Finite elements in space and time

M.J.M. Feron
0520404
Report number: MT 05.23

Supervisor:  dr. ir. R.H.J. Peerlings
1. Introduction

In the past decades finite element methods have been used frequently in a great variety of applications. Particularly methods for approximating stationary problems are well developed. In the present project, time-dependent problems are of special interest. In particular, our attention will be focussed on problems which show large fluctuations in only a small part of the domain. Also, the interesting part of the domain will move through that domain as time changes, see Figure 1 for an example. These characteristics imply an added complexity over stationary problems.

![Figure 1 Typical response considered in the present study](image)

In figure 1 \( u \) represents the variable of interest, \( t \) the temporal coordinate and \( x \) the spatial coordinate. To deal with this kind of problems, the possibilities of finite element approximations in space and time will be explored in this report. History dependent materials will be disregarded, since they would complicate the problem even more. This is undoubtedly interesting to research, but, due to the time available to the author, was not feasible during this study.

A model problem is chosen to act as an example, on which the approximations are tested. Also an analytical solution is determined, because comparison considering accuracy is desirable.

In order to gain more experience in this field of finite element approximations, the usual numerical treatment, in which time and space are discretised separately, and a more general space-time method are presented. For the sake of simplicity only the (spatially) one-dimensional case is discussed. The space-time method has some distinct advantages from algorithmic point of view, specifically when a refined mesh is desired. This is the eventual goal, because the numerically interesting area(s) of the problem would be approximated more accurately using small element sizes.
2 Model problem

There are numerous physical phenomena that can be mathematically described by partial differential equations. When approximation or simulation of these phenomena is desirable, the finite element method comes into the picture as an important tool. In many applications, the dominant features of the solution are concentrated in only a small part of the domain. From an efficiency point of view it can then be interesting to densify the element distribution locally. This is the eventual goal of the research, in which this project is to function as a pilot study.

In this chapter the reader will be familiarized with the convection-diffusion equation. The reason why this equation is presented, is that it is widely used and easy to interpret. Properties and solutions of a specific problem will be discussed. Furthermore, it contains the property just discussed, namely that the fluctuations of the solution are concentrated in a small part of the domain.

2.1 Problem definition

In this report one partial differential equation is of particular interest, i.e. the convection-diffusion equation, which describes e.g. heat transfer in a moving fluid.

\[-\lambda \frac{\partial^2 u}{\partial x^2} + \nu c \frac{\partial u}{\partial x} + c \frac{\partial u}{\partial t} = 0\]  

(2.1)

with

\(\lambda\) = diffusion coefficient  
\(u\) = variable of interest, i.e. temperature  
\(v\) = velocity (dependent on \(x\))  
\(c\) = heat capacity  
\(x\) = spatial coordinate  
\(t\) = time coordinate

The coefficients \(\lambda\) and \(c\) are assumed to be constants. If we add the boundary and initial conditions,

\[u(x,0) = u_i(x)\]  
\[u(0,t) = g(t)\]  
\[\frac{\partial u}{\partial x} \bigg|_{x=L} = q(t)\]

(2.2)

(2.3)

with

\(u_i\) = initial temperature distribution  
\(g(t)\) = time-dependent essential boundary condition  
\(q(t)\) = time-dependent natural boundary condition
we obtain a so called boundary-initial-value problem. Note that $x = 0$ and $x = L$ represent the boundaries of the spatial domain.

Equation (2.1) describes for example a temperature distribution, a one-dimensional flow. The first part is called the diffusion part, and is responsible for the smoothing in time of the temperature distribution due to conduction. The second part of equation (2.1) is called the convective contribution, and represents transport of the heat by the flow of the medium. This is all shown in paragraph 2.2.3.

The first condition shown in 2.2 represents the initial condition, which is defined by the temperature distribution at $t = 0$. The second condition of (2.2) represents the temperature distribution at the left boundary. Furthermore the natural boundary condition (2.3) is applied at the right edge.

Since there is only one second order derivative prescribing the heat flux across this boundary, the partial differential equation is called parabolic (Boersma, 1993). It is possible to determine an analytical solution to the boundary-initial-value problem, as is shown in the next section.

### 2.2 Analytical solution

Determination of an analytical solution is relatively costly, however, finding this solution implies having in-depth knowledge of the behaviour of the equation. A second reason to be interested in an analytical solution is the possibility of comparison of the analytical solution and numerical approximation. In this part of the report the analytical solution will be discussed.

#### 2.2.1 Dimensionless form

The convection-diffusion equation (2.1) can be transformed into a dimensionless form by introducing a characteristic temperature $u^*$, a characteristic time scale $t^*$, a characteristic length $x^*$ and a characteristic velocity $v^*$ according to

\[
\begin{align*}
    u &= U u^* \\
    t &= T t^* \\
    x &= X x^* \\
    v &= V v^*
\end{align*}
\]

such that the normalised quantities $U, T, X$ and $V$ are on the order of one.

Substituting this into (2.1) yields

\[
- \lambda \frac{\partial^2 (U u^*)}{\partial (X x^*)^2} + c V v^* \frac{\partial (U u^*)}{\partial (X x^*)} + c \frac{\partial (U u^*)}{\partial (T t^*)} = 0
\]

(2.5)

For the sake of simplicity we assume $c = 1$. Furthermore dividing by $u^*$ and rewriting yields
Now there are several possibilities regarding the choice of the characteristic constants, which influence the solution. Note that only one alternative is presented here, for more alternatives: (Van Duijn & Peletier, 2004)

If \( v^* > 0 \), let \( x^* \) be a given length (e.g. length of the bar)

\[
t^* = \frac{x^*}{v^*}
\]

(2.7)

\[
\lambda_c = \frac{\lambda}{v^* x^*} \equiv \frac{1}{Pe}
\]

with \( Pe \) the dimensionless Peclet-number

(2.8)

Substituting into (2.6) yields

\[
- \frac{\lambda_c v^* x^*}{x^*} \frac{\partial^2 U}{\partial X^2} + \frac{V v^*}{x^*} \frac{\partial U}{\partial X} + \frac{v^*}{x^*} \frac{\partial U}{\partial T} = 0
\]

(2.9)

and simplifying (2.9)

\[
- \lambda_c \frac{\partial^2 U}{\partial X^2} + V \frac{\partial U}{\partial X} + \frac{\partial U}{\partial T} = 0
\]

(2.10)

Equation (2.10) represents the desired dimensionless form. Now it is possible to evaluate the relative contributions of the different terms to the solution.

