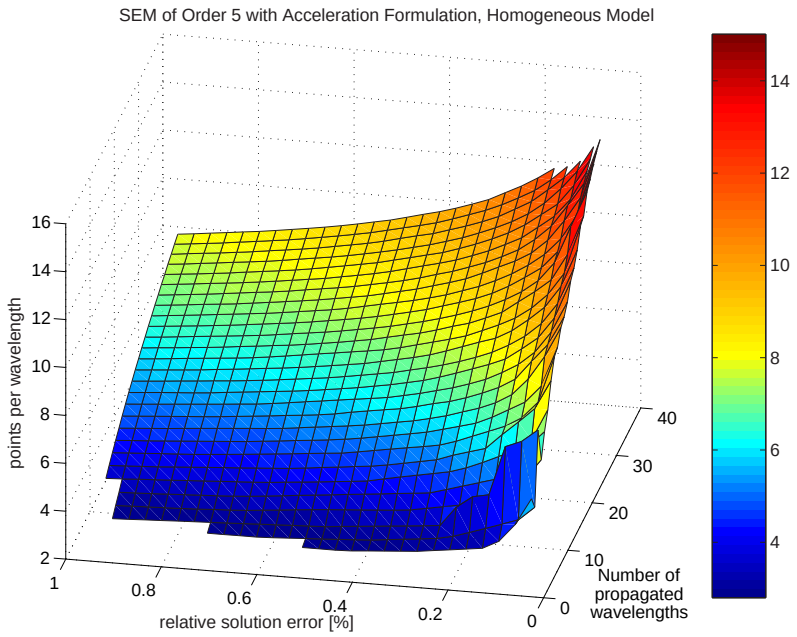


(a) SEM ACF order $N = 5$.

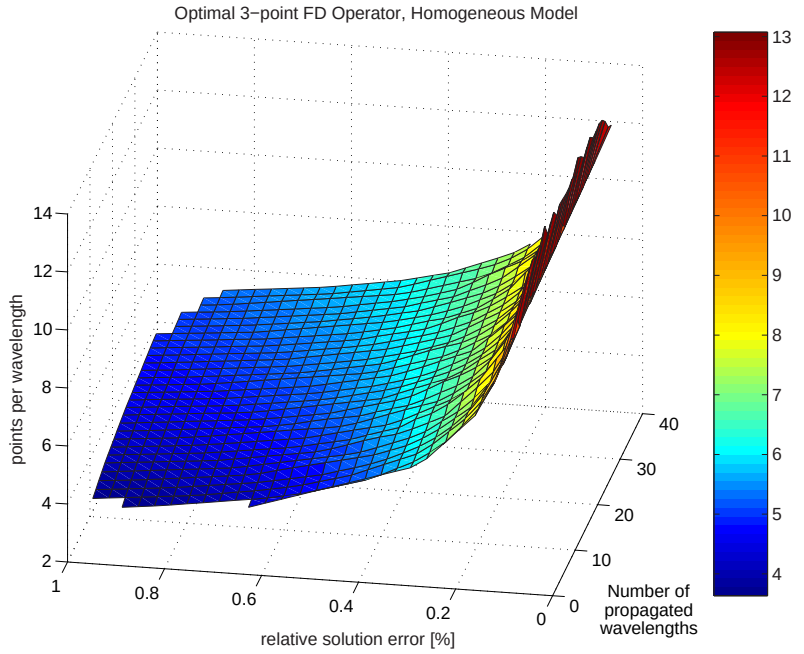


(b) OPO.

Figure 4.16: Relative solution error ϵ of SEM (a) and OPO (b) for homogeneous models. The scale of the z-axis is the same for both plots, only the colouring is different. One can see the different behaviour of the methods with changing *ppw* and *npw*. On the one hand the OPO show an almost linear increase with distance, whereas the error of the SEM grows exponentially. This indicates the optimally reduced numerical dispersion of the former method. On the other hand, the SEM shows a better behaviour with increasing *ppw* but overall OPO are mor accurate.

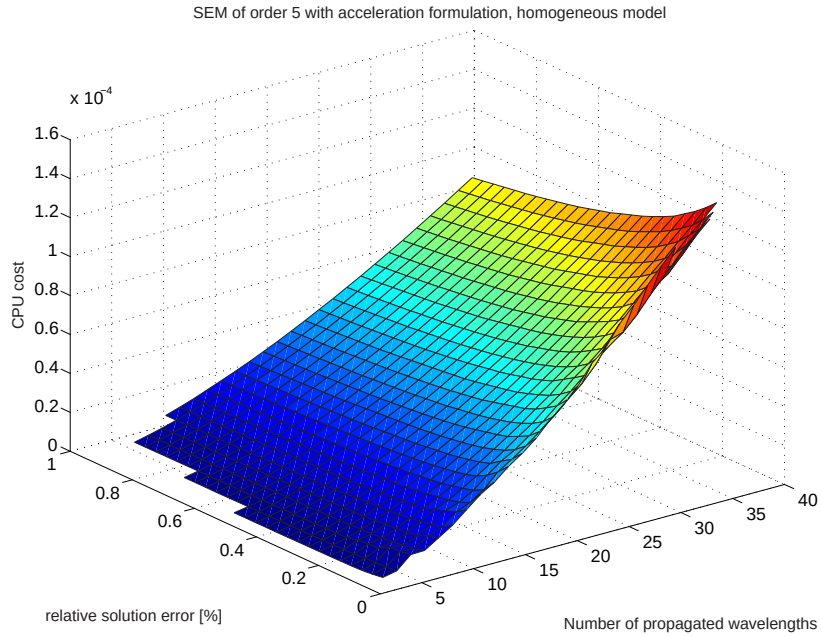


(a) SEM ACF $N = 5$.

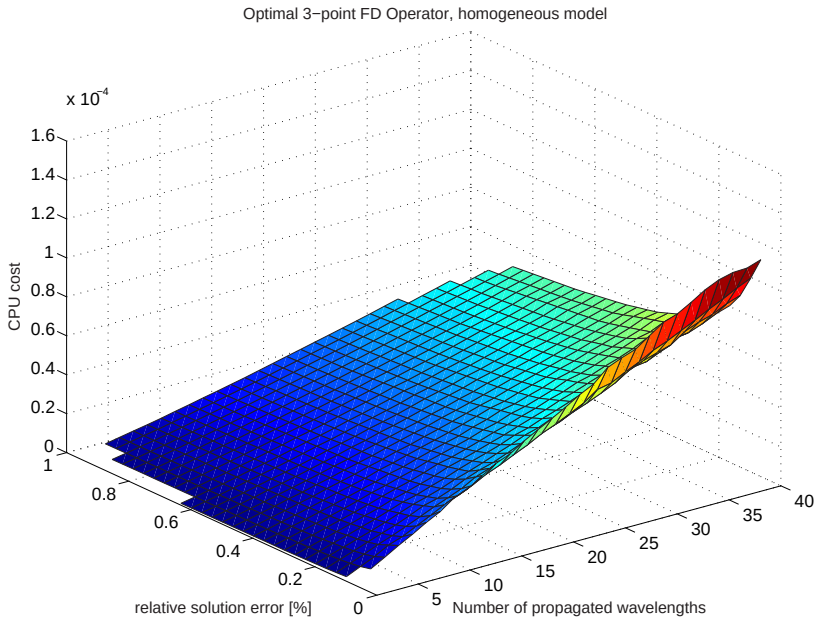


(b) OPO.

Figure 4.17: Minimum points per wavelength needed by SEM (a) and OPO (b) for homogeneous models to achieve a certain error and distance. Colour scales are the same and the range is given on the right. The slope of the SEM surface indicates that the *ppw* needed are almost the same in the range of ϵ plotted here. The Optimal Operator surface grows exponentially with decreasing error. Considering the behaviour of the methods with increasing distance *npw*, again the OPO appear to be better, as the number of *ppw* increases slower with distance as that of the SEM.



(a) SEM ACF $N = 5$.



(b) OPO.

Figure 4.18: Overall CPU cost of the SEM (a) and the OPO (b) for homogeneous models. Obviously the OPO are less expensive in CPU cost than the SEM of order $N = 5$.

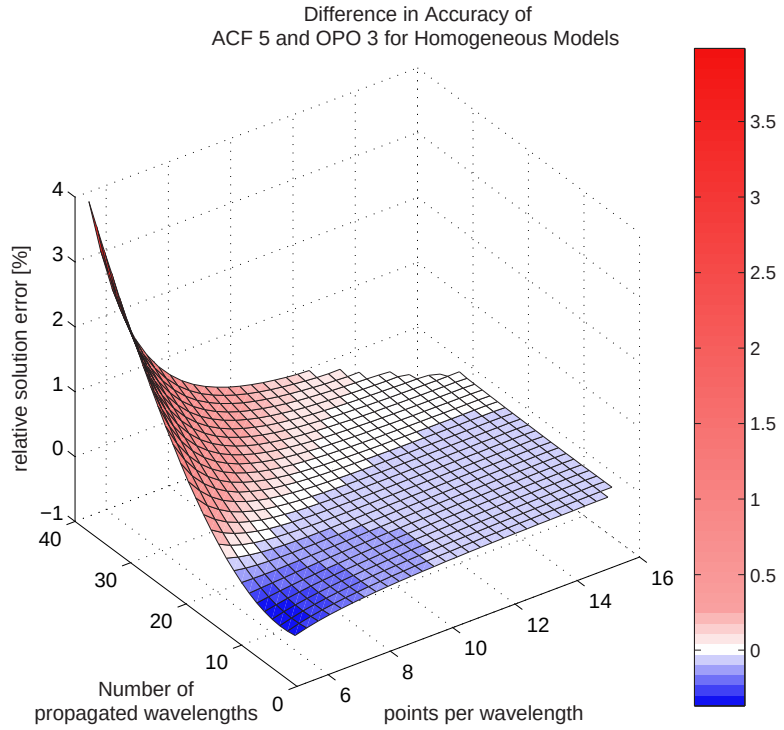


Figure 4.19: Difference in accuracy of SEM ACF $N = 5$ and OPO for homogeneous models. The colour scale is chosen in such a way that zero difference is given in white, positive values in red and negative values in blue. One can see that the SEM is more accurate for less than 10 *npw*. The fact that this number increases steadily up to around 20 when using 16 *ppw* or more again shows that the OPO are more suited for propagating waves for long distances.

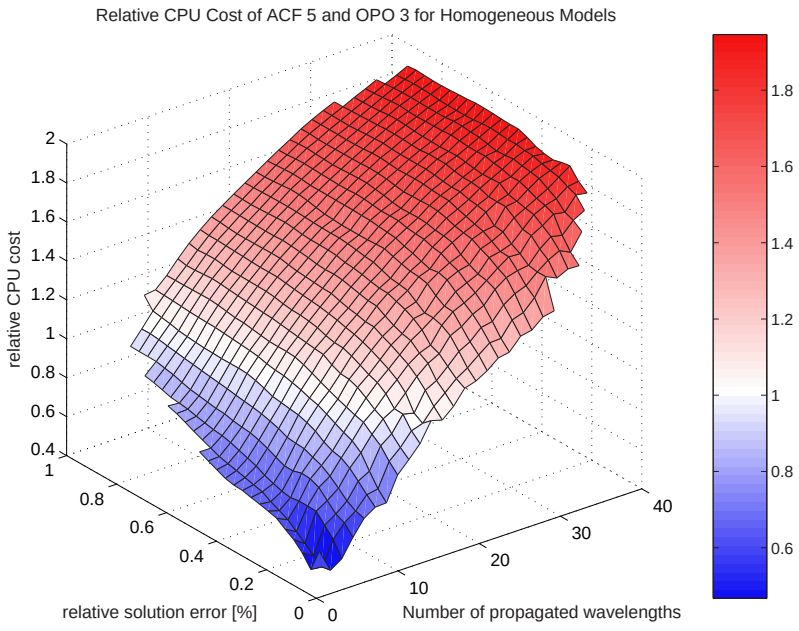


Figure 4.20: Relative CPU cost of SEM ACF $N = 5$ and OPO for homogeneous models. For most cases the OPO need less CPU time per time step since the ratio of $\frac{CostSEM5}{CostOPO}$ is then greater than 1. Only for less than 10 *npw* and for very small errors the SEM appears to be more effective.

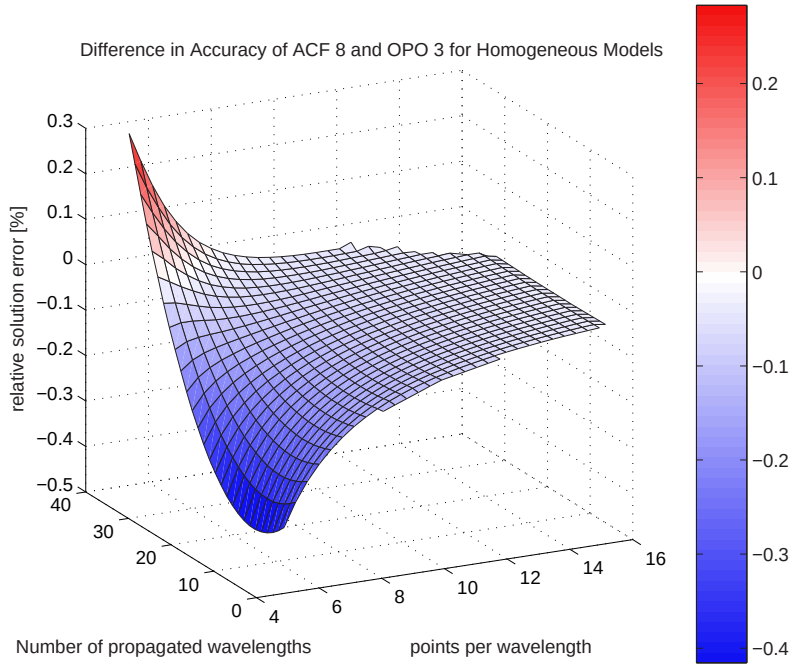


Figure 4.21: Difference in accuracy of SEM ACF $N = 8$ and OPO for homogeneous models. Here the SEM of order eight is much more accurate than the OPO in most of the plotted area. This demonstrates the "spectral" behaviour of the SEM, which means that the solution converges in the same manner as the pseudo-spectral methods with increasing N.

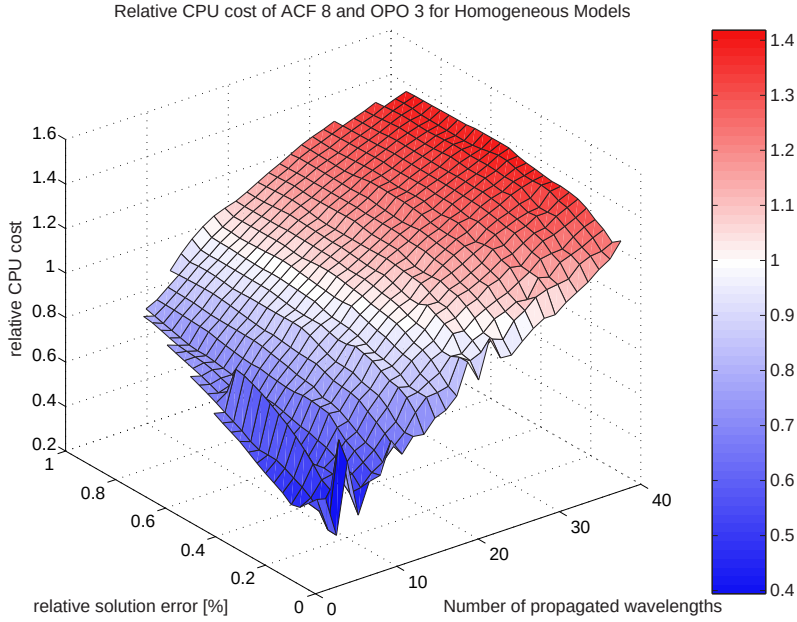


Figure 4.22: Relative CPU cost of SEM ACF $N = 8$ and OPO for homogeneous models. In contrast to the SEM with $N = 5$, using a higher order leads to a better performance. This can be seen from the bigger area, in which the SEM of degree 8 is less expensive in CPU Cost than the OPO, indicated by blue colours. Note that the CPU Cost of SEM $N = 8$, where a Courant value of 0.82 was used is plotted here. Thus the time steps between SEM $N = 8$ and OPO differ. Figure 4.23 shows the comparison in CPU cost, when using the same time step.

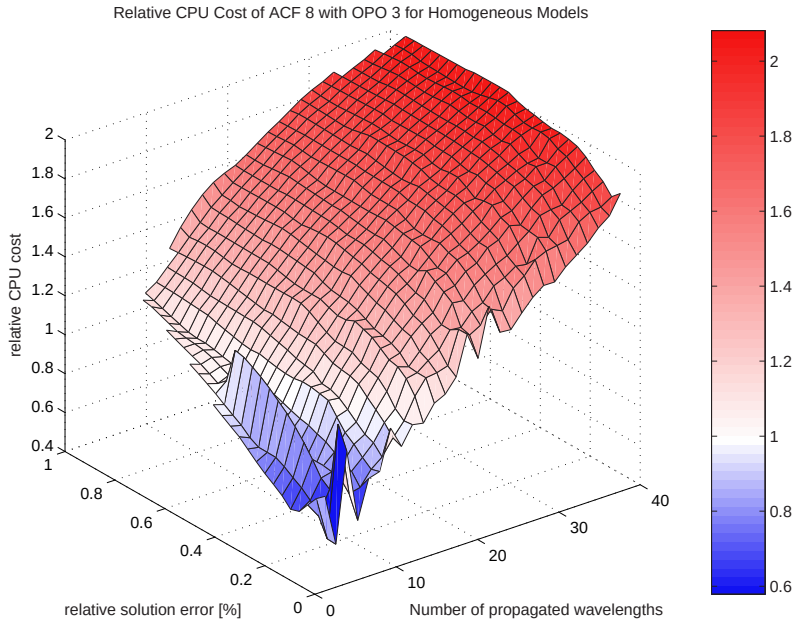


Figure 4.23: Relative CPU cost of SEM ACF $N = 8$ and OPO using the same time step for homogeneous models. This figure illustrates that the performance of the SEM of order eight gets worse when using other time steps. The ratio of $\frac{CostSEM8}{CostOPO}$ is rather the same as the ratio of $\frac{CostSEM5}{CostOPO}$ as can be seen when comparing this plot to the surface in Figure 4.20 (same range for the z-axis).

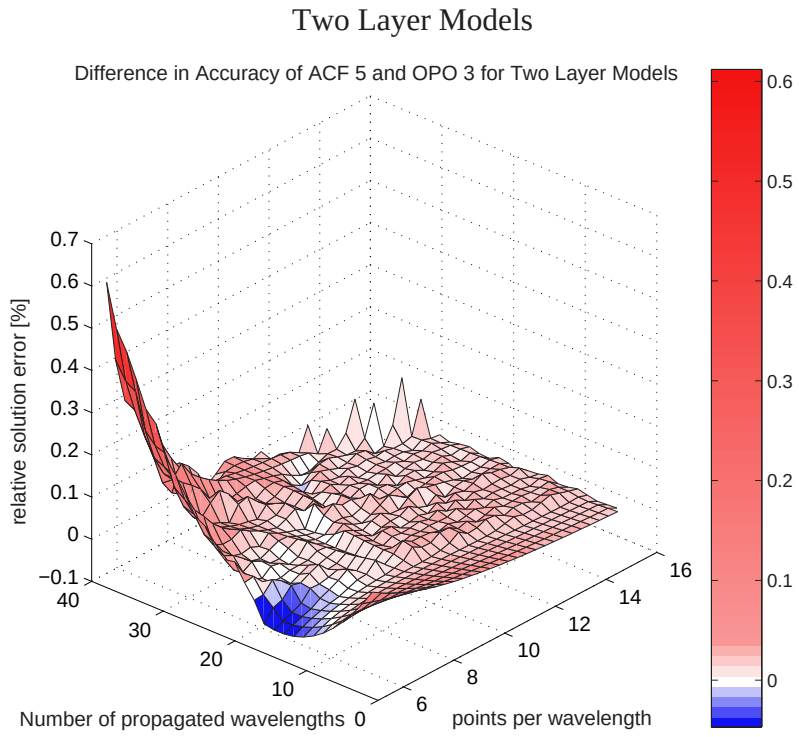


Figure 4.24: Difference in accuracy of SEM ACF $N = 5$ and OPO for heterogeneous models. The differential surfaces of the error ϵ show much more noise. Nevertheless, it is evident that the SEM of order $N = 5$ is less accurate compared to the OPO than for homogeneous models.

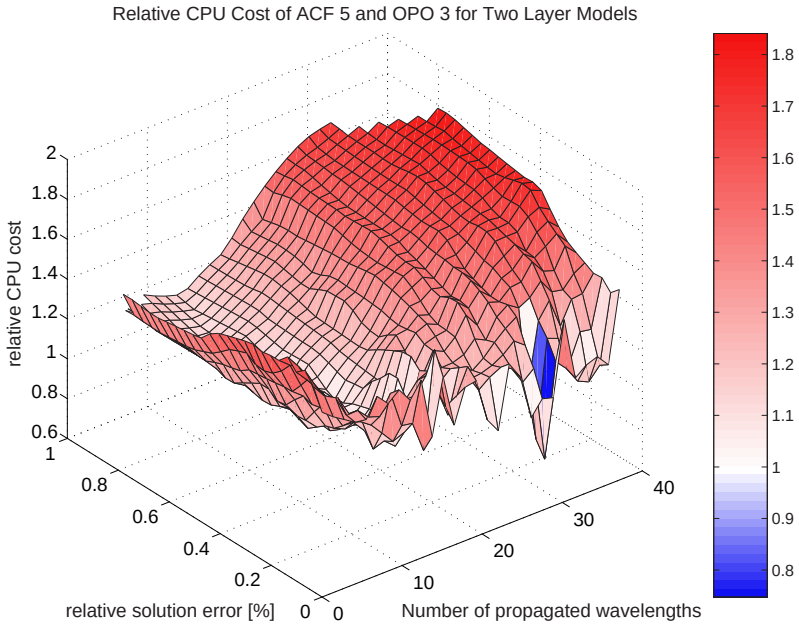


Figure 4.25: Relative CPU cost of SEM ACF $N = 5$ and OPO for heterogeneous models. This figure indicates that the OPO are less expensive than the SEM $N = 5$ for almost all values of npw and ppw. One area can be distinguished where the surface shows a minimum close to $z = 0$.

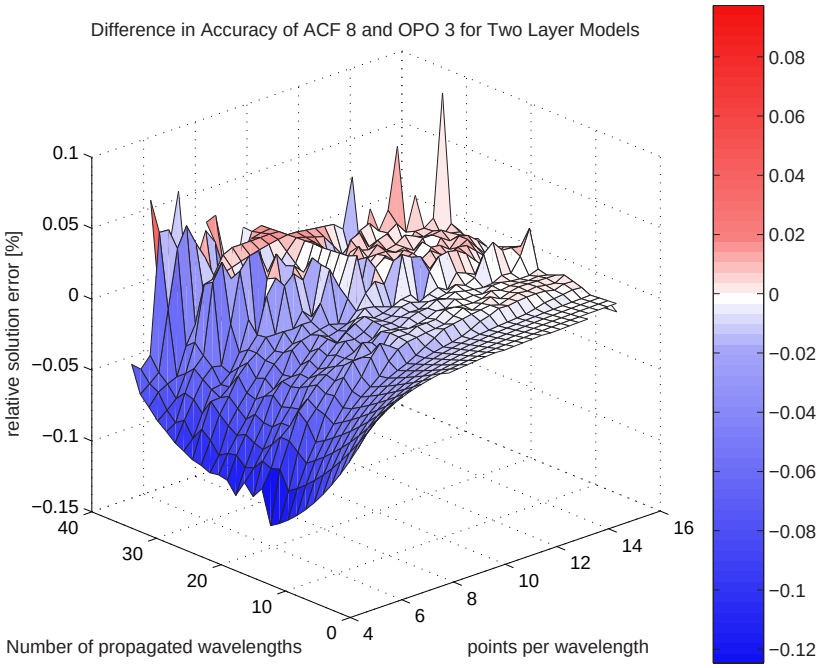


Figure 4.26: Difference in accuracy of SEM ACF $N = 8$ and OPO for heterogeneous models. The accuracy of SEM with order eight is better than of the OPO in a great part of the plot. The step that can be seen in the middle probably results from the interface of the model regions, where wave velocities change. In this area the error shows increased noise and the OPO seem to gain accuracy. The extremely high values in the back come from errors in the evaluation routines.

