Utrecht University

Faculty of Science

Numerically solving the wave equation using the finite element method

Bachelor's Thesis

Jasper Everink

Mathematics

Supervisor:

Dr. Tristan van Leeuwen
Mathematical Institute

7th June 2018

Contents

0 Introduction

0.1 Wave equation

The *wave equation* is a partial differential equation that is used in many field of physics. It is used to model different kind of waves, for example sound waves and the oscillation of strings. The wave equation can be defined in d-dimensions as

$$\frac{\partial^2 u}{\partial t^2} - c^2 \Delta u = 0 \quad \text{with} \quad c > 0, \tag{0.1.1}$$

where u is a function from $\mathbb{R}^d \times \mathbb{R}$ to $\mathbb{R}$ and Δ is the Laplacian over the spatial coordinates defined as

$$\Delta u = \sum_{n=1}^{d} \frac{\partial^2 u}{\partial x_n^2}. \tag{0.1.2}$$

A special case is the *one-dimensional wave equation* which can be more simply written as

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0 \quad \text{with} \quad c > 0. \tag{0.1.3}$$

The function u can be the displacement of a string or membrane of a drum, the height of a water wave or air pressure for sound waves, dependant on the field the equation is used in.

The constant c in the above equations defines the speed at which the wave "moves". To see this we can look at the one dimensional wave equation for which, if we have no other conditions, $u(x,t) = f(x+ct)$ and $u(x,t) = f(x-ct)$ are solutions if the function $f(\eta)$ is twice differentiable. These two solutions have a constant shape which move in time with speed c to the left and right respectively. A general solution to the one-dimensional wave equation was actually derived by d'Alembert and is given by $u(x,t) = f(x-ct) + g(x+ct)$, being the sum of a left and right moving wave.

In this thesis the focus will be on the one-and two-dimensional wave equations over a bounded domain D where the spatial boundary of the domain ∂D is fixed at zero and we have an initial value in time. These initial boundary conditions are specified as follows

$$\begin{aligned} u(x,t) &= 0 \quad \text{for all} \quad x \in \partial D \quad \text{and} \quad t \in \mathbb{R}, \\ u(x,0) &= u_0(x) \quad \text{for all} \quad x \in D \quad \text{and} \\ \frac{\partial u}{\partial t}(x,0) &= u_1(x) \quad \text{for all} \quad x \in D. \end{aligned} \tag{0.1.4}$$

Here, $u(x,t)$ represents the the displacement of a string (in one-dimension) or membrane of a drum (in two-dimensions) that is held fixed at the boundary.

0.2 Finite element method

The *finite element method* is a numerical method generally used to solve differential equations with boundary conditions. The concept of the finite element method is to divide the domain into finitely many smaller areas called elements and approximate the solution of the differential equation on these elements using a suitable set of basis functions. The result is a system of equations which can either be solved directly or by another numerical method.

The finite element method has a few advantages over other numerical methods for solving differential equations (like the finite difference method). Firstly, the finite element method approximates the solution for almost[1] the whole domain, while most other methods only approximate the solution at a discrete set of points. Secondly, with the finite element method we have more freedom over the discretization of the domain. We have the choice to increase precision at parts of the domain by using smaller elements and to

[1]In chapter 4 we will see some cases where we cannot easily find an approximation for over the whole domain.

decrease the precision at parts by using larger elements. Thirdly we can choose different basis functions on the elements to increase or decrease the precision at different parts of the domain or choose basis functions which better approximate the actual solution, for example by using trigonometric functions instead of polynomials.

In this thesis we will use the finite element method for spatial discretization of the wave equation 0.1.1 and solve the resulting system of ordinary differential equations using a finite difference method. In the one-dimensional case the elements will be intervals and we will use arbitrary degree polynomials for the approximation. In the two-dimensional case the elements we will consider are triangular and we will use linear approximation.

0.3 This thesis

Most of the practical side of this thesis is based on "Finite Element Methods: A Practical Guide"[1], but explained through the wave equation. The first two chapters of this thesis cover the one-dimensional case and are based on chapter 2, 3 and 4 of that book[1]. In chapter 3 the numerical method for the one-dimensional case is analysed and certain convergence and stability properties are discussed. Chapter 4 discusses a generalisation of the finite element method to the two-dimensional wave equation. The chapters based on the book[1] use the basic steps as described in subsection 2.9 of that book, which can be summarized as follows:

1 Derive a "weaker" formulation of the differential equation.

2 Define the spatial discretization of the domain into elements and nodes.

3 Define the basis functions for the approximation.

4,5 Derive a system of equations.

6 Solve the system of equations.

The numerical methods derived in the chapters have also been implemented in Python and are used to make most of the plots. An explanation of some important parts of the code can be found in appendix A and the implementations can be found at `www.github.com/jeverink/BachelorsThesis`.

1 One-dimensional wave equation

The first 4 sections of this chapter are based on chapters 2 and 3 of [1].

In this chapter a numerical method is derived for the wave equation in one dimension on the unit interval $[0, 1]$.

The *one-dimensional wave equation* is defined as follows:

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0 \quad \text{for} \quad c > 0, \tag{1.0.1}$$

with initial boundary conditions

$$\begin{aligned} u(0,t) = u(1,t) &= 0 \quad \text{for all} \quad t \in \mathbb{R}, \\ u(x,0) &= u_0(x) \quad \text{for all} \quad x \in [0,1] \quad \text{and} \\ \frac{\partial u}{\partial t}(x,0) &= u_1(x) \quad \text{for all} \quad x \in [0,1]. \end{aligned} \tag{1.0.2}$$

1.1 Weak formulation

First we will derive a different but equivalent formulation for the one-dimensional wave equation defined in 1.0.1. Let $v(x)$ be a differentiable functions such that $v(0) = v(1) = 0$ and therefore zero on the boundaries of the domain. By multiplying the wave equation with this function and integrating it over the unit interval we get following expression

$$0 = \int_0^1 \left(\frac{\partial^2 u}{\partial t^2}(x,t) - c^2 \frac{\partial^2 u}{\partial x^2}(x,t) \right) v(x) dx. \tag{1.1.1}$$

This expression can be rewritten, using integration by parts, to

$$\begin{aligned} 0 &= \int_0^1 \frac{\partial^2 u}{\partial t^2}(x,t) v(x) dx - c^2 \left(\left[\frac{\partial u}{\partial x}(x,t) v(x) \right]_0^1 - \int_0^1 \frac{\partial u}{\partial x}(x,t) \frac{\partial v}{\partial x}(x,t) dx \right) \\ &= \int_0^1 \frac{\partial^2 u}{\partial t^2}(x,t) v(x) dx + c^2 \int_0^1 \frac{\partial u}{\partial x}(x,t) \frac{\partial v}{\partial x}(x,t) dx \\ &= a(u(x,t), v(x)), \end{aligned} \tag{1.1.2}$$

$$\text{with} \quad a(u(x,t), v(x)) := \int_0^1 \frac{\partial^2 u}{\partial t^2}(x,t) v(x) dx + c^2 \int_0^1 \frac{\partial u}{\partial x}(x,t) \frac{\partial v}{\partial x}(x,t) dx. \tag{1.1.3}$$

For the integrals above to exist we require certain properties of u, v and their derivatives.
For this we will use the L_2*-norm* on a domain D. For a function f this is defined by

$$||f||_2 := \int_D |f(x)|^2 dx. \tag{1.1.4}$$

Using this norm a function f is called *square-integrable* if the L_2-norm of f is finite or more specific

$$||f||_2 < \infty. \tag{1.1.5}$$

The set of all square-integrable functions is denoted by L_2. An important property of square-integrable functions is that the integral of the product of two square-integrable functions over the domain D is also finite. This property is important for the existence of the integrals in 1.1.2.

We want the functions u, v and their first derivatives with respect to x to be square-integrable. The set of functions which are square-integrable and whose first derivative is square-integrable is called a *first order Sobelov space* and can be denoted by H^1. The set of functions which are also zero at the boundary of the domain is denoted by H_0^1.

We now want to find a function u which is in H_0^1 for each moment in time t such that for every function v in H_0^1 the equations of 1.1.2 hold.
This gives rise to the *weak formulation* of our one-dimensional wave equation, which can now be stated as follows

Definition 1.1.1 (Weak formulation)**.** Find $u(x,t)$ such that for all $t \in \mathbb{R}$ it holds that $u(\cdot,t) \in H_0^1$ and for all $v \in H_0^1$

$$a(u(x,t), v(x)) = 0, \tag{1.1.6}$$

and the initial boundary conditions 1.0.2 holds.

This is called the weak formulation as the function u only has to hold in 1.1.6 with respect to certain functions v called *test functions.*
Except for the derivation of the weak formulation above the rigorous connection between the weak formulation and the "strong formulation" 1.0.1 and 1.0.2 is beyond the scope of this thesis.

1.2 Nodes and Elements

We will now divide the interval $[0,1]$ into N elements of equal size separated by nodes.
The size of each element is given by $h = \frac{1}{N}$, therefore the *nodes* can be defined as

$$x_i := (i-1)h \quad \text{for} \quad i = 1, \ \dots \ , N+1, \tag{1.2.1}$$

and the *elements* are defined as the intervals

$$E_i := [x_i, x_{i+1}] \quad \text{for} \quad i = 1, \ \dots \ , N. \tag{1.2.2}$$

The nodes and elements are illustrated in the figure below.
Irregular elements are treated in Chapter 2.

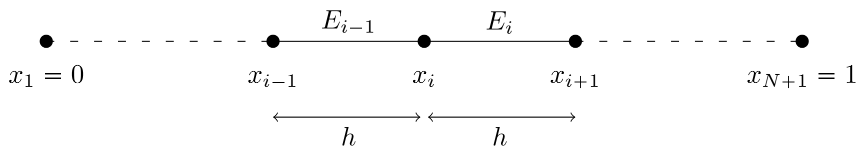


Figure 1.1: A regular partitioning of the unit interval $[0,1]$ in elements and nodes.

1.3 Basis functions

Instead of searching for a function u in H_0^1 such that for all test functions v in H_0^1 the equation of the weak formulation hold, we are going to search in a finite-dimensional subspace of H^1. This space will consist of those functions in H^1 which are piecewise polynomial on each element. In this chapter we will restrict ourselves to piecewise linear functions. We will denote this finite-dimensional set of functions as S^1.

The set S^1 will be defined as the functions v that can be written as a linear combination of finitely many basis functions and that are zero at the boundaries. We will take $N+1$ basis functions, one for each node, and call these $\phi_i(x)$ for $i = 1, \ \dots \ , N+1$. Then the functions v in S^1 can be written as

$$\begin{aligned} v(x) &= \sum_{i=1}^{N+1} v_i \phi_i(x), \quad \text{with} \quad v_i \in \mathbb{R} \quad \text{and} \\ v(0) &= v(1) = 0. \end{aligned} \tag{1.3.1}$$

To get piecewise linear functions on the elements the basis functions are defined as follows.
For $j = 2, \ \dots \ , N$ we will define a basis function which are non-zero and linear on the elements E_{j-1} and

E_j and zero everywhere else. These basis functions can be written as

$$\phi_j(x) = \begin{cases} \frac{x-x_{j-1}}{h} & x_{j-1} \leqslant x \leqslant x_j \\ \frac{x_{j+1}-x}{h} & x_j \leqslant x \leqslant x_{j+1} \,, \quad \text{for} \quad j = 2, \ \dots \ , N. \\ 0 & \text{otherwise} \end{cases} \tag{1.3.2}$$

For $j = 1,\ N+1$ we define basis functions that lie on the boundary of the interval, such that they are non-zero on the elements E_1 and E_N respectively and zero everywhere else. These functions can be written as

$$\begin{aligned} \phi_1(x) &= \begin{cases} \frac{x_2-x}{h} & x_1 \leqslant x \leqslant x_2 \\ 0 & \text{otherwise} \end{cases}, \\ \phi_{N+1}(x) &= \begin{cases} \frac{x-x_N}{h} & x_N \leqslant x \leqslant x_{N+1} \\ 0 & \text{otherwise} \end{cases}. \end{aligned} \tag{1.3.3}$$

These basis functions are illustrated in figure 1.2.

An important property of these basis functions is that $\phi_i(x)$ is precisely one at node x_i and zero on every other node, therefore

$$\phi_i(x_j) = \begin{cases} 1 & \text{when} \quad i = j \\ 0 & \text{when} \quad i \neq j \end{cases}. \tag{1.3.4}$$

This results in the property $v(x_i) = v_i$ for $i = 1, \ \dots \ , N+1$.

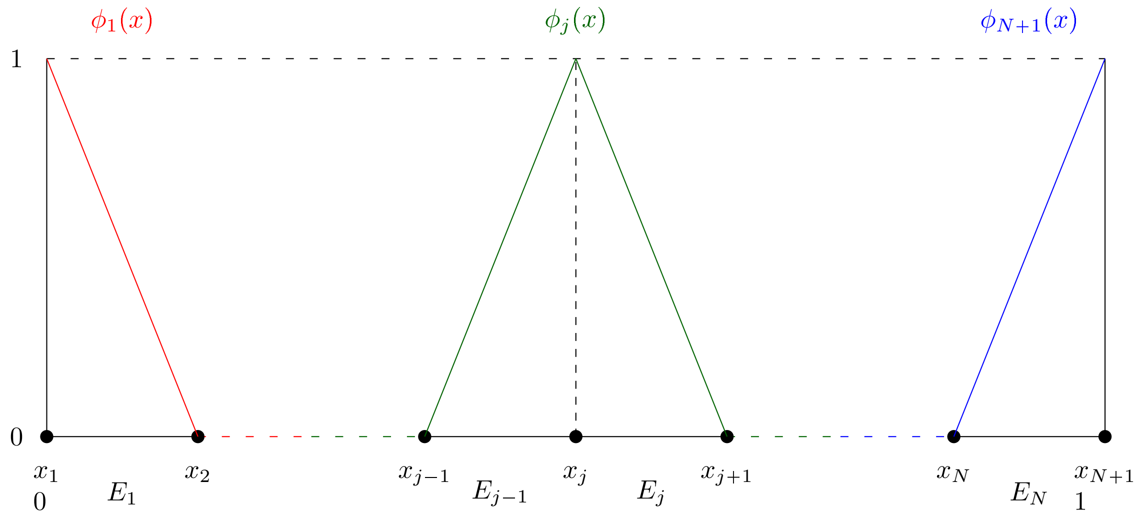


Figure 1.2: The piecewise linear basis functions ϕ_1 , ϕ_j and ϕ_{N+1} .

1.4 Finite element solution

Using the just defined basis functions we can define a *finite element solution* and *finite element test functions* as follows

$$U(x,t) := \sum_{i=1}^{N+1} U_i(t)\phi_i(x), \tag{1.4.1}$$

and

$$V(x) := \sum_{i=1}^{N+1} V_i\phi_i(x). \tag{1.4.2}$$

Instead of searching for a solution in H^1 we will search in S^1. A finite element formulation can then be formulated as follows

Definition 1.4.1 (Finite element formulation)**.** Find $U(x,t)$ such that for all $t \in \mathbb{R}$ it holds that $U(\cdot,t) \in S^1$ and for all $V \in S^1$

$$a(U(x,t),V(x)) = 0, \tag{1.4.3}$$

and the following initial boundary conditions hold

$$\begin{aligned} U(0,t) = U(1,t) &= 0 \quad \text{for all} \quad t \in \mathbb{R}, \\ U(x,0) &= U_{0,0}(x) \quad \text{for all} \quad x \in [0,1] \quad \text{and} \\ \frac{\partial U}{\partial t}(x,0) &= U_{0,1}(x) \quad \text{for all} \quad x \in [0,1]. \end{aligned} \tag{1.4.4}$$

Note that because of the form U takes, it does not have to hold that $U_0(x)$ equals $u_0(x)$ and initially we will choose $U_0(x)$ and $U_1(x)$ such that for each node x_i it holds that $U_i(0) = U_{0,0}(x_i) = u_0(x_i)$ and $\frac{\partial U_i}{\partial t}(0) = U_{0,1}(x_i) = u_1(x_i)$. In chapter 3 we will use a more complex initial value to simplify the numerical analysis.