### 2.2.2 Boundary-initial-value problem

In this section an analytical solution for a specific boundary-initial-value problem (BIVP) is presented with constant boundary and initial values.

\[
U(0, T) = 1 \quad \forall T > 0
\]

(2.11)

\[
U(\infty, T) = 0 \quad \forall T > 0
\]

(2.12)

\[
U(X, 0) = 0 \quad \forall X > 0
\]

(2.13)

Note also that in the following calculation, the dimensionless velocity, \( V = 1 \). Note that the second boundary condition acts at infinity and therefore can not be used in the numerical approximations. There, another boundary condition on the right edge of the domain is used, which has a similar effect as long as characteristic features of the solutions are relatively far from this boundary. Therefore, when comparing the analytical solution to the approximations, bear in mind that a comparison is not possible at the right edge of the domain used.
A straightforward way of solving this problem involves the Laplace transformation. Let \( f : \mathbb{R}^+ \rightarrow \mathbb{R} \) be given. The Laplace transformation then reads

\[
F(p) = \int_0^\infty f(t)e^{-pt}dt \quad p > 0
\]  

(2.14)

This expression can be regarded as an alternative characterization of the function \( f(t) \). The transformation of \( f(t) = 1 \) for \( t > 0 \), results in

\[
F(p) = \int_0^\infty e^{-pt}dt = \frac{1}{p} \quad p > 0
\]  

(2.15)

We can also apply this transformation to the solution \( U(X,T) \) of the partial differential equation (2.11), yielding

\[
U(X,p) = \int_0^\infty U(X,T)e^{-pt}dT \quad p > 0
\]  

(2.16)

This expression can be interpreted as the solution to a boundary value problem (with \( p > 0 \)) in the Laplace domain:

\[-\lambda_x^2 \frac{\partial^2 U}{\partial X^2} + \frac{\partial U}{\partial X} + pU = 0 \quad 0 < x < \infty\]  

(2.17)

\[
U(0, p) = \frac{1}{p}, U(\infty, p) = 0
\]

The solution of (2.17) is given by

\[
U(X,p) = \frac{1}{p} \exp \left( 1 - \sqrt{1 + 4\lambda_x^2 p} \right) \frac{X}{2\lambda_x^2} \quad X > 0
\]  

(2.18)

Transforming back into the time domain is complicated (Van Duijn & Peletier, 2004). After some manipulation the following expression is found:

\[
U(X,T) = \frac{1}{2} e^{x\sqrt{\lambda_x}} \text{erfc} \left( \frac{X + T}{2\sqrt{\lambda_x}T} \right) + \text{erfc} \left( \frac{X - T}{2\sqrt{\lambda_x}T} \right)
\]  

(2.19)

A visualization of this result helps to interpret the partial differential equation above. In the following such a visualization is presented, and the properties of the equation are briefly discussed.
2.2.3 Results

In figure 2 below, the analytical solution is plotted. For \( \lambda_c = 1 \), i.e. convection and diffusion are of equal importance (\( Pe = 1 \)).

![Analytical solution](image)

Figure 2 Analytical solution to the BIVP

In figure 3 slices of figure 2 at two different moments in time are shown, where \( 0 < T1 < T2 \).

![Slices of the analytical solution](image)

Figure 3 Slices of the analytical solution

It can be seen clearly from figure 2 and figure 3 which are the contributions of the convective and diffusion terms in equation (2.1). Due to the convective part of (2.1),
the transition area moves along the spatial coordinate. The contribution of the diffusion part of (2.1) can be recognised in the widening of the transition area as time increases.

It is important to notice that this equation contains a transition area, i.e. a steep gradient, in only a small part of the total domain and that the transition area moves along the spatial domain as time increases. This was exactly the requirement stated in the introduction. In the moving of the transition area, the characteristic velocity can be recognised. In other words, when this characteristic value is increased, the transition area will move faster in time.

The dimensionless Peclet-number represents the relative contribution of diffusion and convection. If $Pe$ is very large, diffusion is negligible. When we regard figure 4 in that case, the $U(X,T2)$ would have the same shape as $U(X,T1)$. 
3 Numerical approximations

In practice, it is very difficult or even impossible to determine an analytical solution, especially for more complex problems. Therefore, approximations are necessary. In the following two methods to obtain approximate solutions are discussed, which are both applicable to space- and time-dependent problems, such as the model problem introduced in the previous chapter.

3.1 Usual numerical treatment

The usual way of approximating non-stationary finite element problems consists of a partial discretisation in the spatial dimension(s), followed by a regular time integration scheme, to account for the temporal dependency. In this part of the report this approximation algorithm is presented. Note that only the spatially one-dimensional problem is discussed.

More about the numerical treatment of problems with temporal and spatial dependencies can be found for instance in Freund, 1996.