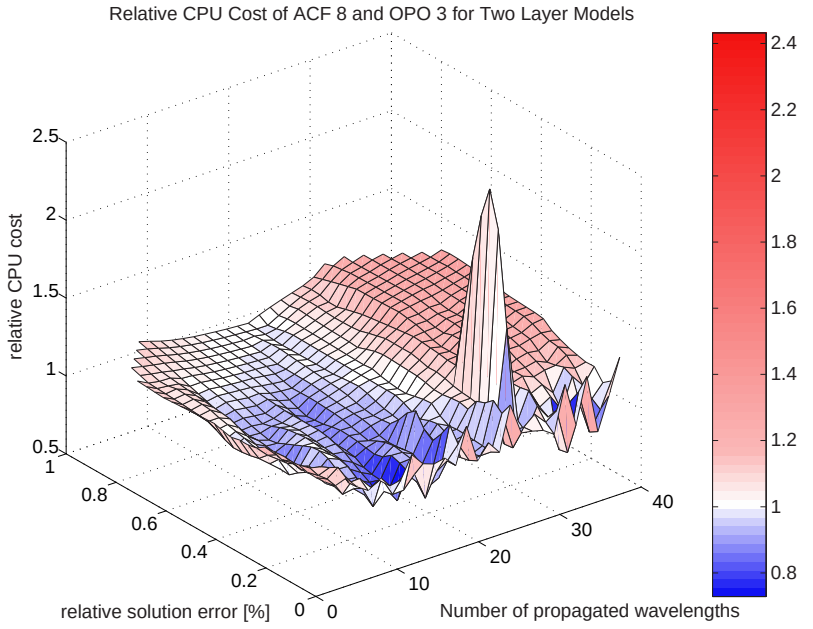


Figure 4.27: Relative CPU cost of SEM ACF $N = 8$ and OPO using the same time step for heterogeneous models. This figure shows again a minimum in the ratio $\frac{CostSEM8}{CostOPO}$, but this time it is more prominent and thus the CPU cost of both methods is more balanced. The peak in the middle probably derives from the high values seen in the surface of the relative solution error.

Chapter 5

Evaluation of the Spectral Element Method in 3-D

In the previous chapter the comparisons between different kinds of spectral element methods and between the best performing one and the "optimal operators" have shown some significant characteristics. The OPO was the method with the lowest CPU time per time step and, in most cases, also the best performing one. The definition of performance was a combination of accuracy and CPU time per time step, called CPU cost.
It is obvious that these comparisons of 1-D simulations can only be a first step. Extrapolation of the behaviour to higher dimensions is not possible without further investigation. In addition, memory demands play an important role now as memory is more limiting than time. Thus, comparisons of 3-D simulations should also contain a test of the accuracy achieved for the same computer memory instead of equally sized models.
In this chapter the results of three-dimensional SEM and FD simulations are presented using explosive and dip-slip sources in a homogeneous rectangular block model. These simple setups allow for the same method of comparison used for the 1-D simulations. Moreover, an interpretation of the results is easier than for complex models.
The choice of the model size was restricted due to the memory available, which was 2 GB for a single machine. In the beginning only a serial SEM code was used, which led to a maximum number of 60, 45 and 30 elements in the $x, y,$ and z direction respectively. The polynomial degree chosen for the spectral elements was $N = 5$. Thus, the overall number of nodes in the grid was 241 x 181 x 121. The memory required was 1.6 GB. In contrast to that, the FD code only needed 0.6 GB for the same overall number of grid points, leading to a ratio of 2.66. The dimensions of the model were 800 by 600 by 400 kilometer leading to cubic elements. Although 2 GB would have been available, these configuration was chosen to achieve cubic elements and a ratio of $x, y,$ and z dimensions of 4:3:2 at the same time.

The source was located at $x = 275$ km and $y = 300$ km and two different depths for the source were chosen: 200 km and 10 km. The Latter position was used to excite strong surface waves. The locations of sources and receivers were chosen to allow for the same evaluation as in 1-D. Thus, the first receiver was located directly above the source and 149 receivers were evenly distributed along the x direction between 275 and 775 km away from the source. Figure 5.1 illustrates the model setup.

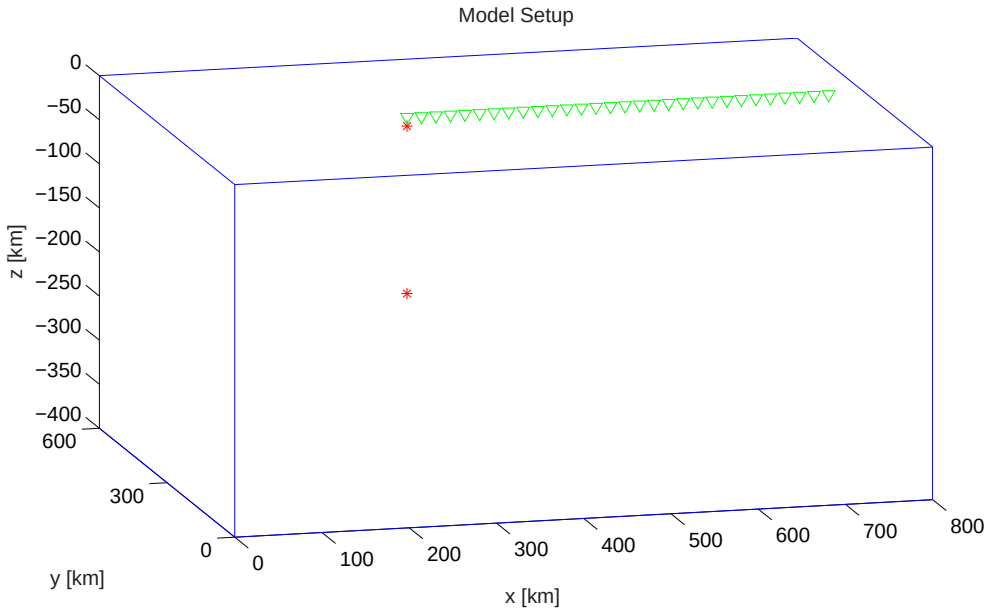


Figure 5.1: Model setup for 3-D simulations. The position of the sources is indicated by red stars. Every fifth receiver is plotted as green triangle.

The idea of this setup was to ensure a maximum distance between the last receiver and the source and at the same time to avoid reflections from the boundaries. Especially the x position of the source and first receiver were chosen in such a way that reflected waves coming from the boundary at $x = 0$ and from the sides at $y = 0$ and $y = 600$ arrive later than the direct wave at the last receiver. Thus, the seismograms could be evaluated for the same time window.
The time step used was 0.1 s and the wave velocities v_s and v_p of shear and compressional waves respectively were 3500 and 6062 $\frac{\text{m}}{\text{s}}$. These values correspond to a ratio of $\frac{v_p}{v_s} = \sqrt{3}$. Choosing the same time step for both methods leads to a similar effective Courant value. The number of time steps simulated was 950, thus covering 95 s. This simulation time is sufficient to record P-waves at the last receiver. S-waves are cut off for the last third of receivers. By doing so, a strong difference in accuracy for the receivers recording P and S-waves and for those, which only record P-waves, could be observed. This difference arises from the fact that S-waves are sampled with less points per wavelength and thus produce a

higher error. Although both methods provide the use of absorbing boundaries, all reflected waves are cut off to be on the save side, as the investigation of boundary effects was not the objective of this study. The seismograms were evaluated with respect to a quasi-analytical solution obtained from the program "Qseis" (Wang [1999]). This program only gives results for vertical 2-D cross-sections, which was the reason for the zero offset of the receiver array in the y direction with respect to the source.
In the first section of this chapter, some seismograms of the SEM are plotted to show the behaviour of this method and to give an example of the output compared. For the first case of an explosive source the seismograms of the quasi-analytical and FD simulations are plotted too, as this is the only case where a difference can be seen by just looking at the different time series. Afterwards, a CPU benchmark is shown and in the last section, comparisons of the accuracy and the derived overall CPU cost are presented. These comparisons are only shown for one component of the seismograms as no additional information can be derived from the others. The seismograms were shifted in time to align the P-phases and the different traces were normalized to the maximum value of all seismograms. As the definition of the source time function was different for the simulations using SEM and FDM, the phase shift is considered to assure equal odds.
The 3-D SEM program code was provided by Dimitri Komatitsch and is a simpler version of the CalTech [2002] code used for simulations in the Los Angeles basin (Komatitsch *et al.* [2003a]). The FD code was written by students of the seismology group of the Institute of Geophysics at the Ludwig-Maximilians-University Munich, which is based on a fourth-order velocity-stress formulation on a staggered grid (Virieux [1986]; Madariaga [1976]). Both methods were used with a second order time integration scheme.

5.1 Seismograms of the Simulations

From the elastodynamic equation one expects only P-waves to be excited in a homogeneous medium when using explosions. The direct P-wave as given by the quasi-analytical solution can be seen in Figures 5.2 and 5.2. When these P-waves reach the free surface P to SV converted waves are generated, which continue to travel along the surface as Rayleigh waves. These are visible for shallow source depths as in Figures 5.8 and 5.10 for explosive and dip-slip sources. The reflected waves are not recorded as all receivers lie on the surface of the model. The amplitude of the Rayleigh waves is expected to be higher for a source being closer to the surface. Thus, placing the source at 10 km depth allows for a good test of the representation of the free surface in both models.
For dip-slip sources both P- and S-waves are expected to travel inside the homogeneous medium (Fig. 5.9). For all cases the velocity at the receivers is plotted,

normalized to the highest amplitude of all seismograms. The y-axis of the plots gives the distance between source and corresponding receiver. The absolute values give the location with respect to the origin of the model and they differ between the methods as the FD model had to be defined from $-\frac{x}{2}$ to $+\frac{x}{2}$ and the SEM model from 0 to x. Only the x- and z-component are plotted as for this configuration no signal appears on the y-component.

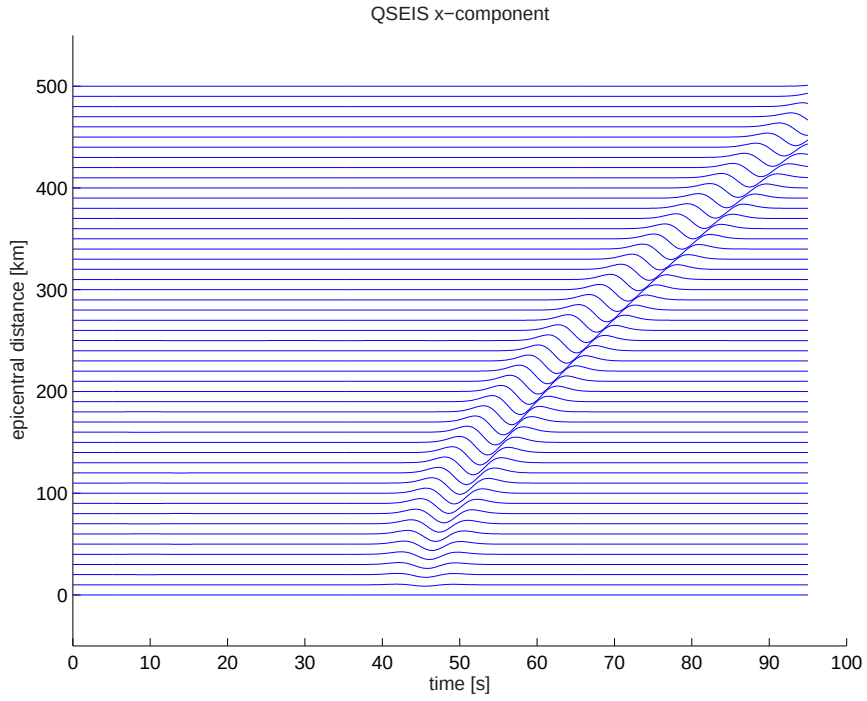


Figure 5.2: Quasi-analytical solution - Explosion at $z = -200$ km (x-component). Only a direct P-wave is observed.

The seismograms of the SEM simulations for an explosive source in 200 km depth show a slightly visible S-wave (5.6). This spurious phase becomes more prominent for less points per wavelength. In Figure 5.7(a) the seismograms were convolved with a ricker wavelet which dominant frequency corresponds to a number of 3 points per minimum wavelength. In addition, the theoretical arrival times for P and direct S-waves are plotted. The amplitude of the S-wave is affected by the relative position of the source inside an element. Figure 5.7(a) shows the seismograms for an arbitrary source position whereas Figure 5.7(b) derives from placing the source exactly at one of the GLL points. It turns out that this is an effect of the source implementation in the SEM formalism. The point source is distributed over the entire element, which leads in the most general case of an arbitrary source position to numerical anisotropy. This effect is reduced when placing the source at an GLL point, especially at position $(\xi, \eta, \gamma) = (0, 0, 0)$ (i.e. the middle of the element). An other way to reduce the effect is to decrease the

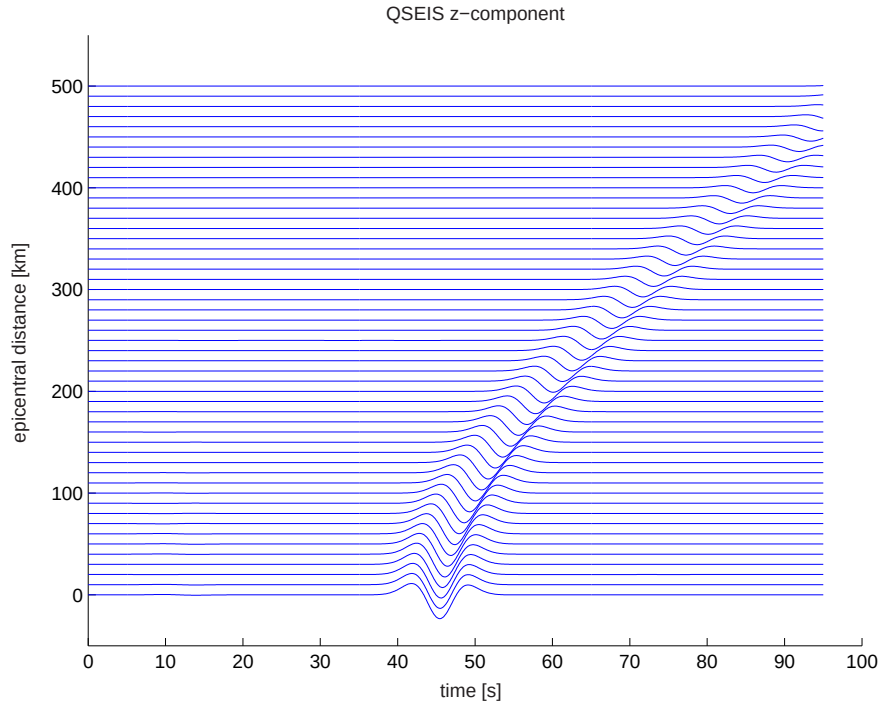


Figure 5.3: Quasi-analytical solution - Explosion at $z = -200$ km.

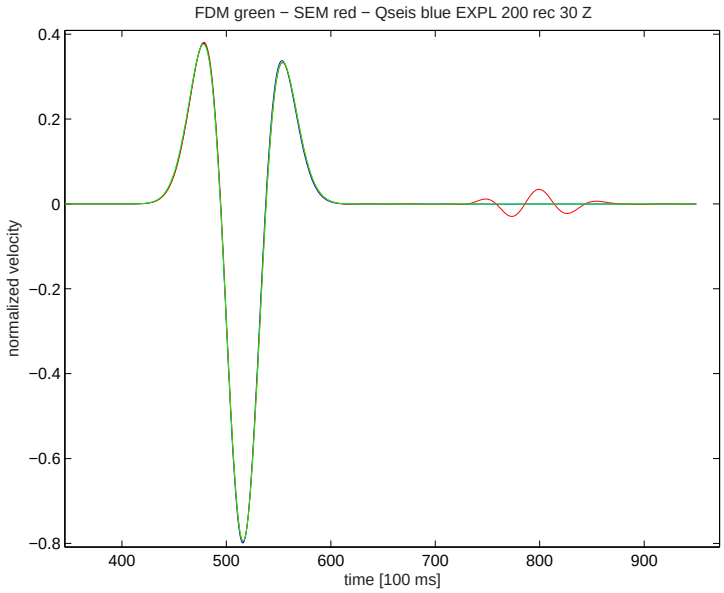
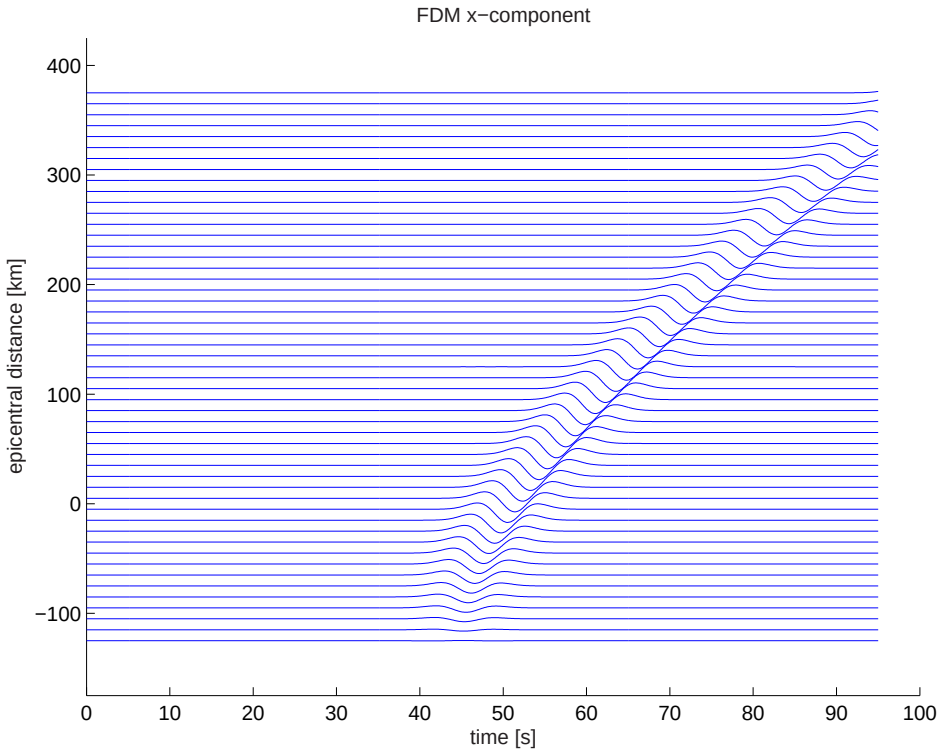
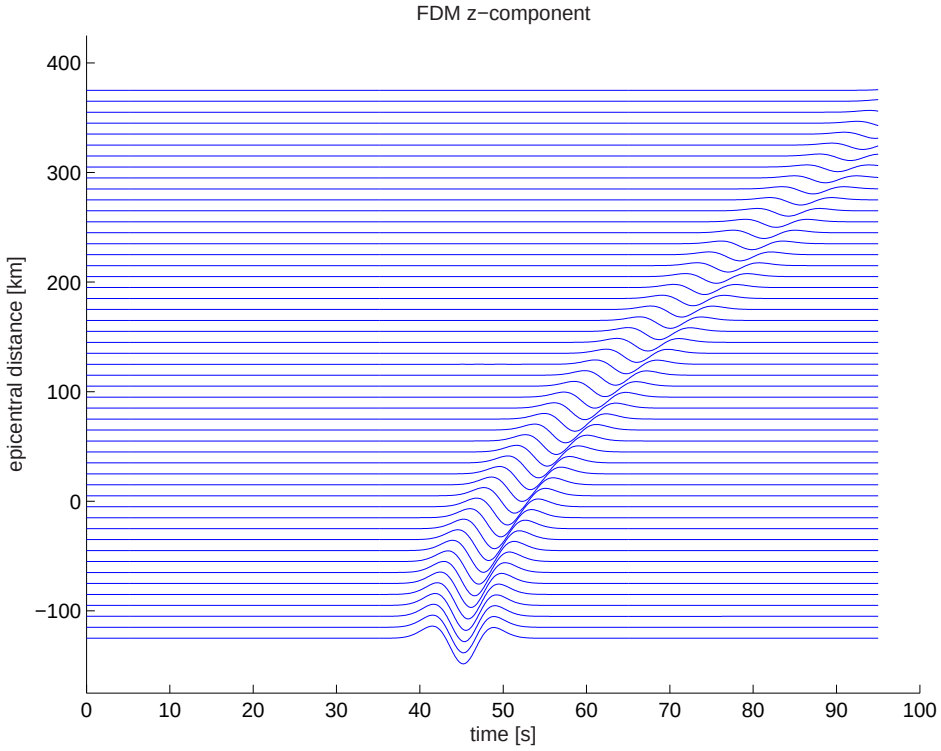


Figure 5.4: Explosion at $z = 200$ km: Comparison of seismograms at receiver 30 at a distance of 97 km.

maximum frequency exited by the source or by increasing the spatial sampling inside the element conataining the source (e-mail from Dimitri Komatitsch).

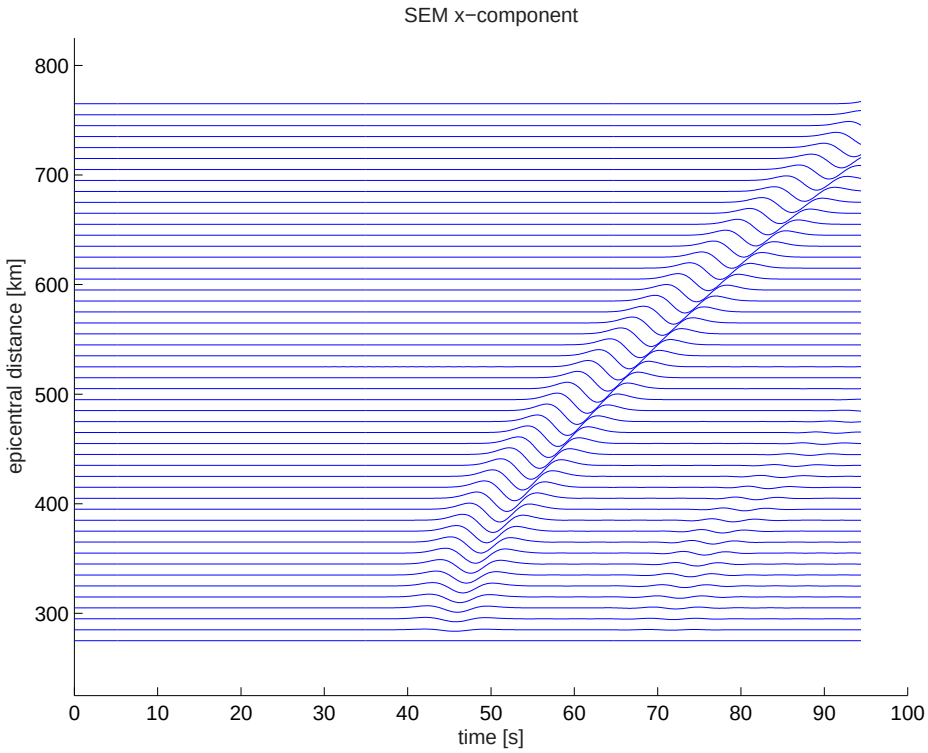


(a)

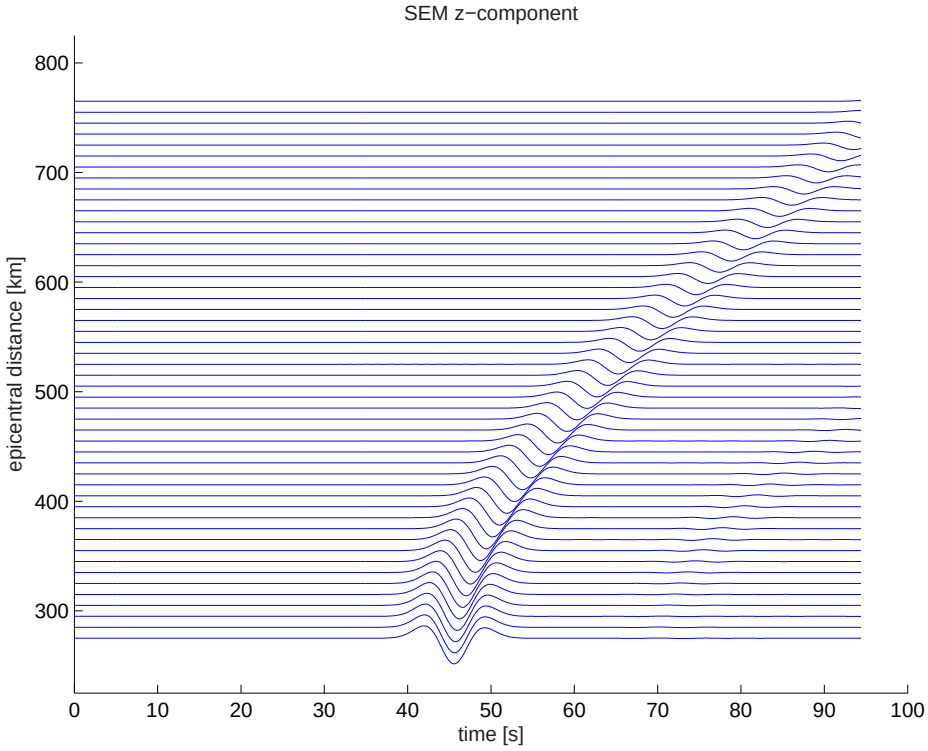


(b)

Figure 5.5: FD solution - Explosion at $z = -200$ km.