For the finite element solution it is enough to test it only against the basis functions. This follows from the linearity of $a(u,v)$ in the second variable and therefore it holds that

$$\begin{aligned} 0 = a(U(x,t),V(x)) &= a(U(x,t), \sum_{i=1}^{N+1} V_i\phi_i(x)) \\ &= \sum_{i=1}^{N+1} V_i a(U(x,t),\phi_i(x)). \end{aligned} \tag{1.4.5}$$

Thus, if it holds for all of the basis functions ϕ_i, it also holds for all functions $V(x)$ in S^1.
For each basis function $\phi_i(x)$ that is not on the edge with $i = 2, \ldots, N$ we can write

$$\begin{aligned} 0 &= a(U(x,t),\phi_j(x)) \\ &= \int_0^1 \left(\sum_{i=1}^{N+1} \frac{\partial^2}{\partial t^2}U_i(t)\phi_i(x)\phi_j(x)\right) dx + c^2 \int_0^1 \left(\sum_{i=1}^{N+1} U_i(t)\frac{\partial}{\partial x}\phi_i(x)\frac{\partial}{\partial x}\phi_j(x)\right) dx \\ &= \sum_{i=1}^{N+1} \frac{\partial^2}{\partial t^2}U_i(t) \int_0^1 \phi_i(x)\phi_j(x)dx + c^2 \sum_{i=1}^{N+1} U_i(t) \int_0^1 \frac{\partial}{\partial x}\phi_i(x)\frac{\partial}{\partial x}\phi_j(x)dx \\ &= \sum_{i=1}^{N+1} \ddot{U}_i(t)T_{i,j} + c^2 \sum_{i=1}^{N+1} U_i(t)S_{i,j}, \\ &\text{with} \quad T_{i,j} := \int_0^1 \phi_i(x)\phi_j(x)dx \quad \text{and} \quad S_{i,j} := \int_0^1 \frac{\partial}{\partial x}\phi_i(x)\frac{\partial}{\partial x}\phi_j(x)dx. \end{aligned} \tag{1.4.6}$$

This results in $N-1$ linear ordinary differential equations. For the two basis functions left out we can use the following additional equations

$$\ddot{U}_1(t) = 0 \quad \text{and} \quad \ddot{U}_{N+1}(t) = 0. \tag{1.4.7}$$

to make sure the boundary conditions hold.
By defining the vector $U(t) := (U_1(t), \ldots, U_{N+1}(t))$ we can use these $N+1$ linear differential equations to write the following ordinary differential equation

$$T\ddot{U}(t) + c^2 SU(t) = 0, \tag{1.4.8}$$

with the coefficients of T and S being those derived in 1.4.6 except for the first and last row whose elements are derived from 1.4.7.
The values of $T_{i,j}$ and $S_{i,j}$ can easily be calculated by splitting their integrals over all the elements like

$$\int_0^1 f(x)dx = \sum_{i=1}^{N} \int_{x_i}^{x_{i+1}} f(x)dx. \tag{1.4.9}$$

As the basis functions only have a maximum of two elements on which they are non-zero, the values of $T_{i,j}$ and $S_{i,j}$ can be calculated by just integrating over the few elements on which both basis functions are non-zero. This allows us to calculate the following coefficients

$$\begin{aligned}
T_{i,i} &= \int_{x_{i-1}}^{x_i} \phi_i(x)^2 dx + \int_{x_i}^{x_{i+1}} \phi_i(x)^2 dx = \frac{2}{3}h, \\
T_{i,i-1} &= \int_{x_{i-1}}^{x_i} \phi_i(x)\phi_{i-1}(x) dx = \frac{1}{6}h, \\
S_{i,i} &= \int_{x_{i-1}}^{x_i} \left(\frac{\partial \phi_i}{\partial x}(x)\right)^2 dx + \int_{x_i}^{x_{i+1}} \left(\frac{\partial \phi_i}{\partial x}(x)\right)^2 dx = \frac{2}{h} \quad \text{and} \\
S_{i,i-1} &= \int_{x_{i-1}}^{x_i} \frac{\partial \phi_i}{\partial x}(x)\frac{\partial \phi_{i-1}}{\partial x}(x) dx = -\frac{1}{h}.
\end{aligned} \tag{1.4.10}$$

For all other values the basis functions do not overlap on non-zero elements and thus their product is zero. A more general way of calculating these coefficients is given in chapter 2.

The result is a second order linear differential equation with constant coefficients 1.4.8. The solution to this equation can than be used in the finite element solution to get a numerical solution to the one-dimensional wave equation stated in the beginning of the chapter.

1.5 Time discretisation

In this section we derive two numerical methods to solve differential equation 1.4.8.

For the first method we will use a second order approximation for the second derivative given by

$$\ddot{U}(t) \approx \frac{U(t+dt) - 2U(t) + U(t-dt)}{dt^2}, \tag{1.5.1}$$

and substituting this in the differential equation results in

$$T\left(\frac{U(t+dt) - 2U(t) + U(t-dt)}{dt^2}\right) + c^2 SU(t) = 0. \tag{1.5.2}$$

After some algebraic manipulation the following equation can be found

$$U(t+dt) = -c^2 dt^2 T^{-1} SU(t) + 2U(t) - U(t-dt). \tag{1.5.3}$$

As the initial values are $U(0)$ and $\dot{U}(0)$ we can use a first order approximation to do the first step

$$U(dt) = U(0) + dt\dot{U}(0), \tag{1.5.4}$$

and use equation 1.5.3 for further iterations.
Notice that 1.5.3 requires the inverse of the sparse (or more specific tridiagonal) matrix T and its inverse is generally not sparse. For a more efficient computation we can use

$$T\left(\frac{U(t+dt) - 2U(t) + U(t-dt)}{dt^2}\right) = -c^2 SU(t). \tag{1.5.5}$$

By first calculating the right-hand side we get a linear system which we can solve for $\frac{U(t+dt)-2U(t)+U(t-dt)}{dt^2}$ from which we can calculate $U(t+dt)$. Using this method the inverse of a sparse matrix does not have to be calculated, but instead a linear system has to be solved each iteration, which is in general more accurate than a matrix inversion.

A second method for solving equation 1.4.8 is to rewrite it as a first order linear differential equation. First define $W(t) := \dot{U}(t)$, then because $\dot{W}(t) = \ddot{U}(t) = -c^2T^{-1}SU(t)$ the differential equation is equivalent to the initial value problem

$$\begin{cases} \dot{W}(t) & = -c^2T^{-1}SU(t), \\ \dot{U}(t) & = W(t) \end{cases} \quad \text{with initial value} \quad \begin{cases} U(0) & = U_{0,0} \quad \text{and} \\ W(0) & = \dot{U}(0) = U_{0,1}. \end{cases} \tag{1.5.6}$$

Then by using a first order approximation

$$\dot{U}(t) \approx \frac{U(t+dt) - U(t)}{dt}, \tag{1.5.7}$$

we get the equations

$$\begin{cases} \frac{W(t+dt)-W(t)}{dt} & = -c^2T^{-1}SU(t), \\ \frac{U(t+dt)-U(t)}{dt} & = W(t), \end{cases} \tag{1.5.8}$$

which can be rewritten to

$$\begin{cases} T\left(\frac{\dot{U}(t+dt)-\dot{U}(t)}{dt}\right) & = -c^2SU(t), \\ U(t+dt) & = U(t) + dt\dot{U}(t). \end{cases} \tag{1.5.9}$$

For the first equation begin by solving the linear system then solve for $\dot{U}(t+dt)$ and the second equation can be calculated directly.

One advantage of the second method is that we do not have to use the initial value to the first step in time and a second advantage will be clear in chapter 3 where we use the $U(t)$ and $\dot{U}(t)$ to calculate the energy of the solution.

1.6 Example

We will now use the second numerical method derived in the previous section to simulate a simple wave. As an example let $c = 1$ so the one-dimensional wave equation becomes

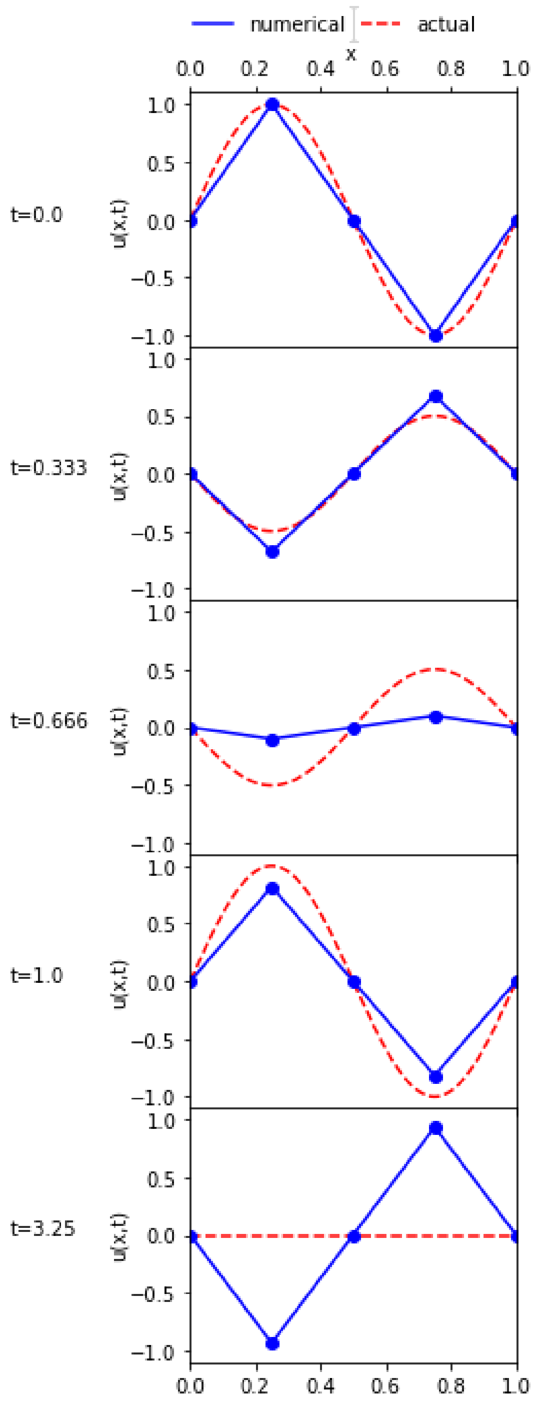
$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0. \tag{1.6.1}$$

For this equation one of the solutions is given by

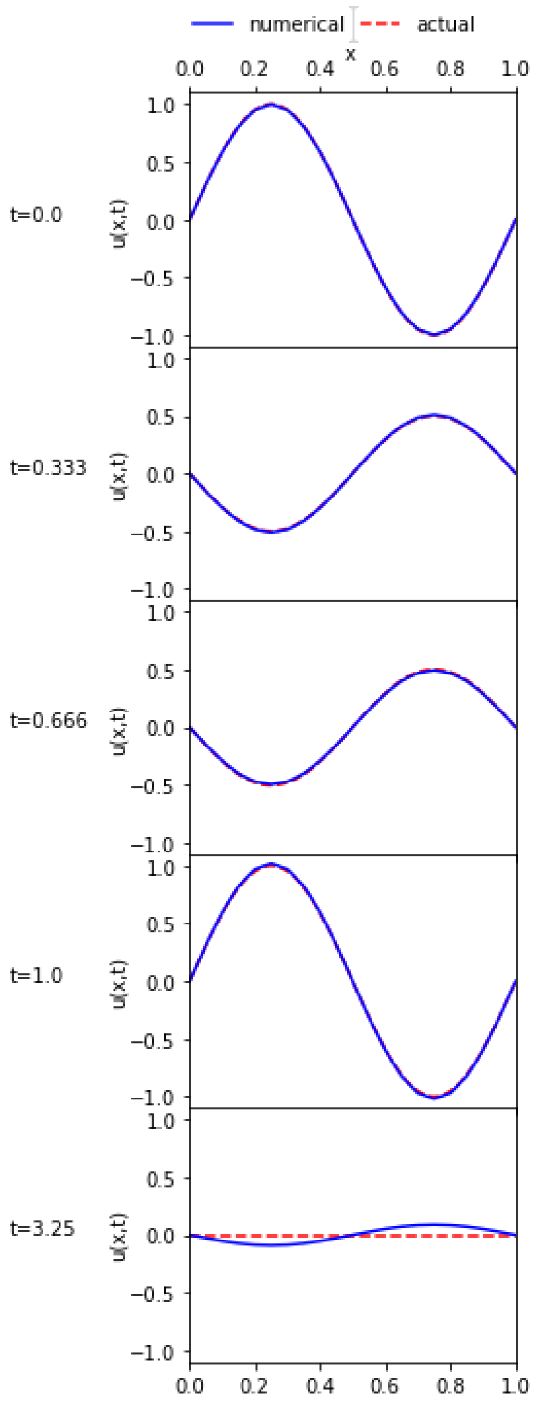
$$u(x,t) = \cos(2\pi t)\sin(2\pi x), \tag{1.6.2}$$

which is periodic in time with period 1 and is called a *standing wave.*

The figures 1.3a and 1.3b show the simulation with time step 0.001 at different points in time for four and respectively twenty elements. It can be seen in 1.3a that the numerical solution has a faster "periodic" movement than the actual solution. The word periodic is put between quotation marks as in 1.4 it can be seen that the amplitude of the numerical solution increases over time.
In figure 1.3b it can be seen that the numerical method more accurately represents the actual solution, but that it also moves faster than the actual solution. And just like with the four element example, the amplitude increases over time as can be seen in 1.5
The behaviours just observed like the periodic movement, increase in amplitude and convergence will be further analysed in chapter 3.



(a) A standing wave simulated with 4 elements.



(b) A standing wave simulated with 20 elements.

Figure 1.3: The standing wave $u(x,t) = \cos(2\pi t)\sin(2\pi x)$ simulated with a time step of 0.001 with 4 (left) and 20 (right) elements.

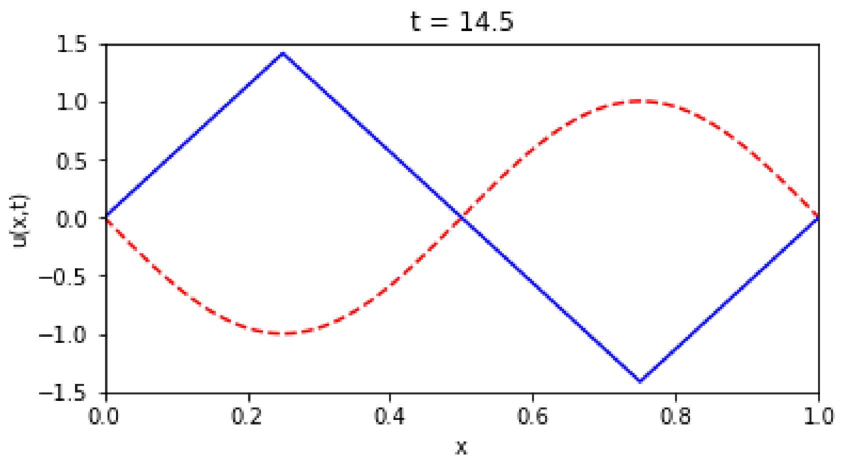


Figure 1.4: A standing wave simulated using 4 elements, showing that the amplitude of the numerical solutions increased over time.

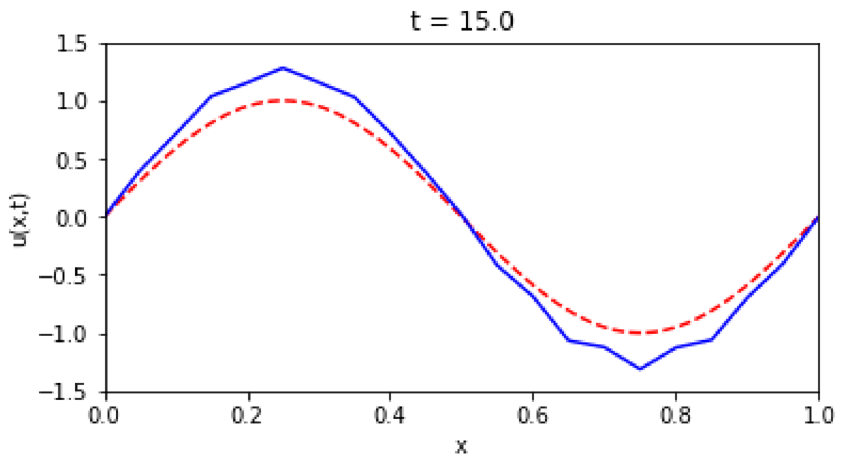


Figure 1.5: A standing wave simulated using 20 elements, showing that the amplitude of the numerical solutions increased over time.

2 Irregular elements and higher-order approximation

The first 3 sections of this chapter are based on chapter 4 of [1].

In this chapter we will generalize the finite element method by first allowing irregular elements and then higher order polynomial approximation.

We will divide an interval $[a, b]$ with $a < b$ into N elements seperated by nodes. The nodes x_i with $i = 1, \dots, N+1$ are chosen such that $a = x_1 < x_2 < \dots < x_N < x_{N+1} = b$ and the elements can than defined the same as 1.2.2, namely

$$E_i := [x_i, x_{i+1}] \quad \text{for} \quad i = 1, \dots, N, \tag{2.0.1}$$

and the size of element E_i is given by h_i or more specific

$$h_i := x_{i+1} - x_i. \tag{2.0.2}$$

This is illustrated in figure 2.1;

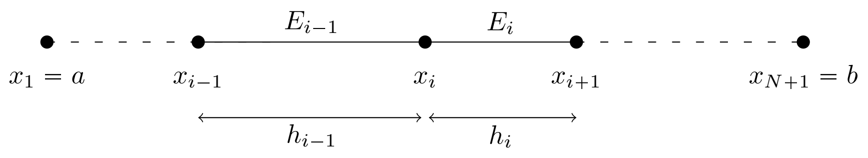


Figure 2.1: An irregular partitioning of the interval $[a, b]$ in elements and nodes.

