DELFT UNIVERSITY OF TECHNOLOGY

REPORT 06-12

Towards solving a moving boundary problem within a multi-layered recording stack

J.H. Brusche, A. Segal, C. Vuik

ISSN 1389-6520

Reports of the Delft Institute of Applied Mathematics

Delft 2006

Towards solving a moving boundary problem within a multilayered recording stack

J.H. Brusche, A. Segal, C. Vuik[1]

August 24, 2006

[1]Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft Institute of Applied Mathematics, P.O. Box 5031, 2600 GA Delft, The Netherlands, phone: 31 15 2787608, fax: 31 15 2787209 (`j.h.brusche@tudelft.nl`)

Contents

1 Introduction **2**
1.1 Optical rewritable recording . . . 2
1.2 Mark formation modeling . . . 2
1.3 Mesh refinement . . . 3
1.4 Numerical results . . . 3

2 Problem description **5**
2.1 The Stefan problem . . . 5
2.2 Two approaches to the Stefan problem . . . 6

3 The enthalpy approach **7**
3.1 The Kirchoff transformation . . . 7
3.2 Nonlinear SOR . . . 7
3.2.1 Method evaluation . . . 8
3.3 Pseudo-Newton method . . . 9
3.3.1 Method description . . . 9
3.3.2 Method evaluation . . . 10

4 The temperature approach **11**
4.1 Method description . . . 11
4.2 Numerical integration . . . 12
4.3 Solving the nonlinear system . . . 15
4.4 The addition of a source term to an isothermal problem . . . 18
4.5 Method evaluation . . . 19

5 Adaptive local mesh refinement **21**
5.1 Introduction . . . 21
5.2 Basic terminology . . . 21
5.3 Local refinement strategies . . . 22
5.4 Implementation of the local mesh refinement . . . 26
5.4.1 Marking elements for refinement . . . 27
5.4.2 Construction of the refined mesh . . . 29
5.4.3 Mapping of the data vectors . . . 32
5.4.4 Adaptation of the time step size . . . 33
5.4.5 Interface outside refinement . . . 34
5.5 Conclusions . . . 37

6 Numerical results **39**
6.1 Introduction . . . 39
6.2 Comparison: Fachinotti vs Nedjar . . . 39
6.2.1 Validity of the comparison . . . 40
6.2.2 Test case 1: 'equal' . . . 40

6.2.3 Test case 2: ’unequal’ . . . 41
6.2.4 Test case 3: ’2D corner’ . . . 42
6.2.5 Nedjar: Newton-Cotes vs Gauss rule . . . 43
6.2.6 Conclusions . . . 43
6.3 Temperature approach . . . 44
6.3.1 Line search . . . 44
6.3.2 Artificial mushy region . . . 45
6.3.3 Adaptive local mesh refinement . . . 47

Chapter 1

Introduction

In this study we report on the research that has been performed in developing a finite element model (FEM) for optical rewritable recording. We will start by giving some background information on optical rewritable recording. Then, a brief overview of the contents of this document is presented.

1.1 Optical rewritable recording

In optical rewritable recording, a disk consists of various layers. The actual recording of data, stored as an array of amorphous regions in a crystalline background, takes place in a specific layer of the recording stack. This layer consists of a so-called phase-change material. The amorphous regions, called marks, are created as a result of very short high intensity pulses of a laser beam that is focused on this active layer. The light energy of the laser is transformed into heat, which locally causes the phase-change material to melt. As soon as the laser is switched off, the molten material solidifies. At the same time, recrystallization occurs in those regions where the temperature is below the melting temperature, but still above the recrystallization temperature. Since the cooling down is very rapid (quenching), almost no recrystallization occurs within the molten region, and thus a solid amorphous region is formed. The same laser beam, but at a lower power level, can be used in a similar way to fully recrystallize the amorphous regions. The recorded data is then erased.

To ensure that the optical system knows its exact location on the disk at all time, a rewritable disk contains a concentric, outwards spiraling groove. Modulations in the refracted light, as a result of the grooves, are processed to position the optical head (tracking). In radial direction, a periodic 'land and groove' structure can thus be discerned. For a Blu-ray recording stack, a typical configuration is depicted in Figure 1.1.

Although much is understood about the concept of optical rewritable recording, many open questions remain. In order to gain better inside in for instance the influence of polarization and wavelength of the incident light or the geometry and composition of the stack on the shape and position of a mark, robust (numerical) modeling is essential. As a result, the occurrence of unwanted effects, such as cross-track cross-talk, can be minimized.

1.2 Mark formation modeling

In this report we will focus on modeling the melting phase of the mark formation process. In contrast to earlier work [3], this requires the contribution of latent heat to be taken into account in the computation of the temperature distribution in the optical recording disk. Latent heat is the amount of heat that is required to produce a phase transition.

The mark formation can be described as a moving boundary problem. Or more precisely, a two-phase Stefan problem, see Chapter 2. A variety of numerical techniques can be found in the

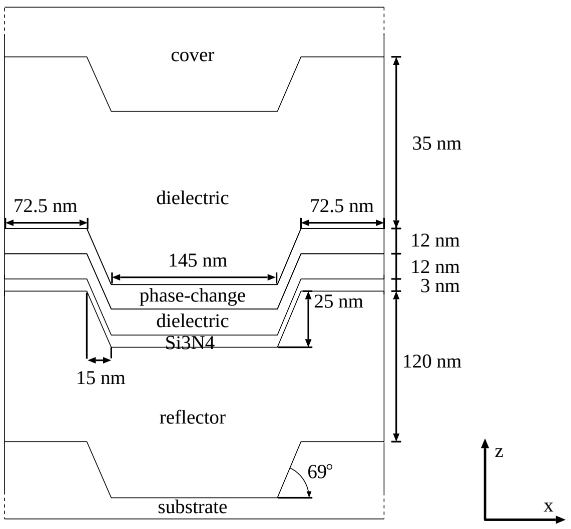


Figure 1.1: Sketch of a typical recording stack for a Blu-ray disk.

literature that can be used to solve such Stefan problems. An elaborate overview of these methods is presented by Brusche [2]. The author concludes that the most suitable numerical approach to solve this particular Stefan problem is an enthalpy method.

Each of the enthalpy based methods discussed in [2] is applicable to domains containing a single material which (partly) solidifies/melts, such as the phase-change layer of a rewritable disk. However, in case of a multi-layered stack we are also interested in the temperature distribution outside the phase-change layer. This extra requirement severely limits the choice of a suitable method.

In Sections 3.2 and 3.3, two enthalpy methods to solve the two-phase Stefan problem are discussed in more detail: the nonlinear Elliott-Ockendon SOR scheme and the relaxed linearized pseudo-Newton method by Nedjar (see [2]). In addition to these enthalpy approaches, in Section 4.1 a temperature based technique is introduced, adopted from Fachinotti et al., in which the temperature is the single dependent variable.

1.3 Mesh refinement

Clearly, a refinement of the computational grid (in both space and time) is one of the most obvious ways to increase the accuracy of the numerically approximated solution. Since globally refining the spatial mesh is highly unfavorable with respect to computational performance and system requirements, a local refinement strategy is desirable. In particular, one would prefer to refine the mesh in a neighborhood of the moving interface, because in regions away from the moving boundary only small fluctuations in the gradient of the solution are to be expected (i.e., the solution is relatively smooth). Of course, controlling the time step size is of great importance as well.

In Chapter 5 we present an adaptive local mesh refinement procedure which has been developed in order to improve the efficiency and accuracy of the mark formation model. The implementation of the code will be discussed in detail.

1.4 Numerical results

In Chapter 6 first a comparison is made between the methods of Nedjar and Fachinotti et al. Of special interest are the differences with respect to the quality of the solution (accuracy/wiggles) and performance (computational time/number of iterations).

Then, we will focus on the Fachinotti method and present some results for selected test problems with and without a source term. In conclusion, results are presented for the application of the adaptive local grid refinement.

Chapter 2

Problem description

The melting of a phase-change material is a complex process. Material specific properties, such as the latent heat, greatly influence the melting behavior. Therefore, some assumptions are made with respect to several physical aspects of the melting process. First of all, it is assumed that the melting of the phase-change material occurs at a melting *point* T_m, rather than along a melting trajectory. In this way, the shape and size of mark are simply determined by the (sharp) moving interface between the solid and liquid state of the (crystalline) phase-change material. Furthermore, the density ρ, latent heat L, heat capacity c, and conductivity κ are taken to be constant per phase. When needed, a subscript s or l is used to distinct between the solid and liquid state, respectively.