3.1.1 Discretisation of the problem

First, we discretise the spatial dimension of the PDE (2.1) in the following steps. The first step is known as the weighted residuals formulation:

$$
\int_0^L \phi \left( - \frac{A}{\partial x^2} \frac{\partial^2 u}{\partial x^2} + \nu \frac{\partial u}{\partial x} + c \frac{\partial u}{\partial t} \right) dx = 0 \quad \forall \phi(x)
$$

(3.1)

In this equation \( \phi(x) \) is a weight function, which satisfies the usual conditions. Then the weak form can be derived by integration by parts

$$
\int_0^L \left( \frac{\partial \phi}{\partial x} \frac{\partial u}{\partial x} + \phi \nu \frac{\partial u}{\partial x} + \phi c \frac{\partial u}{\partial t} \right) dx = \phi \lambda \frac{\partial u}{\partial x} \bigg|_0^L \quad \forall \phi(x)
$$

(3.2)

The temperature \( u \) as well as the weight function \( \phi(x) \) are now discretised using the following finite element approximations

$$
\begin{align*}
  u & \rightarrow u^h(x,t) = N^T(x)u(t) \\
  \phi & \rightarrow \phi^h(x) = N^T(x)\phi = \phi^T N(x) \\
  \frac{\partial u}{\partial x} & \rightarrow \frac{\partial u^h}{\partial x} = \frac{\partial N^T}{\partial x} u(t) \\
  \frac{\partial \phi}{\partial x} & \rightarrow \frac{\partial \phi^h}{\partial x} = \frac{\partial N^T}{\partial x} \phi = \phi^T \frac{\partial N}{\partial x}
\end{align*}
$$

(3.3)
Where \( N(x) \) contains the interpolation functions, which are constant in time, and \( u(t) \) the time-dependent nodal values. Substituting equations of (3.3) into (3.2) and using the fact that the result must hold for all \( \varphi \) yields the following discretised equation:

\[
\int_{0}^{L} \left( \frac{dN}{dx} \frac{dN^T}{dx} + N_{vc} \frac{dN^T}{dx} \right) dxu(t) + \int_{0}^{L} \left( N_{c}N^T \right) dx \frac{\partial u}{\partial t} = N\frac{\partial u}{\partial x}\bigg|_{0}^{L}
\]

(3.4)

which can also be written as

\[
Ku(t) + M \frac{du}{dt} = q
\]

(3.5)

with

\[
K = \int_{0}^{L} \left( \frac{dN}{dx} \frac{dN^T}{dx} + N_{vc} \frac{dN^T}{dx} \right) dx
\]

\[
M = \int_{0}^{L} \left( N_{c}N^T \right) dx
\]

\[
q = N\frac{\partial u}{\partial x}\bigg|_{0}^{L}
\]

Now, the spatial dependency is accounted for in the discretisation just presented. This results in a set of ordinary differential equations, (3.5). In order to account for the temporal dependency, a backward Euler approximation can be used:

\[
\frac{du}{dt} = \frac{1}{\Delta t} \left( u - u' \right)
\]

(3.6)

With \( u' = u(t) \) and \( u = u(t + \Delta t) \) the updated value in each time increment. Substituting (3.6) in (3.5) yields

\[
Ku + M \frac{1}{\Delta t} \left( u - u' \right) = q
\]

(3.7)

or

\[
\left( K + \frac{1}{\Delta t} M \right) u = \left( q + \frac{1}{\Delta t} u' \right)
\]

(3.8)

An approximate solution can now be obtained by solving the linear system of equations (3.8) repeatedly. This solution algorithm can be implemented for example in Matlab, as is done in Appendix A (section 7.1).
3.1.2 Results

Next, the results obtained with the algorithm as developed above are presented. The only goal is to prove that the method works, and the implementation is correct.

Due to the small numbers of elements that is used, the approximations are rather inaccurate. However, despite the coarse discretisation, the features of the analytical result, as shown before in figure 2, can be clearly distinguished. As the spatial and temporal discretisation are refined, the numerical solution indeed converges to the analytical result, apart from a zone near the right boundary (see section 2.2.2).

Figure 4 Result of usual numerical treatment with 10 elements in space and 10 time steps
3.2 Space-time approximation

Another way of approaching space- and time-dependent problems is discussed in the following. This space-time finite element method treats the n-dimensional problem in space plus a time dependency as an (n+1)-dimensional problem. This choice implies some distinct advantages from the algorithmic point of view as it unifies the treatment of stationary and non-stationary cases. Furthermore, from a more practical viewpoint, it allows one to refine the spatial and temporal discretisation locally in a zone which moves in space during the evolution of the temperature distribution.

3.2.1 Problem description

As a starting point of our development we rewrite (2.1) into the form

$$\nabla \cdot (C_1 \cdot \nabla u) + \bar{c}_2 \cdot \nabla u = 0$$

(3.9)

Note that this is only valid for the case that

$$\nabla = \bar{e}_1 \frac{\partial}{\partial x} + \bar{e}_2 \frac{\partial}{\partial t}$$

(3.10)

$$\bar{x} = x\bar{e}_1 + t\bar{e}_2$$

(3.11)

Bear in mind that equation (3.9) may look like a stationary boundary value problem, but due to the definition of equations (3.10) and (3.11) it undoubtedly becomes a non-stationary problem. The tensor $C_1$ and the vector $\bar{c}_2$ are defined as:

$$C_1 = -\lambda \bar{e}_i \bar{e}_i$$

(3.12)

$$\bar{c}_2 = v c \bar{e}_i + c \bar{e}_2$$

(3.13)

The boundary and initial conditions used in this approximation are stated in paragraph 2.1, by equations (2.2) and (2.3). Note that (3.9) may look like an elliptical partial differential equation, but due to the definitions of (3.12) and (3.13) it becomes parabolic. This implies the necessity of two boundary conditions at the left and right boundary, and an initial condition at $t = 0$. However, no condition at $t = \infty$ needs to be stated.