2.1 Canonical element

Instead of defining each basis function separately we will define a *canonical element* denoted E_c which is defined as the unit interval $[0, 1]$. Then, for each element E_i there exists a transformation $\tau_i(x)$ such that $\tau_i(E_c) = E_i$ and the inverse of τ_i exists as illustrated in 2.2. The transformation τ_i is given by

$$\tau_i(x) = x(x_{i+1} - x_i) + x_i = xh_i + x_i. \tag{2.1.1}$$

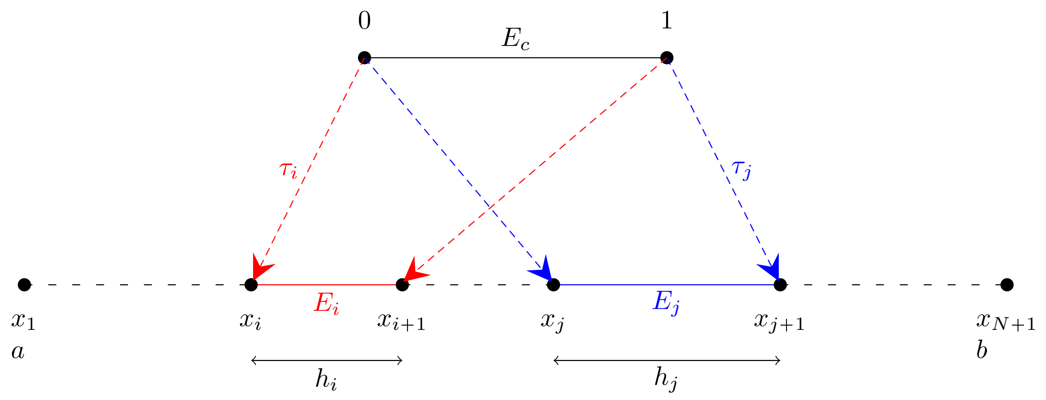


Figure 2.2: The transformations τ_i and τ_j from the canonical element E_c to the irregular elements E_i and E_j .

On the canonical element we can define local basis functions

$$\begin{aligned} \phi_{l,1}(x) &= 1 - x \quad \text{and} \\ \phi_{l,2}(x) &= x, \end{aligned} \tag{2.1.2}$$

which are illustrated in figure 1.2.

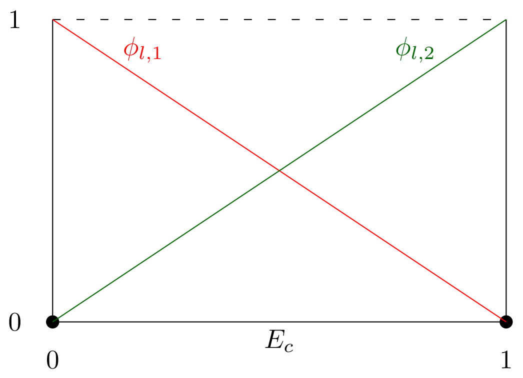


Figure 2.3: The local linear basis functions $\phi_{l,1}$ and $\phi_{l,2}$.

The global basis functions are then definable as

$$\phi_i(x) = \begin{cases} \phi_{l,2}(\tau_{i-1}^{-1}(x)) & x_{i-1} \leqslant x \leqslant x_i, \\ \phi_{l,1}(\tau_i^{-1}(x)) & x_i \leqslant x \leqslant x_{i+1}, \\ 0 & \text{otherwise} \end{cases} \quad \text{for} \quad i = 1,\ \dots\ , N+1, \tag{2.1.3}$$

with exceptions for basis functions corresponding to nodes on the boundary, like in chapter 1.

Using this definition the time and space matrix coefficients from 1.4.10 can be calculated using the following change of variables for a functions f over an element E_k to the canonical element E_c

$$\begin{aligned} &\int_{x_k}^{x_{k+1}} f(x)dx \\ =&\int_{\tau_k(0)}^{\tau_k(1)} f(x)dx \\ =&\int_0^1 f(\tau_k(x))\tau_k'(x)dx \\ =&h_k \int_0^1 f(\tau_k(x))dx. \end{aligned} \tag{2.1.4}$$

If the function $f(x)$ is of the the form $g(\tau_k^{-1}(x))$ on element E_k(for example $f(x) = \phi_{l,2}(\tau_k^{-1}(x))\phi_{l,1}(\tau_k^{-1}(x))$ when calculating the coefficients $T_{i,j}$) the above integral simplifies to

$$\int_{x_k}^{x_{k+1}} g(\tau_k^{-1}(x))dx = h_k \int_0^1 g(x)dx, \tag{2.1.5}$$

which can be calculated easily when $g(x)$ is a polynomial like is the case in this chapter. For example take the regular grid ($h_i = h$) of previous chapter, the coefficient $T_{i,i}$ can be calculated like

$$\begin{aligned} T_{i,i} &= \int_{x_{i-1}}^{x_i} \phi_i(x)^2 dx + \int_{x_i}^{x_{i+1}} \phi_i(x)^2 dx \\ &= h \int_0^1 \phi_{l,2}(x)^2 dx + h \int_0^1 \phi_{l,1}(x)^2 dx \\ &= h \int_0^1 x^2 dx + h \int_0^1 (1-x)^2 dx = h\frac{2}{3}. \end{aligned} \tag{2.1.6}$$

or when using irregular elements

$$T_{i,i} = \frac{1}{3}h_{i-1} + \frac{1}{3}h_i. \tag{2.1.7}$$

2.2 Higher-order polynomials

To increase the order of the polynomials we need more nodes. For dividing the interval $[a, b]$ in N elements and to approximate the solution on each element using a p^{th} degree polynomial we need a total of $Np + 1$ nodes. The nodes x_i with $i = 1, \dots, Np + 1$ such that $a = x_1 < \dots < x_{N+1} = b$ can be split into two groups: exterior and interior nodes.
The *exterior nodes* are x_{ip+1} with $i = 0, \dots, N$ divide the interval into the elements such that

$$\begin{aligned} E_i &:= [x_{ip+1}, x_{(i+1)p+1}] \quad \text{and} \\ h_i &= x_{(i+1)p+1} - x_{ip+1} \end{aligned} \quad \text{for} \quad i = 1, \dots, N. \tag{2.2.1}$$

For each element E_j there exist $p - 1$ *interior nodes* x_{jp+1+i} with $i = 1, \dots, p - 1$. For our purposes we will place these interior nodes uniformly on the elements such that

$$x_{jp+1+i} := x_{jp+1} + i\frac{h_j}{p} \quad \text{for} \quad j = 1, \dots, N \quad \text{and} \quad i = 1, \dots, p - 1. \tag{2.2.2}$$

The interior nodes can also be expressed on the canonical element The nodes and elements are illustrated in figure 2.4.

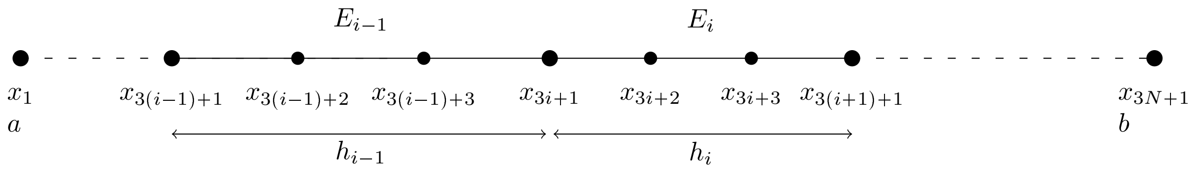


Figure 2.4: An irregular grid for cubic approximation (p = 3).

We now want to define a basis function for each of the nodes. First we will define the local basis functions on the canonical element. As there are $p + 1$ exterior and interior nodes inside each element we need to define $p + 1$ local basis functions.
Let the local nodes $x_{l,i}$ be defined as

$$x_{l,i} := \frac{i - 1}{p} \quad \text{for} \quad i = 1, \dots, p + 1. \tag{2.2.3}$$

We want the corresponding local basis functions $\phi_{l,i}$ to be 1 on the local node $x_{l,i}$ and zero on all other nodes. A suitable set of basis functions are *Lagrange polynomials* which result in the local basis functions

$$\phi_{l,i}(x) := \prod_{\substack{1 \leqslant j \leqslant p+1 \\ j \neq i}} \frac{x - x_{l,j}}{x_{l,i} - x_{l,j}}. \tag{2.2.4}$$

For example, for quadratic approximation ($p = 2$) there are three local basis functions

$$\begin{aligned} \phi_{l,1} &= \frac{(x - \frac{1}{2})(x - 1)}{-\frac{1}{2} \cdot -1} = 2x^2 - 3x + 1, \\ \phi_{l,2} &= \frac{x(x - 1)}{\frac{1}{2} \cdot -\frac{1}{2}} = -4x^2 + 4x \quad \text{and} \\ \phi_{l,3} &= \frac{x(x - \frac{1}{2})}{1 \cdot \frac{1}{2}} = 2x^2 - x. \end{aligned} \tag{2.2.5}$$

The quadratic and cubic cases are illustrated in 2.5 and 2.6.

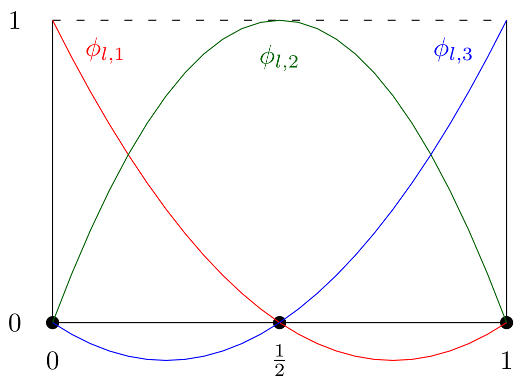


Figure 2.5: The quadratic local basis functions.

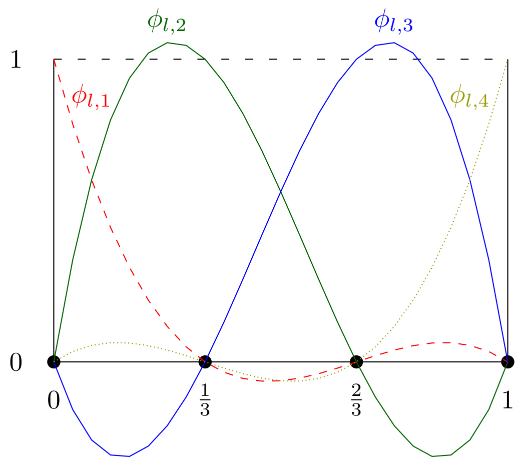


Figure 2.6: The cubic local basis functions.

For exterior nodes we can now define the corresponding global basis functions as

$$\phi_{ip+1}(x) = \begin{cases} \phi_{l,p+1}(\tau_{i-1}^{-1}(x)) & x_{i-1} \leqslant x \leqslant x_i, \\ \phi_{l,1}(\tau_i^{-1}(x)) & x_i \leqslant x \leqslant x_{i+1}, \\ 0 & \text{otherwise} \end{cases} \quad i = 2, \dots, N, \tag{2.2.6}$$

with the same boundary exceptions as earlier.
For interior nodes the global functions can be defined as

$$\phi_{jp+1+i}(x) = \begin{cases} \phi_{l,i}(\tau_j^{-1}(x)) & x_j \leqslant x \leqslant x_{j+1}, \\ 0 & \text{otherwise} \end{cases} \quad j = 2, \dots, N \quad \text{and} \quad i = 2, \dots, p. \tag{2.2.7}$$

Notice that the global basis functions only take non-zero value on the elements that the corresponding node touches and again that for all $i = 1, \dots, Np + 1$ it holds that $\phi_i(x_i) = 1$.

2.3 Matrix construction

A simple way of calculating the coefficients $T_{i,j} = \int_a^b \phi_i(x)\phi_j(x)dx$ and $S_{i,j} = \int_a^b \frac{\partial}{\partial x}\phi_i(x)\frac{\partial}{\partial x}\phi_j(x)dx$ is by rewriting the integrals $\int_a^b$ as a summation of the integrals over all the elements like $\sum_{i=1}^N \int_{E_i}$ and then use the definition of the global basis functions and a change of variables to the canonical elements to simplify the calculation of each integral. We can reduce the amount of integrals by not evaluating those for which we know that the product of the functions is zero.

We can also calculate the coefficient the other way around. We know that the product of two basis functions (and the product of their first derivatives to x) is non-zero on an element if and only if both corresponding nodes touch the element (either as an external or as an internal node), therefore there are $(p+1)^2$ pairs of basis functions on each element whose product is non-zero on that element. These pairs are more specifically pairs of transformed local basis functions on that element. Using this we can start with all values $T_{i,j}$ and $S_{i,j}$ zero and then iterate over all the element and add the integral of the non-zero pairs of that element to the corresponding coefficients. Both methods (as example for $T_{i,j}$) are summarized bellow:

Per coefficient calculation

```
All T[i, j] are 0;
For each coefficient T[i,j]:
 If basis functions i and j have non-zero overlap:
  For each element E[n]:
   T[i, j] += Integrate basis functions i and j over element E[n];
```

Per element calculation

```
Let p be the degree of polynomial;
All T[i, j] are 0;
For each element E[n]:
 For each local basis function i:
  For each local basis function j:
   T[(p*n)+i, (p*m)+j] += Integrate transformed local basis functions i and j over
                                                                           element E[n];
```

2.4 Examples

For the following examples we will choose the constant $c = 1$, the domain will be the interval $[0, 2]$ and the time step will again be 0.001. The interval $[0, 2]$ will be divided into elements of three different sizes, where the smaller elements are on the left size and the largest element is on the right. This division is illustrated by the green dots in 2.7 and 2.8.

For the first example we will use the same solution as in chapter 1 1.6.2 which is given by

$$u(x, t) = \cos(2\pi t)\sin(2\pi x). \tag{2.4.1}$$

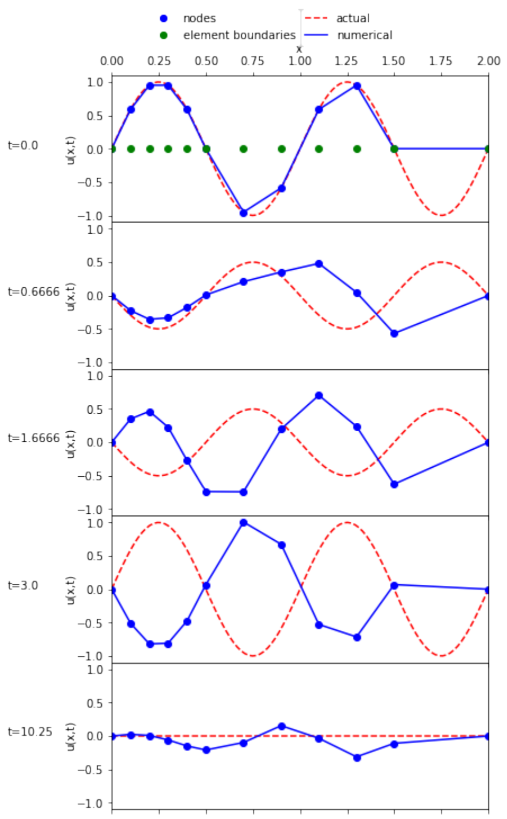
In 2.7a we can see that for first degree approximation we can see that the finer elements can better approximate larger wave, while this is more difficult for the coarser elements. It is even possible that a part of the wave is completely removed when the nodes of an element lie on the zeros of the initial value. As time increases the wave becomes inaccurate quickly with respect to the actual solution.

If we increase the degree of approximation to four we get 2.7b. The increase in degree implies an increase in the amount of nodes and smoothness and thus it initial approximation can better approximate the actual initial value. It is important to notice when comparing 2.7a and 2.7b that even though the solution starts better approximating the actual solution when the degree of polynomials is higher the finer approximation gets unstable faster. This might be explained by the increment of calculations with relatively small values, which can easily introduce rounding errors.

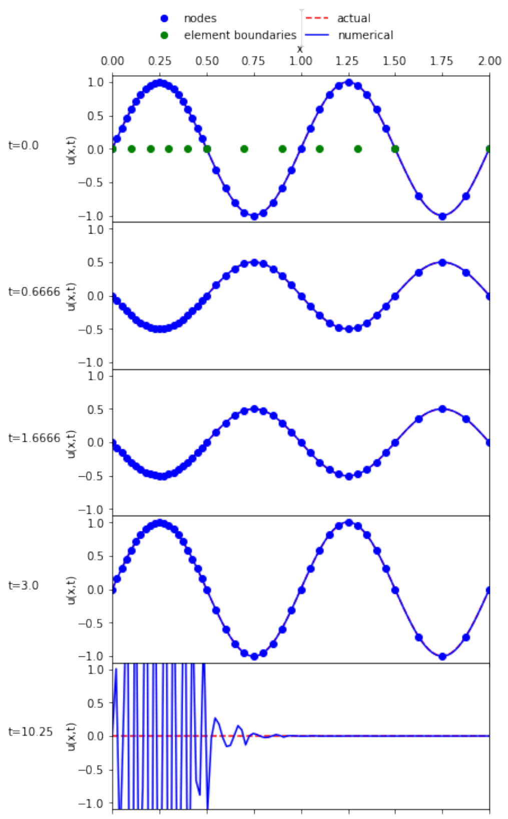
As a second example we will use the sum of two standing waves or more specifically

$$u(x, t) = \cos(2\pi t)\sin(2\pi x) + \cos(4\pi t)\sin(4\pi x). \tag{2.4.2}$$

This example is plotted for different degrees of approximation in 2.8. In this example we see better that increasing the degree of the polynomials improves the numerical solution, but again, making the approximations too fine can introduce instability as can be seen in 2.8d.