2.1 The Stefan problem

Because the position of the moving interface evolves in time, depending on the heat distribution, the melting process is modeled as a moving boundary problem. For an arbitrary domain $\Omega \subset \mathbb{R}^n$ with fixed outer boundary $\delta\Omega$ and moving boundary $\Gamma(t)$, leading to two sub-domains Ω_s and Ω_l such that $\bar{\Omega} = \bar{\Omega}_s \cup \bar{\Omega}_l$ and $\Omega_s \cap \Omega_l = \emptyset$, the two-phase Stefan problem is given by:

$$\begin{cases} \rho c_{s,l} \dfrac{\partial T(\boldsymbol{x},t)}{\partial t} = \kappa_{s,l} \Delta T(\boldsymbol{x},t) + q(\boldsymbol{x},t) & \forall \boldsymbol{x} \in \Omega_{s,l}, t > 0 \qquad (2.1a) \\ \rho L v_n = \left[\kappa_{s,l} \dfrac{\partial T(\boldsymbol{x},t)}{\partial \boldsymbol{n}} \right], \quad T(\boldsymbol{x},t) = T_m & \text{for } \boldsymbol{x} = \Gamma(t), t \geq 0 \qquad (2.1b) \\ T(\boldsymbol{x},0) = \bar{T}_1(\boldsymbol{x}) & \forall \boldsymbol{x} \in \Omega_{s,l} \qquad (2.1c) \end{cases}$$

where $\boldsymbol{n}$ denotes the unit normal vector on the moving interface pointing from the solid domain into the liquid domain, and v_n is the velocity of the moving interface, together with one or more of the following boundary conditions on the complementary parts $\delta\Omega_i, i = 1, 2, 3$ of the fixed outer boundary $\delta\Omega = \bigcup_{i=1}^{3} \delta\Omega_i$:

1. A Dirichlet condition on $\delta\Omega_1$:
$$T = \bar{T}_2(\boldsymbol{x}). \qquad (2.2)$$

2. A Neumann condition on $\delta\Omega_2$:
$$\kappa_{s,l} \frac{\partial T}{\partial \boldsymbol{n}}(\boldsymbol{x}) = \bar{q}(\boldsymbol{x}), \qquad (2.3)$$
where $\bar{q}(\boldsymbol{x})$ is a given normal heat flux.

3. A radiation-type boundary condition on $\delta\Omega_3$:
$$\kappa_{s,l} \frac{\partial T}{\partial \boldsymbol{n}}(\boldsymbol{x}) = \bar{\alpha}(T), \qquad (2.4)$$
where $\bar{\alpha}(T)$ is a function of temperature.

At $t = 0$ the whole domain is taken to be solid. By $[\phi]$ we denote the jump in ϕ defined as:

$$[\phi] = \lim_{\substack{\boldsymbol{x} \longrightarrow \Gamma(t) \\ \boldsymbol{x} \in \Omega_s(t)}} \phi(\boldsymbol{x}, t) - \lim_{\substack{\boldsymbol{x} \longrightarrow \Gamma(t) \\ \boldsymbol{x} \in \Omega_l(t)}} \phi(\boldsymbol{x}, t). \tag{2.5}$$

2.2 Two approaches to the Stefan problem

Two fixed grid approaches to solve the Stefan problem given above can be distinguished: the *enthalpy* formulation and the *temperature* formulation. Under the above-mentioned assumptions, the enthalpy $H(T)$ can be defined as:

$$H(T) = \begin{cases} \rho c_s (T - T_m), & T \leq T_m \\ \rho c_l (T - T_m) + \rho L, & T > T_m \end{cases} \tag{2.6}$$

In the enthalpy formulation the enthalpy H is treated as a second dependent variable besides the temperature T. Using relation (2.6), the heat conduction equation (2.1a) and the Stefan condition (2.1b) are replaced by the well-known enthalpy equation:

$$\frac{\partial H(T)}{\partial t} - \kappa_{s,l} \Delta T = q. \tag{2.7}$$

In the temperature formulation, instead of separating the domain in a liquid and solid part, as via definition (2.6), the enthalpy is written according to its formal definition: as the sum of sensible and latent heat:

$$H(T) = H^{\text{sensible}} + H^{\text{latent}} = \rho c_{s,l} (T - T_m) + \rho L f_l(T), \tag{2.8}$$

where $f_l(T)$ denotes the liquid volume fraction, which in case of isothermal phase-change is equal to the Heavyside step function $\mathcal{H}(T - T_m)$. Definition (2.8) leads to the classical Fourier heat conduction equation, but with an additional term for the latent heat contribution:

$$\rho c_{s,l} \frac{\partial T}{\partial t} + \rho L \frac{\partial f_l}{\partial t} - \kappa_{s,l} \Delta T = q. \tag{2.9}$$

Here, the time derivative of the liquid volume fraction is to be interpreted in the weak sense.

Chapter 3

The enthalpy approach

In this chapter we will discuss two numerical methods for solving the Stefan problem (2.1a-2.1c) that use the enthalpy approach, described in Section 2.2. An SOR scheme, modified for nonlinear systems, is discussed in Section 3.2 and a pseudo-Newton scheme is derived in Section 3.3.

3.1 The Kirchoff transformation

A well known transformation in the literature on enthalpy problems, is the Kirchoff transformation. The main reason for applying this transformation, is that it eliminates the nonlinearity in the diffusion term. As such, in particular when an implicit scheme is used, a gain in efficiency can be obtained, due to faster convergence of the employed iterative scheme. Alexiades and Solomon [1], Chapter 4.3, even emphasize that, specifically when the conductivities of the phases are unequal, and when they are functions of the temperature only and not of position, the Kirchoff method should be used.

The normalized Kirchoff transformed temperature is defined as

$$\tilde{T} := \int_{T_m}^{T} \kappa(\tau)d\tau = \begin{cases} \int_{T_m}^{T} \kappa_s(\tau)d\tau & T < T_m, \\ \int_{T_m}^{T} \kappa_s(\tau)d\tau & T < T_m. \end{cases} \tag{3.1}$$

For constant κ_s, κ_l, we have

$$\tilde{T} = \begin{cases} \kappa_s(T - T_m) & T < T_m \\ 0 & T = T_m \\ \kappa_l(T - T_m) & T > T_m \end{cases}. \tag{3.2}$$

The corresponding enthalpy is

$$\tilde{H} = \begin{cases} \frac{\rho c_s \tilde{T}}{\kappa_s} & \tilde{T} \leq 0 \\ \frac{\rho c_l \tilde{T}}{\kappa_l} + \rho L & \tilde{T} > 0 \end{cases}. \tag{3.3}$$

3.2 Nonlinear SOR

Application of the standard Galerkin FEM to the enthalpy equation (2.7), in combination with one or more of the boundary conditions (2.2)-(2.4), leads to an equivalent system of (nonlinear) equations. In matrix-vector notation this system is given by:

$$M\frac{\partial \tilde{\mathbf{H}}}{\partial t} + S\tilde{\mathbf{T}} = \mathbf{q}, \tag{3.4}$$

where M is the mass matrix and S is the stiffness matrix.

In case Euler backward is applied for the time integration, with a time step size Δt, the fully discretized system at time level $m+1$ is given by:

$$M\frac{\tilde{\mathbf{H}}^{m+1}-\tilde{\mathbf{H}}^{m}}{\Delta t}+S\tilde{\mathbf{T}}^{m+1}=\mathbf{q}^{m+1}. \tag{3.5}$$

Remark that, for simplicity, the time-dependency of the stiffness matrix, which is not constant because the position of the moving interface determines the value of the diffusivity κ for each node at each time level, has been omitted. The mass matrix is always constant, since the conductivity c is contained in the enthalpy.

If we let $\bar{M}$ be the lumped version of the mass matrix M, D the diagonal of the stiffness matrix S, and $\bar{S}$ a matrix containing the off-diagonals of S, (3.5) can be rewritten as:

$$\tilde{\mathbf{H}}^{m+1}+\Delta t\bar{M}^{-1}D\tilde{\mathbf{T}}^{m+1}=\Delta t\bar{M}^{-1}\mathbf{q}^{m+1}-\Delta t\bar{M}^{-1}\bar{S}\tilde{\mathbf{T}}^{m+1}+\tilde{\mathbf{H}}^{m}, \tag{3.6}$$

or

$$\tilde{\mathbf{H}}^{m+1}+Z\tilde{\mathbf{T}}^{m+1}=\mathbf{z}, \tag{3.7}$$

where

$$Z=\Delta t\bar{M}^{-1}D, \tag{3.8}$$
$$\mathbf{z}=\Delta t\bar{M}^{-1}\mathbf{q}^{m+1}-\Delta t\bar{M}^{-1}\bar{S}\tilde{\mathbf{T}}^{m+1}+\tilde{\mathbf{H}}^{m}. \tag{3.9}$$

By denoting the iteration number with the superscript (p) this system of equations can be reduced to (Gauss-Seidel):

$$H_j^{(p+1)}+Z_jT_j^{(p+1)}=z_j^{(p)},\quad j=1,\dots,n \tag{3.10}$$

The Elliott-Ockendon SOR iteration process is then given by, see [4]:

1. Compute Z_j and $z_j^{(p)}$.

2. Compute $\tilde{T}_j^{(p+1)}$ from (3.10), using definition (3.3) for the enthalpy:

$$\tilde{T}_j^{(p+1)}=\begin{cases}\frac{z_j^{(p)}}{\rho c_s+Z_j} & z_j^{(p)}\le 0,\\ 0 & 0<z_j^{(p)}<\rho L,\\ \frac{z_j^{(p)}-\rho L}{\rho c_l+Z_j} & z_j^{(p)}\ge \rho L,\end{cases} \tag{3.11}$$

3. Set $\hat{T}_j^{(p+1)}=T_j^{(p)}+\omega[\tilde{T}_j^{(p+1)}-T_j^{(p)}]$ (Over-relaxation).

4. Set
$$T_j^{(p+1)}=\begin{cases}\hat{T}_j^{(p+1)} & \text{if } \hat{T}_j^{(p+1)}\cdot T_j^{(p)}>0,\\ \tilde{T}_j^{(p+1)} & \text{if } \tilde{T}_j^{(p+1)}\cdot T_j^{(p)}\le 0,\end{cases} \tag{3.12}$$
that is, only over-relax the nodes that have not just changed phase.