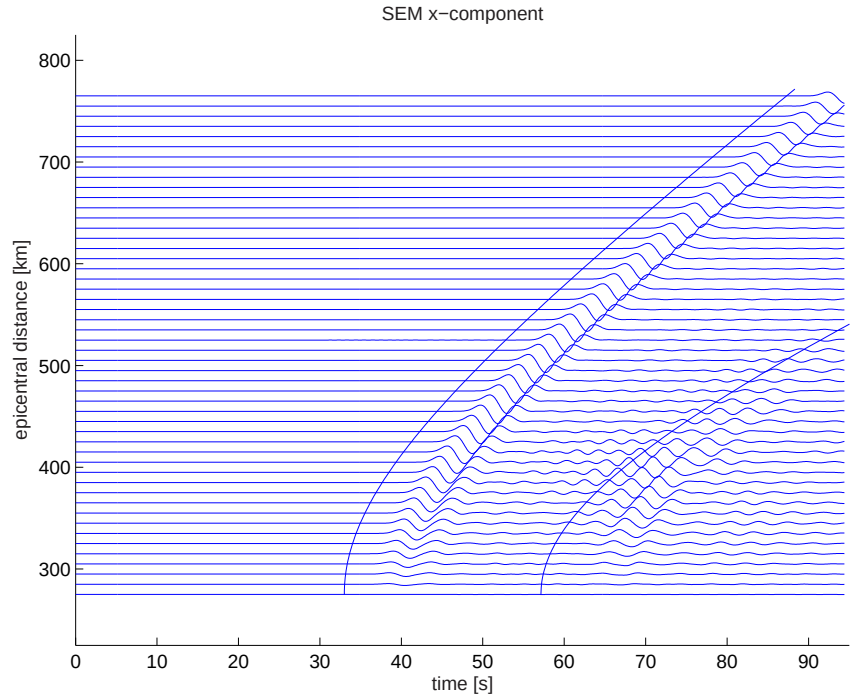


(a)

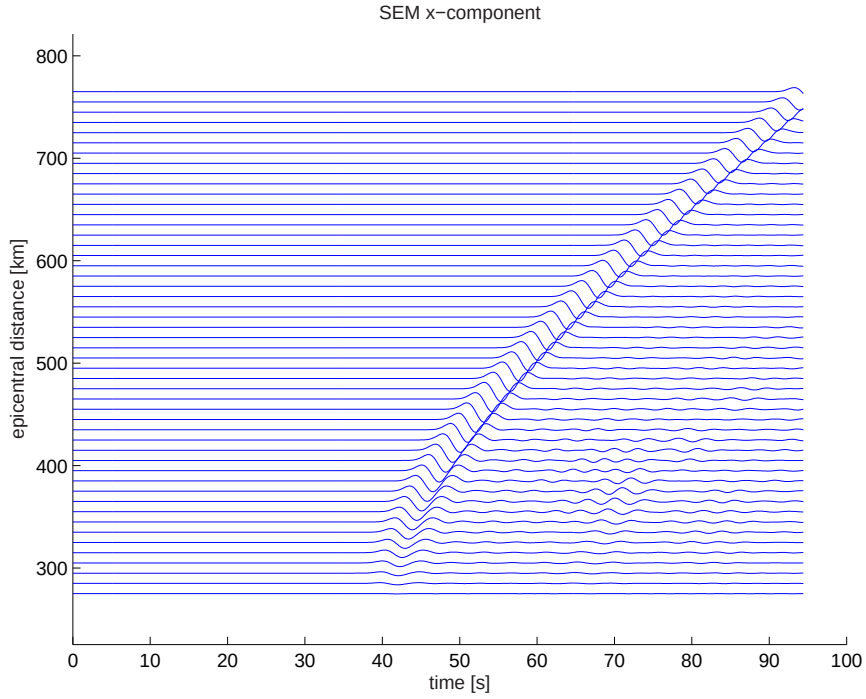


(b)

Figure 5.6: SEM solution - Explosion at $z = -200$ km. Here, a spurious S-wave can be clearly distinguished.

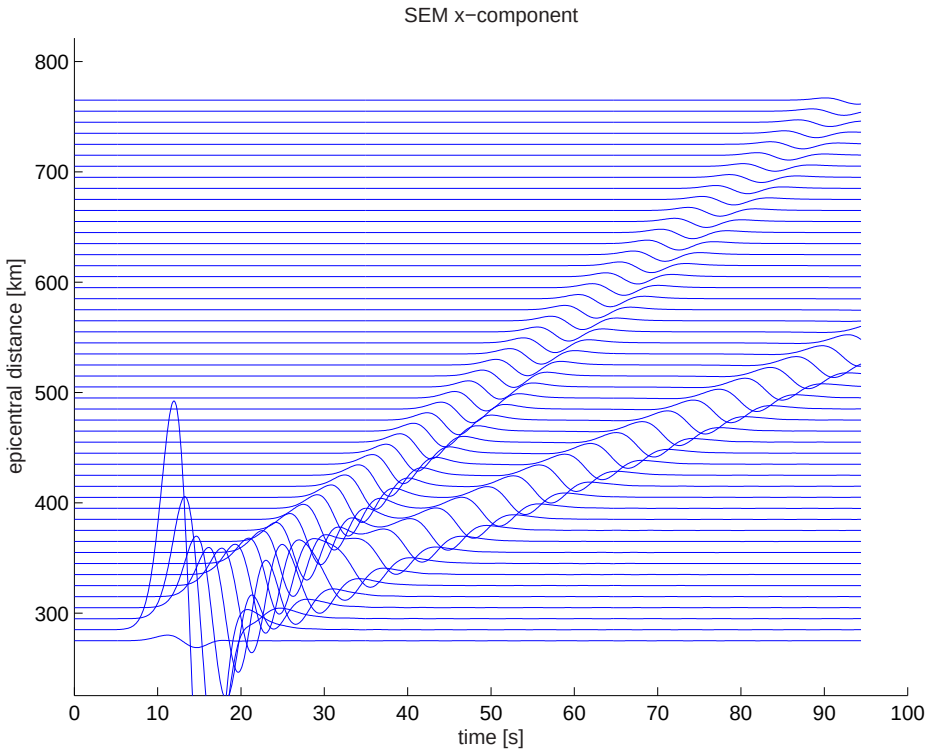


(a)

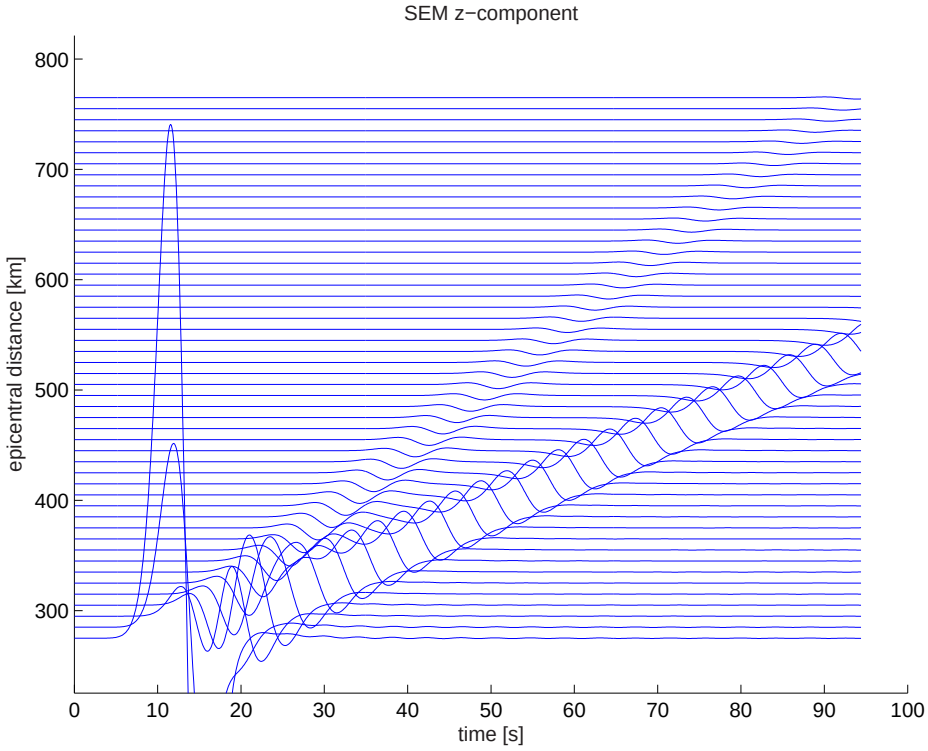


(b)

Figure 5.7: SEM solution - Explosion at $z = 200$ km. The spatial sampling frequency is reduced to 5 points per wavelength for the S-wave for which the erroneous phase is becoming more prominent. (a) Source is placed at an arbitrary position inside the element. Theoretical arrival times are plotted as curved blue lines across the seismograms for P- and S-wave. (b) Source is placed exactly at one of the GLL points $(\xi_2, \eta_4, \zeta 2)$.

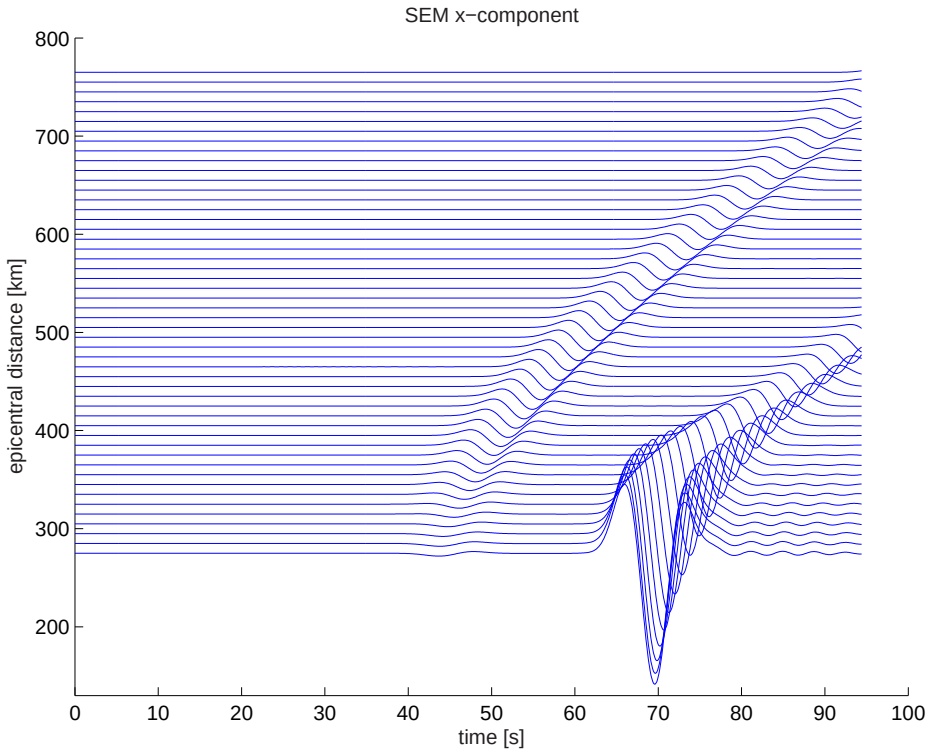


(a)

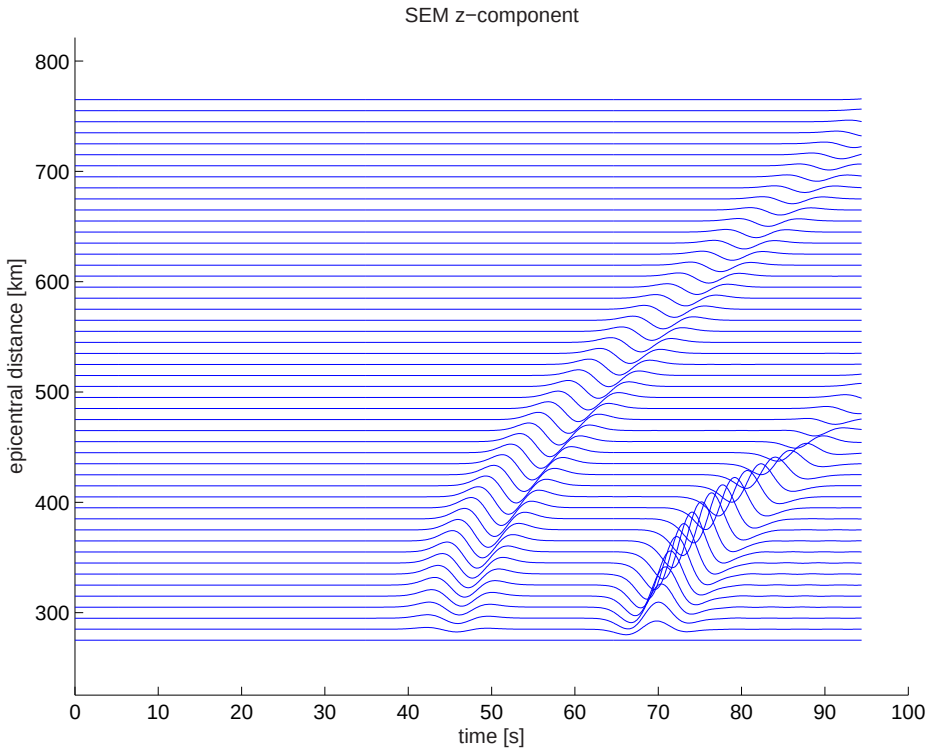


(b)

Figure 5.8: SEM solution - Explosion at $z = -10$ km. Additionally to the P-wave a strong Rayleigh wave can be observed.

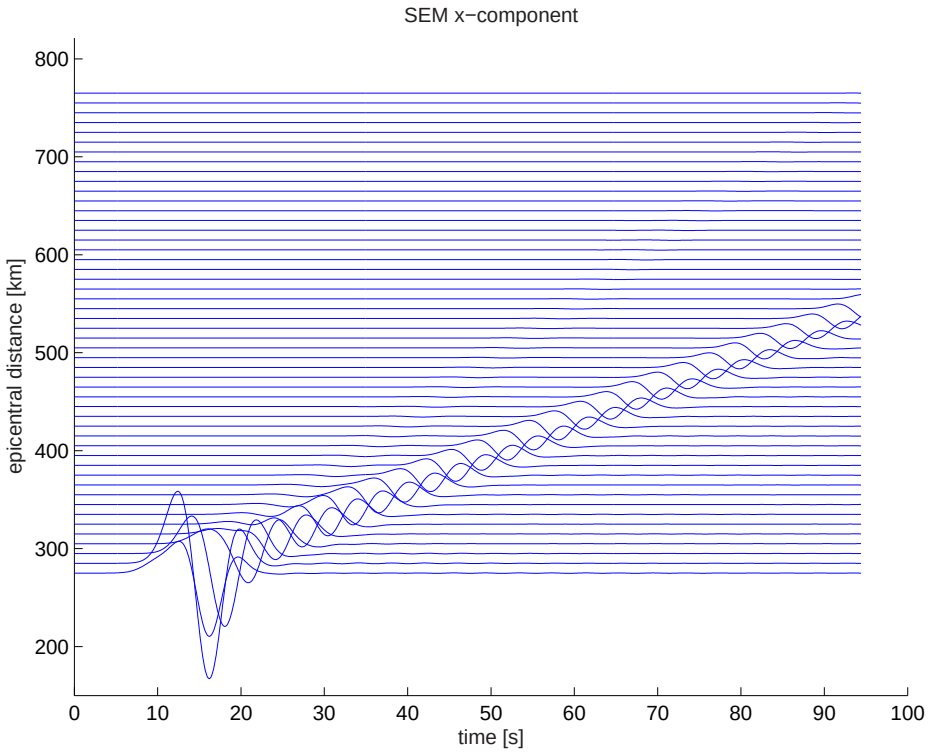


(a)

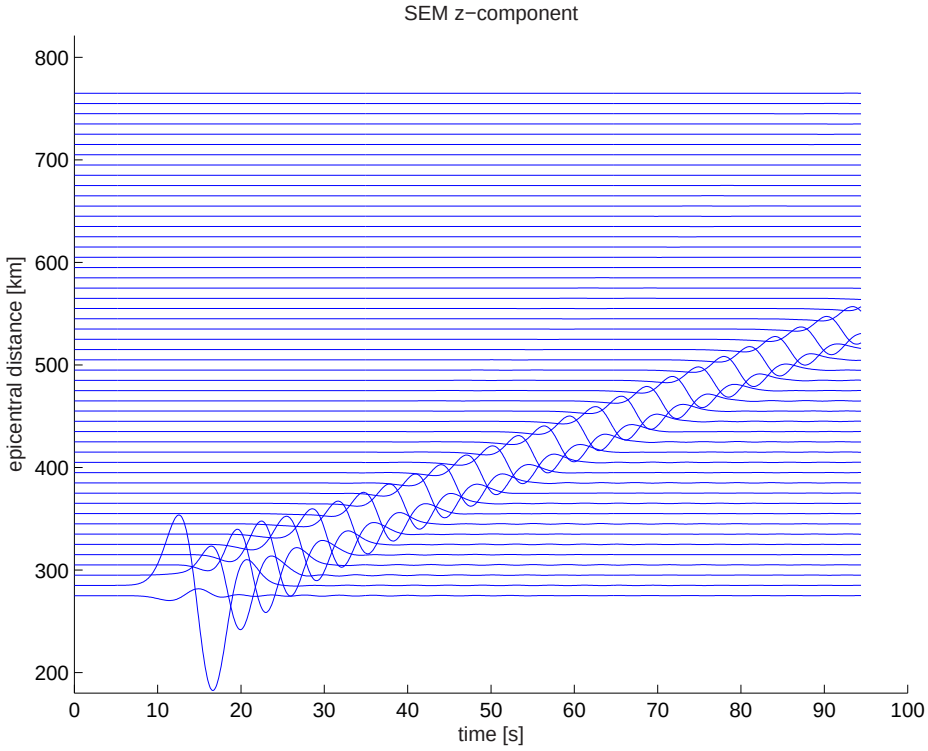


(b)

Figure 5.9: SEM solution - Dip-Slip source at $z = -200$ km. Here, direct P- and S-waves are observed. Small wiggles coming after the S-wave are due to very low spatial sampling of 5 points per wavelength.



(a)



(b)

Figure 5.10: SEM solution - Dip-Slip source at $z = -10$ km. In this case the amplitude of the surface is very big so the P-wave can hardly be seen.

5.2 Benchmark of CPU Time per Time Step

The performance of a numerical technique depends on the time needed to solve a certain problem. In Chapter 4 a method was developed to determine the performance as a combination of accuracy and CPU time per time step. Similar to the 1-D case, a CPU benchmark of the FDM and SEM for three-dimensional simulations was carried out. The number of runs and maximum model sizes are more restricted in this case. Therefore, the sampling points and the number of time steps used to calculate the mean value were significantly lower. One hundred time steps for ten differently sized models were computed.
The result of the benchmark test is shown in Figure 5.11. From this picture it is clear, that the SEM is much faster than the FD method. It has to be stated, that the CPU time strongly depends on the implementation of the program code. This may have an enormous effect of the outcome of the benchmark. All simulations for the CPU benchmark were performed on a AMD Athlon^{TM}XP 2400+ processor.

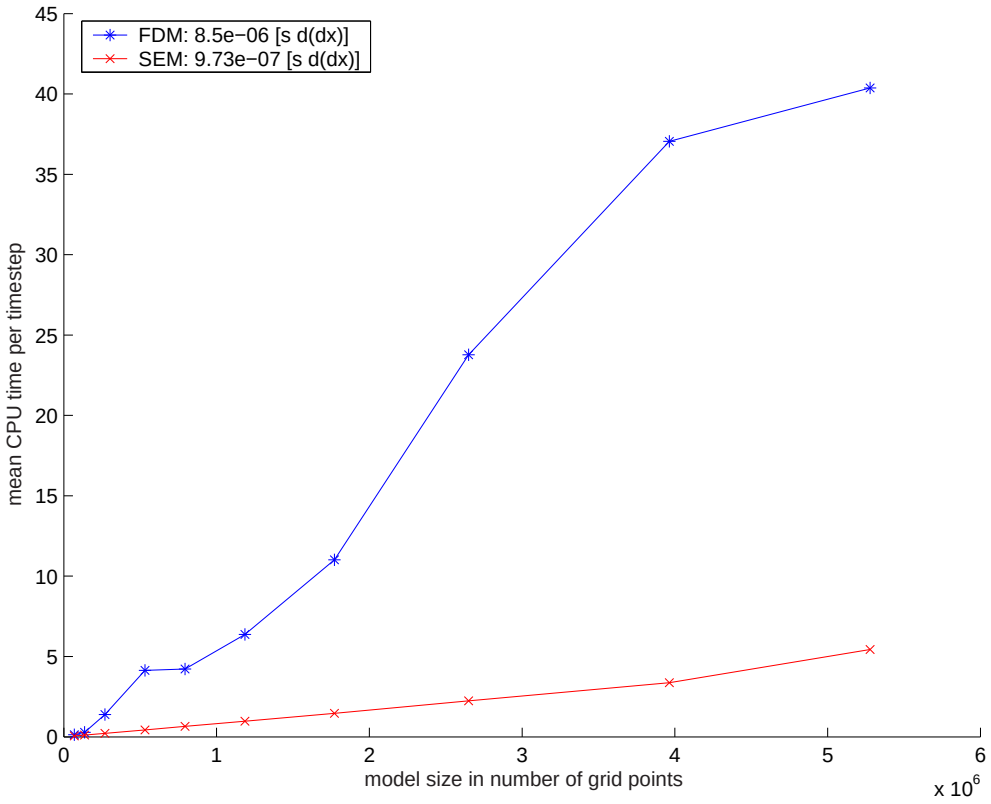
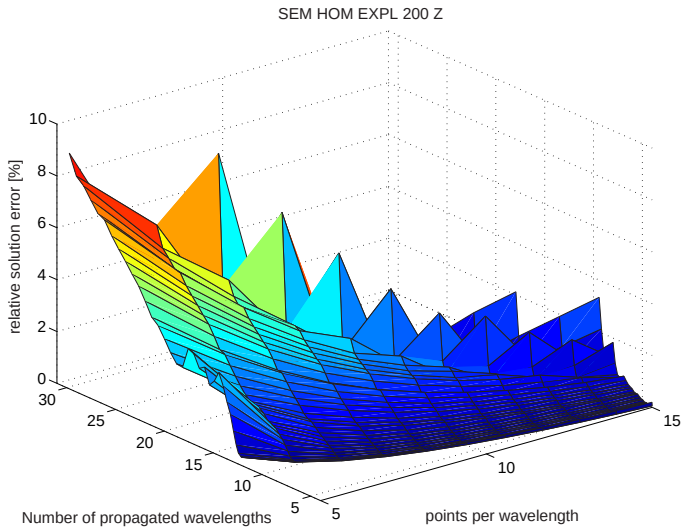
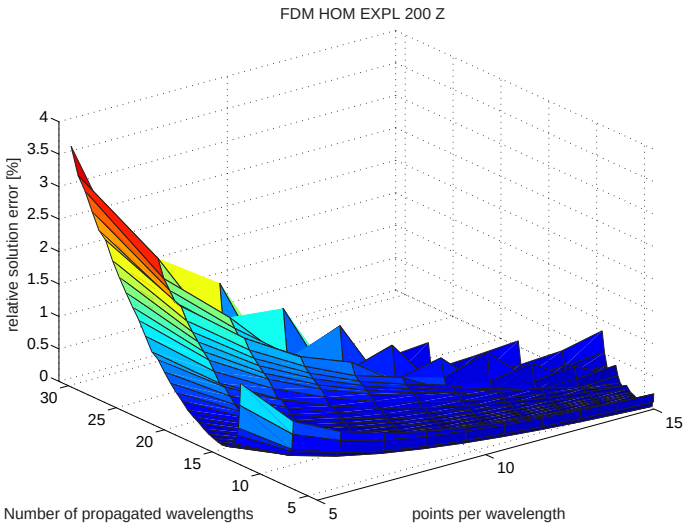


Figure 5.11: CPU benchmark of FDM and SEM for 3-D simulations. The SEM is about 10 times faster than the FDM. The test was performed on a AMD Athlon^{TM}XP 2400+ processor and the memory needed for the biggest model was 1.6 GB in the SEM and 0.6 GB in the FDM simulations.