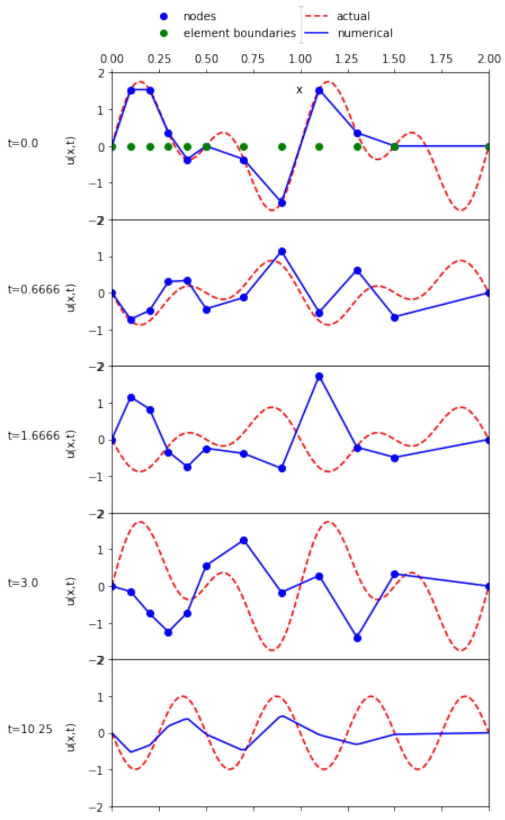


(a) A first degree polynomial approximation.

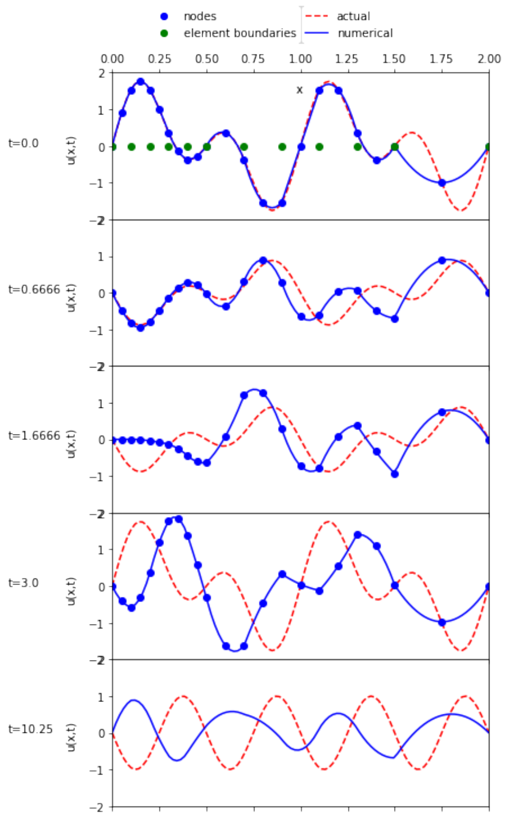


(b) A fourth degree polynomial approximation.

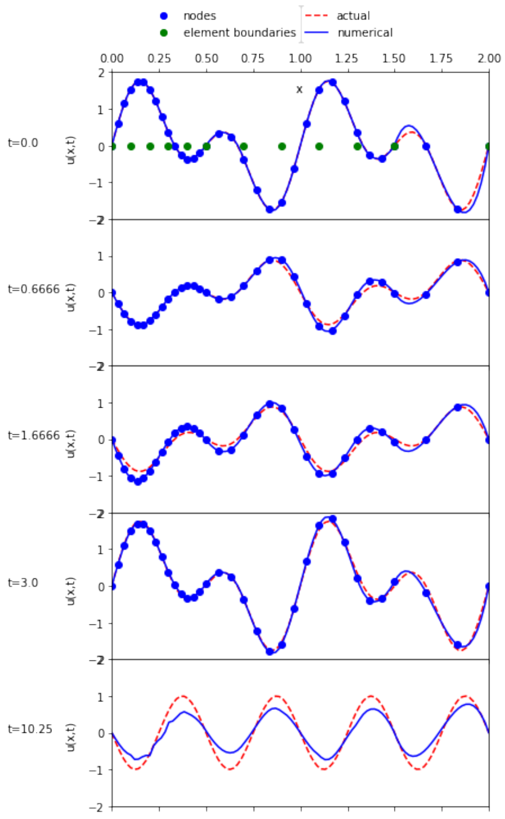
Figure 2.7: The wave $u(x,t) = \cos(2\pi t)\sin(2\pi x)$ simulated using an irregular grid with first (left) and fourth (right) degree polynomials.



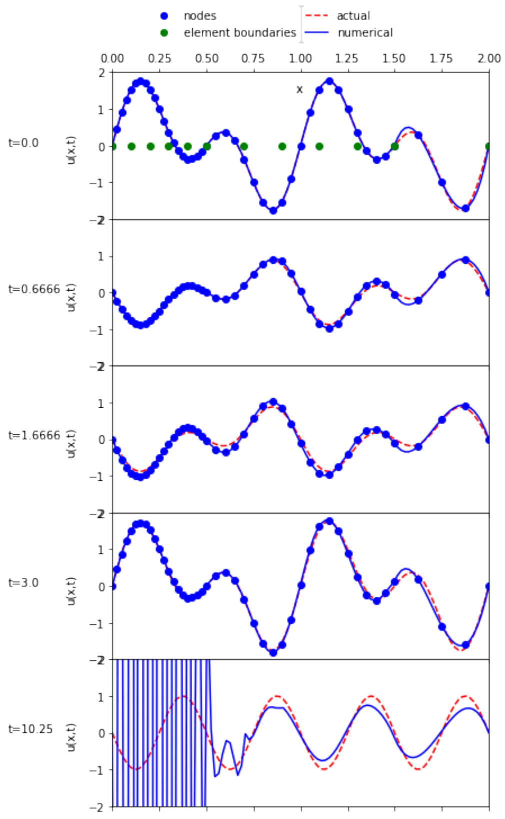
(a) A first degree polynomial approximation.



(b) A second degree polynomial approximation.



(c) A third degree polynomial approximation.



(d) A fourth degree polynomial approximation.

Figure 2.8: The wave $u(x,t) = \cos(2\pi t)\sin(2\pi x) + \cos(4\pi t)\sin(4\pi x)$ simulated using an irregular grid with first, second, third and fourth degree polynomials.

3 Numerical analysis and experiments

In this chapter we will analyse some phenomena we have seen in the examples of the previous chapters. First we will look at the convergence of the numerical solution, then analyse the conservation of energy of solutions and finally we will take a look at the dispersion of waves.

3.1 Convergence

Let u be the solution of the one-dimensional wave equation 1.0.1, U be the solution of the finite element formulation 1.4.1 and 1.4.8 (after spatial discretization) and let $\hat{U}$ be the final numerical solution (after temporal discretization). We will work on the unit interval $[0,1]$ with arbitrary elements of size h_i and approximating the solution using p^{th} degree polynomials

To analyse the convergence we want to find an upper bound for the total error $u-\hat{U}$ with respect to a certain norm. We can split this problem in two parts by using the triangle inequality resulting in

$$\left|\left|u-\hat{U}\right|\right| \leqslant ||u-U|| + \left|\left|U-\hat{U}\right|\right|, \tag{3.1.1}$$

therefore bounding the total error by the sum of the spatial and temporal discretization error.

First we will look at the spatial discretization error. To simplify the analysis we will require the following extra constraints for all test functions V

$$\begin{aligned} \int_0^1 \frac{\partial V}{\partial x}\frac{\partial U_{0,0}}{\partial x}dx &= \int_0^1 \frac{\partial V}{\partial x}\frac{\partial u_0}{\partial x}dx \quad \text{and} \\ \int_0^1 \frac{\partial V}{\partial x}\frac{\partial^2 U_{0,1}}{\partial x \partial t}dx &= \int_0^1 \frac{\partial V}{\partial x}\frac{\partial u_1}{\partial x}dx. \end{aligned} \tag{3.1.2}$$

Just like for the finite element formulation it is enough to test the constraints only for the basis functions, which implies they also hold for all test functions. These constraints are to make sure the initial value of the discretization is close enough to the true initial value. For the finite element formulation discussed in previous chapters the above constraints become two linear systems which can be solved to get initial values for the finite element solution.

Before defining the norm we will use to find an upper bound of the error, we will first define the spatial part of the norm as follows

$$||f(\cdot)||_{H_1} := \sqrt{\int_0^1 (f(x))^2 + \left(\frac{\partial f}{\partial x}(x)\right)^2 dx}, \tag{3.1.3}$$

which can be seen as a form of energy. Then, we can combine this norm with a supremum norm to get the following norm for a set $R \subseteq \mathbb{R}_{\geqslant 0}$:

$$||f||_{L^\infty(R,H_1)} := \sup_{t\in R} ||f(\cdot,t)||_{H_1}. \tag{3.1.4}$$

A useful property we will use is that if $R' \subseteq R$, then $||f||_{L^\infty(R',H_1)} \leqslant ||f||_{L^\infty(R,H_1)}$, because we can never get a bigger value in a supremum when removing elements.

Because we discretize time we only want to use the norm on the times we actually calculate and define the set R of all steps in time starting at 0, with steps of time dt up until a constant T. We will use the norm above with this set R to find an upper bound for the total error.

For the spatial discretization error we will use the following theorem.

Theorem 3.1.1 (Theorem 2.1 from [2]). *Let u be the solution of the one-dimensional wave equation, U be the solution of the finite element formulation such that the constraints 3.1.2 hold, h be the size of the biggest element and $T > 0$ an arbitrary constant, then there exists a constant $C > 0$ such that*

$$||u - U||_{L^\infty([0,T],H^1)} < Ch^p. \tag{3.1.5}$$

Because $R \subset [0, T]$, it follows directly that

$$||u - U||_{L^\infty(R,H^1)} \leqslant ||u - U||_{L^\infty([0,T],H^1)} < Ch^p, \tag{3.1.6}$$

with the constant C from the theorem above. This implies that the spatial discretization up until time T converges to zero when the element sizes go to zero and it converges faster with higher order polynomials. This can be written as $||u - U||_{L^\infty(R,H^1)} = O(h^p)$.

Now we will look at the temporal discretization error. In chapter one we derived a method for solving the equation 1.4.8 $T\ddot{\bar{U}}(t) + c^2 S\bar{U} = 0$ using the equivalent formulation

$$\begin{cases} \dot{\bar{\dot{U}}}(t) &= -c^2 T^{-1} S\bar{U}(t), \\ \dot{\bar{U}}(t) &= \bar{\dot{U}}(t) \end{cases}, \tag{3.1.7}$$

where $\bar{U}(t)$ is a vector of functions such that $\hat{U}(x,t) = \sum_{i=1}^n \bar{U}_i(t)\phi_i(x)$. We will focus on this second method instead of the first method we derived, because this one will also be important in the next section.

The second method 1.5.9 is given by

$$\begin{cases} T\left(\frac{\bar{\dot{U}}(t+dt) - \bar{\dot{U}}(t)}{dt}\right) &= -c^2 S\bar{U}(t), \\ \bar{U}(t+dt) &= \bar{U}(t) + dt\bar{\dot{U}}(t), \end{cases} \tag{3.1.8}$$

or can be written as

$$\begin{cases} \bar{\dot{U}}(t+dt) &= -c^2 dt T^{-1} S\bar{U}(t) + \bar{\dot{U}}(t), \\ \bar{U}(t+dt) &= \bar{U}(t) + dt\bar{\dot{U}}(t). \end{cases} \tag{3.1.9}$$

This method was derived by using a truncated multidimensional version of the following forward difference equation[3]

$$\frac{df}{dt}(t) = \frac{f(t+dt) - f(t)}{dt} - \frac{dt}{2}\frac{d^2 f}{dt^2}(\xi(t)), \quad \xi(t) \in [t,\ t+dt]. \tag{3.1.10}$$

If f is a vector of functions ($f(t) := (f_1(t),\ \dots\ , f_n(t))$) and we apply the above formula to each function of the vector, we get the following vector valued formula

$$\begin{aligned} \frac{df}{dt}(t) &= \frac{f(t+dt) - f(t)}{dt} - \frac{dt}{2}F(t), \quad \text{with} \\ F(t) &= \left\{\frac{d^2 f_i}{dt^2}(\xi_i(t))\right\}_{i=1,\dots,n} \quad \text{and} \quad \xi_i(t) \in [t,\ t+dt]. \end{aligned} \tag{3.1.11}$$

Placing this formula in 3.1.7 for $\dot{\bar{U}}$ we get

$$\begin{aligned} &\begin{cases} \frac{\dot{U}(t+dt) - \dot{U}(t)}{dt} - \frac{dt}{2}F(t) &= -c^2 T^{-1} SU(t), \\ \frac{U(t+dt) - U(t)}{dt} - \frac{dt}{2}G(t) &= \dot{U}(t), \end{cases}, \text{ or} \\ &\begin{cases} \dot{U}(t+dt) &= -c^2 dt T^{-1} SU(t) + \dot{U}(t) + \frac{dt^2}{2}F(t), \\ U(t+dt) &= dt\dot{U}(t) + U(t) + \frac{dt^2}{2}G(t), \end{cases}. \end{aligned} \tag{3.1.12}$$

This results in the following errors

$$\begin{cases}\dot{U}(t+dt)-\dot{\bar{U}}(t+dt) &= -\frac{dt^2}{2}F(t),\\ U(t+dt)-\bar{U}(t+dt) &= -\frac{dt^2}{2}G(t),\end{cases} \quad \text{or} \quad \begin{cases}\dot{U}(x,t+dt)-\dot{\hat{U}}(x,t+dt) &= -\frac{dt^2}{2}F(x,t),\\ U(x,t+dt)-\hat{U}(x,t+dt) &= -\frac{dt^2}{2}G(x,t).\end{cases} \tag{3.1.13}$$

By doing this for n steps such that $dt=\frac{T}{n}$ we get

$$\begin{cases}\dot{U}(x,t+ndt)-\dot{\hat{U}}(x,t+ndt) &= n\ dt^2 C_1 = T\ dt\ C_1,\\ U(x,t+dt)-\hat{U}(x,t+dt) &= n\ dt^2\ C_2 = T\ dt\ C_2,\end{cases} \quad \text{with } C_1 \text{ and } C_2 \text{ constants.} \tag{3.1.14}$$

This results in the global error

$$\left|\left|U-\hat{U}\right|\right|_{L^\infty(R,H_1)} = O(T\ dt). \tag{3.1.15}$$

The result is that, if we let T be a constant as has been done in Theorem 3.1.1, we get the total error

$$\left|\left|u-\hat{U}\right|\right|_{L^\infty(R,H_1)} = \mathcal{O}(h^p+dt), \tag{3.1.16}$$

which implies that, on the interval $[0,T]$, the error converges to zero if both h and dt tend to zero and the convergence speed of the spatial discretization error can be improved by increasing the degree p of polynomials.

3.2 Conservation of energy

In the previous section we have seen a few parameters that are related to the convergence of the numerical solution, namely the size h of the biggest element, the time step dt and the degree p of the polynomials In this section we will use the concept of energy conservation to see how these parameter have an impact on the convergence and stability of the numerical solution.

The concept of conservation of energy is used all over physics and states most of the time that the total amount of energy in a closed systems is constant. This conservation can also hold for the one-dimensional wave equation and can be formulated as

Theorem 3.2.1 (Theorem and prove based on theorem 5.4 in [4])**.** *For a solution u of the one-dimensional wave equation there exists a constant $C>0$, independent of time t, such that*

$$\left|\left|\frac{\partial u}{\partial t}(\cdot,t)\right|\right|^2 + c^2\left|\left|\frac{\partial u}{\partial x}(\cdot,t)\right|\right|^2 = C, \tag{3.2.1}$$

where $||\cdot||$ is the L^2-norm over the domain D.