5. If a convergence criterion, say $\|T_j^{(p+1)}-T_j^{(p)}\|<\textit{tolerance}$, is satisfied, then set $T_j^{m+1}=T_j^{(p+1)}$ and $H_j^{m+1}=z_j^{(p)}-Z_j\cdot T_j^{m+1}$.

3.2.1 Method evaluation

With respect to applying the above technique to obtain the temperature distribution in a multi-layered geometry, the following theoretical and practical difficulties arise:

1. In Example 2 of Section 4.3.5 of [2] it is illustrated how in case of a 1D slab consisting of 2 layers, the enthalpy method can be applied. The key idea is to consider the interface between the two layers as an 'interface' that moves at zero speed. The enthalpy function is now not only dependent on the temperature, but also on position. For the given 1D example, the Ockendon-Elliott SOR method can quite easily be implemented. However, in case of 2D or 3D, it becomes increasingly more difficult (non-smooth edges, corners etc..).
2. The Ockendon-Elliott algorithm is nodal-point based. Especially in case of finite elements, when in particular the enthalpy might be multi-valued in nodal points along the (free) boundaries, it is not clear whether the method might break down or not.
3. The physical parameters and the source term are required in the solver. In a general finite element package implementation, such as for instance SEPRAN [11], this is quite impractical.
4. There is no solid theoretical basis on which to choose the relaxation parameter ω.

3.3 Pseudo-Newton method

In [8] it is described how the fully discretized system (3.5) can be solved using a pseudo-Newton iterative procedure in terms of a temperature increment $\Delta\tilde{\mathbf{T}}_i$. Before we explain how this technique can be adapted to solve for the temperature distribution in a multi-layered domain, we will first briefly repeat the three steps that form the key idea behind the proposed integration algorithm by Nedjar [8]. We conclude this section with an evaluation of the presented method with respect to the mark formation model.

3.3.1 Method description

First, introduce the reciprocal function $\tau : \mathbb{R} \longrightarrow \mathbb{R}$ of (2.6), given by $\tilde{\mathbf{T}} = \tau(\tilde{\mathbf{H}})$ and define

$$\tilde{\mathbf{H}}_{i+1} = \tilde{\mathbf{H}}_i + \Delta\tilde{\mathbf{H}}_i, \tag{3.13}$$

$$\tilde{\mathbf{T}}_{i+1} = \tilde{\mathbf{T}}_i + \Delta\tilde{\mathbf{T}}_i. \tag{3.14}$$

Next, consider a linearization of the function $\tau(\tilde{\mathbf{H}})$:

$$\tilde{\mathbf{T}}_{i+1} = \tilde{\mathbf{T}}_i + \Delta\tilde{\mathbf{T}}_i = \tau(\tilde{\mathbf{H}}_i) + \tau'(\tilde{\mathbf{H}}_i)\Delta\tilde{\mathbf{H}}_i, \tag{3.15}$$

or, rewritten in terms of the enthalpy update $\Delta\tilde{\mathbf{H}}_i$:

$$\Delta\tilde{\mathbf{H}}_i = \frac{1}{\tau'(\tilde{\mathbf{H}}_i)}\left[\Delta\tilde{\mathbf{T}}_i + \left(\tilde{\mathbf{T}}_i - \tau(\tilde{\mathbf{H}}_i)\right)\right]. \tag{3.16}$$

Here, τ' denotes the derivative of τ with respect to its argument. Unfortunately, this derivative can be zero. This is resolved by approximating the fraction in equation (3.16) by a constant μ defined as:

$$\mu = \frac{1}{\max(\tau'(\tilde{\mathbf{H}}_i))}, \tag{3.17}$$

such that the relaxed enthalpy update becomes:

$$\Delta\tilde{\mathbf{H}}_i = \mu\left[\Delta\tilde{\mathbf{T}}_i + \left(\tilde{\mathbf{T}}_i - \tau(\tilde{\mathbf{H}}_i)\right)\right]. \tag{3.18}$$

If we now define

$$\hat{\mathbf{q}} = \mathbf{q}^{m+1} + \hat{M}\tilde{\mathbf{H}}^m, \qquad \hat{M} = \frac{1}{\Delta t}M, \tag{3.19}$$

then substitution of (3.13)-(3.15) and (3.18) into the discretized system (3.5) gives:

$$\hat{M}\left\{\tilde{\mathbf{H}}_i + \mu\left[\Delta\tilde{\mathbf{T}}_i + \left(\tilde{\mathbf{T}}_i - \tau(\tilde{\mathbf{H}}_i)\right)\right]\right\} + S(\tilde{\mathbf{T}}_i + \Delta\tilde{\mathbf{T}}_i) = \hat{\mathbf{q}}. \tag{3.20}$$

A rearrangement of terms finally leads to

$$\left(\mu\hat{M} + S\right)\Delta\tilde{\mathbf{T}}_i = \hat{\mathbf{q}} - \left(\mu\hat{M} + S\right)\tilde{\mathbf{T}}_i - \hat{M}\left(\tilde{\mathbf{H}}_i - \mu\tau(\tilde{\mathbf{H}}_i)\right). \tag{3.21}$$

By rewriting the discretized heat conduction equation in terms of the temperature increment $\Delta\tilde{\mathbf{T}}_i$ for those layers of a recording stack that do not contain a phase change material, it is possible to build a system of equations for the multi-layer as a whole. This means, that any existing finite element code for heat conduction problems in composite domains, can easily be extended to include melting.

3.3.2 Method evaluation

At first glance, the Nedjar method appears to be an elegant solution method for our two-phase Stefan problem. The implicit treatment of the moving interface, which is inherent to the enthalpy approach, makes that the method can handle multiple separate fronts; and the merging and breaking up of these interfaces. The change of phase can be isothermal or mushy. And of course, we have a way of extending the method to problems on 3D multi-layered domains.

However, some critical remarks can be made:

- A difficulty arises in case two adjacent layers of a multi-layered configuration are active layers (i.e., the materials might melt): the enthalpy is then likely to be double-valued on the separating interface. Preferably, this is avoided. Therefore, in case of a multi-layer, the assumption is made that the active layers are separated by at least one non-active layer (i.e., no melting occurs). This works, because in the non-active layers it suffices to solve the heat diffusion equation, since the temperature is the only dependent variable for these layers. Although this assumption seems very limiting when general stack configurations are considered, in practice it is not, since in existing recording stacks, two melting/recording layers are always separated by at least one non-melting/recording layer.

- The evolution of the temperature in time, for a fixed point, shows 'stair-casing'. This undesirable effect has been addressed in an earlier report by Brusche [4]. The staircasing can be slightly reduced by increasing the number of elements. Of course, this might become very impractical, in particular for 3D problems.

- Strong wiggles in the position of the moving interface are observed, even on fine meshes and for small ϵ.

- The choice of the relaxation parameter μ as in (3.17) seems to be the most optimal choice, see Nedjar [8] and References therein. However, as a results of the relaxation, the observed convergence behavior is of course linear, and often many iterations are required to reach convergence (with respect to the desired level of accuracy on the residual).

Chapter 4

The temperature approach

In the previous chapter we have looked at two enthalpy methods: an SOR scheme and a pseudo-Newton approach. Unfortunately, each method turns out to have its own major disadvantages with respect to our goal of finding a suitable method for solving the two-phase Stefan problem (2.1a-2.1c) in 3D multi-layered domains. For the SOR scheme, an extension to coupled domains seems to be nontrivial, or might not even be possible. The extensibility to coupled domains can be overcome by use of the pseudo-Newton approach, as proposed by Nedjar, but the number of iterations per time step turned out to become impractically large.

First, we will give a brief description of the temperature based method of Fachinotti et al. [7]. In particular we will focus on the concept of discontinuous integration, which is the key idea behind this temperature approach. The numerical evaluation of the integrals that arise from the FEM discretization of the temperature equation (2.9) is discussed in Section 4.2.