3.2.2 Discretisation of the problem

Galerkin’s method provides a way to directly recast equation (3.9) into a linear system of equations in terms of nodal values of the approximate solution. (Peerlings, 2003) Using the weighted residuals approach, a weak form can be derived. Multiplication by a test function $\phi(\bar{x})$ and integration on the domain $V$ produces:

$$\int_V \phi(\bar{x}) \nabla \cdot (C_1(\bar{x}) \cdot \nabla u) dV + \int_V \phi(\bar{x}) \bar{c}_2 \cdot (\nabla u) dV = 0 \forall \phi(\bar{x})$$

(3.14)
Where the domain $V$ is a space-time domain, i.e. $(0,L) \times (0,T)$. This equation can be further reshaped into the weak form:

$$
\int_v (\tilde{\nabla} \phi(\bar{x}) \cdot C_1(\bar{x}) \cdot \tilde{\nabla} u) dV - \int_v \phi(\bar{x}) c_2 \cdot (\tilde{\nabla} u) dV = \int_s \phi(\bar{x}) q(\bar{x}) dS \quad \forall \phi(\bar{x}) \quad (3.15)
$$

with

$$
q(\bar{x}) = \tilde{n}(\bar{x}) \cdot C_1(\bar{x}) \cdot \tilde{\nabla} u
$$

Now we can discretise the weak form by replacing the domain, $\phi(\bar{x})$ and $u(\bar{x})$:

1. $V \rightarrow V^h$
2. $u \rightarrow u^h = N^T(\bar{x}) u$
3. $\phi \rightarrow \phi^h = N^T(\bar{x}) \phi = \phi^T N(\bar{x}) \quad (3.16)$
4. $\tilde{\nabla} u \rightarrow \tilde{\nabla} u^h = \tilde{\nabla} N^T u$
5. $\tilde{\nabla} \phi \rightarrow \tilde{\nabla} \phi^h = \tilde{\nabla} N^T \phi = \phi^T \tilde{\nabla} N(\bar{x})$

The discretised domain $V^h$ is constructed from finite elements the space-time domain. This discretised domain maybe slightly different from $V$. Bear in mind that the shape functions denoted by $N(\bar{x})$ are completely different from those in (3.3), due to the fact that the present discretisation contains two dimensions.

Combining equation (3.15) and the discretised fields (3.16) leads to the desired discretised form:

$$
\int_v \tilde{\nabla} N(\bar{x}) \cdot C_1(\bar{x}) \cdot \tilde{\nabla} N^T dV u - \int_v N(\bar{x}) c_2 \cdot \tilde{\nabla} N^T dV u = \int_s N(\bar{x}) q(\bar{x}) dS \quad (3.17)
$$

Or, written in a more compact form

$$
M u = q \quad (3.18)
$$

where

$$
M = K - L
$$

$$
K = \int_v \tilde{\nabla} N(\bar{x}) \cdot C_1(\bar{x}) \cdot \tilde{\nabla} N^T dV
$$

$$
L = \int_v N(\bar{x}) c_2 \cdot \tilde{\nabla} N^T dV
$$

$$
q = \int_s N(\bar{x}) q(\bar{x}) dS
$$

The matrices $K$ and $L$ consist of elementary matrices, $K^e$ and $L^e$ respectively. For simplicity, bilinear isoparametric elements are used in the following. Gauss integration is used to approximate the integrals shown above, i.e.
The Matlab-file using this kind of approximations can be found in Appendix B (section 7.2). In this Matlab-file parameters for the temperature distribution, as described in chapter 2, are used.

### 3.2.3 Results

When all of the above is implemented for example in Matlab, an approximation of the exact analytical solution, as shown in paragraph 2.2, can be computed. In order to be able to make a comparison, we first consider the BIVP presented in chapter two. For the moment, regular meshes are used with equally distributed elements as shown in figure 5.

![Figure 5 Mesh with 10x10 elements, node numbers are represented](image)

This is not a very efficient way of approximating the solution, since the accuracy is limited by the elements transition area along the diagonal from the left bottom to the right top corner. However, this mesh is nevertheless used in order to be able to easily compare the results obtained with those of the previous chapter. When the same parameters, boundary and initial conditions are applied, the result of figure 6 is obtained. Note again that the boundary condition on the right edge of the domain differs slightly from the analytical case, because it is not numerically possible to state a condition at an infinite spatial coordinate.
As expected, figure 6 is very much like the analytical solution presented in figure 2. It is possible to make slices in this plot, similar to the ones shown in figure 3. These slices are compared to the slices of the analytical solution in figure 7 for two different meshes.

Figure 6 Space-time approximation using 10x10 elements

Figure 7 Comparison of the analytical solution with approximations using two different meshes. One mesh consists of 4x4 equally distributed elements the other with 20x20 equally distributed elements.
As was already explained, a uniform mesh is not very efficient for the present problem. An elegant solution is to refine the mesh in the transition area. Such a solution is presented in the next paragraph.

### 3.2.4 Partially densified mesh

As already discussed, a partial densification in those areas of the domain, which are critical for accuracy, would have some distinct advantages from an efficiency point of view. An example of such a mesh is given in figure 8. Knowledge of the influence of the convective contribution is essential, as it has to be known at which speed the transition area moves, in order to apply the refinement in the appropriate region(s). However, one could imagine applying the refinement adaptively, e.g. based on an error estimate.

![Figure 8 Example of improved mesh](image)

When this mesh is applied to the same space-time approach, a new Matlab-file has to be constructed. This file is presented in Appendix C (section 7.3). Figure 9 shows the results obtained with the implementation of this improved mesh. It shows that the mesh of figure 8 is not yet optimal and uses a slightly too narrow refinement zone. However, this result clearly shows the intentions.
As can be seen in figure 9, the approximations in the transition area are more precise. This is exactly what is desired.

### 3.2.5 Comments

As expected, the space-time approach provides us with a flexible tool in the FEM-analysis. However, every method has its own drawbacks, and this is also the case for this method.

Comparing it to the usual numerical treatment, it can be seen that calculation time is significantly higher. This is due to the fact that in this method we once solve a “large” system of equations, whereas in the usual numerical treatment a smaller system of equations is solved repeatedly.

Furthermore, when the number of dimensions is increased, it becomes difficult to interpret and construct the analysis for the (n+1)-dimensional treatment.
4 Conclusion

In this report two methods have been studied to numerically solve a species of problems with temporal and spatial dependencies. For the sake of clarity a high level of simplification is applied.