Proof. Let $v(x)=\frac{\partial u}{\partial t}(x,t)$ be the test function in the weak formulation of the one-dimensional wave equation. This gives

$$\begin{aligned}0 &= \int_D \frac{\partial^2 u(x,t)}{\partial t^2}\frac{\partial u(x,t)}{\partial t}dx + c^2\int_D \frac{\partial u(x,t)}{\partial x}\frac{\partial}{\partial x}\frac{\partial u(x,t)}{\partial t}\\ &= \int_D \frac{1}{2}\frac{\partial}{\partial t}\left(\frac{\partial u(x,t)}{\partial t}^2\right)dx + c^2\int_D \frac{1}{2}\frac{\partial}{\partial t}\left(\frac{\partial u(x,t)}{\partial x}^2\right)dx\\ &= \frac{1}{2}\frac{\partial}{\partial t}\left(\left|\left|\frac{\partial u}{\partial t}(\cdot,t)\right|\right|^2 + c^2\left|\left|\frac{\partial u}{\partial x}(\cdot,t)\right|\right|^2\right),\end{aligned} \tag{3.2.2}$$

which implies that

$$\left|\left|\frac{\partial u}{\partial t}(\cdot,t)\right|\right|^2 + c^2\left|\left|\frac{\partial u}{\partial x}(\cdot,t)\right|\right|^2 = C \quad \text{with} \quad C\in\mathbb{R}. \tag{3.2.3}$$

□

Notice that the proof of theorem 3.2.1 uses the weak formulation, thus when using $v(x) = \dot{U}(x,t)$ in the finite element formulation 1.4.3 it follows that

$$\left|\left|\frac{\partial U}{\partial t}(\cdot,t)\right|\right|^2 + c^2\left|\left|\frac{\partial U}{\partial x}(\cdot,t)\right|\right|^2 = C \quad \text{with} \quad C \in \mathbb{R}. \tag{3.2.4}$$

This implies conservation of energy for the finite element solution of the wave equation.

From the above formulas we can define an energy function as $E(u,t) := ||\dot{u}(\cdot,t)||^2 + c^2\left|\left|\frac{\partial u}{\partial x}(\cdot,t)\right|\right|^2$, such that for a solution u of the weak or finite element formulation there exists a constant $C > 0$ such that $E(u,t) = C$ for all t.

The first error we can notice is that the initial value of the wave equation $u(x,0)$ and $\frac{\partial u}{\partial t}(x,0)$ and its discretized value for the finite element formulation $U(x,0)$ and $\frac{\partial U}{\partial t}(x,0)$ do not necessarily have the same energy, or more specific, $E(u,0)$ and $E(U,0)$ do not have to be equal. But if we make the elements smaller and use higher order polynomials for approximation, we can better approximate the original initial value with U and $\frac{\partial U}{\partial t}$ and therefore better approximate the energy.

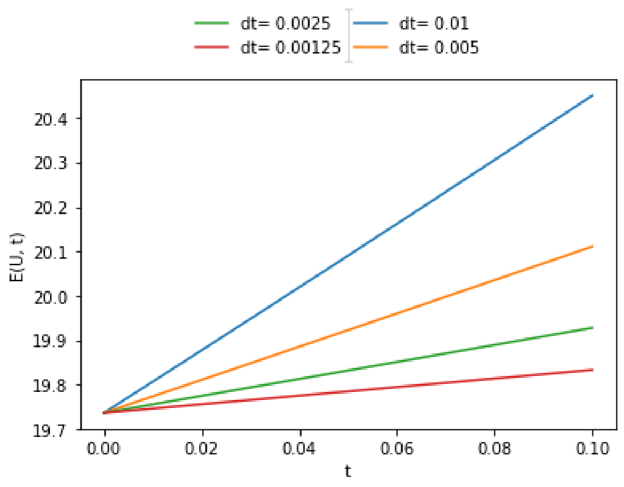
To analyse what will happen to the energy when we discretize the system of ordinary differential equations 1.4.8 we can use a more efficient way of calculating the energy. We can get this by putting the finite element solution $U(x,t) = \sum_{i=1}^n U_i(t)\phi_i(x)$ in the energy function, resulting in the following equations:

$$\begin{aligned}
E(U,t) &= \left|\left|\dot{U}(\cdot,t)\right|\right|^2 + c^2\left|\left|\frac{\partial U}{\partial x}(\cdot,t)\right|\right|^2 \\
&= \int_D\left(\sum_{i=1}^n \dot{U}_i(t)\phi_i(x)\right)^2 dx + c^2\int_D\left(\sum_{i=1}^n U_i(t)\frac{\partial}{\partial x}\phi_i(x)\right)^2 dx \\
&= \int_D \sum_{i=1}^n\sum_{j=1}^n \dot{U}_i(t)\dot{U}_j(t)\phi_i(x)\phi_j(x)dx + c^2\int_D\sum_{i=1}^n\sum_{j=1}^n U_i(t)U_j(t)\frac{\partial}{\partial x}\phi_i(x)\frac{\partial}{\partial x}\phi_j(x)dx, \text{ thus} \\
E(U,t) &= \sum_{i=1}^n\sum_{j=1}^n \dot{U}_i(t)\dot{U}_j(t)T_{i,j} + c^2\sum_{i=1}^n\sum_{j=1}^n U_i(t)U_j(t)S_{i,j}
\end{aligned} \tag{3.2.5}$$

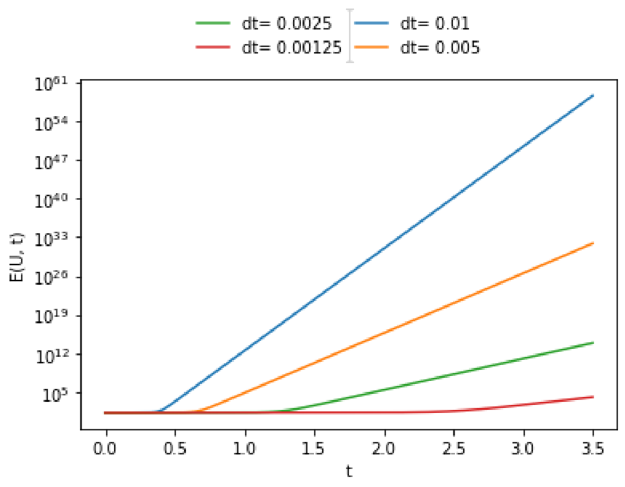
where $T_{i,j}$ and $S_{i,j}$ are defined as in 1.4.6. This is one of the main reasons why we introduced two numerical methods for solving the ODE in chapter 1. The first method was more easily derived, but does not give us concrete values for $\dot{U}_i(t)$, while the second method does give us these values. Combining that with the fact that we already had to calculate most (if not all) values $T_{i,j}$ and $S_{i,j}$ to construct the matrices of the ODE, means that we have all the values to calculate the energy.

Because we know that real solutions of the wave equation and finite element formulation need to have a constant energy, we can use the energy function as a measure of error over time when we have all the values U_i and $\dot{U}_i$ at a moment in time, like for the second method. We can even use this method this analyse numerical solutions for which we don't have the real solutions.

The first place where we saw something that we can analyse through energy is the increase of amplitude over time we could see in figures 1.4 and 1.5 in chapter 1. If we take the same example solution and use ten regularly spaces elements, we can see the energy using different time steps in figures 3.1a and 3.1b.



(a) Energy of the example over a short interval for different time steps.



(b) Energy of the example over a long interval for different time steps.

Figure 3.1: Energy of the example for different time steps.

In the figures we can see that the energy increases over time, but that the amount of increase decreases with the time step. This increase in energy can be seen through the slowly increasing amplitude of the numerical solution. While in the beginning this energy increase is relatively unstable (the energy increases, but not exponentially), we can see in the logarithmic plot 3.1b that after some time there is a huge change in behaviour. After some time the energy starts to increase exponentially, but this exploding change happens later and is less steep when the time step decreases. We can actually see the beginning of this explosive behaviour on the bottom right of figure 1.5, where the solution becomes slightly less sinusoidal and the end result can be seen in 2.7a and 2.8c where the numerical solution suddenly turns chaotic.

This is also why we used a norm that only uses time up until a finite value T instead of all positive numbers. For finite T the energy is bounded, but if we remove this restriction, it might happen that the energy diverges to infinite and the solution will not converge.

Now let us take a look at a more complex situation where we do not know the real solution. For this we will use an example where the initial value consists of two Gaussian peaks, like in figure 3.2, and where the derivative in time is initially zero everywhere.

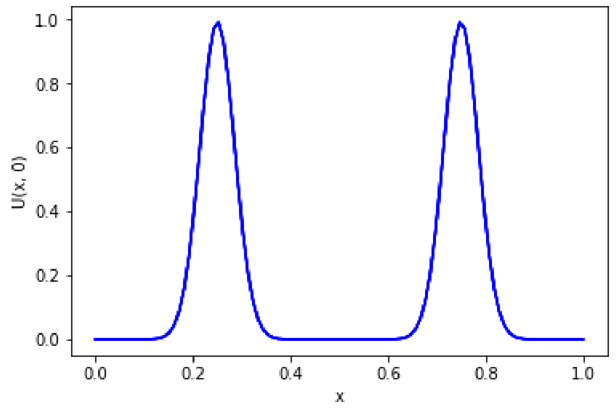
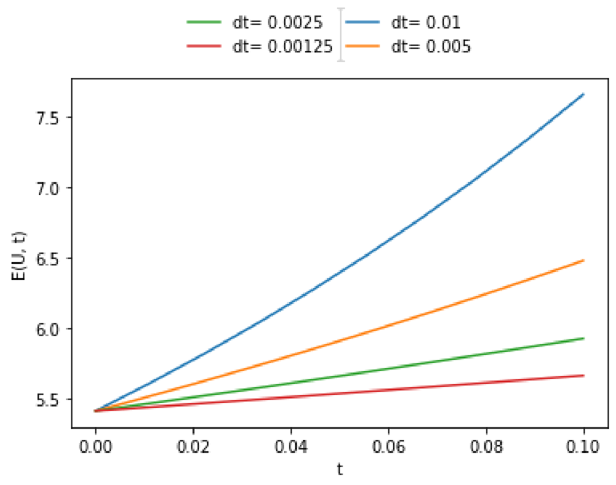
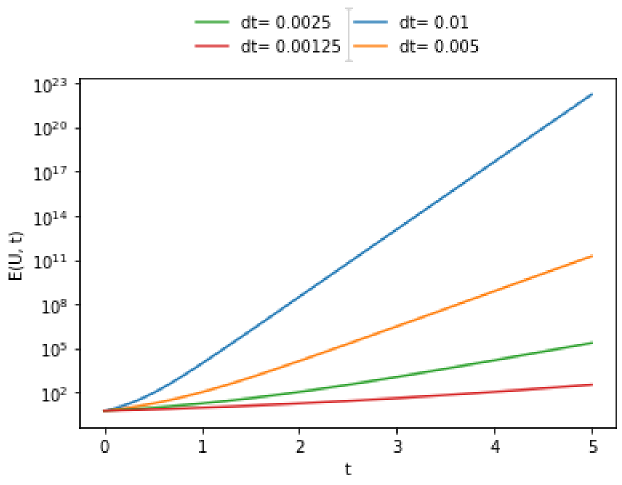


Figure 3.2: Two Gaussian peaks as the initial value for the example.

We can see the energy over time for different time steps for 10 and 20 regular elements in figure 3.3 and 3.4 respectively. In these figures the instability becomes more clear and we can even notice that smaller element like for figure 3.4 is more unstable than in 3.3. Because in both cases the energy increases less rapidly with smaller times steps we need both the time step and the element size to decrease sufficiently fast to not increase the instability.

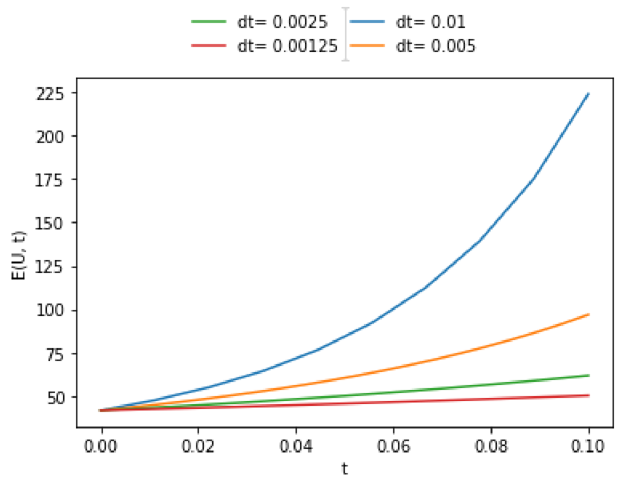


(a) Energy of the example over a short interval for different time steps.

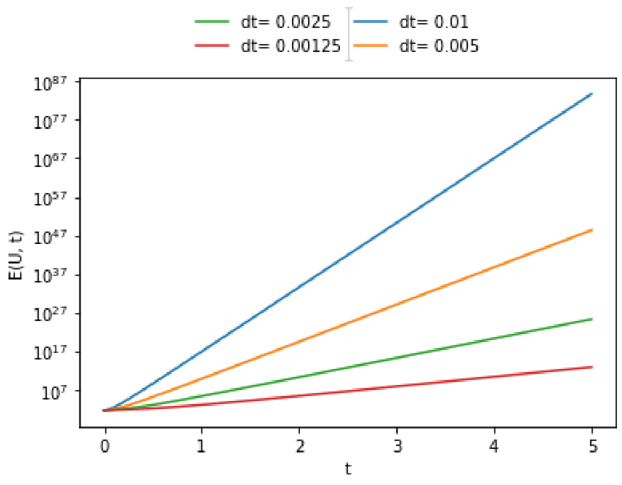


(b) Energy of the example over a long interval for different time steps.

Figure 3.3: The energy of the example simulated using 10 regular elements.



(a) Energy of the example over a short interval for different time steps.



(b) Energy of the example over a long interval for different time steps.

Figure 3.4: The energy of the example simulated using 20 regular elements.

Thus, we can conclude from the analysis of the energy that we require both the decrease in size of elements and of the time step for the convergence of the numerical solution. The size of the elements and the degree of the polynomials for approximation mainly contributes to the convergence of the energy to that of the real energy and the decrease of the time step mainly contributes to decreasing the deviation from the constant energy, but we need to decrease both values sufficiently fast to not increase the instability.

3.3 Dispersion

One phenomenon we have not touched on yet is the different periodic movement a numerical solution might make compared to the actual solution, like could be seen in figure 1.3. We will analyse this by searching a relation between the spatial and temporal frequencies of waves.

Not every wave is a solution of the wave equation. To see this we will take a look at solutions of the form[5]

$$u(x,t) = e^{i(\omega t + kx)} = \cos(\omega t + kx) + i\sin(\omega t + kx), \tag{3.3.1}$$

where ω is the (temporal) frequency and k is called the wave number or spatial frequency. When we put this solution in the wave equation, we can derive a relation as follows:

$$\begin{aligned} \frac{\partial^2}{\partial t^2} e^{i(\omega t + kx)} - c^2 \frac{\partial^2}{\partial x^2} e^{i(\omega t + kx)} &= 0, \\ \left(-\omega^2 + c^2k^2\right) e^{i(\omega t + kx)} &= 0, \\ \omega^2 &= c^2k^2, \\ \omega &= \pm ck. \end{aligned} \tag{3.3.2}$$

This relation between the frequency and the wave number is called the dispersion relation of the wave equation. We can already see in figure 1.3 that the same relation does not hold for the numerical method. The problem we are going to analyse now is if there is such a relation for our numerical method and if there is, how does it relate to the dispersion relation derived above.

The first step is to discretize the solution 3.3.1. If we have n nodes and basis functions then the finite element solution is written as $U(x,t) = \sum_{i=1}^{n} U_i(t)\phi_i(x)$, where $U_i(t)$ is the value at the node x_i at time t. From this we get the discretization

$$U_i(t) = e^{i(\omega t + kx_i)}. \tag{3.3.3}$$

The numerical method 1.5.9 we used could be written as

$$\begin{cases} T\left(\frac{\dot{U}(t+dt)-\dot{U}(t)}{dt}\right) & = -c^2SU(t), \\ U(t+dt) & = U(t) + dt\dot{U}(t), \end{cases} \tag{3.3.4}$$

and if we put the just derived discretization in there we get the following equation

$$\begin{cases} T\left(\frac{i\omega e^{i\omega dt}U(t) - i\omega U(t)}{dt}\right) & = -c^2SU(t), \\ e^{i\omega dt}U(t) & = \left(1 + dti\omega\right)U(t). \end{cases} \tag{3.3.5}$$

Let us first take a look at the second equation in 3.3.5. This equation has to hold for all t and because $U_i(t) = e^{i(\omega t + kx_i)} = 0$ has no solution it implies that $e^{i\omega dt} = 1 + idt\omega$. If we rewrite it using Euler's formula we get $\cos(\omega dt) + i\sin(\omega dt) = 1 + i\omega dt$. From this equation we can conclude that ω is either 0 or complex with non-zero imaginary part, because if ω would be real it must hold that $\sin(\omega dt) = \omega dt$, which can only hold for real numbers if $\omega dt = 0$ and if we let dt be zero we have no use for the numerical method. If we choose ω to be zero then there is no temporal movement in the solution, so the only non-trivial option is for ω to be complex with $\text{Im}(\omega) \neq 0$.