As a result of the inclusion of latent heat, via the liquid volume fraction $f_l(T)$, equation (2.9) is (highly) nonlinear. The corresponding discretized system is therefore solved using an iterative method. In order to improve the convergence behavior of the iterative solver, a line search or backtracking method is often needed. We will come back to this issue in Section 4.3.

Another complication related to the isothermal problem appears with the addition of an external source term. A practical remedy, namely the introduction of an artificial mushy region, is addressed in Section 6.3.2.

In the concluding section a brief evaluation of the temperature method is presented.

4.1 Method description

In the temperature formulation, the conduction of heat is described by equation (2.9):

$$\rho c_{s,l}\frac{\partial T}{\partial t} + \rho L\frac{\partial f_l}{\partial t} - \kappa_{s,l}\Delta T = q. \tag{4.1}$$

Application of the general Galerkin procedure, in which the temperature field is approximated by

$$T(\mathbf{x},t) \approx \sum_{i=1}^{n} \phi_i(\mathbf{x})T_i(t), \tag{4.2}$$

where ϕ_i is a (linear) basis shape function and T_i is the nodal temperature, in combination with an Euler backward time discretization, yields the following discretized system of (nonlinear) equations (at time level $m+1$):

$$M^{m+1}\frac{T^{m+1}-T^m}{\Delta t} + \frac{L^{m+1}-L^m}{\Delta t} + S^{m+1}T^{m+1} = q^{m+1}. \tag{4.3}$$

Note that, when compared to the enthalpy approach, both the mass matrix as well as the stiffness matrix are now time-dependent (conductivity c is now included in the mass matrix; diffusivity κ is included in the stiffness matrix).

The matrix and vector entries for the above system are given by (boundary conditions have been omitted):

$$M_{ij} = \int_{\Omega} \rho c \phi_i \phi_j d\Omega, \tag{4.4}$$

$$S_{ij} = \int_{\Omega} \nabla \phi_i \cdot (\kappa \nabla \phi_j) d\Omega, \tag{4.5}$$

$$L_i = \rho L \int_{\Omega} \phi_i f_l d\Omega, \tag{4.6}$$

$$q_i = \int_{\Omega} \phi_i q d\Omega. \tag{4.7}$$

The distinct feature of the temperature based approach is the use of discontinuous integration in space. The key idea behind discontinuous integration, as for instance described by Fachinotti et al. [7], is that for elements intersected by the moving interface, the integrals (4.4-4.7) are not computed over an element as a whole at once, using for instance an averaged value for the physical parameters, but are instead computed over the liquid and solid subdomains separately. See Figure 4.2 for an illustration.

For problems without an external source term, an accurate evaluation of the discrete balance equation (4.3) is assured, because no regularization of the integrand is required [8]. However, depending on the shape and magnitude of the external source, difficulties with respect to the convergence of the iterative method will arise, if the considered problem is taken to be isothermal. In particular, in case the latent heat is relatively large, some elements will take on values that are (nearly) equal to the melting temperature, for several consecutive time steps. Often, the appearance of these so-called *plateau* leads to a breakdown of the employed numerical method. A practical solution to this problem is the introduction of an artificial mushy region. In Section 6.3.2 the problem of dealing with a plateau will be discussed in more detail.

4.2 Numerical integration

Fachinotti et al. were not the first to apply discontinuous integration. But, as opposed to for instance Crivelli et al. [6], who applied a Gauss quadrature rule, Fachinotti et al. present an analytical integration technique. Fachinotti and coauthors prefer this analytical approach over the use of Gauss quadrature numerical integration, because ”neither extra mapping is required nor summation over sample points.”

However, despite of what is stated by Fachinotti et al., we believe that the use of Newton-Côtes numerical integration rules is a more practical choice, simply because only a (weighted) summation over sample points is required. Besides, the proposed analytical method by Fachinotti et al. requires several matrix-vector/matrix-matrix products and matrix inversions, which can be computationally costly.

The general form of an integration rule for the integration of an integrand Ind(x) over an element e is given by:

$$\int_{e} \text{Ind}(x) dx = \sum_{k=1}^{r} w_k \text{Ind}(x_k), \tag{4.8}$$

in which r is the number of support points of the integration rule, the w_k are the weights (preferably $w_k > 0$), and x_k the support points.

For simplices, i.e. line segments in $\mathbb{R}^1$, triangles in $\mathbb{R}^2$ and tetrahedra in $\mathbb{R}^3$, see Figure 4.1, the Newton-Côtes formulas for the exact integration of a constant or linear function f over a simplex e, are as follows:

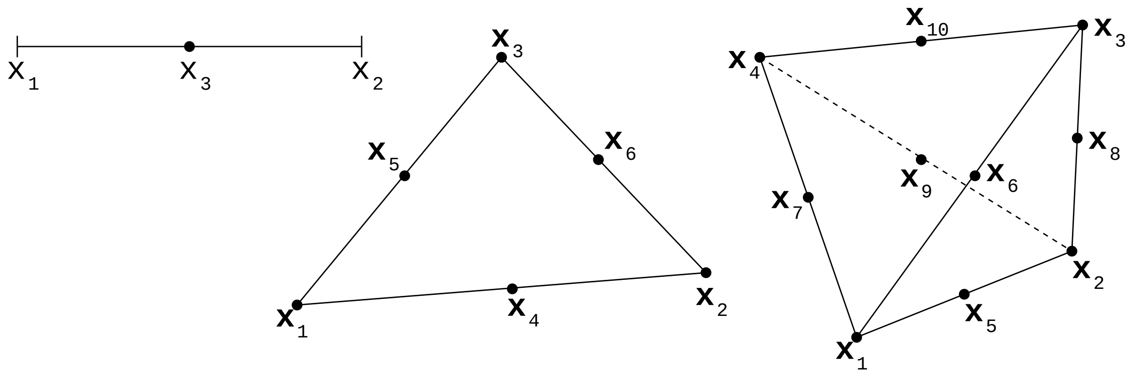


Figure 4.1: First order C^0 elements in $\mathbb{R}^1$, $\mathbb{R}^2$ and $\mathbb{R}^3$.

$$\int_e f \mathrm{d}x =$$

- $\frac{|\Delta_1|}{2!}[\sum_{k=1}^{2} f(x_k)]$ $\quad (1D)$
- $\frac{|\Delta_2|}{3!}[\sum_{k=1}^{3} f(\mathbf{x}_k)]$ $\quad (2D)$
- $\frac{|\Delta_3|}{4!}[\sum_{k=1}^{4} f(\mathbf{x}_k)]$ $\quad (3D)$

(4.9)

where $\Delta_1 = \det \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \end{pmatrix}$,

$$\Delta_2 = \det \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{pmatrix}, \text{and } \Delta_3 = \det \begin{pmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{pmatrix} \tag{4.10}$$

In case f is quadratic, e.g., the entries of the mass matrix, the Newton-Côtes formulas for exact integration are:

$$\int_e f \mathrm{d}x =$$

- $\frac{|\Delta_1|}{3!}[f(x_1) + f(x_2) + 4f(x_3)]$ $\quad (1D)$
- $\frac{|\Delta_2|}{3!}[\sum_{k=4}^{6} f(\mathbf{x}_k)] = \frac{|\Delta_2|}{3!}\sum f(\text{midpoints})$ $\quad (2D)$
- $\frac{|\Delta_3|}{5!}[-\sum_{k=1}^{4} f(\mathbf{x}_k) + 4\sum_{k=5}^{10} f(\mathbf{x}_k)] =$

  $\frac{|\Delta_3|}{5!}\left[-\sum f(\text{vertices}) + 4\sum f(\text{midpoints})\right]$ $\quad (3D)$

(4.11)

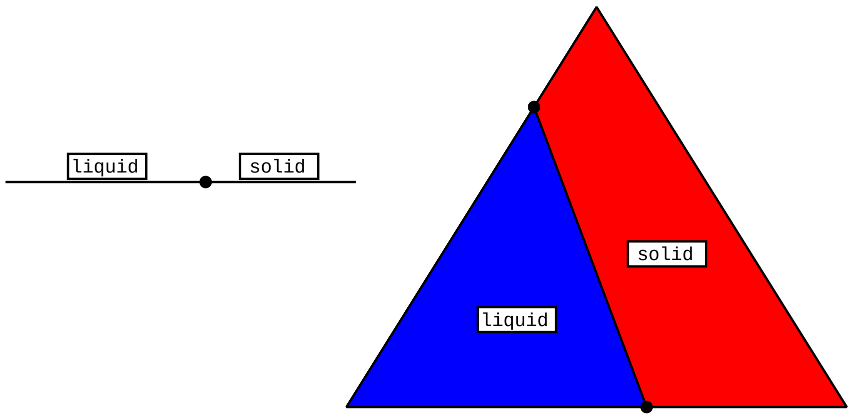


Figure 4.2: For isothermal phase-change the intersection of simplices in 1D and 2D is almost trivial.