5.3 Comparison of Performance: Difference in Accuracy and Relative CPU Cost



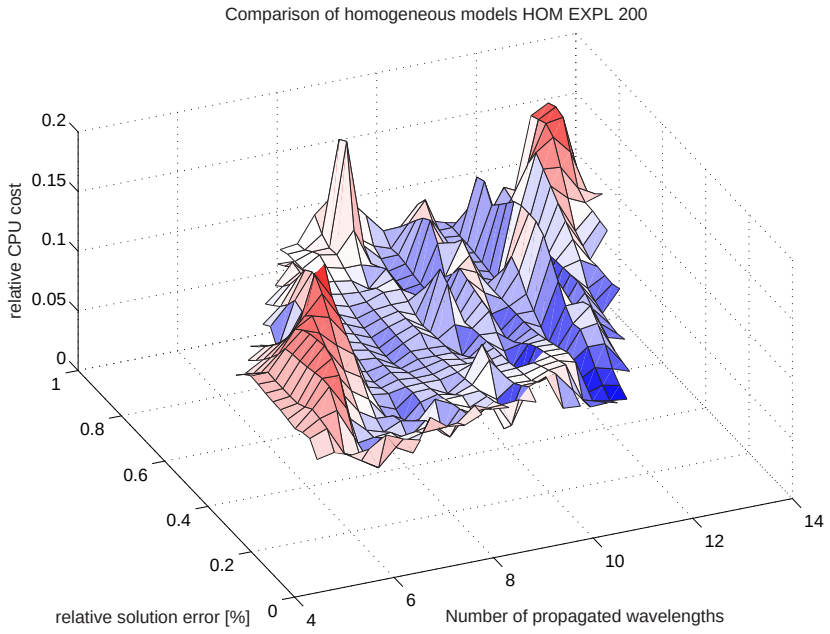
(a) SEM z-component



(b) FDM z-component

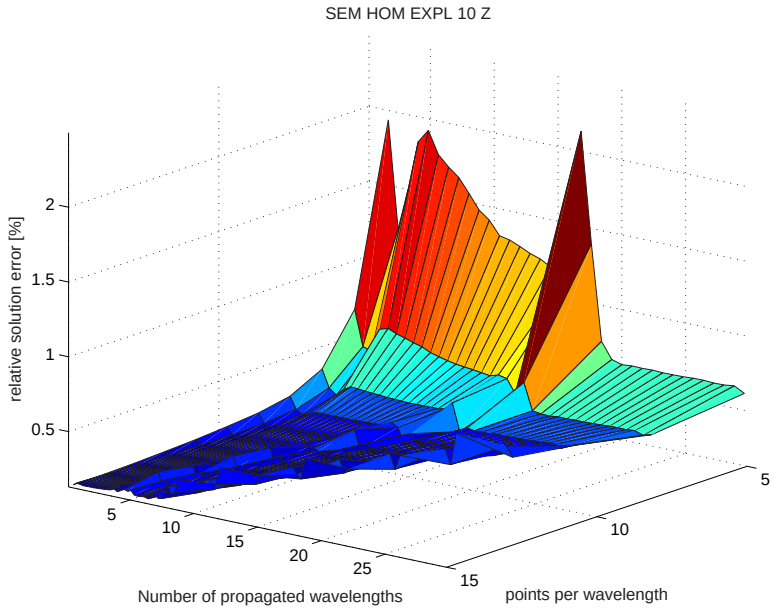
Figure 5.12: Relative solution error - explosion at $z = -200$ km. Both methods clearly show similar response to changes of *ppw* and *npw*. Surprisingly, the error of the FD method is less than the SEM at a factor of 2. Two peaks are visible at around 5 *ppw* and 10-15 *npw* in the error surface of the SEM, indicating the spurious S-wave that occurs for this case.

(a) Difference in accuracy: SEM - FDM z-component

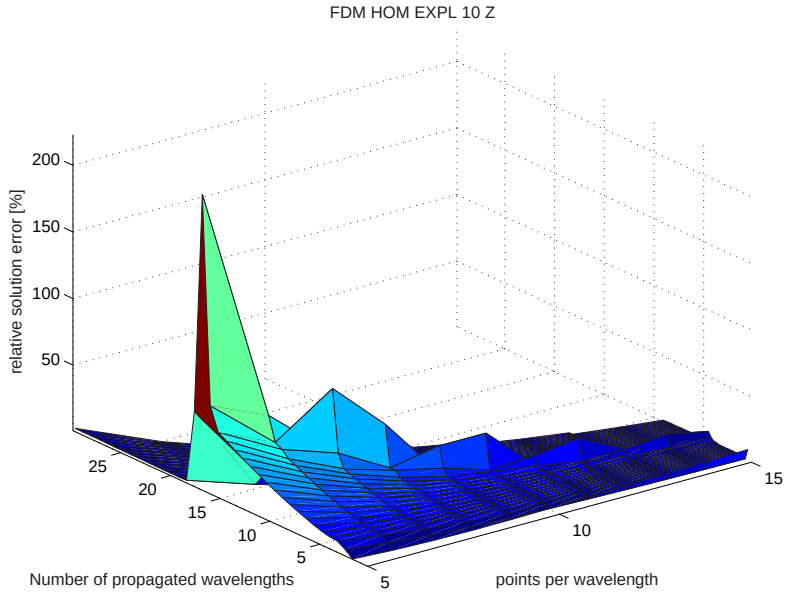


(b) Relative CPU cost: SEM / FDM z-component

Figure 5.13: Comparison of performance between SEM and FDM - explosion at $z = -200$ km. The differential error surface between SEM and FDM represents the higher error of the former. Nevertheless, the SEM is better when comparing overall CPU cost. This is mostly due to the significantly better CPU benchmark test of Section 5.2.

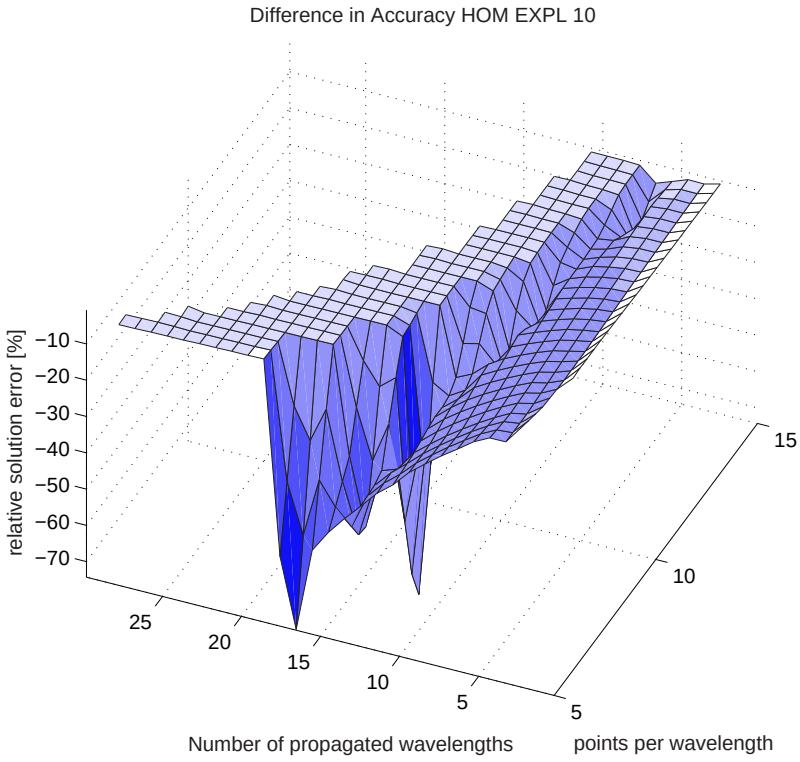


(a) SEM z-component

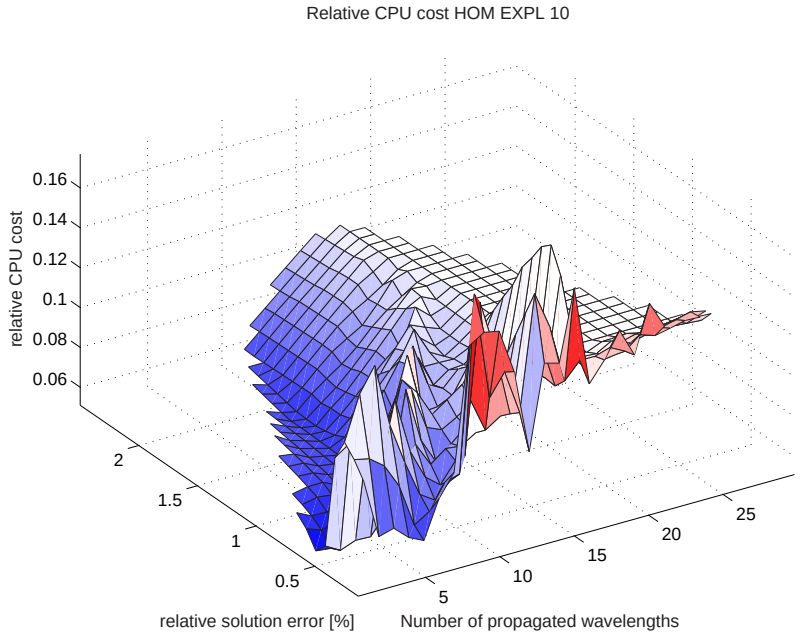


(b) FDM z-component

Figure 5.14: Relative solution error - explosion at $z = -10$ km. These surfaces depict a completely different view. Not only the perspective for the SEM surface is changed by 180 degree (for better display), but also the difference in error is huge. A feature that appears for both methods is the swell running through the middle of the surfaces. This swell results from the surface-wave running out of the considered time window for stations further away from the source. The surface waves produces a higher error than the P-wave, because of fewer points per wavelength at the same frequency. Therefore, at stations where no surface waves arrive, the error is smaller. These figures show that the FD method is not well suite for representing surface waves. The error rises steadily until the surface waves disappear and drops to acceptable values for remote stations.

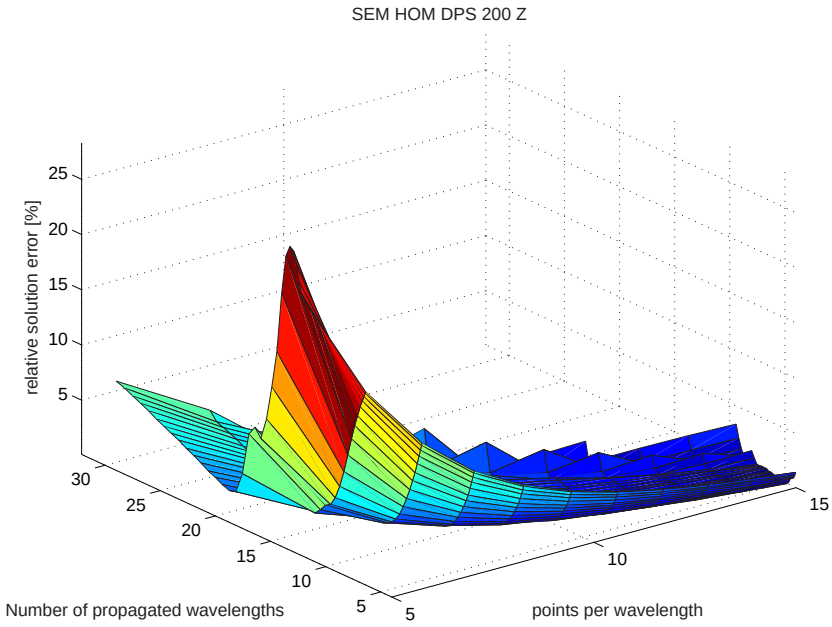


(a) Difference in accuracy: SEM - FDM z-component

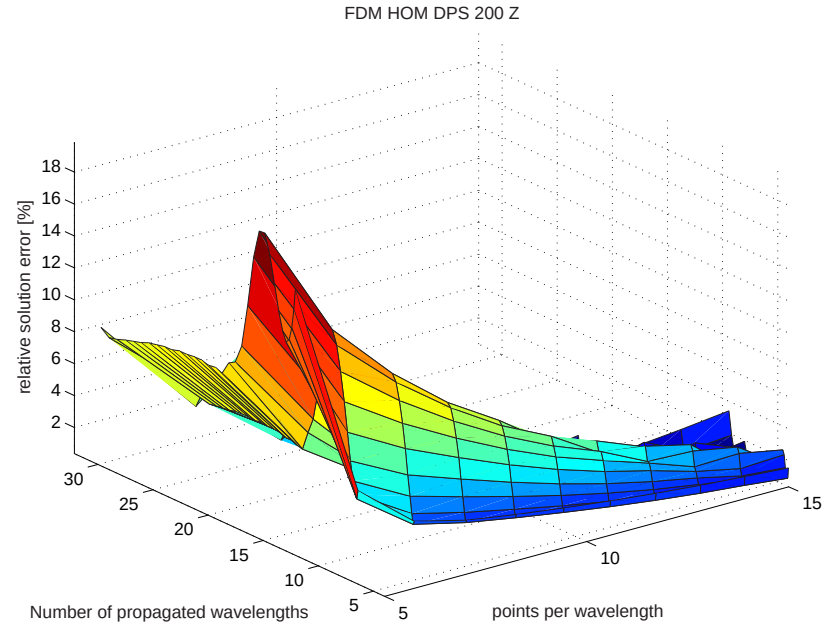


(b) Relative CPU cost: SEM / FDM z-component

Figure 5.15: Comparison of performance between SEM and FDM - explosion at $z = -10$ km. The differential surface displays the same feature mentioned before (Fig. 5.14). CPU cost is by far lower for SEM.

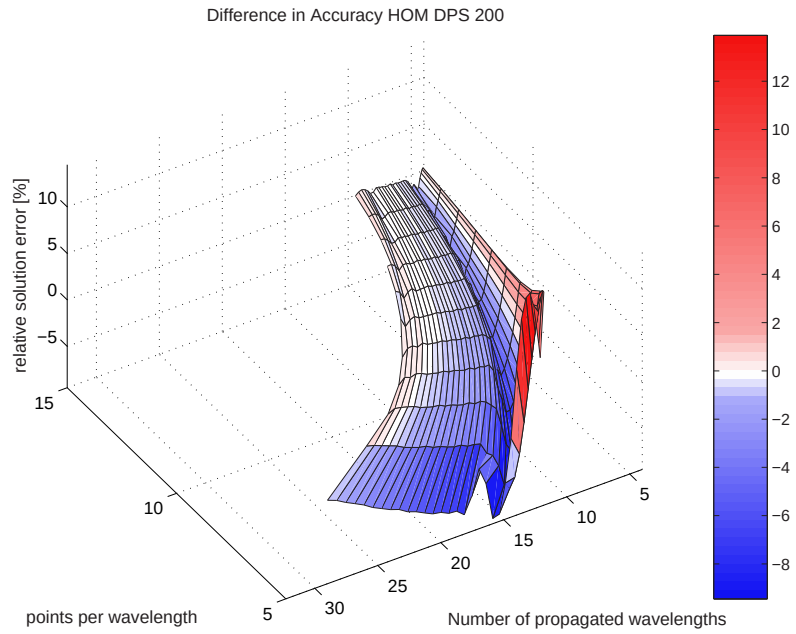


(a) SEM z-component

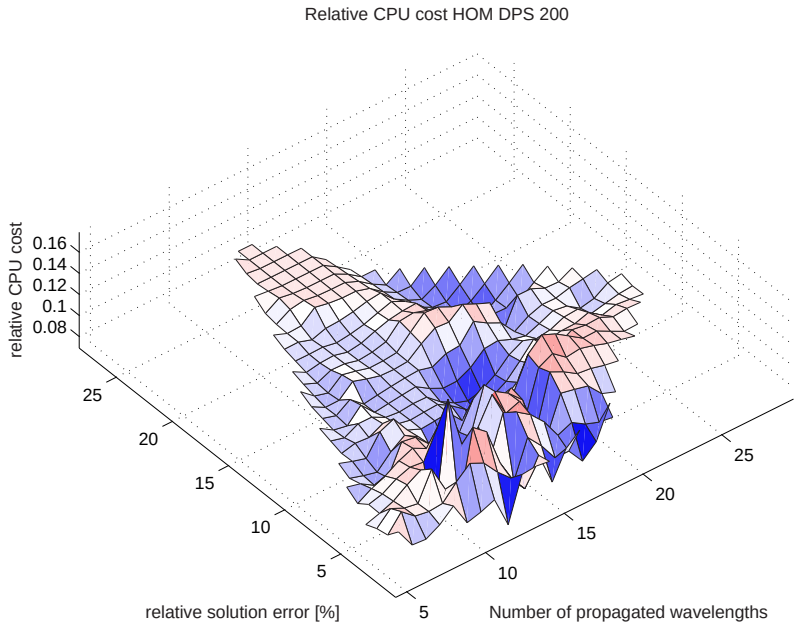


(b) FDM z-component

Figure 5.16: Relative solution error - dip-slip source at $z = -200$ km. For this case the direct S-wave plays a significant role. The high values of both surfaces for few *npw* result from a phase shift of the S-wave (Fig. 5.18). For stations afar, only the P-wave is of relevance for the errors produced.



(a) Difference in accuracy: SEM - FDM z-component



(b) Relative CPU cost: SEM / FDM z-component

Figure 5.17: Comparison of performance between SEM and FDM - dip-slip source at $z = -200$ km. For most of the considered values of *npw* and *ppw* the SEM is more accurate than the FDM. Only a small band at few *npw* remain, where the FDM seems to be better. Expectedly, the FDM is more expensive in terms of CPU cost.

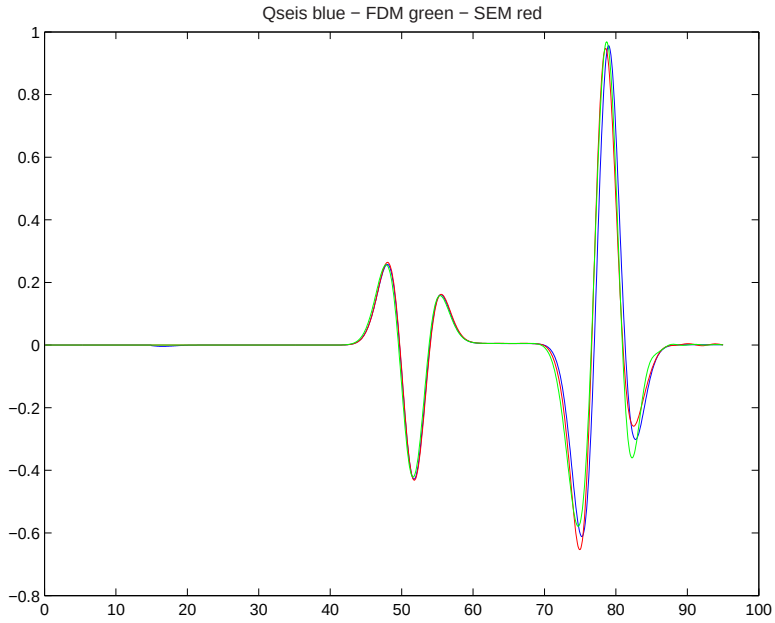


Figure 5.18: Comparison of the velocity of the z-component at receiver 30 between Qseis (blue), FDM (green) and SEM (red) using 5 *ppw* for a dip-slip source in 200 km depth.

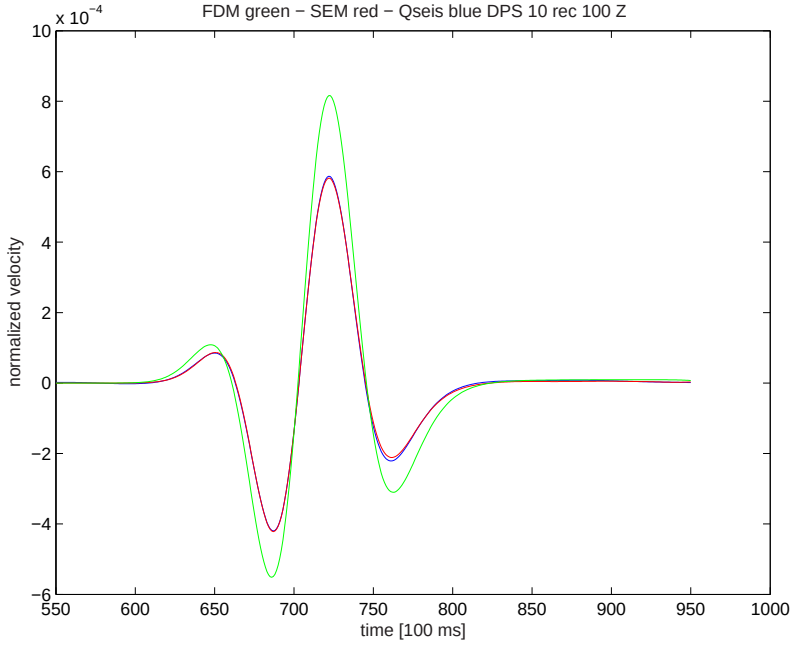
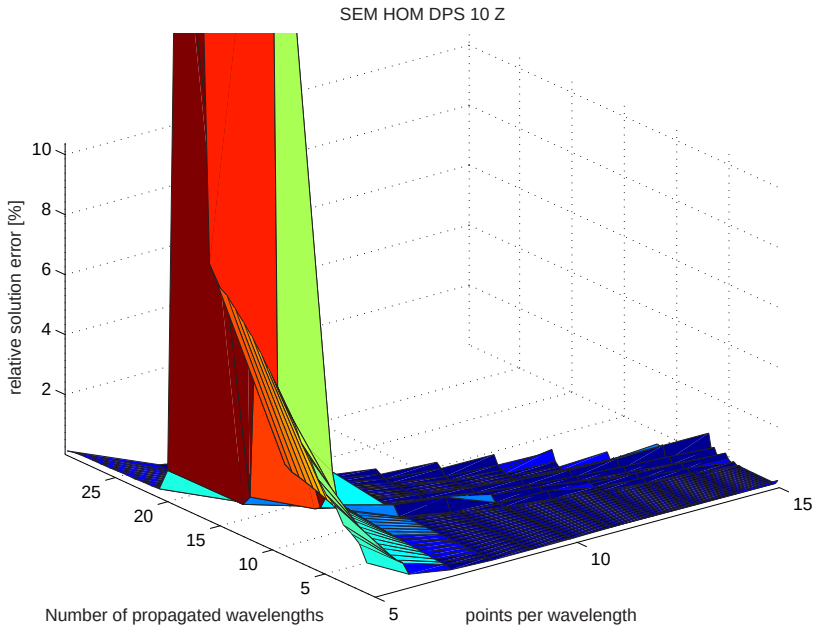
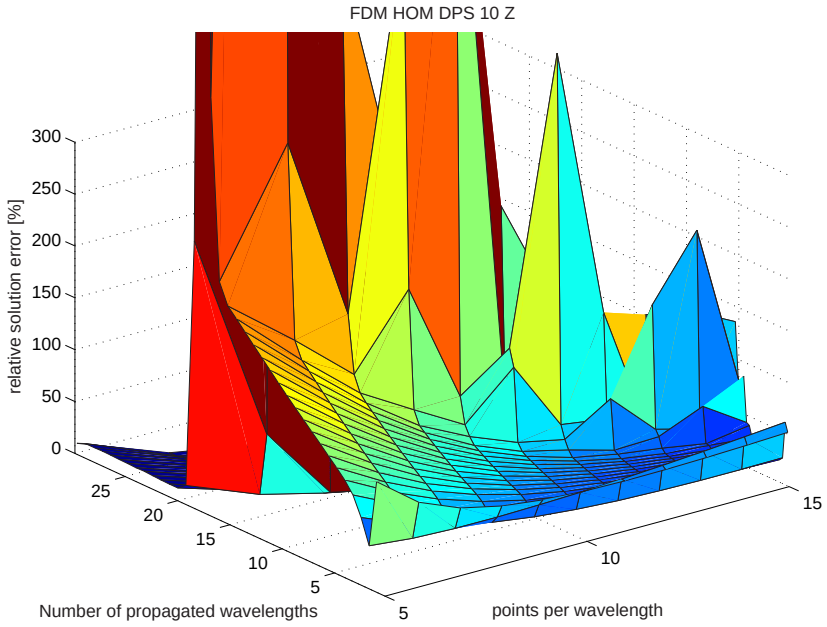


Figure 5.19: Comparison of the velocity of the z-component at receiver 100 between Qseis (blue), FDM (green) and SEM (red) (5 *ppw*) for a dip-slip source in 10 km depth.