The usual numerical treatment uses a discretisation in the spatial dimension(s), followed by a regular time integration scheme to account for the temporal dependency, whereas the (n+1)-dimensional space-time method unifies the treatment of the spatial and temporal dependency. In other words, the space-time method uses one discretisation for the time and spatial coordinates simultaneously, which need not be uniform in either coordinate.

From this study we can conclude, that for the specific problem studied, both the usual numerical treatment as well as the space-time finite element method are satisfactory concerning the solution approximation. The space-time method has some advantages regarding the algorithmics, where it unifies the treatment of spatial and temporal dependencies. However, it is more costly from a calculation point of view.

A partially refined mesh, refined at the part of the domain where the solution fluctuates strongly, has been constructed, and applicability has been demonstrated. As the convective contribution becomes more complex, i.e. time and/or space dependent velocity, the mesh-constructing becomes equally complex. Researching any possibilities regarding the feasibility of the method in approximating these more complex problems, is recommended.

In the following chapter more recommendations are presented concerning research in this field.
5 Recommendations

Some interesting suggestions to continue research in this field are mentioned here. The suggestions can be divided into three categories.

5.1 Multi-dimensional problems

It might be interesting to look further into (spatially) multi-dimensional problems. Because of the use of an (n+1)-dimensional approach, application of the space-time method to three-dimensional problems that also vary with time becomes rather tedious. However, the fact that four dimensions are difficult to physically interpret does not mean that it is impossible to approximate problems in four dimensions. There will be some difficulties in applying the method, but it should be possible.

5.2 Calculation optimization

This is of less importance to the fundamental research in this field, but as the number of dimensions increase, calculation time will increase exponentially. Therefore, optimizations obtained in the one-dimensional case, could have a larger absolute contribution in multi-dimensional cases.

As shown in paragraph 3.2.2 the matrices $K$ and $L$ consist of element matrices. These so called sparse matrices are treated in the implementation as regular full matrices, accounting for a large part of the calculation time. Using more advanced (e.g. sparse) solvers may significantly improve computing times here.

5.3 Applicability to fracture/damage mechanics

As this research is performed within the division of CEM at the technical university of Eindhoven, applicability to damage and fracture mechanics is to be researched. It is the goal to examine whether this technique can be applied in these specific problems. Another form of information, namely the history dependency, complicates the problem significantly. Further research should obtain results about the feasibility of application in this specific field.
6 References


7 Appendices

7.1 Appendix A Solution algorithm usual treatment

The file below is implemented in MatLab

% space discretization 1D with separate time integration scheme
clr

% To keep an eye on computing cost, we keep track of time
tic

tic

% parameters
p = 10; % Space elements
r = 10; % Number of time steps
B = 100; % Length in space(1-dimensional)
Time = 100; % Ending time
Ui = 0; % Initial temperature of the bar
deltat = Time/r; % Time step

% Boundary condition
ub = ones(r,1);

% Making the mesh and defining the number of nodes and elements
meshline(p,r,B,Time)
m = p*(r+1); % number of elements
n = (p+1)*(r+1); % number of nodes
% Now the physical mesh is finished and the next step is to define the master element,
% in this case the master element is a linear element.

% Master element
% C = CurveElement([ Node(Point2([-1 0]),1 )
%                    Node(Point2([ 1 0]),2 ) ])
% Ne = shapefunctions(C)
% gradxiNe = grad(Ne)
% xi = [ Point2([-1/sqrt(3) 0])
%       Point2([+1/sqrt(3) 0]) ];
% w = [1 1]';

% Because the paradox applications don't work in the above, an alternative way is used to
% define the shapefunctions and their derivatives

% parameters
lambda = 1;
v = 1;
c = 1;
% construct system matrices K
nh = p + 1;
h = B/p;
K = zeros(nh, nh);
M = zeros(nh, nh);
for e = 1:p

    xe1 = (e-1) * h;
    xe2 = e    * h;

    Ke = lambda/h * [1 -1; -1 1] - v*c/2 * [1 -1; 1 -1];
    Me = c*h/6 * [2 1; 1 2];

    K([e e+1], [e e+1]) = K([e e+1], [e e+1]) + Ke;
    M([e e+1], [e e+1]) = M([e e+1], [e e+1]) + Me;
end

K, M

% Now the solution algorithm in time has to be constructed

% Initial conditions
u = Ui*ones((p+1),1);
qf = zeros(p,1);        % the column q

% Partition system
iig = [1];
iif = [2:(p+1)];
Mfg = M(iif, iig);
Mff = M(iif, iif);
Kfg = K(iif, iig);
Kff = K(iif, iif);
ug = u(iig);
uf = u(iif);

% Solution algorithm
for f = 1:r
    ufnew = (Kff + (1/deltat)*Mff) \ (qf + (1/deltat)*Mfg*ug + (1/deltat)*Mff*uf -
                   (Kfg + (1/deltat)*Mfg)*ug);
    ugnew = ub(f);
    u = [u;ugnew;ufnew];
    uf = ufnew;
    ug = ugnew;
end
u
% Stopwatch stops, this one includes the calculation cost

% One way of solving the plot problem is to make an object with the grid function
% and then attaching the nodal values in u to the different nodes, in this case we use
% matrix to store all the nodal values, which is later plotted with surface-function.

x1 = [0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1];
U=(reshape(u,(r+1),(p+1)))'

figure(2)
surf(x1,x1,U)
title('Usual numerical treatment','Fontsize',14)
xlabel('Space coord. [-]','Fontsize',14)
ylabel('Time coord. [-]','Fontsize',14)
zlabel('Dimensionless temperature','Fontsize',14)
axis square
axis([0 1 0 1 -.1 1.1])
7.2 Appendix B  Solution algorithm space-time method
This ML-file shows the entire implementation in MatLab for the “new” space-time finite element approximation in one space dimension.