But what happens when we write ω as $\text{Re}(\omega) + i\text{Im}(\omega)$ with $\text{Im}(\omega) \neq 0$. If we put this in the discretized solution we can rewrite it as follows

$$U_i(t) = e^{i(\text{Re}(\omega)t + i\text{Im}(\omega)t + kx_i)} = e^{-\text{Im}(\omega)t}e^{i(\text{Re}(\omega)t + kx_i)}, \tag{3.3.6}$$

then this equation has two possible scenarios (already excluding $\text{Im}(\omega) = 0$), either $\text{Im}(\omega) > 0$ in which case the solution will converge to zero or $\text{Im}(\omega) < 0$ in which case the solution will diverge to infinite. Therefore, if there exists such a numerical solution, it will always either diverge to infinity or dim out to zero.

Let us now take a look at the first equation in 3.3.5. We can rewrite this equation as

$$\frac{i\omega\left(e^{i\omega dt}-1\right)}{dt}TU(t) = -c^2 SU(t). \tag{3.3.7}$$

As an example, we will use the situation as in chapter 1, where the domain is the unit interval $[0,1]$, which we divide in N equally sized elements of size $h = \frac{1}{N}$ such that for $j = 1, \ \dots \ , N+1$ there is a node $x_j = (j-1)h$. In this case we already calculated the values for T and S. If we combine this with equation 3.3.7 we get for $j = 1, \ \dots \ , N$ the following non-trivial equations for which we can derive a dispersion relation as follows

$$\begin{aligned}
\frac{i\omega\left(e^{i\omega dt}-1\right)}{dt}\left(\frac{1}{6}hU_{j-1}(t)+\frac{2}{3}hU_j(t)+\frac{1}{6}hU_{j+1}(t)\right) &= -c^2\left(-\frac{1}{h}U_{j-1}(t)+\frac{2}{h}U_j(t)-\frac{1}{h}U_{j+1}(t)\right),\\
\frac{i\omega\left(e^{i\omega dt}-1\right)}{dt}\left(\frac{1}{6}he^{-ikh}+\frac{2}{3}h+\frac{1}{6}he^{ikh}\right)e^{i(\omega t+kx_j)} &= -c^2\left(-\frac{1}{h}e^{-ikh}+\frac{2}{h}-\frac{1}{h}e^{ihk}\right)e^{i(\omega t+kx_j)},\\
\frac{i\omega\left(e^{i\omega dt}-1\right)}{dt}h^2\left(\frac{1}{6}\left(e^{ikh}+e^{-ikh}\right)+\frac{2}{3}\right) &= -c^2\left(-\left(e^{ihk}+e^{-ikh}\right)+2\right),\\
i\omega\left(e^{i\omega dt}-1\right)h^2\frac{1}{6}\left(\cos(kh)+2\right) &= c^2 dt\left(\cos(kh)-1\right).
\end{aligned} \tag{3.3.8}$$

This results in the dispersion relation

$$\frac{i\omega}{dt}\left(e^{i\omega dt}-1\right) = \frac{6c^2}{h^2}\frac{\cos(kh)-1}{\cos(kh)+2}. \tag{3.3.9}$$

This relation approximates the dispersion relation of the wave equation $\omega = \pm ck$ for small values of ω and k. To notice this we will approximate equation 3.3.9. We will use the the first few terms of the Taylor expansions of e^x and $\frac{\cos(x)-1}{\cos(x)+2}$, which are given as

$$e^x = 1 + x + \mathcal{O}(x^2), \tag{3.3.10}$$

and

$$\frac{\cos(x)-1}{\cos(x)+2} = -\frac{x^2}{6} + \mathcal{O}(x^3). \tag{3.3.11}$$

Filling these approximations in the dispersion relation results in

$$\omega^2 \approx c^2k^2, \tag{3.3.12}$$

for small values of ωdt and kh.

Therefore we might expect the numerical solution to resemble the real solution when ωdt and kh are small. This gives another reason why we need decrease both dt and h sufficiently to improve the accuracy of the dispersion relation.

This chapter we have seen the importance of element sizes and the time step with regard to the accuracy of the solution and the requirement to decrease values sufficiently for both convergence and stability.

4 Two-dimensional wave equation

In this chapter we will numerically solve the two-dimensional wave equation using the same techniques of the first two chapters. The *two-dimensional wave equation* over a domain $D \subset \mathbb{R}^2$ is defined as

$$\frac{\partial^2 u}{\partial t^2} - c^2 \Delta u = 0 \quad \text{for} \quad c > 0, \tag{4.0.1}$$

with initial boundary conditions

$$\begin{aligned} u(x,y,t) &= 0 \quad \text{for all} \quad (x,y) \in \partial D, \\ u(x,y,0) &= u_0(x,y) \quad \text{for all} \quad (x,y) \in D \quad \text{and} \\ \nabla u(x,y,0) &= u_1(x,y) \quad \text{for all} \quad (x,y) \in D. \end{aligned} \tag{4.0.2}$$

The derivation is almost the same as in previous chapters, but the introduction of the second spatial dimension creates a few extra difficulties, but luckily the resulting ordinary differential equation is of the same form as in the one-dimensional case. In this chapter we will mainly focus on the differences and extra difficulties of solving the two-dimensional wave equations compared to the one-dimensional wave equation.

4.1 Higher-dimensional weak formulation

First we will need a weak formulation of the two-dimensional wave equation. Because it is easier, we will actually derive the weak formulation for the wave equation in arbitrary dimensions. For this we will need a generalization of integration by parts called Green's first identity[6], which is formulated as

$$\int_D \psi \Delta \phi dV + \int_D \nabla\psi \cdot \nabla\phi dV = \int_{\partial D} \psi(\nabla\phi \cdot n) dV, \tag{4.1.1}$$

where n is the normal of the hypersurface ∂D.

Just like for the one-dimensional case we will multiply the wave equation with a test function $v(x,y)$, which is zero at the boundary of the domain, and integrate over the whole domain. This results in the follow equation:

$$\begin{aligned} 0 &= \int_D \frac{\partial^2 u}{\partial t^2}(x,y,t) v(x,y) dA - c^2 \int_D \Delta u(x,y,t) v(x,y) dA \\ &= \int_D \frac{\partial^2 u}{\partial t^2}(x,y,t) v(x,y) dA - c^2 \left(\int_{\partial D} v(x,y) \left(\nabla u(x,y,t) \cdot n\right) dA - \int_D \nabla u(x,y,t) \cdot \nabla v(x,y) dA \right) \\ &= \int_D \frac{\partial^2 u}{\partial t^2}(x,y,t) v(x,y) dA + c^2 \int_D \nabla u(x,y,t) \cdot \nabla v(x,y) dA \\ &= a(u(x,y,t), v(x,y)), \\ &\text{with} \quad a(u(x,y,t), v(x,y)) := \int_D \frac{\partial^2 u}{\partial t^2}(x,y,t) v(x,y) dA + c^2 \int_D \nabla u(x,y,t) \cdot \nabla v(x,y) dA. \end{aligned} \tag{4.1.2}$$

We will not discuss the the requirements on $u(x,y,t)$ and $v(x,y)$ in depth, except for the requirement that they both need to be square integrable on the domain such that the integrals above exist.

The result is the two-dimensional weak formulation, where we search for a function $u(x,y,t)$, such that for all test functions $v(x,y)$ it holds that $a(u(x,y,t), v(x,y)) = 0$ and that the conditions 4.0.2 hold.

The space which contains u and the test functions is infinite-dimensional, therefore we will search in a finite-dimensional subspace spanned by global basis functions $\phi_i(x,y)$. The result is the finite element formulation where we search for the finite element solution

$$U(x,y,t) := \sum_{i=1}^{n} U_i(t) \phi_i(x,y), \tag{4.1.3}$$

such that for all finite element test functions

$$V(x,y) := \sum_{i=1}^{n} V_i \phi_i(x,y), \tag{4.1.4}$$

it holds that

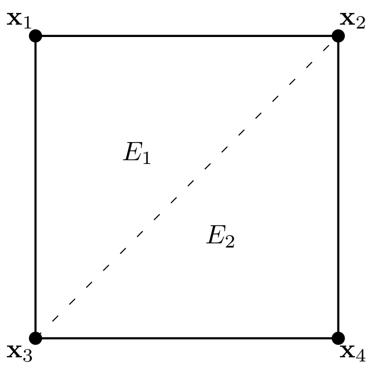
$$a(U(x,y,t), V(x,y)) = 0, \tag{4.1.5}$$

and because of the bilinearity of $a(\cdot,\cdot)$ it is enough to only test $U(x,y,t)$ against the global basis functions $\phi_i(x,y)$.

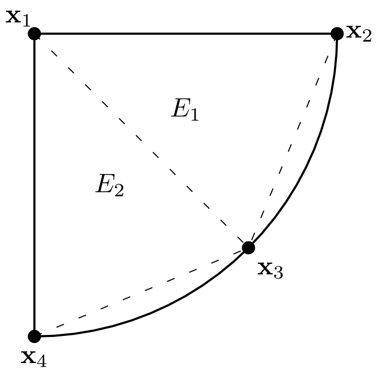
Before we can define the actual global basis functions we first need to define the elements we are going to approximate the solution on.

4.2 Elements and Nodes

Let there be N elements and n nodes. The nodes are $\mathbf{x}_i \in \mathbb{R}^2$ for $i = 1, \ldots, n$ and the elements $E_j \subset \mathbb{R}^2$ for $j = 1, \ldots, N$ are defined by three nodes for which we will use the indices $E_{j,i}$ such that the three corners of element E_j are $\mathbf{x}_{E_{j,1}}$, $\mathbf{x}_{E_{j,2}}$ and $\mathbf{x}_{E_{j,3}}$, which we will from now on write as $\mathbf{x}_{j,1}$, $\mathbf{x}_{j,2}$ and $\mathbf{x}_{j,3}$ respectively. It is important that all elements are orientated the same way, thus all corners are either indexed clockwise or counter-clockwise. Not orientating the elements the same can lead to unexpected results. The figures 4.1a and 4.1b are examples of discretization of a square domain and partial discretization of a quarter circle domain respectively.



(a) Discretization of a square



(b) Partial discretization of a quarter circle

These examples show one of the problems that arises when we go from one to two dimensions. In one dimension, every connected domain is an interval which we can always fully subdivide in more intervals, but in two dimensions we cannot always fully subdivide a connected domain in finitely many triangles like in figure 4.1b. We can however use more elements to better approximate the domain or use special curved elements, but for simplicity we will only discuss triangular elements in this thesis.

Another problem we get when going to a higher dimension is that subdividing elements does not necessarily increase accuracy. For example, in the case of a triangular domain we can continuously subdivide the elements like in figure 4.2. The result is many stretched out elements and the approximation between the extremities of the elements does not improve. Therefore, while in one dimension an important measure for the convergence was the length of the intervals, in two (and higher) dimensions this generalizes to the diameter (furthest distance between points) of the elements.

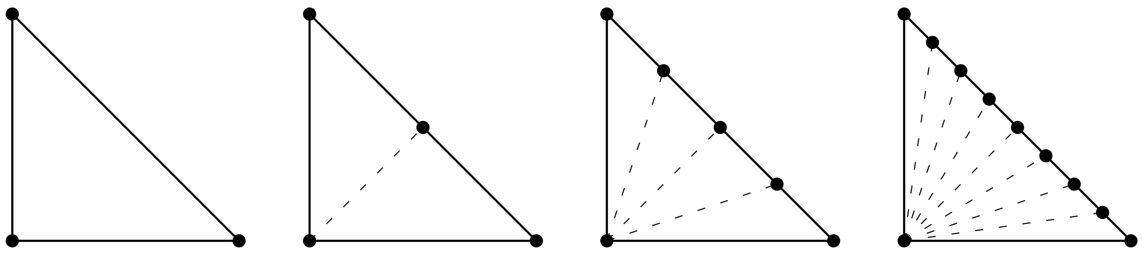

(a) One triangle (b) Two triangles (c) four triangles (d) eight triangles

Figure 4.2: Repeated subdivision

4.3 Canonical Elements and Basis function

Just like in chapter 2 we will make sure that each element is just a transformation from a simpler canonical element. As all our elements are triangles, we will define our element E_c as the triangle spanned by the points $(0, 0)$, $(1, 0)$ and $(0, 1)$ as illustrated in figure 4.3.

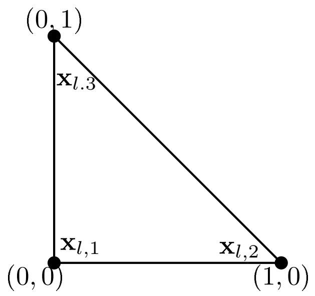


Figure 4.3: Triangular canonical element E_c

We can then get each triangular element spanned by the points $\mathbf{x}_{j,1}$, $\mathbf{x}_{j,2}$ and $\mathbf{x}_{j,3}$ using the transformation τ_j given by

$$\tau_i(X, Y) = (1 - X - Y)\mathbf{x}_{j,1} + X\mathbf{x}_{j,2} + Y\mathbf{x}_{j,3}, \tag{4.3.1}$$

which has an inverse if the triangle is not degenerate (the three points do not lie on the same line). The transformation is illustrated in figure 4.4.

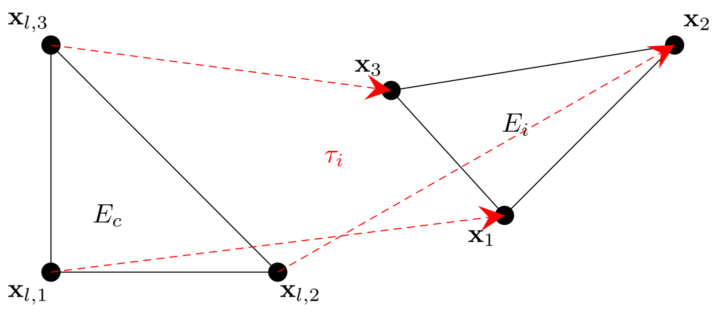
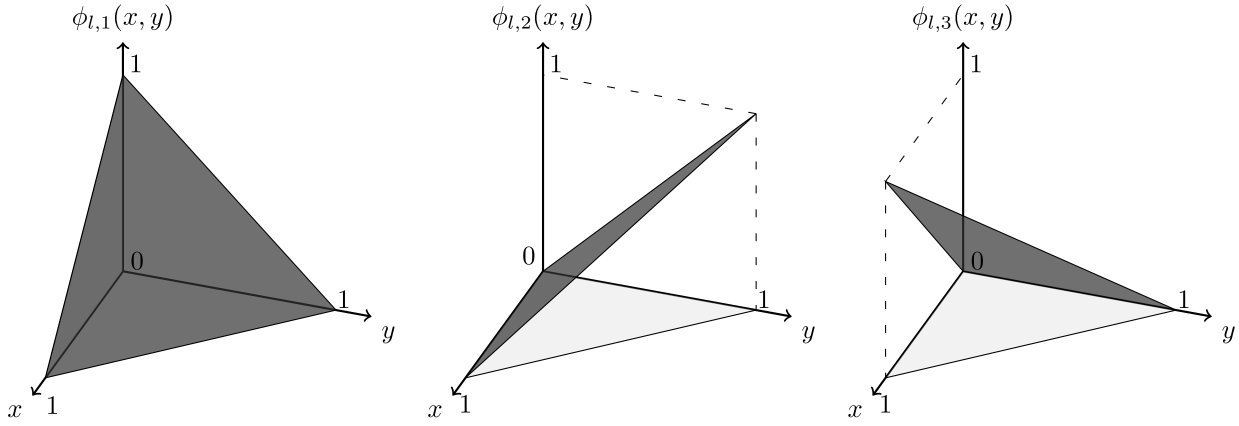


Figure 4.4: Transformation τ_i from the canonical element E_c to the element E_i .

Before defining the global basis functions we have to first define the local basis functions on the canonical element. We again want the local basis functions to be linear on the canonical element, one on their respective point and zero on all other nodes. These basis functions can simply be defined as

$$\begin{aligned} \phi_{l,1}(x,y) &= 1 - x - y, \\ \phi_{l,2}(x,y) &= x \quad \text{and} \\ \phi_{l,3}(x,y) &= y, \end{aligned} \tag{4.3.2}$$

and are illustrated in figure 4.5.



(a) Local basis function $\phi_{l,1}(x,y)$ (b) Local basis function $\phi_{l,2}(x,y)$ (c) Local basis function $\phi_{l,3}(x,y)$

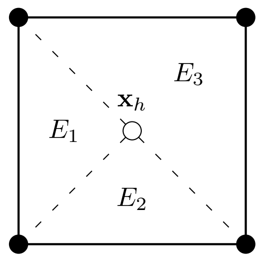
Figure 4.5: two-dimensional local basis functions

We now have everything to define the global basis functions. We will construct a single global basis function for each node $\mathbf{x}_i$ as follows. If the node $\mathbf{x}_i$ is a corner point of an element E_j such that $\mathbf{x}_i = \mathbf{x}_{j,k}$, therefore $\mathbf{x}_i$ is the k^{th} corner point of E_j, then the corresponding basis function $\phi_i(x,y)$ is given on that element by

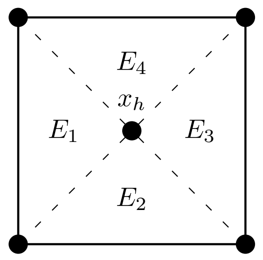
$$\phi_i(x,y) = \phi_{l,k}(\tau_j^{-1}(x,y)), \tag{4.3.3}$$

and is zero on all elements the node is not a corner point of or is not covered by an element.