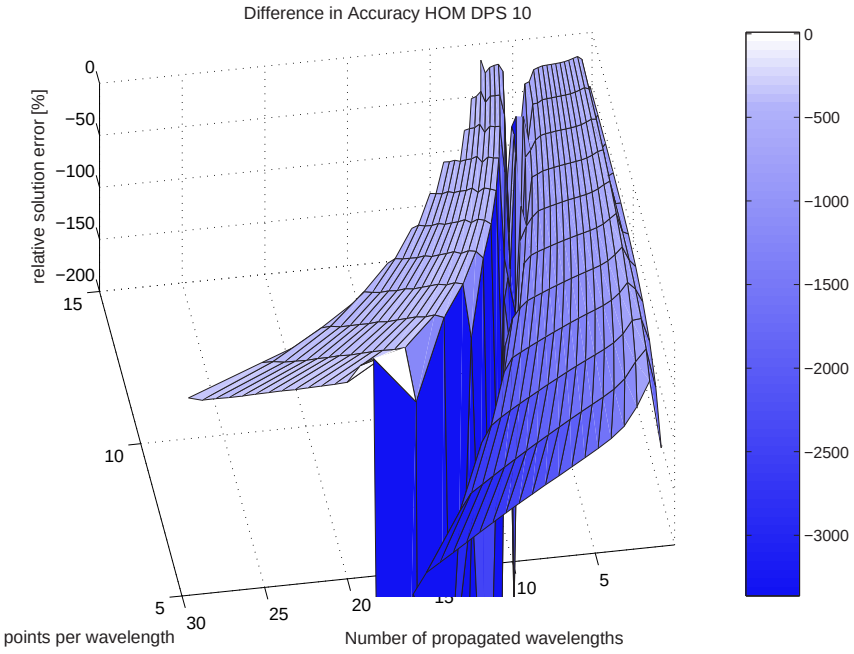


(a) SEM z-component

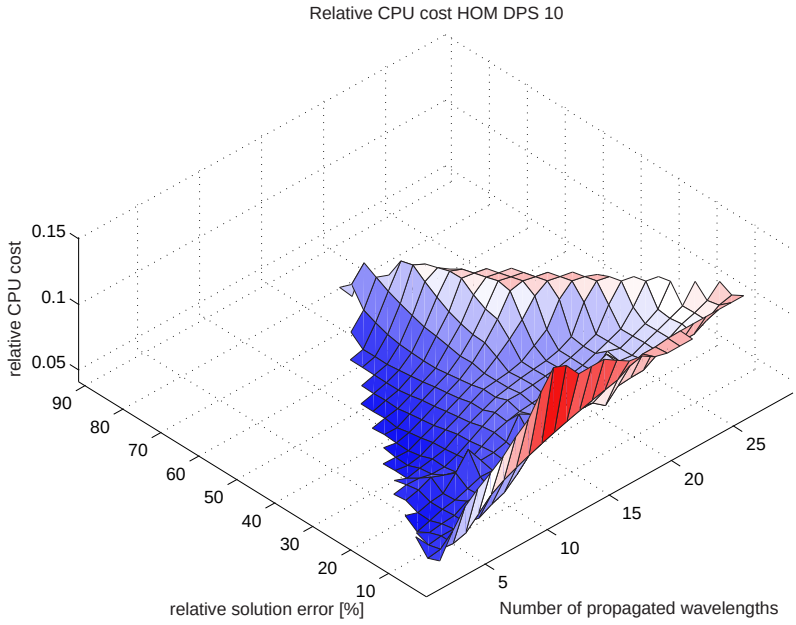


(b) FDM z-component

Figure 5.20: Relative solution error - dip-slip source at $z = -10$ km. Here the error of the SEM is below 2 % for most *ppw* and *npw*. At the stations where the strong surface wave travel out of the corresponding time window, the error rises enormous. The FD method show similar trends but at gigantic errors. The behaviour of error with decreasing *ppw* is also different and more pronounced. Nevertheless, the error is of around 20 % for stations that record only the P-wave.



(a) Difference in accuracy: SEM - FDM z-component



(b) Relative CPU cost: SEM / FDM z-component

Figure 5.21: Comparison of performance between SEM and FDM - dip-slip source at $z = -10$ km. From these pictures it is clear that in the SEM the representation of surface waves is much better than in FDM. Relative CPU cost is below a maximum factor of $\frac{SEM}{FDM} = 0.2$ for all investigated models.

Chapter 6

Discussion

After a describtion of the spectral element method in the first part of this thesis, some characteristic features of the performance of the SEM in 1- and 3-D were outlined in the second part. The previous chapter has demonstrated that the SEM is the favorite method for 3-D calculations when compared to a classical finite difference method. Nevertheless, in Chapter 4 the results presented reveal that the "optimal operators", an optimized FD method, may offer an alternative. In this context some considerations are of interest to the discussion whether one or the other method should be preferred. In order to put the results in perspective one has to look at the complete picture. The comparisons performed depend on the prerequisites of the simulations to a great extent.

To start with the 1-D case, one has to bear in mind that none of the FORTRAN 90/95 codes is written by professional programmers and moreover, neither of them is designed to run optimally fast on a certain machine. Nevertheless, as only the mean CPU time needed per time step was considered in the present tests, all features that are not necessary for the solution of the wave equation were turned off. For example the writing of seismograms or the plotting of the 1-D "wave-field" are not essential during the time loop.
In addition, the CPU benchmark is difficult as information on the load of the processor by the Linux system itself is not easily available. The maximum performance of each simulation in CPU load was thus not recorded. Therefore, one has to be careful when judging by this CPU benchmark alone. A consequently performed CPU benchmark should include comparisons on different machines and architectures and also several different compilers. However, it has to be stated again, that care was taken to test the programs on exactly the same computer. On this machine no other processes, apart from those needed by the operating system, were running to ensure a self-consited comparison.

Regarding the 3-D codes, the SEM code is certainly the more professional one and thus for direct comparison the odds are more on the SEM side. Nevertheless,

as all programs of each respective comparison ran under the same conditions, the trend given by the CPU benchmark can be used as a first clue. A future step should consist of revising the programs, especially the FD code, to speed them up to an optimum.

A further issue is the memory required by the simulations. As stated before, the 3-D FD code requires much less memory than the 3-D SEM code for the same model size. At this point, it has to be made clear, that most of the additional memory needed in the SEM results from the storage of the Jacobi-Matrices and their Jabobians. These are a consequence of the mapping and thus allow for deforming the elements and to adapt them to a certain shape. The FD code is at this time only designed for equal grid spacing in all dimensions. Therefore the codes offer different possibilities and cannot be compared directly in terms of memory. A fair comparison of memory requirement could consist of choosing the same mapping for all elements in the SEM and thus to reduce the number of Jacobians to a single one.
The memory itself has an influence of the accuracy that can be achieved for a considered simulation. Starting from given model dimensions, the memory available limits the spatial sampling. The sampling itself governs the accuracy and thus, an overall comparison must consist of a combination of the performance criterion used in this thesis, the CPU cost and a test of accuracy when using the same memory.
Moreover, the choice of time step dt and thus of the Courant value for the simulations can be compared. In turn it might be interesting to compare methods, with each using the highest Courant value possible.

An additional point that has to be discussed is the numerical dispersion and the reliability of the quasi-analytical solution. From Figures 5.18 and 5.19 it becomes clear that the error[1] of the numerical solution is governed by the phase shift of the S- or the surface waves. Thus, the relative error between SEM and FDM may vary to a great extent, when shifting the seismograms to reduce the error in phase of the S- and surface wave respectively. In this work the seismograms were shifted to align the P phases.
Some problems occured in the quasi-analytical solution obtained from "Qseis". Small waves could be distinguished, which were not resulting from physics. Care has to be taken therefore, when computing solution errors with such "quasi"-analytical solutions. But, as both methods were compared to the same reference, this only affects the absolute values, which cancel out in relative comparisons.

One can see from the above that many issues influence the output of comparisons as such performed during this work. Keeping in mind the previous considerations,

[1] as defined in this thesis

one can state that on the one hand the SEM represents surface waves by far better than the FD method. This is not surprising as the free surface condition is naturally included in the SEM formalism. On the other hand, it was demonstrated that the SEM is not suited for proper simulation of explosive sources. Numerical anisotropy resulting from the source implementation is leading to spurious waves, which may be misinterpreted in more complex cases than the homogeneous one considered here.
In general the FDM needs much more points per wavelength than the SEM and the maximum of 15, for which the resulting errors were computed, is still not enough. Therefore, by looking at the results of the 1-D case, it seems highly reasonable to pursue investigation on the optimal operators. The comparisons show less numerical dispersion for OPO than for SEM and depending on the choice of order for the spectral elements, the OPO are in many cases more accurate. The overall CPU cost of both methods, by which was judged in this thesis, illustrates that the OPO are preferable. But, up to now it is not clear how the OPO perform for several boundary conditions, of which the free surface condition is of most interest. Reducing the errors produced by FDM for surface waves seems inevitable after this study. Therefore, a test of boundary conditions between OPO and SEM in 1-D should be the next step. In a following step, 3-D comparisons to SEM will then give more information on the performance of optimized FD methods.

At last, it should be emphasized that additional aspects were not treated in this thesis. One of the undisputable advantages of the SEM is the representation of complex geometry. Simulations published up to the present day have shown that the SEM is capable to solve a variety of problems that emerge in geoscience.

Summary

The objective of this thesis was to review and describe the spectral element method for the simulation of elastic wave propagation in detail and to compare it with different finite difference methods. Especially an optimized version of the FD operators, the optimal operators, was of interest herefore.

The SEM is a sophisticated numerical technique and is an extension of the finite element method. In the first part, the theory and concepts of the SEM were explained in detail for the one-dimensional wave equation in Chapter 2 at first, and later also for 3-D cases. It was demonstrated that the SEM is based on the variational formulation of the PDE and that this can be used for a convenient decomposition of the model domain into elements. For simplification of further calculations a coordinate transformation was defined to map each of those elements onto the interval $\mathbf{\Lambda} = [-1, 1]_{n_d}$. All functions are then considered on this standard domain for the elements separately. When differentiated or integrated functions are needed, the Jacobi-Matrix or its determinant the Jacobian appear as additional terms in the equations.
In order to further discretize the model, the functions are interpolated on Λ using Lagrange polynomials and the Lagrangian interpolation scheme. Numerical integration is done by means of the Gauss-Lobatto-Legendre quadrature, for which the associated collocation points and integration weights are needed. Applying interpolation and numerical integration to the decomposed variational formulation results in two matrices defined for every element. These are the mass and stiffness matrix, respectively. Assembling the elemental matrices leads to a global linear system of equations, that can be solved in time using a matrix formulation. The key feature of the SEM is the combination of Lagrange polynomials defined on the Gauss-Lobatto-Legendre collocation points and the GLL quadrature. It leads to an exactly diagonal mass matrix, whose inversion is thus trivial for the solution of the global linear system. A different type of interpolating functions was also introduced for the use in SEM in Chapter 2.

In the second part of the thesis, results of different simulations and comparisons first between different time schemes for the SEM and later between the best performing SEM and two FD methods were shown. It turned out that a new

criterion for the comparison of the performance of numerical methods had to be defined. This criterion was called "CPU cost" and consisted of a combination of accuracy and CPU time per time step. A specially created model setup could be used in a first step to calculate the relative solution error as a function of distance and spatial numerical sampling. The former was expressed by dimensionless numbers of propagated wavelengths and the latter is typically expressed as points per wavelength. The results obtained from these calculations can be displayed as surfaces. Inverting the surfaces leads to a relation giving the minimum *ppw* needed to limit the error at distance *npw*. In a second step, a benchmark of the CPU time per time step was performed to obtain the additional CPU time needed for every additional grid point. Multiplication of this value with *npw* and the associated minimum *ppw* gives the overall CPU cost for a certain problem.
Applying this method to 1-D and 3-D problems is possible, only for the latter some restrictions due to the special model setup were needed. A comparison of 1-D SEM simulations using different time integration schemes indicated that a predictor-multicorrector Newmark scheme is not the most efficient one in terms of CPU cost, although in most cases the most accurate. Instead, an explicit second order Newmark scheme using an acceleration formulation turned out to be the best performing one. When comparing one-dimensional SEM simulations of different polynomial order, it becomes evident that the performance increases with order N not only in accuracy but also in CPU time per time step. Nevertheless, the time step has to be reduced for greater N because of the stability criterion leading to a greater number of time steps for the same simulation time.
The focus of the evaluation was on comparing the SEM simulations with OPO simulations for one-dimensional models. As these optimal operators are not yet developed for boundary conditions, the models were built in such a way that reflected waves from the boundaries were not recorded at the receivers in the time window considered for the calculation of errors.
The benchmark of the CPU time per time step recovered that the OPO is the fastest method compared to the SEM with several different polynomial orders.
For spectral elements with polynomial degree 5 the maximum possible Courant value of the stability criterion was chosen. According to this a Courant value for the OPO was used that led to the same time step as for the SEM. This resulted in a better performance of the OPO for most *ppw* and *npw* investigated here. The outcome of the comparison changed when using spectral elements with polynomial order 8, but only when a maximum possible Courant value was used. Extrapolation to the same time step as for OPO shows that again the OPO performs better.

A last topic covered in this work was the comparison of the SEM with FDM simulations for simple homogeneous three-dimensional models. Again, as for the 1-D case the, "CPU cost" was used as performance criterion. Two kinds of sources were applied. An explosive source and a dip-slip source which was considered to

excite strong surface waves when put close to the free surface.
To a great surprise, the SEM exhibited a clearly visible direct S-wave in the case of an explosion inside a homogeneous medium. The amplitude of the S-wave was found to depend on the location inside the element containing the source. In addition, the frequency band chosen for the source affected the S-wave. Numerical anisotropy of the source implementation in the SEM is considered as explanation. The comparisons of accuracy between SEM and FDM show significant difference, especially for surface waves. The SEM is much more accurate, which reconfirms that FDM needs much more points per wavelength than the SEM. Comparing the overall CPU cost, the SEM is by far the better performing method.

The results of the second part of this thesis indicate that it might be interesting to further investigate the optimal operators. Especially the implementation of optimized boundary conditions and extension to 3-D simulations may lead to an alternative to the well-performing spectral element method.

Appendix A

A.1 The Integration Weights and Collocation Points of the Gauss-Lobatto-Legendre quadrature

polynomial degree N	collocation points ξ_i	integration weights ω_i
2:	0	1.3333333333333333
	± 1	0.3333333333333333
3:	± 0.4472135954999579	0.8333333333333334
	± 1	0.1666666666666667
4:	0	0.7111111111111111
	± 0.6546536707079772	0.5444444444444445
	± 1	0.1000000000000000
5:	± 0.2852315164806451	0.5548583770354862
	± 0.7650553239294647	0.3784749562978470
	± 1	0.0666666666666667
6:	0	0.4876190476190476
	± 0.4688487934707142	0.4317453812098627
	± 0.8302238962785670	0.2768260473615659
	± 1	0.0476190476190476
7:	± 0.2092992179024789	0.4124587946587038
	± 0.5917001814331423	0.3411226924835044
	± 0.8717401485096066	0.2107042271435061
	± 1	0.0357142857142857

Table A.1: Collocation points and integration weights of the Gauss-Lobatto-Legendre quadrature for order $N = 2, \ldots, 7$.

polynomial degree N	collocation points ξ_i	integration weights ω_i
8:	0	0.3715192743764172
	± 0.3631174638261782	0.3464285109730463
	± 0.6771862795107377	0.2745387125001617
	± 0.8997579954114602	0.1654953615608055
	± 1	0.0277777777777778
9:	± 0.1652789576663870	0.3275397611838976
	± 0.4779249498104445	0.2920426836796838
	± 0.7387738651055050	0.2248893420631264
	± 0.9195339081664589	0.1333059908510701
	± 1	0.0222222222222222
10:	0	0.3002175954556907
	± 0.2957581355869394	0.2868791247790080
	± 0.5652353269962050	0.2480481042640284
	± 0.7844834736631444	0.1871698817803052
	± 0.9340014304080592	0.1096122732669949
	± 1	0.0181818181818182
11:	± 0.1365529328549276	0.2714052409106962
	± 0.3995309409653489	0.2512756031992013
	± 0.6328761530318606	0.2125084177610211
	± 0.8192793216440067	0.1579747055643701
	± 0.9448992722228822	0.0916845174131962
	± 1	0.0151515151515152
12:	0	0.2519308493334467
	± 0.2492869301062400	0.2440157903066763
	± 0.4829098210913362	0.2207677935661101
	± 0.6861884690817575	0.1836468652035501
	± 0.8463475646518723	0.1349819266896083
	± 0.9533098466421639	0.0778016867468189
	± 1	0.0128205128205128

Table A.2: Collocation points and integration weights of the Gauss-Lobatto-Legendre quadrature for order $N = 8, \ldots, 12$.

A.2 The Fortran SEM Program Code for solving 1-D wave equation

The simulations outlined in Chapter 4 were performed with a serial Fortran code. In addition to the standard libraries of the intel Fortran compiler, the libraries of the freely available "`pgplot`" (© CalTech) package were used. They provide several subroutines for 2- and 3-D plotting and can be downloaded from `ftp://ftp.astro.caltech.edu/pub/pgplot/pgplot5.2.tar.gz` (Pearson [2002]).
In the following the reader can find:

- the parameter file of the 1-D code
- the main program `elastic_1D`
- a selection of subroutines

The main program is given in full length, whereas only those subroutines are printed, that are especially needed for SEM simulations. These are in order of their appearance in the Appendix:

– `diff_lagrange`, calculates the collocation points, integration weights and the derivatives of the Lagrange polynomials at all collocation points. It makes use of three subroutines designed for spectral methods taken from Funaro [1993], which were mentioned in Section 2.2.2. These are:

  ◇ `VALEPO`, calculates the values of Legendre polynomials at a given point
  ◇ `ZELEGL`, calculates the nodes of the GLL integration quadrature
  ◇ `DELEGL`, calculates the first derivative of the Lagrange polynomials at the GLL quadrature points

  The calculation of integration weights was implemented by myself using the formula in equation (2.52) taken from Abramowitz & Stegun [1984].

– `create_x_irreg`, creates the 1-D grid, a vector containing the x-coordinates of the nodes

– `connect_vec`, assembles global vectors from elemental vectors (e.g. mass "matrix", see Section 2.1.5)

– `connect_mat`, assembles global matrices from elemental matrices (e.g. stiffness matrix)

– `create_S`, creates the condensed stiffness matrix shown in Section 2.2.7 without using the subroutine `connect_mat`

- `calculate_F`, calculates the forces at each node of the grid, following the theoretical considerations of Section 2.2.8
- `get_func_at_pos`, calculates the value of a function at any point inside one element, which values are known at the collocation points. It makes use of subroutine
  - ◇ `lagrange_any`, which was taken from the CALTECH [2002] SEM code written by Dimitri Komatitsch and co-workers. It is the realisation of equation (2.23)
- `calc_jacobian_1D`, calculates the Jacobi-Matrix, the Jacobian and the Jacobi-Matrix of the inverse mapping
- `get_shape_1D`, calculates the 1-D shape functions at each node of all elements

Subroutine `lagrange_any` was used in the original version by CALTECH [2002]. Subroutines `get_shape_1D` is based on the subroutine `get_shape_2D` from that code, but had to be changed to 1-D serial applications. As mentioned, the subroutines `VALEPO`,`ZELEGL` and `DELEGL` were taken from a set of routines by FUNARO [1993].
Subroutines, that are called by the main program, but not printed here are:

- `create_sem_fname`, defines a string used for output files
- `gauss, ricker, ricker2`and `delta`, define different source time functions
- `setup_ac_array`, defines different receiver arrays, specially designed for the comparisons shown in 4
- `plot_1D_snapshot`, plots 2-D graphs of the displacement field (snapshots) during runtime
- `calcanalyt2lay_sem`, calculates the corresponding analytical solution for a homogeneous or two layer case for a delta-shaped source time function. Original version `calcanalyt2lay` was written by Tobias Metz (METZ [2003])
- `conv_fft`, calculates the convolution of two vectors using fast-fourier-transform (fft)
- `plot_ac_seismogram`, plots different types of seismograms, depending on the input of the program
- `plot_initial`, plots the initial displacement field

- `write_seismograms`, creates output of the program, three files for every receiver: synthetic and analytical solution and additional information
- `write_setup`, creates output file containing information about the simulation

In addition, the subroutine `multiply_S_U` is printed, which performs the matrix-vector multiplication of the condensed stiffness matrix **K** with the displacement vector **u**. The shape of the condensed stiffness matrix and the rules of the multiplication were explained in Section 2.2.7, pages 42 - 44. The stiffness matrix is denoted by S in the codes for historic reasons.