% Space and time discretisation

tic  % record time it takes to execute

% construct domain and field
clr

% gridformation
p = 10;
r = 10;
L = 100;
Time = 100;

% Boundary condition
G=ones(r,1);

% construct domain and field
V = Rect([0 L], [0 Time]);
Vh = grid(V, p, r);
plot(Vh)
drawnow
m = numberOfElements(Vh)
n = numberOfNodes(Vh)

% define parameters
lambda = 1;  % convection coefficient
c = 1;
v = 1;
Ui = 0;  % initial temperature of the bar

% master element and Gauss points
Q = Quad( [ Node(Point2([-1 -1]), 1)
            Node(Point2([+1 -1]), 2)
            Node(Point2([+1 +1]), 3)
            Node(Point2([-1 +1]), 4) ] );

Ne = shapeFunctions(Q);
gradiNe = grad(Ne);

xi = [ Point2([-1/sqrt(3) -1/sqrt(3)]
            Point2([+1/sqrt(3) -1/sqrt(3)])
            Point2([+1/sqrt(3) +1/sqrt(3)])
            Point2([-1/sqrt(3) +1/sqrt(3)]) ]);

w = [1;1;1;1];
% build stiffness matrix
K = zeros(n, n);
L = zeros(n, n);
C1= Tensor22([-lambda 0;0 0]);
c2= Vector2([c*v c]);
for e = 1:m

xe = posVector( nodes(element(Vh, e)) );
x = Ne' * xe;
J = dyadicProduct(gradxiNe', xe);
gradNe = inv(J) * gradxiNe;

Ke = zeros(4, 4);
Le = zeros(4, 4);
for k = 1:4
    Ke = Ke + w(k) * feval( gradNe * C1 * gradNe' * det(J) , xi(k) );
    Le = Le + w(k) * feval( Ne' * c2 * gradNe' * det(J) , xi(k) );
end

% assembly
ii = nodeTags(element(Vh, e));
K(ii, ii) = K(ii, ii) + Ke;
L(ii, ii) = L(ii, ii) + Le;
end

% Define replacement matrix
M = K - L;

% partition system
Qg1 = [];
for i = 1 : p
    qg1 = (p+1)*i + 1;
    Qg1 = [ Qg1 qg1 ];
end
iig = [1:(p+1) Qg1];

Qf1 = [];
for i = 1 : p
    qf1 = [ i*(p+1)+2:i*(p+1) + p + 1] ;
    Qf1 = [ Qf1 qf1 ];
end
iif = Qf1;

Mgg = M(iig, iig);
Mfg = M(ii, iig);
Mgf = M(iig, iif);
Mff = M(iif, iif);

ug = zeros(size(iig));
ug(1:(p+1)) = Ui;
ug((p+2):(p + length(G) + 1)) = G;
ug((p + 2 + length(G)):length(iig)) = Ui;
ug

qf = zeros(length(iiif), 1);

L1=length(qf)

% solve for unknowns
uf = Mff \ (qf - Mfg*(ug'));

u = zeros(n, 1);
u(iig) = ug;
u(iiif) = uf;
u

x1 = [0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1];
U=(reshape(u,(p+1),(r+1)))';

t=toc

figure(2)
surf(x1,x1,U)
title('Stationary initial value problem','Fontsize',14)
xlabel('Space coord. [-]','Fontsize',14)
ylabel('Time coord. [-]','Fontsize',14)
zlabel('Dimensionless temperature','Fontsize',14)
axis square
axis([-1 1 1 1])
7.3 Appendix C  Partially densified mesh for space-time

In this file a constructed, more efficient mesh is implemented in MatLab.

% This file will create a mesh with extra elements on the diagonal

% Parameters
L = 100;    % space
Time = 100;    % time

% plot geometry
clf reset
plot([0 L L 0 0], [0 0 Time Time 0], 'k--')
axis equal
hold on
drawnow