The definition of the basis functions does give rise to a problem called hanging nodes. In figure 4.6a we see node $\mathbf{x}_h$, which is at the corner of the elements E_1 and E_2, but lies on an edge of E_3, therefore the corresponding basis function ϕ_h is one at x_h for the elements E_1 and E_2, but is zero on the whole of E_3, causing a discontinuity on the edges connecting E_1 and E_2 to E_3. If the value $U_h(t)$ is non-zero, then this discontinuity will also be in the finite element solution. Because we do not want this discontinuity, we also do not want these hanging nodes. Luckily we can easily "unhang" a hanging node by splitting the element it lies on the edge of in two new elements, both having the hanging element in the corner, this is illustrated in 4.6b.



(a) Example of a hanging node x_h.



(b) Unhanging of the node x_h.

Figure 4.6: Example of a hanging node.

4.4 Finite Element Solution

The rest of the derivation looks a lot like chapter 1 and 2. We defined the finite element solution as a linear combination of the global basis functions like

$$U(x, y, t) := \sum_{i=1}^{n} U_i(t)\phi_i(x, y). \tag{4.4.1}$$

Filling this in the finite element formulation results in almost the same equations like in 1.4.6, which for nodes x_j not on the boundary are

$$\begin{aligned} &\sum_{i=1}^{n} \ddot{U}_i(t)T_{i,j} + c^2 \sum_{i=1}^{n} U_i(t)S_{i,j} = 0, \\ \text{with}\quad &T_{i,j} := \int_D \phi_i(x, y)\phi_j(x, y)dA \quad \text{and} \quad S_{i,j} := \int_D \nabla\phi_i(x, y) \cdot \nabla\phi_j(x, y)dA. \end{aligned} \tag{4.4.2}$$

If we combine this with an equation $\ddot{U}_j(t) = 0$ for each node x_j that lies on the boundary of the domain we can construct the same ordinary differential equation as in the first two chapters, namely

$$T\ddot{U}(t) + c^2 SU(t) = 0, \tag{4.4.3}$$

Which we can solve in the same way as in chapter 2.

The only problem we have now is how to calculate the coefficient $T_{i,j}$ and $S_{i,j}$, but luckily we can also use here a substitution of variables. Firstly we write every integral $\int_D$ to $\sum_{i=1}^{n} \int_{E_i}$ [2] such that in each integral over an element E_j we can write the basis functions as $\phi_i(x, y) = \phi_{l,k}(\tau_j^{-1}(x, y))$, where k is the index of the local basis function the global basis function takes on element j, or else the basis function is zero. Then for functions of the form $f(\tau_j^{-1}(x, y))$, such as the integrands in $T_{i,j}$ and $S_{i,j}$, it holds that

$$\int_{E_j} f(\tau_j^{-1}(x, y))dA = \int_{E_c} f(x, y)JdA, \tag{4.4.4}$$

where J is the determinant of the Jacobian matrix of $\tau_j(x, y)$. If the element E_j is the triangle with the points (x_1, y_1), (x_2, y_2) and (x_3, y_3) at the corner, we can calculate the value J as follows

$$J = \det\left(\frac{\partial\tau_j(x, y)}{\partial x} \middle| \frac{\partial\tau_j(x, y)}{\partial y}\right) = \det\begin{pmatrix} x_2 - x_1 & x_3 - x_1 \\ y_2 - y_1 & y_3 - y_1 \end{pmatrix} = (x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1). \tag{4.4.5}$$

Therefore J is a constant independent on the variables of integration and can be placed outside of the integral.

Now the only things we need to calculate the the coefficients $T_{i,j}$ and $S_{i,j}$ are the values $\int_{E_c} \phi_{l,i}(x, y)\phi_{l,j}(x, y)dA$ and $\int_{E_c} \nabla\phi_{l,i}(x, y) \cdot \nabla\phi_{l,j}(x, y)dA$ (for which no actual integration is required, because the derivative are constant) for $i, j = 1, 2, 3$. These values are summarized in the tables 2 and 3.

Table 1: Values of the integrals of the local basis functions.

Table 2: $\int_{E_c} \phi_{l,i}(x, y)\phi_{l,j}(x, y)dA$

i \ j	1	2	3
1	$\frac{1}{12}$	$-\frac{1}{24}$	$-\frac{1}{24}$
2	$-\frac{1}{24}$	$\frac{1}{4}$	$\frac{1}{8}$
3	$-\frac{1}{24}$	$\frac{1}{8}$	$\frac{1}{12}$

Table 3: $\int_{E_c} \nabla\phi_{l,i}(x, y) \cdot \nabla\phi_{l,j}(x, y)dA$

i \ j	1	2	3
1	1	$-\frac{1}{2}$	$-\frac{1}{2}$
2	$-\frac{1}{2}$	$\frac{1}{2}$	0
3	$-\frac{1}{2}$	0	$\frac{1}{2}$

[2] In cases where we can not fully tile the domain with triangles, we lose integration over the uncovered domain, but luckily the global basis functions are defined such that they are zero on the uncovered areas.

This table allows us to quickly calculate parts of the coefficients of $T_{i,j}$ and $S_{i,j}$, for example, if the element E_j is the triangle with the points (x_1, y_1), (x_2, y_2) and (x_3, y_3), then we get

$$\int_{E_j} \phi_{l,1}(\tau_j^{-1}(x,y))\phi_{l,1}(\tau_j^{-1}(x,y))dV = \frac{1}{12}((x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)), \tag{4.4.6}$$

and

$$\int_{E_j} \nabla\phi_{l,2}(\tau_j^{-1}(x,y)) \cdot \nabla\phi_{l,1}(\tau_j^{-1}(x,y))dV = \frac{1}{2}((x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)). \tag{4.4.7}$$

Now that we have a way to construct matrices of the system of ordinary differential equations we need to discretize it in time, but luckily we can use the exact same methods as we have seen in chapter 1.

4.5 Examples

We will now take a look at a few examples. First, let the constant c be 1 and the domain D be the square $[0, 2]^2$. We will use the following initial values:

$$u_0(x,y) = \cos\left((x-1)\frac{\pi}{2}\right)\cos\left((y-1)\frac{\pi}{2}\right), \tag{4.5.1}$$

and

$$u_1(x,y) = 0. \tag{4.5.2}$$

We will split the domain regularly using n_x and n_y squares in the x-and y-axis respectively and divide each square in two triangles as illustrated in figure 4.7.

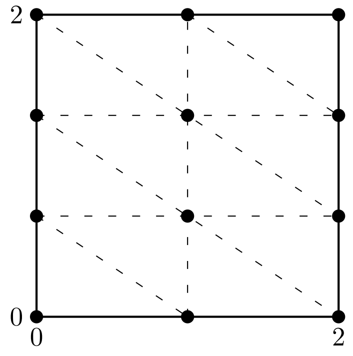
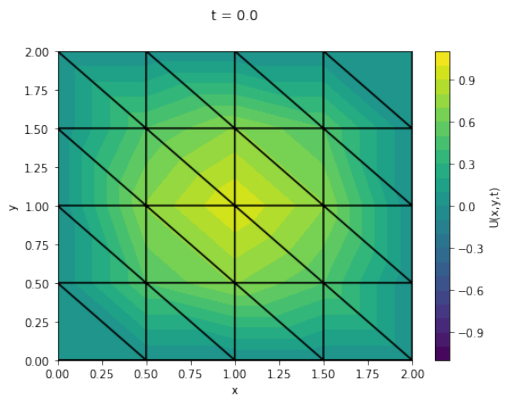
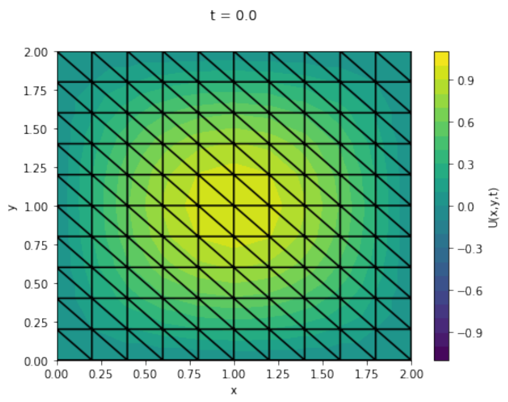


Figure 4.7: Example discretization using $n_x = 2$ and $n_y = 3$.

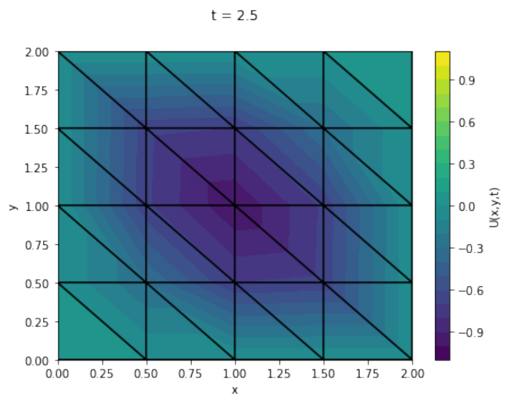
In figures 4.8 we can see two examples with time step 0.01 and different amount of triangles. These figures show for different domain discretizations the simulation at different moments in time. It is interesting to see that after some time the shape of the wave stretches in the direction of the diagonals of the elements.



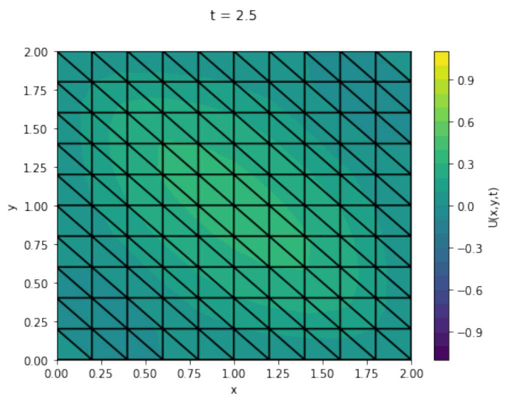
(a) A 4×4 grid at time $t = 0$.



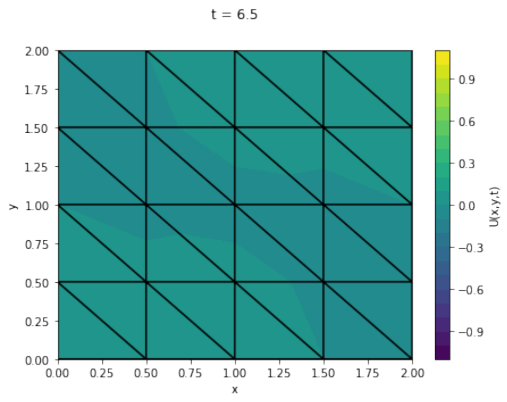
(b) A 10×10 grid at time $t = 0$.



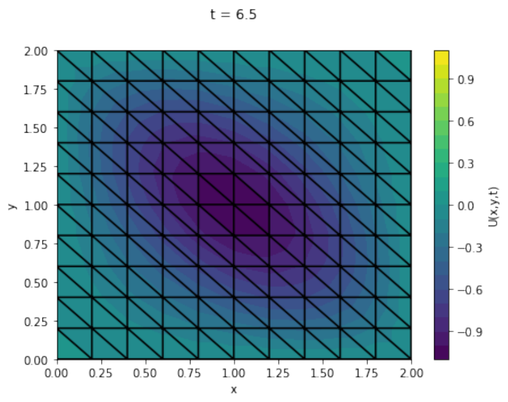
(c) A 4×4 grid at time $t = 2.5$.



(d) A 10×10 grid at time $t = 2.5$.



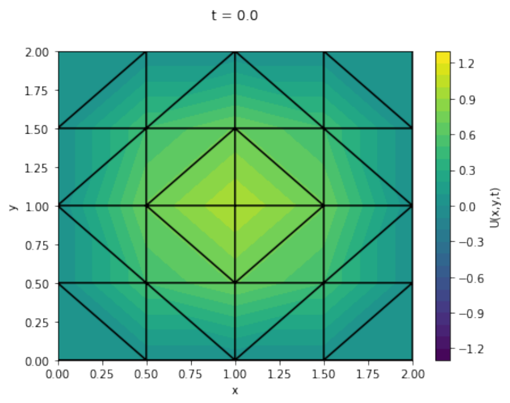
(e) A 4×4 grid at time $t = 6.5$.



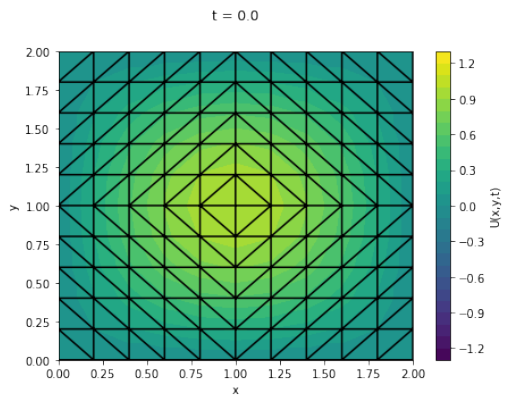
(f) A 10×10 grid at time $t = 6.5$.

Figure 4.8: Simulation of the initial values $u_0(x,y) = \cos\left((x-1)\frac{\pi}{2}\right)\cos\left((y-1)\frac{\pi}{2}\right)$ and $u_1(x,y) = 0$ with time step 0.01 and with $(n_x, n_y) = (4,4)$ (left) and $(n_x, n_y) = (10,10)$ (right) at different moments in time.

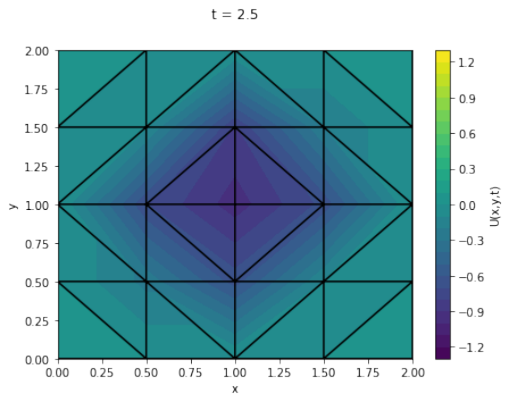
In an attempt to decrease the stretching we can discretize the square differently, like can be seen in figure 4.9. In this discretization we take into account the symmetric and radial properties of the initial value and domain. The simulation behaves similarly to those in figure 4.8, but with better symmetry. There is still some stretching to the top, which might be due to numerical errors.



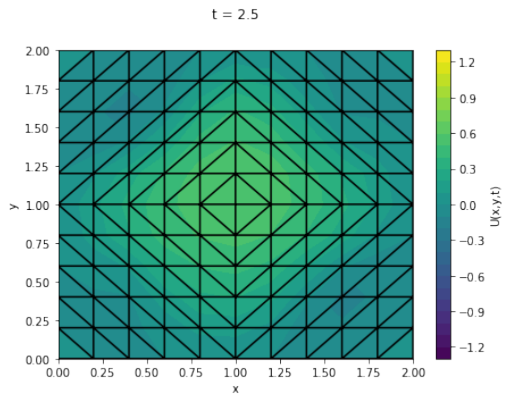
(a) A 4×4 grid at time $t = 0$.



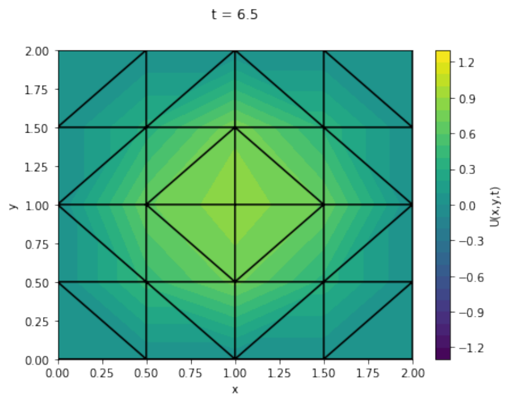
(b) A 10×10 grid at time $t = 0$.



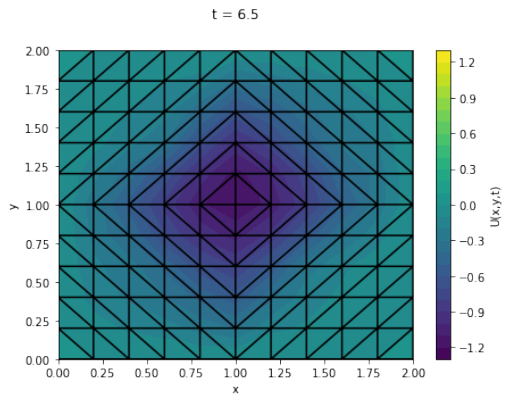
(c) A 4×4 grid at time $t = 2.5$.