The Parameter File (page 140)

The user can change several parameters to fit his requirements. He can choose:

- the order N of interpolating Lagrange polynomials
- the number of elements ne
- the type of the model (homogeneous,"two layer" model,heterogeneous)
- the type of the mesh, i.e. regular or irregular.
- the Length of the model
- the time to be simulated
- the Courant number of the stability criterion (time step dt is computed according to the stability criterion (2.58) during runtime
- four different boundary conditions ("free surface", rigid,periodic and absorbing boundaries)
- between a source or initial displacement field
- the type of the source: a delta peak in time, a gaussian and two ricker wavelets are implemented
- the source position (only on GLL collocation points)
- the dominant frequency of the source time function
- four different receiver arrays
- several display options (runtime plotting, "gif" of "ps" output)
- an option to obtain `ascii` output

The Main Program (pages 141-141)

- Initialization
- Reading of input files:
  - Parameter file (for details see description in Appendix A.2, page 137)
  - meshfile (if irregular mesh is used)
  - density and *mu* data files
- Preparations for the calculation of the stiffness and mass matrix
  - Calculation of collocation points and integration weights
  - Calculation of the first derivative of the Lagrange polynomials at the collocation points
- Calculation of shape functions
- Calculation of Jacobi-Matrices and Jacobians
- Generation of a non-equidistant x-vector for run-time plotting of the displacement field (Gauss-Lobatto-Legendre (GLL) points of each element are not evenly distributed inside the intervall [-1,1])
- Calculation of the time step dt depending on the stability criterion
- Generation of the source signal (e.g. delta-peak or ricker-wavelet) or reading of initial displacement field data
- Calculation of the elemental stiffness matrices (skipped, if calculation of forces in time loop is preferred)
- Calculation of the elemental mass matrices
- Generation of the "connectivity-matrix" for the 1-D case
- Assembly of the global matrices $\mathbf{M}$ and if used $\mathbf{K}$
- setup of receivers
- Inversion of the mass-matrix
- Time-Loop: Integration in the time domain (explicit scheme)
  - Calculation of forces (if used)
  - implementation of boundary conditions
  - Calculation of displacement $\mathbf{u}_{t_{i+1}}$ at the next time step
- writing seismograms to output files