% The array of coordinates, as used in MARC-MENTAT
C1 = 100*[0.000e+0 0.000e+0 0.000e+0
5.000e-2 0.000e+0
1.000e-1 0.000e+0
2.000e-1 0.000e+0
4.000e-1 0.000e+0
6.000e-1 0.000e+0
8.000e-1 0.000e+0
1.000e+0 0.000e+0
0.000e+0 5.000e-2
5.000e-2 5.000e-2
1.000e-1 5.000e-2
1.500e-1 5.000e-2
0.000e+0 1.000e-1
5.000e-2 1.000e-1
1.000e-1 1.000e-1
1.500e-1 1.000e-1
2.000e-1 1.000e-1
3.000e-1 1.000e-1
5.000e-2 1.500e-1
1.000e-1 1.500e-1
1.500e-1 1.500e-1
2.000e-1 1.500e-1
2.500e-1 1.500e-1
0.000e+0 2.000e-1
1.000e-1 2.000e-1
1.500e-1 2.000e-1
2.000e-1 2.000e-1
2.500e-1 2.000e-1
3.000e-1 2.000e-1
4.000e-1 2.000e-1
6.000e-1 2.000e-1
8.000e-1 2.000e-1
<p>| 1.000e+0 | 2.000e-1 |
| 1.500e-1 | 2.500e-1 |
| 2.000e-1 | 2.500e-1 |
| 2.500e-1 | 2.500e-1 |
| 3.000e-1 | 2.500e-1 |
| 3.500e-1 | 2.500e-1 |
| 1.000e-1 | 3.000e-1 |
| 2.000e-1 | 3.000e-1 |
| 2.500e-1 | 3.000e-1 |
| 3.000e-1 | 3.000e-1 |
| 3.500e-1 | 3.000e-1 |
| 4.000e-1 | 3.000e-1 |
| 5.000e-1 | 3.000e-1 |
| 2.500e-1 | 3.500e-1 |
| 3.000e-1 | 3.500e-1 |
| 3.500e-1 | 3.500e-1 |
| 4.000e-1 | 3.500e-1 |
| 4.500e-1 | 3.500e-1 |
| 0.000e+0 | 4.000e-1 |
| 2.000e-1 | 4.000e-1 |
| 3.000e-1 | 4.000e-1 |
| 3.500e-1 | 4.000e-1 |
| 4.000e-1 | 4.000e-1 |
| 4.500e-1 | 4.000e-1 |
| 5.000e-1 | 4.000e-1 |
| 6.000e-1 | 4.000e-1 |
| 8.000e-1 | 4.000e-1 |
| 1.000e+0 | 4.000e-1 |
| 3.500e-1 | 4.500e-1 |
| 4.000e-1 | 4.500e-1 |
| 4.500e-1 | 4.500e-1 |
| 5.000e-1 | 4.500e-1 |
| 5.500e-1 | 4.500e-1 |
| 3.000e-1 | 5.000e-1 |
| 4.000e-1 | 5.000e-1 |
| 4.500e-1 | 5.000e-1 |
| 5.000e-1 | 5.000e-1 |
| 5.500e-1 | 5.000e-1 |
| 6.000e-1 | 5.000e-1 |
| 7.000e-1 | 5.000e-1 |
| 4.500e-1 | 5.500e-1 |
| 5.000e-1 | 5.500e-1 |
| 5.500e-1 | 5.500e-1 |
| 6.000e-1 | 5.500e-1 |
| 6.500e-1 | 5.500e-1 |
| 0.000e+0 | 6.000e-1 |
| 2.000e-1 | 6.000e-1 |
| 4.000e-1 | 6.000e-1 |
| 5.000e-1 | 6.000e-1 |
| 5.500e-1 | 6.000e-1 |</p>
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.000e-1</td>
<td>6.000e-1</td>
</tr>
<tr>
<td>6.500e-1</td>
<td>6.000e-1</td>
</tr>
<tr>
<td>7.000e-1</td>
<td>6.000e-1</td>
</tr>
<tr>
<td>8.000e-1</td>
<td>6.000e-1</td>
</tr>
<tr>
<td>1.000e+0</td>
<td>6.000e-1</td>
</tr>
<tr>
<td>5.500e-1</td>
<td>6.500e-1</td>
</tr>
<tr>
<td>6.000e-1</td>
<td>6.500e-1</td>
</tr>
<tr>
<td>6.500e-1</td>
<td>6.500e-1</td>
</tr>
<tr>
<td>7.000e-1</td>
<td>6.500e-1</td>
</tr>
<tr>
<td>7.500e-1</td>
<td>6.500e-1</td>
</tr>
<tr>
<td>5.000e-1</td>
<td>7.000e-1</td>
</tr>
<tr>
<td>6.000e-1</td>
<td>7.000e-1</td>
</tr>
<tr>
<td>6.500e-1</td>
<td>7.000e-1</td>
</tr>
<tr>
<td>7.000e-1</td>
<td>7.000e-1</td>
</tr>
<tr>
<td>7.500e-1</td>
<td>7.000e-1</td>
</tr>
<tr>
<td>8.000e-1</td>
<td>7.000e-1</td>
</tr>
<tr>
<td>8.500e-1</td>
<td>7.500e-1</td>
</tr>
<tr>
<td>9.000e-1</td>
<td>7.500e-1</td>
</tr>
<tr>
<td>7.000e-1</td>
<td>7.500e-1</td>
</tr>
<tr>
<td>7.500e-1</td>
<td>7.500e-1</td>
</tr>
<tr>
<td>8.000e-1</td>
<td>7.500e-1</td>
</tr>
<tr>
<td>8.500e-1</td>
<td>7.500e-1</td>
</tr>
<tr>
<td>9.000e-1</td>
<td>7.500e-1</td>
</tr>
<tr>
<td>1.000e+0</td>
<td>8.000e-1</td>
</tr>
<tr>
<td>2.000e-1</td>
<td>8.000e-1</td>
</tr>
<tr>
<td>4.000e-1</td>
<td>8.000e-1</td>
</tr>
<tr>
<td>6.000e-1</td>
<td>8.000e-1</td>
</tr>
<tr>
<td>7.000e-1</td>
<td>8.000e-1</td>
</tr>
<tr>
<td>7.500e-1</td>
<td>8.000e-1</td>
</tr>
<tr>
<td>8.000e-1</td>
<td>8.000e-1</td>
</tr>
<tr>
<td>8.500e-1</td>
<td>8.000e-1</td>
</tr>
<tr>
<td>9.000e-1</td>
<td>8.000e-1</td>
</tr>
<tr>
<td>1.000e+0</td>
<td>8.000e-1</td>
</tr>
<tr>
<td>7.500e-1</td>
<td>8.500e-1</td>
</tr>
<tr>
<td>8.000e-1</td>
<td>8.500e-1</td>
</tr>
<tr>
<td>8.500e-1</td>
<td>8.500e-1</td>
</tr>
<tr>
<td>9.000e-1</td>
<td>8.500e-1</td>
</tr>
<tr>
<td>9.500e-1</td>
<td>8.500e-1</td>
</tr>
<tr>
<td>7.000e-1</td>
<td>9.000e-1</td>
</tr>
<tr>
<td>8.000e-1</td>
<td>9.000e-1</td>
</tr>
<tr>
<td>8.500e-1</td>
<td>9.000e-1</td>
</tr>
<tr>
<td>9.000e-1</td>
<td>9.000e-1</td>
</tr>
<tr>
<td>9.500e-1</td>
<td>9.000e-1</td>
</tr>
<tr>
<td>1.000e+0</td>
<td>9.000e-1</td>
</tr>
<tr>
<td>8.500e-1</td>
<td>9.500e-1</td>
</tr>
<tr>
<td>9.000e-1</td>
<td>9.500e-1</td>
</tr>
<tr>
<td>9.500e-1</td>
<td>9.500e-1</td>
</tr>
<tr>
<td>1.000e+0</td>
<td>9.500e-1</td>
</tr>
<tr>
<td>0.000e+0</td>
<td>1.000e+0</td>
</tr>
<tr>
<td>2.000e-1</td>
<td>1.000e+0</td>
</tr>
<tr>
<td>4.000e-1</td>
<td>1.000e+0</td>
</tr>
</tbody>
</table>
% Generate nodes
N = Node(Point2);
for i = 1:137
    N(i,1) = Node( Point2(C1(i,:)), i);
end
n = i;