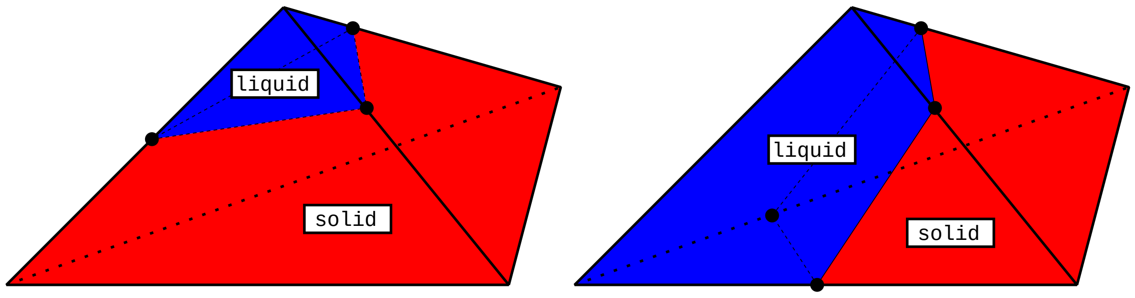


Figure 4.3: In 3D, the intersection of a linear tetrahedron is either a triangle or a quadrilateral.

In case of 1D isothermal phase-change, an element intersected by the moving interface is always subdivided into two line segments. Each segment is occupied by either one of two phases of the material, see Figure 4.2. For each of these segments, the computation of the integrals (4.4)-(4.7) by means of a Newton-Côtes rule, is almost trivial. One should only take care to insert the correct values for the physical parameters.

As for 2D isothermal phase-change, a triangle can be intersected in three ways, all of which are topologically equivalent. For an intersection as depicted in Figure 4.2, the discontinuous integration of an integrand over the whole element is performed in two steps. First, the integral over the liquid subtriangle is computed, using the appropriate physical parameters for the liquid phase. Then, in order to evaluate the integral over the trapezium shaped solid region, we first compute the integral over the whole triangle, as if the element were not intersected, and thereafter subtract the integral part over the subtriangle. Alternatively, the trapezium shaped region can be further subdivided into subtriangles, and the integrals evaluated on each separate subtriangle.

In 3D we are faced with a bit more of a challenge. First of all, the area of the intersection of a tetrahedron can be either a triangle or a quadrilateral, see Figure 4.3. Furthermore, note that both of these intersections can occur in such a way that different 'corners' of the tetrahedron are 'cut off', each of which is topologically equivalent.

In order to compute the integrals over the liquid and the solid subdomains, it is of course convenient to do so in the most efficient way. In case the intersection area is a triangle, the tetrahedral domain is subdivided into a tetrahedron and a trapezoid. While an integral over the tetrahedral subdomain is easily computed using the Newton-Côtes integration rule, the evaluation of the integral over the remainder is somewhat more complicated. One choice would be to subdivide the remainder into three tetrahedra, compute the integral values over these tetrahedra and finally add the contributions.

A more elegant approach is of course to first compute the integral over the whole tetrahedron, using the correct physical parameters corresponding to the phase of the remainder, and then to evaluate and subtract the integral part over the tetrahedral subdomain.

When the area of the intersection is a quadrilateral, one has no other choice than to subdivide each of the two trapezoidal subdomains into three tetrahedrals each and coherently adding the subsequent integral terms.

In general, the use of integration rules with negative weights should be avoided, because as a results of these weights, negative entries can appear in the element mass matrix. In addition, it should be noted that some caution is required in case the total integral over an element is computed by means of subtraction of the contributions over the subelements. For example, consider an intersected element in 2D and assume that all integrals are evaluated using first order Newton-Côtes (cf. (4.9), 2D). First, the element mass matrix for one phase, say, the solid phase, is computed for the whole element, which results in a diagonal matrix. Next, the element mass matrices are computed for the triangular subelement for both the solid and liquid parameters. The resulting element mass matrices are full matrices, although the off-diagonal elements will be relatively small with respect to the diagonal elements. Now, depending on the parameter values corresponding to both phases, the subtraction of the element mass matrix for the solid phase from the element mass matrix corresponding to the whole element, and the addition of the element mass matrix for the liquid phase, can result in negative entries on the diagonal and off-diagonals of the final element mass matrix. As a consequence, when the total mass matrix is no longer positive definite, the discrete system (4.3) will become unstable.

4.3 Solving the nonlinear system

In the temperature formulation, the heat conduction equation (4.1) is highly nonlinear, due to the addition of the latent heat term $\frac{\partial f_l}{\partial t}$. In practice, the solution to the corresponding discretized system of equations (4.3) can be found by either using Picard iterations or by use of Newton-Raphson. We will discuss both approaches.

Let us consider the residual form of the system of equations (4.3):

$$\Psi(T^{m+1}) = M^{m+1}\frac{T^{m+1}-T^m}{\Delta t} + \frac{L^{m+1}-L^m}{\Delta t} + S^{m+1}T^{m+1} - q^{m+1} = 0. \tag{4.12}$$

Starting with an initial guess $T_0^{m+1} = T^m$, the subsequent approximations of the solution $T_i^{m+1}, i = 1, 2, \ldots$ to (4.12), by means of Picard iterations, are computed via:

$$T_i^{m+1} = \Psi(T_{i-1}^{m+1}). \tag{4.13}$$

Unfortunately, for our application, this basic iterative scheme hardly ever leads to convergence: in most cases 'flip-flop', in which the solution jumps between two different states, is observed. A means of improving the convergence behavior is the use of weighted under-relaxation:

$$T_i^{m+1} = T_{i-1}^{m+1} + \omega\left[\Psi(T_{i-1}^{m+1}) - T_{i-1}^{m+1}\right]. \tag{4.14}$$

The problem here, is that we do not know any means of selecting an optimal value for ω, other than by trial-and-error. Furthermore, if a certain value of ω improves convergence for one time step, this does not automatically guarantee it will also improve the convergence for subsequent time steps.

The nonlinear system (4.12) can also be solved by means of the well known Newton-Raphson iterative scheme. The main advantage of this approach is that locally quadratic convergence is assured, as soon as the norm of the difference between the numerical solution and the exact solution is less than the radius of convergence.

For the ith iterate of the Newton-Raphson scheme we have:

$$T_i^{m+1} = T_{i-1}^{m+1} + \Delta T_{i-1}^{m+1}, \tag{4.15}$$

where ΔT_{i-1}^{m+1} is the solution of the linear system

$$J(T_{i-1}^{m+1})\Delta T_{i-1}^{m+1} = -\Psi(T_{i-1}^{m+1}). \tag{4.16}$$

The Jacobian J is given by

$$J(T_{i-1}^{m+1}) = \frac{\partial \Psi}{\partial T} = \frac{1}{\Delta t}\left(M^{m+1} + \frac{\partial L}{\partial T}\right) + S^{m+1}. \tag{4.17}$$

The partial derivative of L with respect to the temperature T needs special attention. Following the definition of $L_i^{e_i}$ it is easily derived that

$$\begin{aligned} \frac{\partial L_i^{e_i}}{\partial T_j} &= \rho L \int\limits_{\Omega^{e_i}} \frac{\partial H(T - T_m)}{\partial T_j}\phi_j d\Omega^{e_i} & (4.18)\\ &= \rho L \int\limits_{\Omega^{e_i}} \delta(T - T_m)\phi_i\phi_j d\Omega^{e_i}. & (4.19) \end{aligned}$$

It is shown in Fachinotti [7] that the integral (4.19) over Ω^{e_i}, at the finite element level, can be rewritten as an integral over the segment of the interface Γ^{e_i} contained in element e_i:

$$\rho L \int\limits_{\Omega^{e_i}} \delta(T - T_m)\phi_i\phi_j d\Omega^{e_i} = \rho L \int\limits_{\Gamma^{e_i}} \frac{\phi_i\phi_j}{\|\nabla T\|} d\Gamma^{e_i}. \tag{4.20}$$

As with the Picard iterations, in practice, divergence of the Newton-Raphson scheme is often observed when highly nonlinear problems are considered. In particular, this concerns problems for which the latent heat is relatively large in comparison to the other physical parameters. The reason for the non-convergence of the Newton-Raphson scheme, is that the computed Newton step ΔT_{i-1}^{m+1}, is too large.

A reasonable strategy to overcome this problem is to minimize the 2-norm of the residual

$$\psi = \|\Psi\|_2^2 = \Psi^{\mathrm{T}}\Psi, \tag{4.21}$$

since every solution to (4.12) minimizes (4.21). Note that the Newton step $p = \Delta T_{i-1}^{m+1}$ is a descent direction for ψ:

$$p^{\mathrm{T}}\nabla\psi = (-J^{-1}\Psi)^{\mathrm{T}}(\psi^{\mathrm{T}}J) = -\Psi^{\mathrm{T}}\Psi < 0. \tag{4.22}$$

However, there may exist local minima of (4.21) that are no solutions to (4.12). The only remedy then is to start with a different initial guess.