(d) A 10×10 grid at time $t = 2.5$.



(e) A 4×4 grid at time $t = 6.5$.



(f) A 10×10 grid at time $t = 6.5$.

Figure 4.9: Simulation of the initial values $U_{0,0}(x,y) = \cos\left((x-1)\frac{\pi}{2}\right)\cos\left((y-1)\frac{\pi}{2}\right)$ and $U_{0,1}(x,y) = 0$ on a symmetric discretization with time step 0.01 and with $(n_x, n_y) = (4,4)$ (left) and $(n_x, n_y) = (10,10)$ (right) at different moments in time.

We can also have more complex domains than rectangles. As the next example we will use the domain as in figure 4.10[3]

[3]Some interesting property about this domain is explained at `https://en.wikipedia.org/wiki/Hearing_the_shape_of_a_drum`.

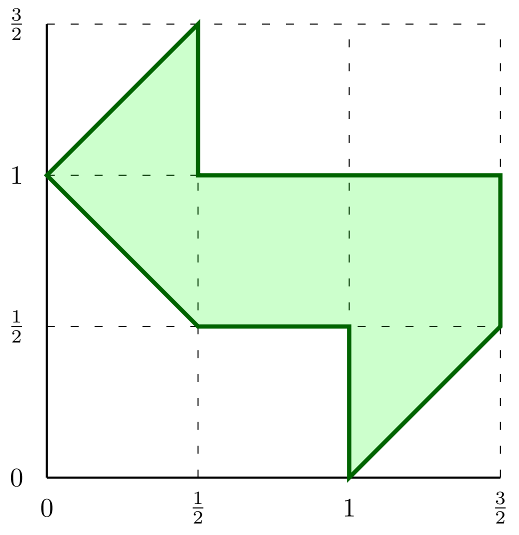
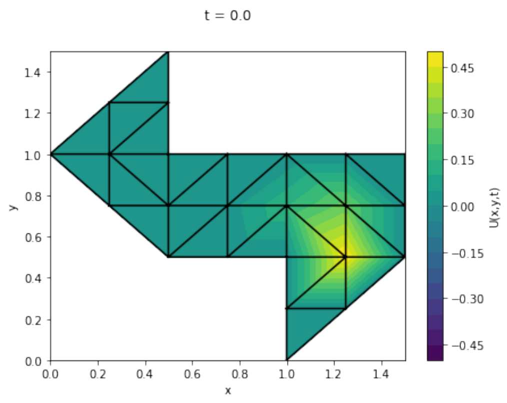
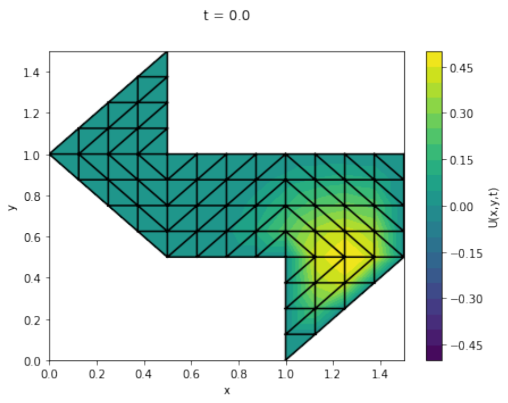


Figure 4.10: A more complex domain in two dimensions.

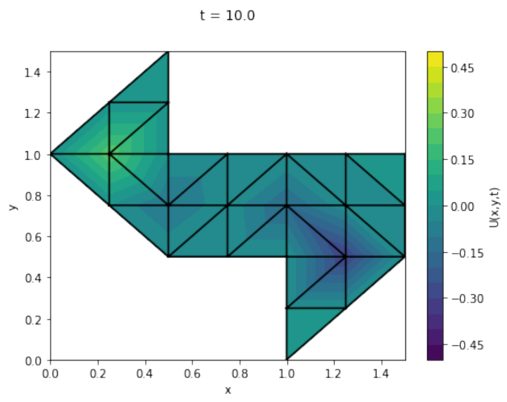
The domain has been discretized in two different ways and with initially a Gaussian peak centred around $(1.5, 0.5)$. These discretizations are simulated with a time step of 0.01 and plotted at different moment in time in figure 4.11. The wave that starts in the bottom-right corner creates smaller waves that move to the top-left corner, just like we expect. We can also see that, just like in the previous examples, the increase of the amount of elements slows down the propagation of the wave.



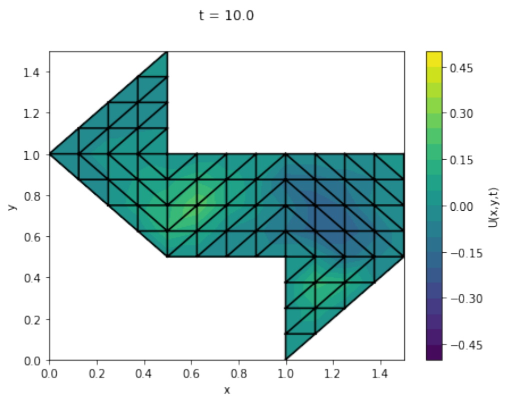
(a) 28 Elements at $t = 0$.



(b) 112 Elements at $t = 0$.



(c) 28 Elements at $t = 10$.



(d) 112 Elements at $t = 10$.

Figure 4.11: Simulation on a non-rectangular domain with initially a Gaussian peak for two different discretizations with timestep 0.01.

5 Conclusion

In this thesis we have seen how to use the finite element method to numerically solve the one- and two-dimensional wave equation where the solution is fixed at zero on the boundary of the domain and with an initial value in time. We used the finite element method to discretize in space and a finite difference method to discretize in time. We have seen that with this method we have a lot of freedom with both how to discretize the domain and how to approximate the solutions.

We have also analysed the one-dimensional case both concretely and experimentally. Firstly, we have seen that we need both the size h of the biggest element and the time steps dt to go to zero for the error to converge to zero and that the degree of the polynomials p increases the rate of convergence of the spatial discretizations. More specifically, that the order of convergence of the total error is $\mathcal{O}\left(h^p + dt\right)$. Secondly, we have seen that solutions of both the wave equation and the finite element solution have constant energy and through experiments that, most of the time, the energy of the numerical solution diverges to infinity over time, but we can reduce the initial error through refining the elements and the error over time by decreasing the time steps. Thirdly, we have seen that there are restrictions on what kind of waves are solutions of the wave equation and the numerical method and that for small values of h and dt the numerical restrictions approximate those of the wave equation.

Many things we have seen could be done differently, like different types of basis functions and element shapes, and the analysis of the one-dimensional case was also not totally rigorous everywhere as we have only really discussed the stability of the solutions through experiments, but this is all just the tip of the iceberg with regards to the finite element method and its variant. But, lets leave that as an adventure for the readers and writer of this thesis.

A Implementation

In this section some of the important parts of the implementions are explained.

The implementations can be found at `www.github.com/jeverink/BachelorsThesis` and are coded in IPython Notebooks. They can be run using either Jupyter [4] or Cocalc [5], but Jupyter is recommended, due to the used animation features. Each notebook is split into different cells, but it is recommended to run all cells (Cell → Run all) due to their dependencies.

A.1 One-dimensional wave equation

The notebook discussed in this section is ***Chapter_1.ipynb***

Matrix construction

Let N and $n := N + 1$ be the amount of elements and nodes respectively and h is the size of an element. As we were able to calculate closed-form expressions for the coefficients in 1.4.10 for the values $T_{i,j}$ and $S_{i,j}$. These expressions can be used to easily construct the matrices T and S using the following code:

Construction of T

```
# Time coefficient matrix
T = np.zeros((n, n));

T[0, 0] = 1; # Left boundary
T[N, N] = 1; # Right boundary

for i in range(1, N):
  for j in range(0, N+1):
    if(i==j):
       T[i, j] = (2.0/3.0)*h;
    if(abs(i-j) == 1):
       T[i, j] = (1.0/6.0)*h;
```

Construction of S

```
# Space coefficient matrix
S = np.zeros((n, n));




for i in range(1, N):
  for j in range(0, N+1):
    if(i==j):
      S[i, j] = (2.0/h);
    if(abs(i-j) == 1):
      S[i, j] = -(1.0/h);
```

Both matrices start out with all zero coefficients. Firstly the first and last row are filled with the boundary equations derived in 1.4.7 and then the rests of the coefficients are filled in as derived in 1.4.10.

A.2 Irregular elements and higher-order approximation

The notebook discussed in this section is ***Chapter_2.ipynb***

Lagrange polynomials

As we want to be able to to approximate the solution on each element with polynomials of arbitrary degree p we need to be able to construct the local basis functions, which are Lagrange polynomials. This can be achieved using the following code which uses SymPy [6] for symbolic expressions:

[4] `www.jupyter.org/`
[5] `www.cocalc.com/`
[6] `www.sympy.org/`

Lagrange polynomials

```python
x = Symbol('x');
def lagrange(p, i):
  width = 1.0/p;
  this = i * width;
  num = 1; # Numerator
  for j in range(0, p + 1):
    if(j != i):
      num = Mul(num, x - (j*width));
  denom = 1; # Denominator
  for j in range(0, p + 1):
    if(j != i):
      denom = Mul(denom, this - (j*width));
  denom = Pow(denom, -1);
  return Mul(num, denom);
```

To easier understand how this calculates the Lagrange polynomials we can rewrite its formula 2.2.4 as follow:

$$\phi_{l,i}(x) := \prod_{\substack{1\leqslant j\leqslant p+1 \\ j\neq i}} \frac{x - x_{l,j}}{x_{l,i} - x_{l,j}} = \left(1 \cdot \prod_{\substack{1\leqslant j\leqslant p+1 \\ j\neq i}} x - x_{l,j}\right)\left(1 \cdot \prod_{\substack{1\leqslant j\leqslant p+1 \\ j\neq i}} x_{l,i} - x_{l,j}\right)^{-1}. \tag{A.2.1}$$

This formulation is almost equivalent to the way it is calculated in the code.

Matrix construction

To construct the matrices T and S we need to calculate the coefficients $T_{i,j}$ and $S_{i,j}$ which requires integration over the different elements. Let h be the size of an element, phi1 and phi2 be to local basis functions and trans and invtrans be the transformation and respectively inverse transformation of the canonical element to the global element, then this results in the following methods:

Integration for T

```python
def integ(phi1, phi2, h):
  product = Mul(phi1, phi2);
  return h*integrate(product,
                     (x, 0, 1));
```

Integration for S

```python
def diffInteg(phi1, phi2, h,
              trans, invtrans):
  # Transform local to global
  gPhi1 = phi1.subs(x,
                    invtrans);
  gPhi2 = phi2.subs(x,
                    invtrans);
  # Take global derivative
  derGPhi1 = diff(gPhi1, x);
  derGPhi2 = diff(gPhi2, x);
  # Transform global to local
  derPhi1 = derGPhi1.subs(x,
                          trans);
  derPhi2 = derGPhi2.subs(x,
                          trans);
  product = Mul(derPhi1, derPhi2);
  return h*integrate(product,
                     (x,0,1));
```

In the case of T we can directly integrate the product of the two basis functions and multiply that with the size of the element because of 2.1.5. In the case of S we cannot directly take the derivative of the local basis function because we require the derivative of the global basis functions and the easiest way to fix this is to first transform the local basis functions to the global basis functions, then take their derivative with respect to x and finally transform them back to the canonical element.

Let e be a list such that for an element E_i the left external node is e[i] and the right external node is e[i+1], then using the above defined integration methods we can now construct the matrices T and S as follows:

Construction of T

```python
# Time coefficient matrix
T = np.zeros((n, n));

for i in range(0, N):
  h = e[i + 1] - e[i];


  for j in range(0, degree + 1):
    for m in range(0, degree + 1):
      T[(degree * i) + j,
        (degree * i) + m]
       += integ(phi[j], phi[m],
                h);


for i in range(0, n):
  T[0, i] = 0;
  T[n-1, i] = 0;
T[0, 0] = 1;
T[n-1, n-1] = 1;
```

Construction of S

```python
#Time coefficient matrix
S = np.zeros((nodeCount, nodeCount));

for i in range(0, N):
  h = e[i + 1] - e[i];
  trans = (h * x) + (e[i]);
  invtrans = (x - e[i])/h;
  for j in range(0, degree + 1):
    for m in range(0, degree + 1):
      S[(degree * i) + j,
        (degree * i) + m]
       += diffInteg(phi[j], phi[m],
                    h,
                    trans, invtrans);

for i in range(0, n):
  S[0, i] = 0;
  S[n-1, i] = 0;
```

We start with all zero matrices and then fill it up with the coefficients $T_{i,j}$ and respectively $S_{i,j}$ including the first and final row. We than overwrite the the first and final row of both matrices with the respective values for the equations of the basis functions at the boundary of the domain.

Finally we need to be able to construct the continuous numerical solution from the vector of values at the nodes. If u is the vector at time t and pos is an x position on the domain, we can calculate the values $U(x,t)$ using the following method:

Evaluation

```python
# Evaluation
def ev(u, pos):
  for i in range(0, N):
    if(e[i] <= pos <= e[i+1]):
      h = e[i + 1] - e[i];
      localPos = (pos - e[i])/h;
      sum = 0;
      for p in range(0, degree+1):
        sum += u[(i*degree) + p]*phi[p].subs(x, localPos);
      return sum;
    return 0;
```

First we search the element the x position is in and then transform it to canonical element, then by knowing the vector and the element the position is in, we can calculate the actual value as the linear sum of the basis functions evaluated at the local coordinate with coefficients from the vector. An alternative method would be to simply calculate the dot product of the vector with the vector of global basis functions evaluated at the coordinate x, but this is more expensive and a waste of calculations.

A.3 Numerical analysis and experiments

*The notebook discussed in this section is **Chapter_3.ipynb***

Energy

In chapter 3 we have seen the equation 3.2.5 which we can use to easily calculate the energy of a finite element solution. If u and uDer are the vectors U and $\dot{U}$ respectively and Tc[i][j] and Sc[i][j] are the coefficients $T_{i,j}$ and $S_{i,j}$ respectively, then the function can be simply implemented as follows:

Energy calculation

```python
# Energy function
def energy(u, uDer):
  acc = 0;
  for i in range(0, nodeCount):
    for j in range(0, nodeCount):
      acc += Tc[i][j] * uDer[i] * uDer[j];
      acc += (c**2) * Sc[i][j] * u[i] * u[j];
  return acc;
```

The method is general enough that, if Tc and Sc are defined, the energy can be used for finite element solutions in arbitrary dimensions.

A.4 Two-dimensional wave equation

*The notebook discussed in this section is **Chapter_4.ipynb***

Matrix construction

The biggest difference between the one-and two-dimensional case is the way the matrices T and S are constructed. Before we construct them we first need to discuss how we store the nodes and elements. For a node x_i are x[i] and y[i] the x and y coordinates and b[i] is contains a boolean whether the node lies on the boundary of the domain and therefore needs special treatment due to boundary conditions. Then triangles is a list of triples containing the indices of the three corner nodes of the triangular element. Finally we let A and Ad be the tables 2 and 3 respectively. Using these values we can implement the matrix construction as follows:

Construction of T and S

```
T = np.zeros((n, n));
S = np.zeros((n, n));

for (i1, i2, i3) in triangles:
  inds = [i1, i2, i3];
  J = (x[i2]-x[i1])*(y[i3]-y[i1])-(x[i3]-x[i1])*(y[i2]-y[i1]);
  for i in range(0, 3):
    for j in range(0, 3):
      T[inds[i], inds[j]] += J*A[i][j];
      S[inds[i], inds[j]] += J*Ad[i][j];

for i in range(0, n):
  if(b[i]):
    for j in range(0, n):
      S[i,j] = 0;
      if(i == j):
        T[i, j] = 1;
      else:
        T[i, j] = 0;
```

In the first loop, all the coefficients $T_{i,j}$ and $S_{i,j}$ are calculated by summing the contributions of all elements. These coefficients could then be copied to use them later for calculating the energy of a finite element solution. The second loop is to change all rows corresponding to a node on the boundary and replace them with the equations $\dot{U}_i(t) = 0$.

References

[1] J. Whiteley, *Finite Element Methods: A Practical Guide* (Springer, 2017).

[2] S. Adjerid, Computer Methods in Applied Mechanics and Engineering **191**, 4699 (2002), ISSN 0045-7825, URL `http://www.sciencedirect.com/science/article/pii/S0045782502004000`.

[3] U. M. Ascher and C. Greif, *A First Course in Numerical Methods* (2011).

[4] M. G. Larson and F. Bengzon, *The Finite Element Method: Theory, Implementation, and Applications* (Springer, 2013).

[5] L. N. Trefethen, *Finite Difference and Spectral Methods for Ordinary and Partial Differential Equations* (1996), URL `http://people.maths.ox.ac.uk/trefethen/pdetext.html`.

[6] W. A. Strauss, *Partial Differential Equations: An Introduction* (2008), 3rd ed.

[7] S. C. Brenner and L. R. Scott, *The Mathematical Theory of Finite Element Methods* (Springer, 2008).