% plot nodes
for i = 1 : n
    x = coordinates(N(i));
    plot(x(1), x(2), 'b.', 'MarkerSize', 8)
    text(x(1), x(2), sprintf(' %d', i), 'Color', 'b', 'FontSize', 8, ...
         'HorizontalAlignment', 'left', 'VerticalAlignment', 'bottom')
end
drawnow

% Generating the elements using the connectivity extracted from MARC-MENTAT
% First the connectivity array is inserted, each row presents the node-numbers with
% which one Quad-element is formed

C2 = [1 2 10 9
      2 3 11 10
      11 3 4 12
      17 4 5 18
      17 16 12 4
      5 6 31 30
      30 29 18 5
      6 7 32 31
      7 8 33 32
      9 10 14 13
     10 11 15 14
     15 11 12 16
     13 14 19 24
     15 20 19 14
     15 16 21 20
     16 17 22 21
     22 17 18 23
     29 28 23 18
     20 25 24 19
     20 21 26 25
     21 22 27 26
     27 22 23 28
     24 25 39 51
     25 26 34 39]
% Generate elements

78  79  106  105
79  80  107  106
80  81  93  107
81  82  88  93
83  89  88  82
83  84  90  89
84  85  91  90
91  85  86  92
98  97  92  86
98  86  87  99
114  113  99  87
89  94  93  88
89  90  95  94
90  91  96  95
96  91  92  97
94  108  107  93
94  95  100  108
96  101  100  95
96  97  102  101
97  98  103  102
103  98  99  104
113  112  104  99
101  109  108  100
101  102  110  109
102  103  111  110
111  103  104  112
105  106  131  130
106  107  132  131
107  108  133  132
108  109  120  133
109  110  115  120
111  116  115  110
111  112  117  116
112  113  118  117
118  113  114  119
125  124  119  114
116  121  120  115
116  117  122  121
117  118  123  122
123  118  119  124
121  134  133  120
121  122  126  134
123  127  126  122
123  124  128  127
124  125  129  128
127  135  134  126
127  128  136  135
128  129  137  136);
E = Quad(N(1:4));

for e = 1 : 122
    E(e) = Quad(N(C2(e,:)));  
end 

m = e;

% plot elements 
for e = 1:m 
    xe = coordinates(nodes(E(e)));  
    plot(xe([1:4 1], 1), xe([1:4 1], 2), 'r-')  
    x = mean(xe([1:4 , :]));  
    text(x(1), x(2), sprintf('%d', e), 'Color', 'r', 'FontSize', 8, ...  
        'HorizontalAlignment', 'center', 'VerticalAlignment', 'middle')  
end

drawnow 

% Construct the mesh 
Vh = Mesh;  
Vh = insert(Vh, N);  
Vh = insert(Vh, E);  

% indices of boundary nodes 
ii_bottom = [1:8];  
ii_left = [9 13 24 51 78 105 130];

% Normal n+1 dimensional treatment as seen before 

% define parameters 
lambda = 1;  
% convection coefficient 
c = 1;  
v = 1;  
Ui = 0;  
% initial temperature of the bar 
G = 1;  
% boundary condition 

% master element and Gauss points
Q = Quad( [ Node(Point2([-1 -1]), 1)  
            Node(Point2([+1 -1]), 2)  
            Node(Point2([+1 +1]), 3)  
            Node(Point2([-1 +1]), 4) ] );

Ne = shapeFunctions(Q);  
gradxiNe = grad(Ne);

xi = [ Point2([-1/sqrt(3) -1/sqrt(3)])  
        Point2([+1/sqrt(3) -1/sqrt(3)])  
        Point2([+1/sqrt(3) +1/sqrt(3)])  
        Point2([-1/sqrt(3) +1/sqrt(3)]) ];
w = [1;1;1;1];

% build stiffness matrix
K = zeros(n, n);
L = zeros(n, n);
C1 = Tensor22([-lambda 0;0 0]);
c2 = Vector2([c*v c]);
for e = 1:m

xe = posVector( nodes(element(Vh, e)));
x = Ne' * xe;
J = dyadicProduct(gradxiNe', xe);
gradNe = inv(J) * gradxiNe;

Ke = zeros(4, 4);
Le = zeros(4, 4);
for k = 1:4
    Ke = Ke + w(k) * feval( gradNe * C1 * gradNe' * det(J), xi(k) );
    Le = Le + w(k) * feval( Ne * c2 * gradNe' * det(J), xi(k) );
end

% assembly
ii = nodeTags(element(Vh, e));
K(ii, ii) = K(ii, ii) + Ke;
L(ii, ii) = L(ii, ii) + Le;

end

% Define replacement matrix
M = K - L;

% partition system
iig = [ii_bottom ii_left];

Mgg = M(iig, iig);
Mfg = M(iif, iig);
Mgf = M(iig, iif);
Mff = M(iif, iif);

ug = zeros(15,1);
ug(1:8,1) = Ui;
ug(9:15,1) = G;
ug

qf = zeros(length(iif), 1);
lqf = length(qf);
lug = length(ug)
% solve for unknowns
uf = Mff \ (qf - Mfg*(ug));
u = zeros(n, 1);
u(iig) = ug;
u(iif) = uf;
u

figure(2)
Uh = ScalarField(Vh);
Uh(nodes(Vh)) = u;
plot(Uh)
title('Diagonally densified mesh','Fontsize',14)
xlabel('x-coordinate [-]','Fontsize',14)
ylabel('time coordinate [-]','Fontsize',14)
zlabel('Dimensionless temperature distribution','Fontsize',14)
axis square