Because p is a descent direction, there exists an α such that $\psi(T_{i-1}^{m+1} + \alpha p)$ is reduced sufficiently. When a satisfactory α is found, the temperature is updated as:

$$T_i^{m+1} = T_{i-1}^{m+1} + \alpha\Delta T_{i-1}^{m+1}. \tag{4.23}$$

Two questions now remain to be answered. What should be the criterion for accepting a step? And how to determine α? It is certainly not sufficient to simply require that $\psi(T_i^{m+1}) < \psi(T_{i-1}^{m+1})$. First of all, ψ could be decreasing too slowly relative to the step lengths. Secondly, one can have a sequence where the step lengths are too small relative to the initial rate of decrease of ψ.

The first problem is fixed by requiring that for $0 < \lambda < 1$:

$$\begin{aligned} \psi(T_i^{m+1}) &\le \psi(T_{i-1}^{m+1}) + \lambda\nabla\psi\left(T_i^{m+1} - T_{i-1}^{m+1}\right), & (4.24)\\ &= \psi(T_{i-1}^{m+1}) + \alpha\lambda\nabla\psi\Delta T_{i-1}^{m+1}. & (4.25) \end{aligned}$$

According to [10], $\lambda = 10^{-4}$ is a good choice. The second problem can be fixed by requiring that the rate of decrease of $\psi(T_i^{m+1})$ to be greater than some fraction β of the rate of decrease of $\psi(T_{i-1}^{m+1})$.

If the above two conditions are not satisfied for the current Newton update ΔT_{i-1}^{m+1}, a very practical procedure to determine α, taken from [10], is the following line search procedure.

Define

$$g(\alpha) \equiv \psi\left(T_{i-1}^{m+1} + \alpha p\right), \tag{4.26}$$

so that

$$g'(\alpha) = p^{\mathrm{T}} \nabla \psi. \tag{4.27}$$

We start with $g(0)$ and $g'(0)$ available (the initial guess). Recall that we only need to backtrack when the full Newton step (e.i., $\alpha = 1$), is not acceptable. We then also have $g(1)$ at our disposal. The idea now is to model g as a polynomial $\tilde{g}$ in α and choose α such that it minimizes the model. This α is found by setting $\tilde{g}'(\alpha) = 0$ and then to solve for α.

In [10] $g(\alpha)$ is modeled as a quadratic polynomial $\tilde{g}(\alpha) = a\alpha^2 + b\alpha + c$ for the first step of the line search:

$$\tilde{g}(\alpha) = [g(1) - g(0) - g'(0)]\alpha^2 + g'(0)\alpha + g(0), \tag{4.28}$$

where a, b and c are easily determined using the known values of $g(1)$, $g(0)$ and $g'(0)$. A minimum is found when

$$\alpha = -\frac{g'(0)}{2[g(1) - g(0) - g'(0)]}. \tag{4.29}$$

On second and subsequent backtracks, $g(\alpha)$ is modeled as a cubic polynomial in α. Here, the previous value $g(\alpha_1)$ and the second most recent value $g(\alpha_2)$ are used:

$$\tilde{g}(\alpha) = a\alpha^3 + b\alpha^2 + g'(0)\alpha + g(0). \tag{4.30}$$

Requiring this expression to give the correct values of g at α_1 and α_2 leads to two equations that can be solved for the coefficients a and b:

$$\begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{\alpha_1 - \alpha_2} \begin{bmatrix} 1/\alpha_1^2 & -1/\alpha_2^2 \\ -\alpha_2/\alpha_1^2 & \alpha_1/\alpha_2^2 \end{bmatrix} \cdot \begin{bmatrix} g(\alpha_1) - g'(0)\alpha_1 - g(0) \\ g(\alpha_2) - g'(0)\alpha_2 - g(0) \end{bmatrix}. \tag{4.31}$$

For the cubic model (4.30) the minimum is at:

$$\alpha = \frac{-b + \sqrt{b^2 - 3ag'(0)}}{3a}. \tag{4.32}$$

It can be shown that, since the Newton step failed, $\alpha \lesssim \frac{1}{2}\alpha_1$ for small λ. Besides, we need to guard against too small a value of α. It is thus advisable to require that at most $\alpha_{\min} = 0.1\alpha_1$.

One could ask oneself whether the use of a cubic polynomial approximation is really necessary. We therefore propose the use of a quadratic model for all backtracking steps:

$$\tilde{g}(\alpha) = a\alpha^2 + b\alpha + c, \tag{4.33}$$

where a, b and c are found using the known values of $g(\alpha_1)$, $g(\alpha_2)$ and $g'(\alpha_2)$ and are given by:

$$\begin{aligned} a &= [g(\alpha_1) - g(\alpha_2) + g'(\alpha_2)(\alpha_2 - \alpha_1)] / (\alpha_2 - \alpha_1)^2, & (4.34) \\ b &= g'(\alpha_2) - 2a\alpha_2, & (4.35) \\ c &= g(\alpha_2) - g'(\alpha_2)\alpha_2 + a\alpha_2^2. & (4.36) \end{aligned}$$

The minimum for this quadratic form is found at

$$\alpha = \alpha_2 - g'(\alpha_2)/2a. \tag{4.37}$$

For many problems it often even suffices to simply model $g(\alpha)$ as a linear function. Using only $g(\alpha_1)$ and $g'(\alpha_1)$, $\tilde{g}(\alpha)$ can be defined as:

$$\tilde{g}(\alpha) = g(\alpha_1) - g'(\alpha_1)\alpha_1. \tag{4.38}$$

Instead of computing α from $\tilde{g}'(\alpha_1) = 0$, which would make no sense, we solve for α from $\tilde{g}(\alpha) = 0$ to find:

$$\alpha = \alpha_1 - g(\alpha_1)/g'(\alpha_1). \tag{4.39}$$

Fachinotti et al. seem to take a slightly different approach to minimize the residual norm. They determine the parameter α by requiring orthogonality between the residual vector $\Psi(T_i^{m+1})$ and the search direction ΔT_{i-1}^{m+1}, i.e.,

$$G^i = \left[\Delta T_{i-1}^{m+1}\right]^{\mathrm{T}} \Psi(T_i^{m+1}) = 0. \tag{4.40}$$

For the first linesearch step, G^i is approximated by a linear polynomial, while quadratic regression is applied for the successive backtracking steps.

Nonetheless, (4.40) does not make much sense: if we wish to have $\Psi(T_i^{m+1}) = 0$, then why would it be any better to take the inner product with the search direction? Note that if we multiply the residual by the Jacobian matrix, the following holds

$$(\Delta T_{i-1}^{m+1})^{\mathrm{T}} \left[\Psi(T_{i-1}^{m+1})^{\mathrm{T}} J(T_{i-1}^{m+1})\right] = (\Delta T_{i-1}^{m+1})^{\mathrm{T}} \nabla\psi = g'(\alpha), \tag{4.41}$$

which is exactly (4.27)! We therefore suspect that Fachinotti et al. made a mistake in their article!!

4.4 The addition of a source term to an isothermal problem

Most test cases found in the literature on Stefan problems consider moving fronts induced by boundary conditions on the fixed boundaries of the computational domain. This restriction also holds for the paper by Fachinotti et al. However, in optical rewritable recording, marks are formed as a result of the absorption of the energy contained in the incident laser light. In other words, our isothermal model for the mark formation process, introduced in Chapter 2, is governed by a source term, not by boundary conditions on the fixed boundaries.

Besides, without loss of generality, it may be assumed that the amorphous mark is to be created in a fully crystalline background. For our isothermal model, this implies that initially there exists no moving boundary. Unfortunately, in practice the eventual creation of a new moving front can lead to poor convergence of the Newton Raphson scheme. At first, a reduction of the time step size can proof beneficial, but for increasingly larger values of the latent heat, a breakdown of the iterative procedure is imminent.

To illustrate this problematic behavior of the numerical solution, let us consider a temperature distribution at time level $t = t_n$, as sketched in Figure 4.5. The time step size is $\Delta t = 1$. Now assume that at the next time instance, part of the numerical solution reaches the melting temperature T_m, at which point the latent heat come into effect. At this stage, typically an overshoot of one of the nodes is observed. Moreover, the number of Newton iterations increases considerable and backtracking is used heavily.

Next, assume that we successfully acquired a solution at $t = t_{n+1}$ (which in general is not even guaranteed), but failed to obtained a solution at time level t_{n+2} (within the specified maximum number of Newton iterations). Further assume that a second attempt, this time using a time step that is half the size of the original one, is successful. Most likely, the newly obtained solution shows several overshoots and undershoots. With each consecutive attempt, as the time step is contiguously reduced, the acquired solutions will oscillate between the two states shown in Figure 4.5. Not surprisingly, the amplitudes of the oscillations decrease with the reduction of the time step size.

A reasonable explanation for the mentioned consistent divergence of the Newton scheme is that for any *feasible* time step size, at some time level there does not exist a solution to the isothermal problem within the discrete Galerkin subspace. If the instantaneous jump in the enthalpy becomes too large, the isothermal model is thus no longer usable.