Additional Subroutines

```
5       !1) = N

.true.  !2) = reg_grid_flag

200     !3) = ne

so      !4) = mod_id

0.60    !5) = tmax

1.      !6) = Length

0.82    !7) = Courant_nr

a       !8) = bcondition

.true.  !9) = src_flag

57      !10) = src_ne

1       !11) = src_N

r2      !12) = stfpar

30.00   !13) = frq

d0      !14) = array_id

0.2     !15) = distance

0       !16) = pd (plot device)

90      !17) = plot_rec_nr

.true.  !18) = convolute

.false. !19) = percent

.true.  !20) = txt

!_____________________________________________________________________________
!                                 Descriptions
!_____________________________________________________________________________

!1) = N: order of Lagrange polynomials for interpolation on the elements

!2) = reg_grid_flag: ".true." if you want to use a regular grid! Else "DATA_mesh/mesh1D.dat" is read

!3) = ne: number of elements to use in the simulation

!4) = mod_id: two character string: type 'so' for homogeneous model , 'st' for two layered case,
!'sh' for heterogeneous.
!If 'st' is given, the files "DATA_mue/et_s00_00two_mue.txt" and
!"DATA_rho/et_s00_00two_rho.txt" are read. They both have two entries for Lame constant mue and rho
!respectively.
!If 'sh' is given, the files "DATA_mue/eh_sXX_YYYYY_mue.txt" and "DATA_rho/eh_sXX_YYYYY_rho.txt" are

!read, where XX is the corresponding order N<100 and YYYYY is the number of elements ne < 100000.
!See description in DATA_***/README for detailed information on these files.

!5) = tmax: total time to be simulated

!6) = Length: length of the model

!7) = Courant_nr: Courant number for stability criterion

!8) = bcondition: boundary condition? One character string. 'f' for free surface, 'r' for rigid
!boundary, 'a' for absorbing boundaries or 'p' for periodic boundary conditions

!9) = src_flag: ".true." if source shall be implemented, if not program needs initial conditions
!from files "DATA_initial/U.txt" and "~/Uold.txt"

!10) = src_ne:  source position? (e.g. 2 for element 2 or any number less or equal to ne)

!11) = src_N: source position inside the element given by src_ne? (e.g. between 0 and N-1)")

!12) = stfpar: source time function? 'ga' for gaussian, 'r1' for ricker wavelet,
!'r2' for first derivative of the ricker, or 'de' for a delta peak!

!13) = frq:  if plot_dev is not 3 or stfpar is not 'de': the dominant frequency of the
!ricker wavelet used for plotting or simulation, respectively.

!14) = array_id: type of receiver array (character(len=2):a*=240,b=150,c=100,d*=1;
!*0 = anywhere, *1 = on GLL points)

!15) = distance: if array_id =d (only one receiver): Please enter the source-receiver distance
!(<= Length - x(source))

!16) = pd: plot device (0=xwindow,1=gif,2=ps,3=no plot,4=like 0 but plot only seismograms,
!not snapshots)

!17) = plot_rec_nr: if more than one receiver are used, plot which receiver (number)

!18) = convolute: convolution with ricker wanted for plotting, when simulating with a delta peak?
!".true." for yes

!19) = percent: run time message for percent of nt wanted? ".true." for yes

!20) = txt: if output files (seismograms at all receivers and additional setup file) are wanted
!type ".true."
```

```
PROGRAM elastic_1D

!Program to solve the one-dimensional wave equation with a spectral element method

!written by Bernhard Schuberth 2003
!in the course of the diploma thesis
!????? Title ??????
!at the Departement for Earth- and Environmental Sciences, Geophysics Section, of the
!Ludwig-Maximilians-University Munich

!------------------------------------------------------------------------------

!modules used

use plot_mod                    !plotting tools based on pgplot
use el_spec_plot                !plotting subroutines needed for this program
use source                      !creates different source time functions
use write_data                  !used for writing seismograms to outputfiles
use el_array                    !specific arrays for comparison with optimal FD operators
use sem_spec_modules            !specific subroutines needed for SEM
use jacobian_mod                !calculation of the Jacobi-Matrix and Jacobian
use shape_mod                   !calculation of shape functions of the mapping
use sem_analyt_mod              !calculation of the corresponding analytical solution
use signal_processing           !subroutines for fft and convolution
use minmax1d                    !maxi/min of a vector, used because of problems in "minval"
use minmax                      !maximum or minimum of 2-D or 3-D matrices

!------------------------------------------------------------------------------


implicit none

double precision, dimension(:,:), allocatable :: me                         !elemental mass matrices
double precision, dimension(:,:), allocatable :: DL             !derivatives of Lagrange polynomials

double precision, dimension(:,:), allocatable :: rho_e                      !density in all elements
double precision, dimension(:,:), allocatable :: mue_e     !elasticity coefficient mu in all elements
double precision, dimension(:,:), allocatable :: ve           !wave velocity derived from rho and mu

double precision, dimension(:,:), allocatable :: greens                     !stores all seismograms
double precision, dimension(:,:), allocatable :: analytical  !all corresp. analytical solutions (AS)

double precision, dimension(:,:), allocatable :: Jacobi           !Jacobi-Matrix (J-M)of the mapping
double precision, dimension(:,:), allocatable :: Jacobian                    !determinant of J-M
double precision, dimension(:,:), allocatable :: Jacobi_inverse          !J-M of the inverse mapping
double precision, dimension(:,:), allocatable :: shape1D             !shape functions of the mapping
double precision, dimension(:,:), allocatable :: dershape1D       !derivatives of the shap functions

double precision, dimension(:), allocatable :: U          !displacement at global nodes (timestep i)
double precision, dimension(:), allocatable :: Unew     !displacement at global nodes (timestep i+1)
double precision, dimension(:), allocatable :: Uold     !displacement at global nodes (timestep i-1)
double precision, dimension(:), allocatable :: U_init              !initial displacement field at t=0
double precision, dimension(:), allocatable :: Uold_init        !initial displacement field at t= -1
double precision, dimension(:), allocatable :: Mass         !global mass matrix (stored as a vector)
double precision, dimension(:), allocatable :: MassInv                  !inverse of the mass matrix
double precision, dimension(:), allocatable :: F                                !forces at global nodes
double precision, dimension(:), allocatable :: Rho_global    !globally defined density needed for AS
double precision, dimension(:), allocatable :: Mue_global !globally defined elasticity coeff. for AS

double precision, dimension(:), allocatable :: x                    !x-vector (spatial coordinate)
double precision, dimension(:), allocatable :: dx                  !distance between global x points
double precision, dimension(:), allocatable :: xmesh   !defining the 1-D mesh (2 points per element)
double precision, dimension(:), allocatable :: de                          !length of the elements
double precision, dimension(:), allocatable :: xi         !GLL quadrature points (local coordinates)
double precision, dimension(:), allocatable :: xidiff                !distance between GLL points
double precision, dimension(:), allocatable :: weights    !weights of the GLL integration quadrature

double precision, dimension(:), allocatable :: t                                        !time vector
double precision, dimension(:), allocatable :: src          !source time function for the simulation

double precision, dimension(:), allocatable :: stf             !source time function for convolution
double precision, dimension(:), allocatable :: cpu    !cpu time for each time step (processor clock)
double precision, dimension(:), allocatable :: cpu_dt !cpu time for each time step(for benchmarking)

double precision, dimension(:), allocatable :: seismogram        !convoluted seismogram for plotting
double precision, dimension(:), allocatable :: conv_ana                  !convoluted AS for plotting
double precision, dimension(:), allocatable :: diff_ana    !error in AS <= discrete sampling in time
double precision, dimension(:), allocatable :: rectimes           !special time marks needed for AS
double precision, dimension(:), allocatable :: reftimes           !special time marks needed for AS
double precision, dimension(:), allocatable :: srd      !source receiver distance for each receiver
double precision, dimension(:), allocatable :: srd_diff               !distances between receivers
double precision, dimension(:), allocatable :: receiver_pos            !x coordinates of receivers
double precision, dimension(:), allocatable :: receiver_dx !distance of receiver to nearest gridnode
```

```
double precision  dt                                                        !duration of time step
double precision  dxmin                                                      !minimum value of dx
double precision  v_max                                                      !maximum value of ve
double precision  v_min                                                      !minimum value of ve
double precision  pt                  !period of the dominant frequency of a wavelet given by 1/frq

double precision  source_pos                                          !x coordinate of the source
double precision  diff_greens !error in seismogram due to discrete time-sampling (only for plotting)
double precision  cpu_dt_mean                                !mean value of cpu-time per time step

double precision Length                                                      !Length of the model
double precision dx_mean                                  !average of distance between grid nodes

double precision bv1, bvng           !values used for calculation of absorbing boundaries at (1,ng)

double precision maxU, minU           !maximum and minimum values of displacement used for plotting
double precision src_max, src_min     !maximum and minimum values of displacement used for plotting
double precision tmp                                                        !dummy variable


integer,dimension(:,:), allocatable :: C                                    !Connectivity-Matrix

integer,dimension(:), allocatable :: receiver_ix
integer,dimension(:), allocatable :: num_el

integer, dimension(1:2) :: el_step_ix

integer N,ne,ng,NGNOD  !polyn. order,number of elements,number of grid points,number of anchor nodes
integer M,nt, source_ix, rec_num, dngix,step_ix
integer i,j,k,l                                                             !indices

character(len=2) deg                                                        !string for N
character(len=3) gif_num                          !used for correct numbering of gif-output

character(len=50) mue_file, rho_file                                        !input files
character(len=50) fname1, setupfile,plot_file                               !output files
character(len=50) poldeg,xlab, ylab, time                  !strings used for plotting
character(len=100) plot_dev


!Definition of fixed Parameters:

integer, parameter :: NGNOD = 2 !number of anchor nodes used for the shape functions of the elements

!------------------------------------------------------------------------------

!variables that can be changed by the user by editing the file "parfile" (see also for descriptions)

double precision tmax                          !the duration of wave propagation to be simulated
double precision Courant_nr                          !Courant number for the stability criterion
double precision frq                !dominant frequency of wavelet to be simulated or convoluted
double precision distance                      !distance of receiver, when only one is used

integer pd                                                                  !plot device
integer src_ne                                        !number of element to contain source
integer src_N                              !GLL point at which the source shall be located
integer plot_rec_nr                            !plot the seismogram of receiver number ...


character(len=2) mod_id                                                     !flag for model type
character(len=2) stfpar                                                     !flag for source type
character(len=2) array_id                                                   !flag for array type
character(len=1) bcondition                                  !flag for boundary conditions
logical convolute                                  !flag for convolution before plotting
logical txt                                                  !flag for creating ascii output
logical percent                                 !flag for run-time display of percentage
logical src_flag                                !flag if point source or initial conditions
logical reg_grid_flag                                       !flag if regular or irregular grid

!end of initialization


!------------------------------------------------------------------------------


!Read parameter file "./parfile"

open(unit=19,file='parfile')

!input of the parameters N (degree of Polynomials), ne (number of elements)

       read(19,*) N
```

```
        if (N ≥ 100) then
                write(0,*) "Order N > 100 is not possible."
                stop
        endif

        read(19,*) reg_grid_flag
        read(19,*) ne
        read(19,*) mod_id
        read(19,*) tmax
        read(19,*) Length
        read(19,*) Courant_nr
        read(19,*) bcondition
        read(19,*) src_flag
        read(19,*) src_ne
        read(19,*) src_N
        read(19,*) stfpar
        read(19,*) frq
        read(19,*) array_id
        read(19,*) distance
        read(19,*) pd
        read(19,*) plot_rec_nr
        read(19,*) convolute
        read(19,*) percent
        read(19,*) txt

close(19)

!get the source into the correct element given by "src_ne"

if (src_N > N) then
do while (src_N > N)
src_N=src_N-N
enddo
endif

mod_id = trim(mod_id)

deg = nletterstring(N,2)

!dominant period of the wavelet

pt=1./frq

!create filename for this run
!----------------------------

fname1=''

if (mod_id == 'so') then
        call create_sem_fname(fname1,'eo_s',N,2,ne,5,array_id)
elseif (mod_id == 'st') then
        call create_sem_fname(fname1,'et_s',N,2,ne,5,array_id)
elseif (mod_id == 'sh') then
        call create_sem_fname(fname1,'eh_s',N,2,ne,5,array_id)
endif

!check for irregular grid
!------------------------

if (reg_grid_flag == .true.) then

        allocate(xmesh(0:ne),de(1:1))
        xmesh(0)=0
        de(1)=1./float(ne)*Length
        do i=1,ne
          xmesh(i)=xmesh(i-1) + de(1)
        enddo
else

!if irregular grid get mesh data form file "./DATA_mesh/mesh1D.dat"

  open(unit=24,file='DATA_mesh/mesh1D.dat')
i=0
  do
        read(24,*,end=500)
        i=i+1
  enddo
  500 close(24)

  ne=i-1
  allocate(xmesh(0:ne),de(1:ne))

  open(unit=24,file='DATA_mesh/mesh1D.dat')
i=0
  do
        read(24,*,end=700) xmesh(i)
        i=i+1
  enddo
  700 close(24)

  do i=1,ne
        de(i)=xmesh(i)-xmesh(i-1)
  enddo

  Length=xmesh(ne)-xmesh(0)
endif

!------------------------------------------------------------------------------

!starting the program
!--------------------

!calculating the total number of nodes ng (global numbering)

ng = N * ne + 1

M = N + 1

!allocating all matrices

allocate(me(0:N,1:ne),rho_e(0:N,1:ne),mue_e(0:N,1:ne),ve(0:N,1:ne))
allocate(Mass(1:ng),MassInv(1:ng),U(1:ng),F(1:ng),x(1:ng),Uold(1:ng),Unew(1:ng),dx(1:ng-1))
allocate(Rho_global(1:ng),Mue_global(1:ng))
allocate(weights(0:N),DL(0:N,0:N),C(1:M,1:ne),xi(0:N),xidiff(1:N))
allocate(U_init(1:ng),Uold_init(1:ng))
allocate(Jacobi(0:N,1:ne),Jacobian(0:N,1:ne),Jacobi_inverse(0:N,1:ne))
allocate(shape1D(1:NGNOD,0:N),dershape1D(1:NGNOD,0:N))

!clearing all variables

Mass=0
MassInv=0
U=0
me=0
F=0
DL=0
Jacobi=0
Jacobian=0
Jacobi_inverse=0
shape1D=0
dershape1D=0
C=0
Uold=0
Unew=0
Rho_global=0
Mue_global=0
rho_e=0
mue_e=0
minU=huge(U)
maxU=tiny(U)


!read the information on elasticity coefficient mu from file
!-----------------------------------------------------------

if (mod_id == 'sh') then
        mue_file='DATA_mue/'//fname1(1:12)//'_mue.txt'
        open(unit=25,file=mue_file)
        i=0
        do i=1,ne
           do j=0,N
                read(25,*) mue_e(j,i)
           enddo
        enddo
elseif (mod_id == 'so') then
        mue_file='DATA_mue/'//fname1(1:4)//'00_00hom_mue.txt'
        open(unit=25,file=mue_file)
        read(25,*) mue_e(0,1)
        mue_e = mue_e(0,1)
        step_ix = 0
        Mue_global = mue_e(0,1)

elseif (mod_id == 'st') then
        mue_file='DATA_mue/'//fname1(1:4)//'00_00two_mue.txt'
        open(unit=25,file=mue_file)
        read(25,*) mue_e(0,1)
        read(25,*) mue_e(N,ne)
        read(25,*) el_step_ix(1)
```

```
        mue_e(:,1:el_step_ix(1)-1) = mue_e(0,1)
        mue_e(:,el_step_ix(1):ne) = mue_e(N,ne)
        step_ix = N * (el_step_ix(1)-1) + 1

        !creating the "global" Mue-vector
        !(only needed for the calcualtion of the analytical solution)!
        !-------------------------------------------------------------

        Mue_global(1:step_ix-1) = mue_e(0,1)
        Mue_global(step_ix:ng) = mue_e(N,ne)

endif
write(0,*) ''
write(0,*) '*************** Finished reading mue data! ****************'
close(25)

!read the density information from file
!--------------------------------------

if (mod_id == 'sh') then
        rho_file='DATA_rho/'//fname1(1:12)//'_rho.txt'
        open(unit=26,file=rho_file)
        i=0
        do i=1,ne
           do j=0,N
                read(26,*) rho_e(j,i)
           enddo
        enddo
elseif (mod_id == 'so') then
        rho_file='DATA_rho/'//fname1(1:4)//'00_00hom_rho.txt'
        open(unit=26,file=rho_file)
        read(26,*) rho_e(0,1)
        rho_e = rho_e(0,1)
        step_ix = 0
        Rho_global = rho_e(0,1)

elseif (mod_id == 'st') then
        rho_file='DATA_rho/'//fname1(1:4)//'00_00two_rho.txt'
        open(unit=25,file=rho_file)
        read(25,*) rho_e(0,1)
        read(25,*) rho_e(N,ne)
        read(25,*) el_step_ix(2)
        if (el_step_ix(1) ≠ el_step_ix(2)) then
                write(0,*) '*************************************************************'
                write(0,*) '***                 Warning!                      ***'
                write(0,*) '***      Properties change in different elements      ***'
                write(0,*) '***          Not a two layer case anymore          ***'
                write(0,*) '*** Calculation of an analytical solution is not possible!***'
                write(0,*) '*************************************************************'
        endif
        rho_e(:,1:el_step_ix(2)-1) = rho_e(0,1)
        rho_e(:,el_step_ix(2):ne) = rho_e(N,ne)
        step_ix = N * (el_step_ix(2)-1) + 1

        !creating the "global" Rho-vector
        !(only needed for the calcualtion of the analytical solution)!
        !-------------------------------------------------------------

        Rho_global(1:step_ix-1) = rho_e(0,1)
        Rho_global(step_ix:ng) = rho_e(N,ne)

endif
write(0,*) ''
write(0,*) '************** Finished reading density data! **************'
close(26)

!calculating the wave velocity inside each element
!-------------------------------------------------

do i=1,ne
   do j=0,N
        ve(j,i)=sqrt(mue_e(j,i)/rho_e(j,i))
   enddo
enddo

write(0,*) ''
write(0,*) 'step_ix = ', step_ix


!calculating the derivatives of the lagrange polynomials at the gll nodes
!and getting the collocation points xi(i) and the weights weights(i) for
!the gll integration rule
!-----------------------------------------------------------------------
```

```
call diff_lagrange(N,DL,weights,xi)


!defining the mapping function and the Jacobi matrix
!---------------------------------------------------

call get_shape1D(shape1D,dershape1D,xi)

call calc_jacobian1D(Jacobi,Jacobian,Jacobi_inverse,shape1D,dershape1D,xmesh)

!creating the correct x vector with non equidistant points

call create_x_irreg(x,xmesh,shape1D)

do i=1,ng-1
        dx(i)=x(i+1)-x(i)
enddo

!defining the time vector
!------------------------

  !getting the stability criterion right!

        !checking if Courant number is small enough for absorbing and periodic boundary conditions

        if (bcondition == 'a') then
          if (Courant_nr > 0.42) then
                Courant_nr = 0.42
                write(0,*) ''
                write(0,*) ''
                write(0,*) '*************************** WARNING! ***************************'
                write(0,*) ''
                write(0,*) '** Courant value is too high for absorbing boundary conditions! **'
                write(0,*) ''
                write(0,*) '** Changing the Courant value to 0.42 (maximum value possible)! **'
                write(0,*) ''
                write(0,*) '****************************************************************'
                write(0,*) ''
          endif
        endif


  dxmin = min_vector(dx)

  v_max=max_matrix2d(ve)

  dt = Courant_nr * dxmin/v_max          !Stability Criterion!!!!!!!!!!!!!

  write(0,*) ''
  write(0,*) 'dt = ', dt

  !calculating the number of time steps to be evaluated

  nt = idint(tmax / dt)

  !allocating all vectors, which length is dependent on nt

  allocate(t(1:nt),src(1:nt),stf(1:nt),cpu(1:nt),cpu_dt(1:nt-1),rectimes(1:nt),reftimes(1:nt))

  t=0
  src=0

  !time vector

  t = (/ (float(i), i=1,nt) /)*dt

!defining the source term or the initial conditions, depending on src_flag
!-------------------------------------------------------------------------


if (src_flag == .true.) then                          !point source is used

  !creating two source time functions: delta peak in time for
  !the solution of the wave equation => Green´s function of the
  !model space
  !and a ricker wavelet with dominant period pt which is later
  !convoluted with the Green´s function to obtain a synthetic
  !seismogram

  if (stfpar == 'ga') then
        call gauss(src,dt,pt)
        stf=src
  elseif (stfpar == 'r1') then
        call ricker(src,dt,pt,nt)
```

```
        stf=src
  elseif (stfpar == 'r2') then
        call ricker2(src,dt,pt,nt)
        stf=src
  elseif (stfpar == 'de') then
        call delta(nt,1,dt,src)
    if (pd ≠ 3) then
        call ricker2(stf,dt,pt,nt)
    endif

  endif

else                                          !initial conditions are used

  open(unit=19,file='DATA_initial/'//fname1(1:13)//'in1.txt')
  do i=1,ng
        read(19,*) U_init(i)
  enddo
  close(19)

  open(unit=19,file='DATA_initial/'//fname1(1:13)//'in2.txt')
  do i=1,ng
        read(19,*) Uold_init(i)
  enddo
  close(19)

  U=U_init
  Uold=Uold_init

endif


! creating the elemental mass matrices me
!----------------------------------------

        do i=1,ne
           do j=0,N
                me(j,i) = rho_e(j,i) * Jacobian(j,i) * weights(j)
           enddo
        enddo


!creating the connectivity matrix for the 1D case
!------------------------------------------------

C=0

do i=1,ne

    do j=1,M

        C(j,i)=j+(i-1)*N

    enddo

enddo


!connecting the vectors me to M
!------------------------------

call connect_vec(N,ne,ng,me,Mass,C)

!if periodic boundary conditions are used connect first and last element

   if (bcondition == 'p') then
        tmp=0.
        tmp=Mass(1)
        Mass(1)=Mass(1)+Mass(ng)
        Mass(ng)=Mass(ng)+tmp
   endif

deallocate(me)


!Adding the boundary conditions to the source vector F
!-> see subroutine calculate_F in module sem_spec_modules.f90


!setting up the receiver array according to array_id
!---------------------------------------------------

!calculating the source position

  source_ix = (src_ne-1)*N + src_N
  source_pos = x(source_ix)

!calculating the receiver positions

        if (array_id(1:1) == 'd') then
        rec_num = 1
        plot_rec_nr = 1

        elseif (array_id(1:1) == 'a') then
                rec_num = 240

        elseif (array_id(1:1) == 'b') then
                rec_num = 150

        elseif (array_id(1:1) == 'c') then
                rec_num = 100

        endif

        allocate(receiver_ix(1:rec_num),srd(1:rec_num),srd_diff(1:rec_num-1))
        allocate(receiver_pos(1:rec_num),num_el(1:rec_num))
        allocate(receiver_dx(1:rec_num))

        srd = 0.
        srd_diff = 0.
        receiver_pos = 0.
        receiver_ix = 0
        receiver_dx = 0.

        if (array_id(1:1) == 'd') srd(1) = distance
        call setup_ac_array(array_id,source_ix,receiver_ix,rec_num,srd,srd_diff,&
                            receiver_pos,receiver_dx,num_el,x,xmesh)


!allocating the seismogram

  allocate(greens(1:nt,1:rec_num))


!if plot_dev is not 3:
!preparing the plot
!------------------

if (pd ≠ 3) then

        if (N ≥ 10) then
                poldeg = "Polynomials of degree "//deg(:)
        else
                poldeg = "Polynomials of degree "//deg(2:)
        endif

        xlab="x"
        xlab=trim(xlab)
        ylab="Displacement U"
        ylab=trim(ylab)

        allocate(analytical(1,1:nt),diff_ana(1:rec_num))
        allocate(conv_ana(1:nt))
        conv_ana=0
        analytical(1,nt/2:nt)=1.
        !call conv_fft(analytical(plot_rec_nr,:),stf,conv_ana)
        call conv_fft(analytical(1,:),stf,conv_ana)

        src_max=1.2*max_vector(conv_ana)
        src_min=1.2*min_vector(conv_ana)

        deallocate(conv_ana,analytical,diff_ana)

endif


!calculating the solution to the wave equation
!---------------------------------------------

!inverting the mass matrix

do i=1,ng
   MassInv(i) = 1. / Mass(i)
enddo

deallocate(Mass)
```

```
!************************************ time loop ********************************************
!-------------------------------------------------------------------------------------------

do i=1,nt

   if (i == 1) then
        write(0,*) ''
        write(0,*) '******************** Time loop started! ********************'
        write(0,*) ''
   endif

   !compute the force at every grid point of the mesh for timestep i

   F=0

   if (bcondition == 'a') then                                  !absorbing boundaries
        bv1=rho_e(0,1)*ve(0,1)*((U(1)-Uold(1))/dt)
        bvng=rho_e(N,ne)*ve(N,ne)*((U(ng)-Uold(ng))/dt)

        call calculate_F(N,ne,DL,weights,mue_e,U,F,C,Jacobian,Jacobi_inverse,ve(src_N,src_ne),&
                         src(i),src_ne,src_N,bv1,bvng,stfpar,dt)
   else

        call calculate_F(N,ne,DL,weights,mue_e,U,F,C,Jacobian,Jacobi_inverse,ve(src_N,src_ne),&
                         src(i),src_ne,src_N,stfpar=stfpar,dt=dt)
   endif

   if (bcondition == 'p') then
        tmp=0.
        tmp=F(1)
        F(1)=F(1)+F(ng)
        F(ng)=F(ng)+tmp
   endif

   !checking for boundary conditions

   if (bcondition == 'r') then

     Unew(2:ng-1) = dt**2. * MassInv(2:ng-1) * F(2:ng-1) + 2.*U(2:ng-1) - Uold(2:ng-1)

   else

     Unew = dt**2. * MassInv * F + 2.*U - Uold

   endif

   !creating the Green´s functions

   do j=1,rec_num

        if (array_id(2:2) == '0') then          !receivers somewhere inside the elements

          call get_func_at_pos(N,num_el(j),Unew,greens(i,j),receiver_pos(j),xi,C,xmesh)

        elseif (array_id(2:2) == '1') then      !receivers exactly on GLL points
          greens(i,j) = Unew(receiver_ix(j))
        endif

   enddo


   !if plot_dev is not 3:
   !---------------------

   if (pd ≠ 3 ∧ pd ≠ 4) then

        !preparing the plot

        write(time,'(F10.5)') t(i)
        time = trim(time)

        !plot the displacement at time (i) on the intervall [0,1]


        if (min_vector(Unew) < src_min) then
                minU=1.2*min_vector(Unew)
        else
                minU=src_min
        endif
        if (max_vector(Unew) > src_max) then
                maxU=1.2*max_vector(Unew)
        else
                maxU=src_max
        endif
```

```
        if (pd == 1) then
          if (i == 1) j=0
          if (i == 1 ∨ mod(i,50) == 0) then
            j=i/50 + 1
            gif_num=nletterstring(j,3)
            plot_file=trim('DATA_TMP/'//trim(fname1(1:13))//'nr'//trim(gif_num))
            call plot_1d_snapshot(ng,real(x),real(Unew),xlab,ylab,poldeg,trim(plot_file),&
                                  pd,time,i,real(minU),real(maxU),real(source_pos))
          endif
        else
          call plot_1d_snapshot(ng,real(x),real(Unew),xlab,ylab,poldeg,'DATA_TMP/'//fname1,&
                                pd,time,i,real(minU),real(maxU),real(source_pos))
        endif
   elseif (pd == 4 ∧ i == nt) then
        pd = 0
   endif


   !prepare for next time step

   Uold=U
   U=Unew

   call cpu_time(cpu(i))

        if (percent == .true. ∧ pd ≠ 0 ∧ nt > 5000) then

        if (mod(i,10) == 0) then
                write(0,*) (float(i)/float(nt)*100.),' % done'
                write(0,*) 'cpu time =', cpu(i) - cpu(1)
        endif
        endif


enddo

!************************************ end of time loop ***********************************
!-------------------------------------------------------------------------------------------

        write(0,*) ''
        write(0,*) '******************** Time loop ended! **********************'
        write(0,*) ''


!calculating the mean value of cpu time needed for one time step dt

do i=1,nt-1
   cpu_dt(i) = cpu(i+1) - cpu(i)
enddo

cpu_dt_mean = sum(cpu_dt)/size(cpu_dt)


!if gif- or ps-file is plotted

if (pd == 1) then
   call pgpage
endif

if (pd == 2) then
   call pgpage
endif

!calculate the corresponding analytical solution

if (pd ≠ 3 ∨ txt == .true.) then

  !analytical solution only available for homogeneous or two layered model
  if (mod_id == 'so' ∨ mod_id == 'st') then
        allocate(analytical(1:rec_num,1:nt),diff_ana(1:rec_num))
        if (array_id(2:2) == '1' ∧ el_step_ix(1) == el_step_ix(2)) then
        call calcanalyt2lay_sem(analytical,receiver_ix,dt,dx,source_ix,Rho_global,Mue_global,src,&
                                step_ix,1,diff_ana,rectimes,reftimes)
        elseif (array_id(2:2) == '0' ∧ el_step_ix(1) == el_step_ix(2)) then
        call calcanalyt2lay_sem(analytical,receiver_ix,dt,dx,source_ix,Rho_global,Mue_global,src,&
                                step_ix,1,diff_ana,rectimes,reftimes,receiver_dx)
        endif
  endif
endif

!if plot_dev is not 3:
!---------------------
```

```
if (pd ≠ 3) then                                                    !only done when plotting is on

 !if point source was used
 if (src_flag == .true.) then

        allocate(conv_ana(1:nt))
        conv_ana=0
        call conv_fft(analytical(plot_rec_nr,:),stf,conv_ana)    !get convoluted analytical solution

  !if simulation used delta peak in time then check for convolution-flag "convolute"
  if (stfpar == 'de' ∧ convolute == .true.) then

        allocate(seismogram(1:nt))

        call conv_fft(greens(:,plot_rec_nr),stf,seismogram)   !get convoluted seismogram at receiver

        !plot greens function, analytical greens function together with convoluted seismogram and
        !analytical solution
        call plot_ac_seismogram(t,greens(:,plot_rec_nr),analytical(plot_rec_nr,:),&
                                'DATA_TMP/'//fname1,pd,seismogram,conv_ana,stf,&
                                diff_ana(plot_rec_nr),diff_greens)

  !if simulation used delta peak in time and convolution is not wanted
  elseif (stfpar == 'de' ∧ convolute == .false.) then

        !plot only greens function and analytical greens function
        call plot_ac_seismogram(t=t,greens=greens(:,plot_rec_nr),heavyside=&
                                analytical(plot_rec_nr,:),plot_outfile=&
                                'DATA_TMP/'//fname1,pd=pd,diff_ana=diff_ana(plot_rec_nr))

  !if simulation used a band limited signal (ricker, gaussi, etc.)
  elseif (stfpar ≠ 'de') then

        !plot the simulated seismogram and the convoluted corresponding analytical solution
        call plot_ac_seismogram(t=t,greens=greens(:,plot_rec_nr),heavyside=conv_ana(1:nt),&
        plot_outfile='DATA_TMP/'//fname1,pd=pd,stf=stf,diff_ana=diff_ana(plot_rec_nr))

  endif

 !if initial conditions were used instead of a source
 else

        !plot initial conditions
        call plot_initial(x,U_init,Uold_init,plot_outfile='DATA_TMP/'//fname1,pd=pd)

        !plot only seismogram, analytical solution is not considered here
        call plot_ac_seismogram(t=t,greens=greens(:,plot_rec_nr),&
                                heavyside=(/ (0.*dble(i),i=1,size(greens)) /),&
                                plot_outfile='DATA_TMP/'//fname1,pd=pd)

 endif

endif


!output file wanted?
!-------------------

if (txt == .true.) then

        write(0,*) ''
        write(0,*) '********** Start writing seismograms! **********'
        write(0,*) ''

   do j=1,rec_num

   !creating the analytical solution

        if (array_id(2:2) == '0') then
                dngix = srd(j)/(srd_diff(1)/(1200/rec_num))
        elseif (array_id(2:2) == '1') then
                dngix = receiver_ix(j) - source_ix
        endif

   !writing seismograms and receiver info to output files

        if (stfpar ≠ 'de') then
        i=size(greens(:,plot_rec_nr))+size(stf)-1
        if(allocated(conv_ana) == .false.) then
                allocate(conv_ana(1:nt))
        endif
        conv_ana = 0.
        call conv_fft(analytical(j,:),stf,conv_ana)
        analytical(j,:) = conv_ana(1:nt)
        endif

        call write_seismograms(fname1,greens(:,j),analytical(j,:),srd(j),j,3,dngix,&
                                        diff_ana=diff_ana(j))
  enddo

!calculating the corresponding mean dx for every array

if (array_id(2:2) == '0') then
        dx_mean = srd_diff(1)/(1200/rec_num)
elseif (array_id == 'd0') then
        dx_mean = 0.
        dx_mean = srd_diff(1)/(srd_diff(1)/(L/float(ng-1)))
elseif (array_id == 'a1') then
        dx_mean = sum(srd_diff)/float((size(srd_diff)*5))
elseif (array_id == 'b1') then
        dx_mean = sum(srd_diff)/float((size(srd_diff)*8))
elseif (array_id == 'c1') then
        dx_mean = sum(srd_diff)/float((size(srd_diff)*12))
elseif (array_id == 'd1') then
        dx_mean = (x(receiver_ix(1)) - x(source_ix))/float(receiver_ix(1) - source_ix)
endif

  !write additional data to setup file

        setupfile='DATA_TMP/'//trim('usedsetup_'//trim(fname1)//'.m')
        v_min = min_matrix2d(ve)
        if (reg_grid_flag == .true.) then
        call write_setup(setupfile,N,ne,mod_id,array_id,dt,nt,Length,v_min,&
                                tmax,cpu_dt_mean,dx_mean,ng,rec_num,source_ix,de(1))
        else
        call write_setup(setupfile,N,ne,mod_id,array_id,dt,nt,Length,v_min,&
                                tmax,cpu_dt_mean,dx_mean,ng,rec_num,source_ix)
        endif

        write(0,*) ''
        write(0,*) '********** Finished writing seismograms! **********'
        write(0,*) ''

endif

!display the avarage cpu-time per time step

write(0,*) ''
write(0,*) 'cpu_dt_mean = ',cpu_dt_mean
write(0,*) ''


END PROGRAM elastic_1D
```

```
MODULE sem_spec_modules


implicit none

!*****************************************************************
************

INTERFACE
        SUBROUTINE diff_lagrange(N,DL,rho,xi)

                Implicit none

                double precision xi(0:N),LL(0:N),VN(0:N),rho(0:N)
                double precision DL(0:N,0:N)
                integer N,i,j,k

        END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE create_x_irreg(x,xmesh,shape1D)

        implicit none

        double precision, dimension(:,:) :: shape1D
        double precision, dimension(:) :: x, xmesh
        integer N,ne,ng,NGNOD, ispec,i,j
  END SUBROUTINE
END INTERFACE

INTERFACE
        SUBROUTINE connect_vec(N,ne,ng,el,GL,C)

                double precision el(1:N+1,1:ne), GL(1:ng)

                integer C(1:N+1,1:ne)
                integer N,ne,ng
                integer i,j,k

        END SUBROUTINE
END INTERFACE

INTERFACE
        SUBROUTINE connect_mat(N,ne,ng,el,GL,C)

                double precision el(1:N+1,1:N+1,1:ne), GL(1:ng,1:ng
)

                integer C(1:N+1,1:ne)
                integer N,ne,ng
                integer i,j,k

        END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE create_S(N,ne,ng,se,S)

        implicit none

        double precision, dimension(0:N,0:N,1:ne) :: se
        double precision, dimension(:,:) :: S
        double precision, dimension(0:2*N) :: se_lang

        integer N,ne,ng,i,j
  END SUBROUTINE create_S
END INTERFACE

INTERFACE
  SUBROUTINE calculate_F(N,ne,DL,weights,lambdae,U,F,C,Jacobian,Jaco
bi_inverse,ve_src,src,src_ne,src_N,bv1,bvng,stfpar,dt)

        implicit none

        double precision, dimension(1:N+1,1:ne) :: lambdae
        double precision, dimension(1:N+1,1:N+1) :: DL
        double precision, dimension(:,:) :: Jacobian, Jacobi_invers
e
        double precision, dimension(:) :: U, F
        double precision, dimension(1:N+1) :: fe, weights, tmpx1

        double precision src, ve_src, sigma, tmp
        double precision, optional :: bv1, bvng, dt
        integer, dimension(1:N+1,1:ne) :: C
        integer i,j,k,l, ispec, N, ne, src_ne, src_N

        character(len=2), optional :: stfpar
  END SUBROUTINE
END INTERFACE

INTERFACE
  SUBROUTINE get_func_at_pos(N,num_el,func,outval,x,xigll,C,x_anchor
)

        implicit none

        double precision, dimension(:),intent(in) :: func,x_anchor
        double precision, intent(in) :: x
        double precision, dimension(1:N+1) :: l,dl,xigll
        double precision outval, xi_local

        integer, dimension(:,:),intent(in) :: C
        integer i,j,k, N,num_el
  END SUBROUTINE
END INTERFACE

END MODULE sem_spec_modules


!_____________________________________________________________________
_____________

SUBROUTINE diff_lagrange(N,DL,rho,xi)
!
!calculates the derivatives of the Lagrange Polynomials at the GLL
quadrature
!points and stores them in Matrix DL (coloumn = d (L)/d (xi) = dL,
row = dL at
!point xi

Implicit none

double precision xi(0:N),LL(0:N),VN(0:N),rho(0:N)
double precision DL(0:N,0:N)
integer N,i,j,k



!Calculate the Gauß-Lobatto-Legendre Quadrature points and the valu
es of the
!Legendre polynomials VN at the gll nodes

call main_pol(xi,N,VN)

!calculate the weights rho(i) at every gll node for the gll integra
tion rule

rho = 2 / ( float((N+1)*N) * VN**2)

!calculating the derivatives of the Polynomials at the  Gauß-Lobatt
o-Legendre Quadrature points

do i=0,N

    LL=0
    LL(i)=1
    call DELEGL(N,xi,VN,LL,DL(:,i))

enddo

END SUBROUTINE diff_lagrange

!_____________________________________________________________________
_____________
SUBROUTINE main_pol(xi,N,VN)

IMPLICIT NONE

DOUBLE PRECISION X,xi(0:N),Y,DY,D2Y,ET(0:1000),VN(0:1000)
INTEGER j,k,N

DO j=0,1000
X=-1+REAL(j)/1000.*2
CALL ZELEGL(N,ET,VN)
ENDDO
DO j=0,N

        xi(j)=ET(j)

ENDDO

END SUBROUTINE main_pol


!_____________________________________________________________________
_____________
      SUBROUTINE VALEPO(N,X,Y,DY,D2Y)

!*   COMPUTES THE VALUE OF THE LEGENDRE POLYNOMIAL OF DEGREE N
!*   AND ITS FIRST AND SECOND DERIVATIVES AT A GIVEN POINT

!*   N  = DEGREE OF THE POLYNOMIAL
!*   X  = POINT IN WHICH THE COMPUTATION IS PERFORMED

!*   Y  = VALUE OF THE POLYNOMIAL IN X
!*   DY = VALUE OF THE FIRST DERIVATIVE IN X
!*   D2Y= VALUE OF THE SECOND DERIVATIVE IN X


!      IMPLICIT DOUBLE PRECISION (A-H,O-Z)

      IMPLICIT NONE
      DOUBLE PRECISION Y,DY,D2Y,X,YP,DYP,D2YP,C1,C2,C3,C4,YM,DYM,D2
YM
      INTEGER N,I


         Y   = 1.

         DY  = 0.
         D2Y = 0.


      IF (N .EQ. 0) RETURN


         Y   = X

         DY  = 1.
         D2Y = 0.


      IF(N .EQ. 1) RETURN


         YP   = 1.

         DYP  = 0.
         D2YP = 0.

      DO I=2,N

         C1 = DFLOAT(I)
         C2 = 2.*C1-1.

         C4 = C1-1.
         YM = Y

         Y  = (C2*X*Y-C4*YP)/C1
         YP = YM

         DYM  = DY
         DY   = (C2*X*DY-C4*DYP+C2*YP)/C1

         DYP  = DYM
         D2YM = D2Y

         D2Y  = (C2*X*D2Y-C4*D2YP+2.D0*C2*DYP)/C1
         D2YP = D2YM
       ENDDO


      RETURN
END SUBROUTINE VALEPO


!_____________________________________________________________________
_____________
      SUBROUTINE ZELEGL(N,ET,VN)
```

```
!*   COMPUTES THE NODES RELATIVE TO THE LEGENDRE GAUSS-LOBATTO FORMULA
!*   N  = ORDER OF THE FORMULA
!*   ET = VECTOR OF THE NODES, ET(I), I=0,N
!*   VN = VALUES OF THE LEGENDRE POLYNOMIAL AT THE NODES, VN(I), I=0,N

!      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      IMPLICIT NONE

      DOUBLE PRECISION SN,X,Y,DY,D2Y,PI,C,ETX
      DOUBLE PRECISION ET(0:1000),VN(0:1000)
      INTEGER N,N2,I,IT

!     DIMENSION ET(0:*), VN(0:*)

      IF (N .EQ. 0) RETURN

         N2 = (N-1)/2
         SN = DFLOAT(2*N-4*N2-3)
         ET(0) = -1.
         ET(N) = 1.
         VN(0) = SN
         VN(N) = 1.
      IF (N .EQ. 1) RETURN

         ET(N2+1) = 0.
         X = 0.
      CALL VALEPO(N,X,Y,DY,D2Y)
         VN(N2+1) = Y
      IF(N .EQ. 2) RETURN

         PI = 3.14159265358979323846
         C  = PI/DFLOAT(N)
      DO I=1,N2
        ETX = DCOS(C*DFLOAT(I))
        DO IT=1,8
         CALL VALEPO(N,ETX,Y,DY,D2Y)
         ETX = ETX-DY/D2Y
        ENDDO
        ET(I) = -ETX
        ET(N-I) = ETX
        VN(I) = Y*SN
        VN(N-I) = Y
      ENDDO

      RETURN
      END SUBROUTINE ZELEGL


!_____________________________________________________________________
_____



      SUBROUTINE DELEGL(N,ET,VN,QN,DQN)


!*********************************************************************
******
!*  COMPUTES THE DERIVATIVE OF A POLYNOMIAL AT THE LEGENDRE GAUSS-LOBATTO
!*  NODES FROM THE VALUES OF THE POLYNOMIAL ATTAINED AT THE SAME POINTS
!*   N   = THE DEGREE OF THE POLYNOMIAL

!*   ET  = VECTOR OF THE NODES, ET(I), I=0,N

!*   VN  = VALUES OF THE LEGENDRE POLYNOMIAL AT THE NODES, VN(I), I=0,N
!*   QN  = VALUES OF THE POLYNOMIAL AT THE NODES, QN(I), I=0,N

!*   DQN = DERIVATIVES OF THE POLYNOMIAL AT THE NODES, DQZ(I), I=0,N
!*********************************************************************
******
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)

!      DIMENSION ET(0:*), VN(0:*), QN(0:*), DQN(0:*)

double precision ET(0:*), VN(0:*), QN(0:*), DQN(0:*)
         DQN(0) = 0.D0

      IF (N .EQ. 0) RETURN
```

```
      DO 1 I=0,N

          SU = 0.D0

          VI = VN(I)

          EI = ET(I)

      DO 2 J=0,N

      IF (I .EQ. J) GOTO 2

          VJ = VN(J)

          EJ = ET(J)

          SU = SU+QN(J)/(VJ*(EI-EJ))

2     CONTINUE

          DQN(I) = VI*SU

1     CONTINUE



          DN = DFLOAT(N)

          C  = .25D0*DN*(DN+1.D0)

          DQN(0) = DQN(0)-C*QN(0)

          DQN(N) = DQN(N)+C*QN(N)


      RETURN

      END SUBROUTINE DELEGL



!_____________________________________________________________________
_____________
!_____________________________________________________________________
_____________

SUBROUTINE create_x_irreg(x,xmesh,shape1D)

!creates a non equidistant x vector of length ng corresponding to the gll nodes
!in each element and depending on the x-vector xmesh.

implicit none

double precision, dimension(:,:) :: shape1D
double precision, dimension(:) :: x, xmesh
integer N,ne,ng,NGNOD, ispec,i,j

N=size(shape1D,2)-1
ne=size(xmesh)-1
ng=size(x)
NGNOD=size(shape1D,1)


do ispec=1,ne
  do i=1,N
    do j=1,NGNOD
       x((ispec-1)*N+i) = x((ispec-1)*N+i) + shape1D(j,i)*xmesh(ispec-1+j)
    enddo
  enddo
enddo

do j=1,NGNOD
       x(ng) = x(ng) + shape1D(j,N+1)*xmesh(ne-1+j)
enddo

END SUBROUTINE create_x_irreg

!_____________________________________________________________________
_________________
```

```
!_____________________________________________________________________
_________________

SUBROUTINE connect_vec(N,ne,ng,el,GL,C)

!subroutine for compiling a global vector GL (adding the components
 el(i,j) at the correct places)

double precision el(1:N+1,1:ne), GL(1:ng)

integer C(1:N+1,1:ne)
integer N,ne,ng
integer i,j,k

do i=1,ne

    do j=1,N+1

        GL(C(j,i))=GL(C(j,i))+el(j,i)

    enddo

enddo

END SUBROUTINE connect_vec


!_____________________________________________________________________
_____________
!_____________________________________________________________________
_____________

SUBROUTINE connect_mat(N,ne,ng,el,GL,C)

!subroutine for compiling a global matrix GL (adding the components
 el(i,j) at the correct places)

double precision el(1:N+1,1:N+1,1:ne), GL(1:ng,1:ng)

integer C(1:N+1,1:ne)
integer N,ne,ng
integer i,j,k

do i=1,ne

    do j=1,N+1

        do k=1,N+1

            GL(C(k,i),C(j,i))=GL(C(k,i),C(j,i))+el(k,j,i)

        enddo

    enddo

enddo

END SUBROUTINE connect_mat

!_____________________________________________________________________
_____________

SUBROUTINE create_S(N,ne,ng,se,S)

implicit none

double precision, dimension(0:N,0:N,1:ne) :: se
double precision, dimension(:,:) :: S
double precision, dimension(0:2*N,1:ne-1) :: se_lang

integer N,ne,ng,i,j

do i=1,ne-1
        se_lang(0:N-1,i)=se(N,0:N-1,i)
        se_lang(N,i)=se(N,N,i)+se(0,0,i+1)
        se_lang(N+1:2*N,i)=se(0,1:N,i+1)
enddo

do i=1,N
        S(i,1:N+1)=se(i-1,:,1)

        S((ne-1)*N+i+1,N+1:2*N+1)=se(i,:,ne)
enddo

S((ne-1)*N+1,:)=se_lang(:,ne-1)
```

```
do i=1,ne-2

        S(i*N+1,:)=se_lang(:,i)

        do j=2,N
        S(i*N+j,(N+2-j):(2*N+2-j))=se(j-1,:,i+1)
        enddo
enddo


END SUBROUTINE create_S


!-----------------------------------------------------------------
---------------

SUBROUTINE calculate_F(N,ne,DL,weights,lambdae,U,F,C,Jacobian,Jacobi
_inverse,ve_src,src,src_ne,src_N,bv1,bvng,stfpar,dt)

implicit none

double precision, dimension(1:N+1,1:ne) :: lambdae
double precision, dimension(1:N+1,1:N+1) :: DL
double precision, dimension(:,:) :: Jacobian, Jacobi_inverse
double precision, dimension(:) :: U, F
double precision, dimension(1:N+1) :: fe, weights, tmpx1

double precision src, ve_src, sigma, tmp
double precision, optional :: bv1, bvng, dt

integer, dimension(1:N+1,1:ne) :: C
integer i,j,k,l, ispec, N, ne, src_ne, src_N

character(len=2), optional :: stfpar

do ispec=1,ne

    fe=0

    do i=1,N+1
        tmp = 0.
      do l=1,N+1

        tmp= tmp + U(C(l,ispec)) * DL(i,l)      !calculating du/dxi
 = epsilon in 1D
      enddo

        ! calculating the stress in physical coordinates => multipl
y with dxi/dx = Jacobi_inverse
        sigma = lambdae(i,ispec) * tmp * Jacobi_inverse(i,ispec)

        ! Jacobian needed for integration and second dxi/dx needed
for derivation of
        !test function in physical coordinates
        tmpx1(i) = Jacobian(i,ispec) * sigma * Jacobi_inverse(i,isp
ec)
    enddo

    do i = 1,N+1

        tmp = 0.

        do l=1,N+1

           tmp = tmp + tmpx1(l) * DL(l,i) * weights(l)
        enddo

        fe(i) = - tmp

        if (present(bv1) .and. present(bvng)) then
           if (i == 1 .and. ispec == 1) fe(i)=fe(i)-bv1
           if (i == (N+1) .and. ispec == ne) fe(i)=fe(i)-bvng
        endif

        if (ispec == src_ne .and. i == src_N) then
          if ((present(stfpar) .or. present(dt)) == .false. .or. stfpar
== 'de') then
            fe(i) = fe(i) + 2. * src * ve_src
          else
            fe(i) = fe(i) + 2. * src * ve_src / dt
          endif
        endif

    enddo

    do i = 1,N+1
        F(C(i,ispec))=F(C(i,ispec))+fe(i)
    enddo

enddo


END SUBROUTINE calculate_F


!-----------------------------------------------------------------
---------------

SUBROUTINE get_func_at_pos(N,num_el,func,outval,x,xigll,C,x_anchor)

use lagrange_poly

implicit none

double precision, dimension(:),intent(in) :: func,x_anchor
double precision, intent(in) :: x
double precision, dimension(1:N+1) :: l,dl,xigll
double precision outval, xi_local

integer, dimension(:,:),intent(in) :: C
integer i,j,k, N,num_el

xi_local = 0.
outval = 0.

xi_local = -1. + (x - x_anchor(num_el))/(x_anchor(num_el+1) - x_anc
hor(num_el))*2.

call lagrange_any(xi_local,N+1,xigll,l,dl)

do i=1,N+1
        outval = outval + func(C(i,num_el)) * l(i)
enddo


END SUBROUTINE get_func_at_pos

!-----------------------------------------------------------------

!=====================================================================
!
!          S p e c f e m 3 D  G l o b e  V e r s i o n  3 . 3
!          ---------------------------------------------------
!
!                 Dimitri Komatitsch and Jeroen Tromp
!    Seismological Laboratory - California Institute of Technology
!        (c) California Institute of Technology September 2002
!
!   A signed non-commercial agreement is required to use this progr
am.
!  Please check http://www.gps.caltech.edu/research/jtromp for deta
ils.
!           Free for non-commercial academic research ONLY.
!      This program is distributed WITHOUT ANY WARRANTY whatsoever.
!      Do not redistribute this program without written permission.
!
!=====================================================================

MODULE lagrange

END MODULE

!-----------------------------------------------------------------

  subroutine lagrange_any(xi,NGLL,xigll,h,hprime)

! subroutine to compute the Lagrange interpolants based upon the GL
L points
! and their first derivatives at any point xi in [-1,1]

  implicit none

  integer NGLL
  double precision xi,xigll(NGLL),h(NGLL),hprime(NGLL)

  integer dgr,i,j
  double precision prod1,prod2

  do dgr=1,NGLL
  prod1 = 1.0d0
  prod2 = 1.0d0
  do i=1,NGLL
    if(i .ne. dgr) then
      prod1 = prod1*(xi-xigll(i))
      prod2 = prod2*(xigll(dgr)-xigll(i))
    endif
  enddo
  h(dgr)=prod1/prod2

  hprime(dgr)=0.0d0
  do i=1,NGLL
    if(i .ne. dgr) then
      prod1=1.0d0
      do j=1,NGLL
        if(j .ne. dgr .and. j .ne. i) prod1 = prod1*(xi-xigll(j))
      enddo
      hprime(dgr) = hprime(dgr)+prod1
    endif
  enddo
  hprime(dgr) = hprime(dgr)/prod2

  enddo

  end subroutine lagrange_any

!
!=====================================================================
===
!

! subroutine to compute the derivative of the Lagrange interpolants
! at the GLL points at any given GLL point

  double precision function lagrange_deriv_GLL(I,j,ZGLL,NZ)

!-----------------------------------------------------------------
------

!
!     Compute the value of the derivative of the I-th
!     Lagrange interpolant through the
!     NZ Gauss-Lobatto Legendre points ZGLL at point ZGLL(j)
!
!-----------------------------------------------------------------
------

  implicit none

  integer i,j,nz
  double precision zgll(0:nz-1)

  integer degpoly

  double precision, external :: pnleg,pndleg

  degpoly = nz - 1
  if (i == 0 .and. j == 0) then
    lagrange_deriv_GLL = - dble(degpoly)*(dble(degpoly)+1.d0) / 4.d
0
  else if (i == degpoly .and. j == degpoly) then
    lagrange_deriv_GLL = dble(degpoly)*(dble(degpoly)+1.d0) / 4.d0
  else if (i == j) then
    lagrange_deriv_GLL = 0.d0
  else
    lagrange_deriv_GLL = pnleg(zgll(j),degpoly) / &
      (pnleg(zgll(i),degpoly)*(zgll(j)-zgll(i))) &
      + (1.d0-zgll(j)*zgll(j))*pndleg(zgll(j),degpoly) / (dble(degp
oly)* &
      (dble(degpoly)+1.d0)*pnleg(zgll(i),degpoly)*(zgll(j)-zgll(i))
*(zgll(j)-zgll(i)))
  endif

  end function lagrange_deriv_GLL
```

```
MODULE jacobian_mod

INTERFACE
  SUBROUTINE calc_jacobian1D(Jacobi,Jacobian,Jacobi_inverse,shape1D,
dershape1D,xmesh)

        implicit none

        double precision, dimension(:,:) :: Jacobi, Jacobian, Jacob
i_inverse
        double precision, dimension(:,:) :: shape1D,dershape1D
        double precision, dimension(:) :: xmesh
        double precision xelm

        integer ispec, i,j,k, NGLLX, NGNOD, ne
  END SUBROUTINE
END INTERFACE

END MODULE


!_____________________________________________________________________
_____


SUBROUTINE calc_jacobian1D(Jacobi,Jacobian,Jacobi_inverse,shape1D,der
shape1D,xmesh)

!subroutine calculates the Jacobi-Matrix, the Jacobian of the
!Mapping Function and the Jacobi-Matrix of the inverse Mapping
!Function for a 1-D SEM  (main program: elastic_1D)

implicit none

double precision, dimension(:,:) :: Jacobi
double precision, dimension(:,:) :: Jacobian
double precision, dimension(:,:) :: Jacobi_inverse

double precision, dimension(:,:) :: shape1D
double precision, dimension(:,:) :: dershape1D
double precision, dimension(:) :: xmesh
double precision xelm

integer ispec, i,j,k, NGLLX, NGNOD, ne

NGNOD=size(shape1D,1)
NGLLX=size(shape1D,2)

ne = size(Jacobi,2)

do ispec=1,ne
  do i=1,NGLLX
    do j=1,NGNOD
       xelm = xmesh(ispec-1+j)
       Jacobi(i,ispec) = Jacobi(i,ispec) + dershape1D(j,i)*xelm
       Jacobian(i,ispec) = Jacobi(i,ispec)
       Jacobi_inverse = 1./Jacobian
    enddo
  enddo
enddo

END SUBROUTINE
```

```
MODULE shape_mod

! Changed to 1D by Bernhard Schuberth 02.06.2003 for serial applica
tions

!========================================================================
===
!
!          S p e c f e m 3 D  G l o b e  V e r s i o n  3 . 3
!          ---------------------------------------------
!
!                  Dimitri Komatitsch and Jeroen Tromp
!    Seismological Laboratory - California Institute of Technology
!        (c) California Institute of Technology September 2002
!
!    A signed non-commercial agreement is required to use this prog
ram.
!   Please check http://www.gps.caltech.edu/research/jtromp for det
ails.
!           Free for non-commercial academic research ONLY.
!      This program is distributed WITHOUT ANY WARRANTY whatsoever.
!      Do not redistribute this program without written permission.
!
!========================================================================
===

INTERFACE
  SUBROUTINE get_shape1D(shape1D,dershape1D,xigll)

        implicit none

        double precision xigll(:)
        double precision shape1D(:,:)
        double precision dershape1D(:,:)

        integer i,j,k,ia, NGLLX, NGNOD

        double precision xi
        double precision l1xi,l2xi,l3xi
        double precision l1pxi,l2pxi,l3pxi

        double precision sumshape,sumdershapexi
        double precision, parameter :: TINYVAL=1e-9

        double precision, parameter :: ONE = 1.
        double precision, parameter :: HALF = 1./2.
        double precision, parameter :: ZERO = 0.
        double precision, parameter :: TWO = 2.
  END SUBROUTINE
END INTERFACE

END MODULE


!_____________________________________________________________________

  subroutine get_shape1D(shape1D,dershape1D,xigll)

  implicit none

! Gauss-Lobatto-Legendre points of integration
  double precision xigll(:)

! 1D shape functions and their derivatives
  double precision shape1D(:,:)
  double precision dershape1D(:,:)

  integer i,j,k,ia, NGLLX, NGNOD

! location of the nodes of the 1D quadrilateral elements
  double precision xi
  double precision l1xi,l2xi,l3xi
  double precision l1pxi,l2pxi,l3pxi

! for checking the 1D shape functions
  double precision sumshape,sumdershapexi
  double precision, parameter :: TINYVAL=1e-9

! defining 1, 1/2 and 0 and 2

  double precision, parameter :: ONE = 1.
  double precision, parameter :: HALF = 1./2.
  double precision, parameter :: ZERO = 0.
  double precision, parameter :: TWO = 2.
```

```
NGNOD = size(shape1D,1)
NGLLX = size(shape1D,2)


if (NGNOD == 2) then

! generate the 1D shape functions and their derivatives (2 nodes)

  do i=1,NGLLX

  xi=xigll(i)

  l1xi=(ONE - xi) * HALF
  l2xi=(ONE + xi) * HALF

  l1pxi=-HALF
  l2pxi=HALF

!     corner nodes

      shape1D(1,i)=l1xi
      shape1D(2,i)=l2xi

      dershape1D(1,i)=l1pxi
      dershape1D(2,i)=l2pxi

  enddo

elseif (NGNOD == 3) then

! generate the 1D shape functions and their derivatives (3 nodes)

  do i=1,NGLLX

  xi=xigll(i)

  l1xi=HALF*xi*(xi-ONE)
  l2xi=ONE-xi**2
  l3xi=HALF*xi*(xi+ONE)

  l1pxi=xi-HALF
  l2pxi=-TWO*xi
  l3pxi=xi+HALF


!     corner nodes

      shape1D(1,i)=l1xi
      shape1D(2,i)=l2xi
      shape1D(3,i)=l3xi

      dershape1D(1,i)=l1pxi
      dershape1D(2,i)=l2pxi
      dershape1D(3,i)=l3pxi

  enddo

endif

! check the shape functions
  do i=1,NGLLX

      sumshape=ZERO

      sumdershapexi=ZERO

      do ia=1,NGNOD

        sumshape=sumshape+shape1D(ia,i)

        sumdershapexi=sumdershapexi+dershape1D(ia,i)

      enddo

!     the sum of the shape functions should be 1
      if(abs(sumshape-ONE) > TINYVAL) print *, 'error in 1D shape functions
'

!     the sum of the derivatives of the shape functions should be 0
      if(abs(sumdershapexi) > TINYVAL) &
        print *, 'error in xi derivatives of 1D shape function'

  enddo

  end subroutine get_shape1D
```

```
!_____________________________________________________________________
!Excerpt of the 1-D wave propagation code with Newmark-type
!integration scheme, using an acceleration formulation and no
!additional iterations.
!_____________________________________________________________________

!time loop

do i=1,nt

   ! update displacement using finite difference time scheme

     do j=1,ng
       U(j) = U(j) + dt*Veloc(j) + (dt**2)/2. * Accel(j)
       Veloc(j) = Veloc(j) + dt/2. * Accel(j)
       Accel(j) = 0.
     enddo

   !compute the force at every grid point of the mesh for
   !timestep i

         call calculate_F(N,ne,DL,weights,mue_e,U,Accel,C,Jacobian,&
                          Jacobi_inverse,ve(src_N,src_ne),src(i),&
                          src_ne,src_N,stfpar=stfpar,dt=dt)

   !checking for boundary conditions

         do j=1,ng
                Accel(j) = Accel(j) * MassInv(j) / 2.
                Veloc(j) = Veloc(j) + dt/2. * Accel(j)
                U(j) = U(j) + dt*Veloc(j) + (dt**2)/2. * Accel(j)
         enddo

enddo


!_____________________________________________________________________
!Excerpt of the 1-D wave propagation code with Newmark-type
!predictor-corrector integration scheme, using an acceleration
!formulation and one iteration.
!_____________________________________________________________________

Vpred = 0.

    deltat = dt
    deltatover2 = 0.5d0*dt
    deltatsqover2 = 0.5d0*dt*dt


!time loop

do i=1,nt

! compute predictors

    do j=1,ng
        U(j) = U(j) + deltat*Veloc(j) + deltatsqover2*Accel(j)
        Vpred(j) = Veloc(j) + deltatover2*Accel(j)
        Veloc(j) = Vpred(j)
        Accel(j) =0.
    enddo

  do iter = 1,2 !NITER

! compute internal forces

         call calculate_F(N,ne,DL,weights,mue_e,U,Accel,C,Jacobian,&
                          Jacobi_inverse,ve(src_N,src_ne),src(i),&
                          src_ne,src_N,stfpar=stfpar,dt=dt)

    do j=1,ng
     if(iter == 1) then
      Accel(j) = Accel(j)*MassInv(j)
     endif
      Veloc(j) = Vpred(j) + deltatover2*Accel(j)
    enddo

  enddo

enddo
```

```
SUBROUTINE multiply_S_U(nn,nng,stiff,U_int,stiffmU)

!calculates the matrix-vector product of the condensed stiffness ma
trix stiff, created by
!create_S, with the displacement vector U_int

implicit none

double precision, dimension(1:nng,0:2*nn) :: stiff
double precision, dimension(1:nng) :: U_int,stiffmU

integer j,k,nn,nng

   stiffmU = 0

   do j=1,nn

      do k=0,2*nn
         stiffmU(j) = stiffmU(j) + stiff(j,k) * U_int(k+1)
      enddo
   enddo

   do j=nn+1,nng-nn

      do k=-nn,nn
         stiffmU(j) = stiffmU(j) + stiff(j,k+nn) * U_int(j+k)
      enddo
   enddo

   do j=nng-nn+1,nng

      do k=0,2*nn
         stiffmU(j) = stiffmU(j) + stiff(j,k) * U_int(nng-2*nn+k)
      enddo
   enddo

END SUBROUTINE multiply_S_U
```