To counteract this impracticality, we propose the introduction of an artificial mushy region. This regularization of the enthalpy allows the approximating solution to lie within a small band

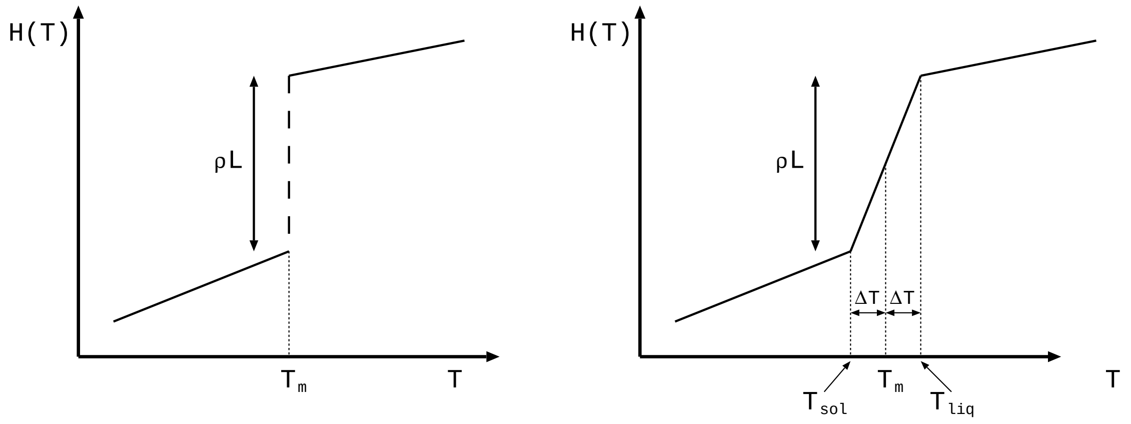


Figure 4.4: Enthalpy H as function of temperature T. Isothermal (left) and mushy (right).

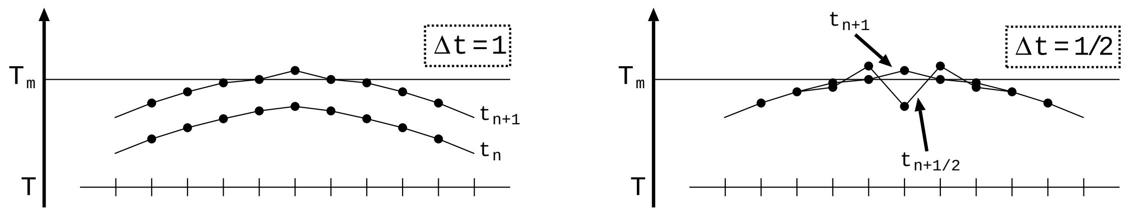


Figure 4.5: The appearance of a new (source term induced) moving interface: evolution of the numerical solution in time. Isothermal case. The numerical solution mainly oscillates between two states; amplitudes decrease with the reduction of the time step size.

(the mushy region) around the melting temperature T_m. The bounds of the artificial mushy region are determined by two values of the temperature, which we denote by T_{sol} and T_{liq}, see Figure 4.4. These bounds are determined via a parameter T_Δ as:

$$T_{\text{sol}} = T_m - T_\Delta, \qquad T_{\text{liq}} = T_m + T_\Delta \tag{4.42}$$

In general, a value of, say, 0.1% of the melting temperature, is sufficient to sustain good convergence behavior of the Newton scheme, while at the same time the time step size can be kept constant. Smaller values of T_Δ may require a few reductions of the time step size, although with respect to the accuracy, the gain is minimal.

The effect of the artificial mushy region on the evolution in time of the temperature distribution, taken from the previous example, is sketched in Figure 4.6. Note that, since the enthalpy is regularized, it is also continuous. Therefore, the enthalpy can take on any value in the range of the jump from 0 to ρL. And thus, as can be seen from the figure, the artificial region allows for minor oscillations of the numerical solution around the melting temperature.

The application and performance of the artificial mushy region will be further discussed in Section 6.3.2. In particular, we will focus on the choice of the regularization parameter T_Δ.

4.5 Method evaluation

For a large variety of test problems, the above discussed temperature approach outperforms the enthalpy methods. Computational costs seem to be less and, probably most importantly, the moving interface is found to show hardly no wiggles and stair-casing is considerably reduced.

Nonetheless, a few issues should be addressed:

- Although the temperature method performs extremely well for relatively small values of the latent heat L, the performance deteriorates when L becomes relatively large. This problem

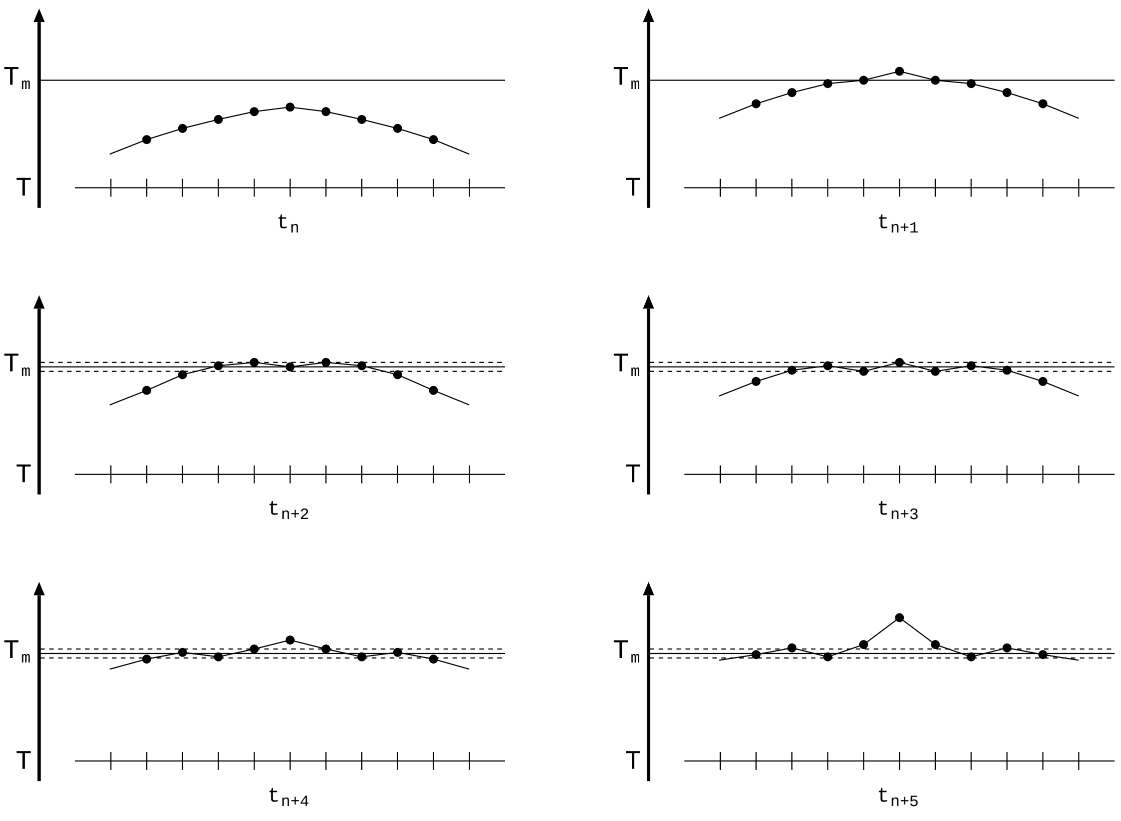


Figure 4.6: The appearance of a new (source term induced) moving interface: evolution of the numerical solution in time. Mushy case. Only minor oscillations within the mushy band; no time step size reduction.

can be (partly) remedied with the addition of a line search procedure to the Newton Raphson solver. A good estimate of the parameter α, used to scale the Newton step, can be determined using either linear, quadratic or cubic regression.

- The addition of an external source to the temperature formulation, in combination with larger values for the latent heat, gives rise to plateau: temperature values of all the nodes of one or more elements take on values that are (up to a very small tolerance, say $O(10^{-6})$) equal to the melting temperature T_m during several consecutive time steps. As a result, the convergence of the Newton scheme is heavily affected, and the method often breaks down completely. This complication can be resolved by introducing an artificial mushy region.

Chapter 5

Adaptive local mesh refinement

In this chapter several strategies considering adaptive local mesh refinement are discussed and a refinement strategy is selected. In Section 5.4 the implementation of the preferred strategy is described in detail.

5.1 Introduction

The accuracy of a finite element solution is affected by the total number and size of the elements that are used. On finer grids, a higher level of accuracy can be expected than on coarse grids. Globally refining an existing mesh, however, is very impractical. Computational time and storage space can soon become huge, especially in 3D.

An obvious means of resolving this major drawback of global refinement is to limit the refinement to selected regions of the computational domain. Typically, these regions are appointed based on a priori knowledge about particular characteristics of the solution or the geometry of the domain. In case of Stefan problems, the estimated position of the moving boundary presents a good indication of where to refine the mesh. Of course, as a result of the movement of the interface, different regions can be assigned to be refined during a computation.