List of Figures

List of Tables

Bibliography

Abramowitz, M.; Stegun, I. A. (Editors), *Pocketbook of Mathematical Functions, Abridged ed. of "Handbook of Mathematical Functions"*. Harri Deutsch, Frankfurt a. M., **1984**.

Aki, K.; Richards, P. G., *Quantitative Seismology*. University Science Books, Sausalito, California, **2002**.

Ampuero, J.-P.; Vilotte, J.-P.; Festa, G., *Dynamic rupture with the spectral element method: where do we stand?*. In *Proc. of the Workshop on Numerical Modeling of Earthquake Source Dynamics (1.-3. Sept. 2003)*, Smolenice Castle, Slovak Republic, **2003**, p. 9.

Bronstein, J. N.; Semendjajew, K. A.; Musiol, G.; Mühlig, H., *Taschenbuch der Mathematik, 4. Auflage*. Harri Deutsch, Frankfurt a. M., **1999**.

CalTech, **2002**, URL `http://www.gps.caltech.edu/~jtromp/research/downloads.html`. Last modified 11.06.2002.

Capdeville, Y., *Méthode couplée éléments spectraux - solution modale pour la propagation d'ondes dans la Terre* à *l'échelle globale*. Ph.D. thesis, Université Paris 7, Paris, France, **2000**.

Capdeville, Y.; Chaljub, E.; Vilotte, J.-P.; Montagner, J.-P., *Coupling the spectral element method with a modal solution for elastic wave propagation in global earth models*. Geophys. J. Int., 152, **2003**, 34–67.

Chaljub, E., *Modélisation numérique de la propagation d'ondes sismiques en géométrie sphérique: application* à *la sismologie globale (Numerical modelling of the propagation of seismic waves in spherical geometrie: applicaton to global seismologyi)*. Ph.D. thesis, Université Paris VII Denis Diderot, Paris, France, **2000**.

Chaljub, E.; Capdeville, Y.; Vilotte, J.-P., *Solving elastodynamics in a fluid solid heterogeneous sphere: a parallel spectral element approximation on non-conforming grids*. J. Comput. Phys., 187, **2003**, 457–491.

CLAYTON, R.; ENGQUIST, B., *Absorbing boundary conditions for acoustic and elastic wave equations*. Bull. Seismol. Soc. Am., 67(6), **1977**, 1529–1540.

CLOUGH, R. W., *The finite element in plane stress analysis*. In *Proc. 2nd ASCE Conf. on Electronic Computation*, Pittsburgh, Pa., **1960**.

COLLINO, F.; TSOGKA, C., *Application of the perfectly matched absorbing layer model to the linear elastodynamic problem in anisotropic heterogeneous media*. Geophysics, 66(1), **2001**, 294–307.

FORNBERG, B., *The pseudospectral method - Comparisons with finite differences for the elastic wave equation*. Geophysics, 52(04), **1987**, 483–501.

FUNARO, D., *FORTRAN Routines for Spectral Methods*. Instituto di Analisi Numerica, Pavia, **1993**.

GABLE, C.; CHERRY, T., **2000**, URL
`http://www.ees.lanl.gov/EES5/geomesh/catalogue/NTS_TYBO_EMBED_00/methods1.html`.
last modified 21.11.2000; e-mail: gable@lanl.gov, tcherry@lanl.gov.

GALERKIN, B. G., *Series solution of some problems of elastic equilibrium of rods and plates (russian)*. Vestn. Inzh. Tech., 19, **1915**, 897–908.

GELLER, R. J.; TAKEUCHI, N., *A new method for computing highly accurate dsm synthetic seismograms*. Geophys. J. Int., 123, **1995**, 449–470.

HUGHES, T. J. R., *The Finite Element Method - Linear Static and Dynamic Finite Element Analysis*. Prentice Hall, Englewood Cliffs, London, **1987**.

JUNG, M.; LANGER, U., *Methode der finiten Elemente für Ingenieure*. Teubner Verlag, Wiesbaden, **2001**.

KELLY, K. R.; MARFURT, K. J., *Editor's introduction to Chapter 1 - Classical finite-difference and finite-element methods*. In MARFURT, K. J. (Editor), *Numerical modeling of seismic wave propagation*, Soc. Expl. Geophys., **1990**, pp. 1–3.

KOMATITSCH, D., *Méthodes spectrales et éléments spectraux pour l'équation de l'élastodynamique 2D et 3D en milieu hétérogène (Spectral and spectral-element methods for the 2D and 3D elastodynamics equations in heterogeneous media)*. Ph.D. thesis, Institut de Physique du Globe, Paris, France, **1997**.

KOMATITSCH, D.; BARNES, C.; TROMP, J., *Simulation of anisotropic wave propagation based upon a spectral element method*. Geophysics, 65(4), **2000**, 1251–1260.

KOMATITSCH, D.; LIU, Q.; TROMP, J.; SÜSS, P.; STIDHAM, C.; SHAW, J. H., *Simulations of strong ground motion in the Los Angeles Basin based upon the spectral-element method*. Bull. Seismol. Soc. Am., **2003**a. Submitted.

KOMATITSCH, D.; MARTIN, R.; TROMP, J.; TAYLOR, M. A.; WINGATE, B. A., *Wave propagation in 2-D elastic media using a spectral element method with triangles and quadrangles*. J. Comput. Acoust., 9(2), **2001**, 703–718.

KOMATITSCH, D.; TROMP, J., *Introduction to the spectral-element method for 3-D seismic wave propagation*. Geophys. J. Int., 139, **1999**, 806–822.

KOMATITSCH, D.; TROMP, J., *Modeling of seismic wave propagation at the scale of the Earth on a large Beowulf*. Proceedings of the ACM/IEEE Supercomputing SC'2001 conference, **2001**a. Published on CD-ROM and on www.sc2001.org.

KOMATITSCH, D.; TROMP, J., *Modeling seismic wave propagation on a 156 GB PC cluster*. Linux Journal, October 2001 issue, **2001**b, 38–45.

KOMATITSCH, D.; TROMP, J., *Spectral-element simulations of global seismic wave propagation-I. Validation*. Geophys. J. Int., 149, **2002**a, 390–412.

KOMATITSCH, D.; TROMP, J., *Spectral-element simulations of global seismic wave propagation-II. 3-D models, oceans, rotation, and self-gravitation*. Geophys. J. Int., 150, **2002**b, 303–318.

KOMATITSCH, D.; TROMP, J., *A Perfectly Matched Layer absorbing boundary condition for the second-order seismic wave equation*. Geophys. J. Int., 154, **2003**, 146–153.

KOMATITSCH, D.; TSUBOI, S.; CHEN, J.; TROMP, J., *A 14.6 billion degrees of freedom, 5 teraflop, 2.5 terabyte earthquake simulation on the earth simulator*. Proceedings of the ACM/IEEE Supercomputing SC'2003 conference, **2003**b. In press.

KOMATITSCH, D.; VILOTTE, J.-P., *The spectral-element method: an efficient tool to simulate the seismic response of 2D and 3D geological structures*. Bull. Seismol. Soc. Am., 88(2), **1998**, 368–392.

KOMATITSCH, D.; VILOTTE, J.-P.; VAI, R.; CASTILLO-COVARRUBIAS, J. M.; SÁNCHEZ-SESMA, F. J., *The Spectral Element method for elastic wave equations: application to 2D and 3D seismic problems*. Int. J. Numer. Meth. Engng., 45, **1999**, 1139–1164.

KREISS, H. O.; OLIGER, J., *Comparison of accurate methods for the integration of hyperbolic equations*. Tellus, 24, **1972**, 199–215.

MADARIAGA, R., *Dynamics of an expanding circular fault*. Bull. Seismol. Soc. Am., 65, **1976**, 163–182.

MADAY, Y.; PATERA, A. T., *Spectral element methods for the incompressible Navier-Stokes equations*, **1989**, 71–143. Editors: A. K. Noor and J. T. Oden.

METZ, T., *Optimale Operatoren fr die numerische Simulation von Wellenausbreitung - Ableitung,Implementierung und Verifikation in 1,2und 3D*. Master's thesis, Dept. für Geo- und Umweltwissenschaften, Sektion Geophysik, Ludwig-Maximilians-Universität, München, Germany, **2003**. Unpublished.

MÜLLER-HANNEMANN, M., **2000**, URL `http://www.math.tu-berlin.de/~mhannema/pictures/hex/dfold.html`. Last modified: 05.02.2000; e-mail: mhannema@math.TU-Berlin.DE.

PADOVANI, E.; PRIOLO, E.; SERIANI, G., *Low- and high-order finite element method: Experience in seismic modeling*. J. Comput. Acoust., 2(4), **1994**, 371–422.

PATERA, A. T., *A spectral element method for fluid dynamics: laminar flow in a channel expansion*. J. Comput. Phys., 54, **1984**, 468–488.

PEARSON, T., **2002**, URL `http://www.astro.caltech.edu/ tjp/pgplot/`. Last modified 20.10.2003.

PRIOLO, E., *Earthquake ground motion simulation through the 2-D spectral element method*. In *Paper presented at the Int. Conf. on Computational Acoustics (ICTCA 1999)*, Trieste, Italy, **1999**.

PRIOLO, E.; CARCIONE, J. M.; SERIANI, G., *Numerical simulation of interface waves by high-order spectral modeling techniques*. J. Acoust. Soc. Am., 95(2), **1994**, 681–693.

PRIOLO, E.; SERIANI, G., *A numerical investigation of Chebyshev spectral element method for acoustic wave propagation*. In *Proc. 13th IMACS Conf. on Comp. Appl. Math.,v. 2*, Dublin, Ireland, **1991**, pp. 551–556.

RITZ, W., *Über eine neue methode zur lösung gewisser variations - probleme der mathematischen physik*. J. Reine Angew. Math., 135, **1909**, 1–61.

SCHWARZ, H. R., *Methode der Finiten Elemente, 2. Auflage*. Teubner, Stuttgart, **1984**.

SERIANI, G., *3-D large-scale wave propagation modeling by a spectral element method on a Cray T3E multiprocessor*. Comp. Meth. Appl. Mech. Eng., 164, **1998**, 235–247.

SERIANI, G.; PRIOLO, E., *Spectral element method for acoustic wave simulation in heterogeneous media*. Finite Elements in Analysis and Design, 16, **1994**, 337–348.

STACEY, R., *Improved transparent boundary formulations for the elastic wave equation*. Bull. Seismol. Soc. Am., 6(78), **1988**, 2089–2097.

TURNER, M. J.; CLOUGH, R. W.; MARTIN, H. C.; TOPP, L. J., *Stiffness and deflection analysis of complex structures*. J. Aero. Sci., 23, **1956**, 805–823.

UDIAS, A., *Principles of Seismology*. Cambridge Univ. Press, Cambridge, **1999**.

VIRIEUX, J., *P-sv wave propagation in heterogeneous media: velocity-stress finite-difference method*. 51(4), **1986**, 889–901.

WANG, R., *A simple orthonormalization method for stable and efficient computation of green's functions*. Bull. Seismol. Soc. Am., 89(3), **1999**, 733–741.

ZIENKIEWICZ, O. C.; TAYLOR, R. L., *The Finite Element Method, vol. 1*. Butterworth-Heinemann, Oxford, **2000**.

Selbständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel und Quellen angefertigt habe.

München, den