Ideally, a typical local refinement code has the following properties:

- Adaptive: an advancing interface or changes in geometry can be captured.
- Preserving: the quality of the mesh does not degrade during successive refinements.
- Efficient: the amount of refinement and the corresponding gain in accuracy are balanced.
- Practical: the construction of the refinement and the data structure used, are uncomplicated.

The refinement strategy and data structure discussed in this chapter have been developed as an integral part of the Sepran [11] finite element package. Since the larger part of the information on the structure of the mesh is available within the Sepran code, the data structure presented here is specifically designed to facilitate the addition of the local adaptive mesh refinement to the Sepran package. An important feature of our design is that the implementation of the data structure is such that it is independent of the spatial dimension.

Although most of the principles of the local mesh refinement as described in this chapter are generally applicable, some aspects specifically relate to Stefan problems, and solid-liquid transitions in particular.

5.2 Basic terminology

Before discussing the local mesh refinement, it is convenient to first introduce some basic terminology, which is used throughout the text.

By *refinement* we mean the subdivision of an element into sub-elements. Our subdivision is always the result of the division of at least one edge of the element into two new edges. A new node is added at the intersection point of the divided edge. This node is always connected to either an 'opposite' vertex of the original element, or another added node. *Hanging nodes*, i.e., nodes that are not a vertex of all sub-elements containing this node, are thus avoided. As such, a 1D line element is always divided into two line elements, a 2D triangular element is at most subdivided into 4 subtriangles, and a 3D tetrahedral element is at most divided into 9 subtretrahedra. Throughout this chapter it is assumed that an edge is always split up into two equally sized subedges.

If all of the subelements of an already refined element are refined as well, we say that the initial element is refined twice, or put differently, that the *level of refinement (LOR)* of the initial element is *of the order two.*

Two elements are called *direct neighbors*, in case they share at least one vertex. These elements are said to be *at 'distance' 1.* The 'distance' between an element and a direct neighbor of a direct neighbor of this element is equal to 2, etc.

The initial mesh, i.e., the mesh at the beginning of a computation at $t = 0$, is called the *basis mesh.* Elements of the basis mesh are referred to as *basis elements.*

In case of isothermal moving boundary problems, an element is called an *intersected element*, when at least one of the vertices has a temperature value larger than the melting temperature T_m, and at least one node for which $T < T_m$. If a mushy region is considered, the element is an intersected element when at least one of the nodes has a temperature smaller than the solidification temperature T_{sol} or the liquification temperature T_{liq}, and at least one node for which the temperature is larger. When the whole element is mushy ($T_{\text{sol}} < T < T_{\text{liq}}$), it is also considered to be intersected.

When an element of the basis mesh is appointed to be refined, the element is said to be marked for refinement, or simply *marked.*

5.3 Local refinement strategies

Whenever a local refinement of a mesh is required, the first decision to be made is which elements are to be refined. A generally accepted and commonly used criterion for element refinement is based on a computed element wise error of the approximated solution with respect to some reference solution. Those elements for which the error is larger than a given tolerance, are marked to be refined.

However, in general this approach requires some error estimate based on theory or knowledge on the exact (analytical) solution. For practical applications, such reference solutions generally do not exist. An obvious alternative is to refine in a neighborhood where difficulties or inaccuracies can be expected. A priori knowledge of these locations often exists, based on for instance properties of the computational domain. One could think of composite geometries with discontinuous material properties, or kinks. In particular, the position of the moving boundary, as for example in our mark formation model, gives a very good indication of where mesh refinement can be beneficial.

A problem that arises when moving boundary problems are considered, is that the interface displacement can be rather unpredictable. Although the interface velocity can be estimated, given the governing equation and the values for the physical parameters, the merging and breaking up of multiple fronts most likely gives rise to irregularities. An obvious cause of these difficulties lies of course in the discontinuity of the derivative of the temperature across the moving interface.

In general, it is to be ensured that after one time step of the numerical scheme, the moving interface always ends up in the same, or at most adjacent basis elements. This strict requirement guarantees that the latent heat contribution is dealt with appropriately (in discrete sense). One might consider this as a kind of Courant-Friedrich-Lewy criterion. Since the moving interface is restricted to advance at most to adjacent elements, it follows that the minimal region to be marked for refinement consists of the intersected elements, and their direct neighbors. The collection of marked elements forms a so-called *band of refinement.* The width of this band of refinement, an integer value referred to as the *width of the refinement*, is defined as the maximum minimal 'distance' between a marked (non-intersected) element and the intersected elements. For example, in case only the direct neighbors of the intersected elements are marked for refinement, it has a value of 1. If, in addition, the direct neighbors of these direct neighbors are also marked, then the width of the refinement is equal to 2, etc.

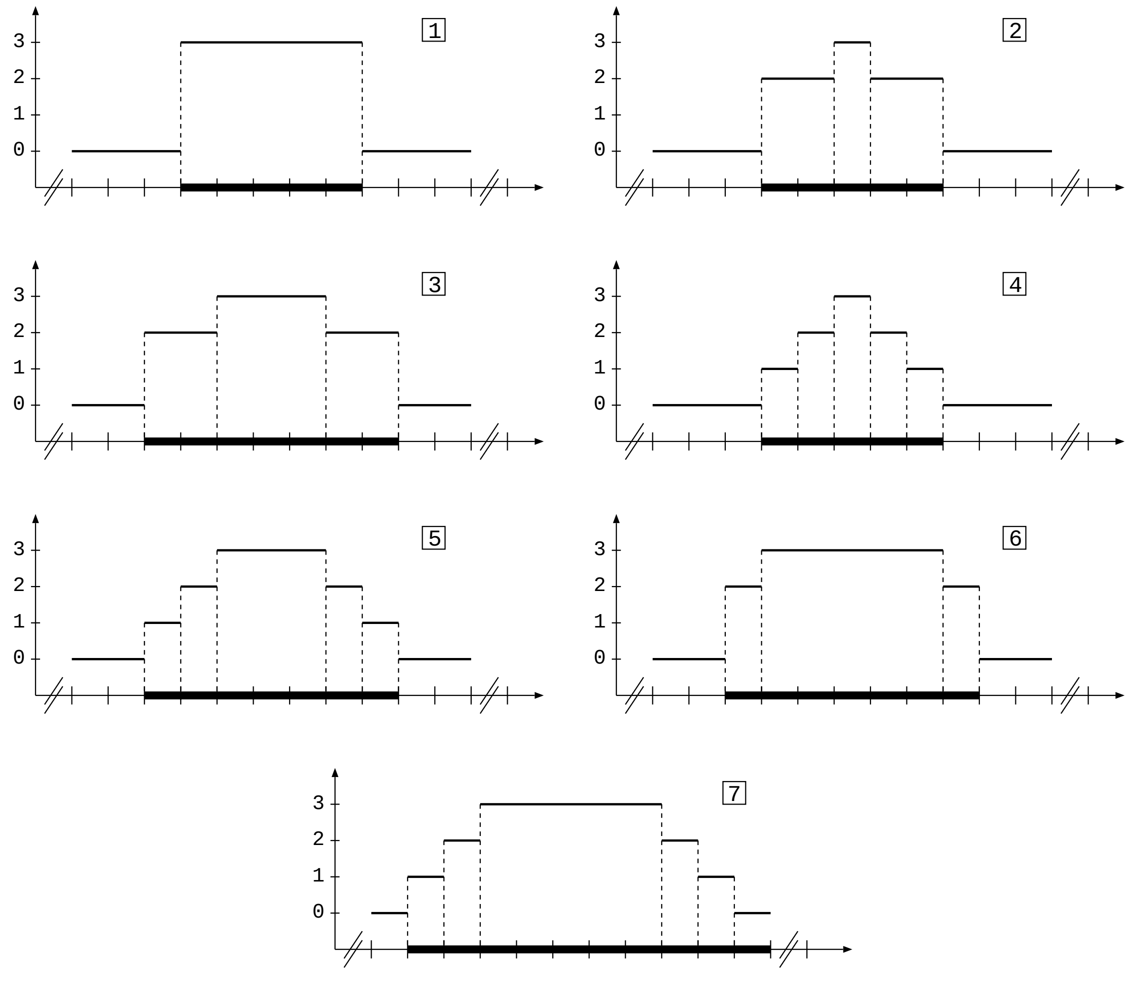


Figure 5.1: Graphical representation of seven means of assigning an LOR to marked elements (bold elements). The central element is assumed to be intersected. The height of the bar on top of each element indicates the assigned LOR (vertical axis).

Because of our CFL-like criterion, it can thus be concluded that the minimal width of the refinement is equal to 1. The final width of the refinement depends on two choices. Firstly, the maximum 'distance' from the intersected elements at which elements are assigned the highest LOR, must be selected. Secondly, in case more elements are to be refined, it should be decided whether these basis elements are all refined at a lower, but constant, LOR, or that the assigned LOR is gradually reduced depending on the 'distance' from the intersected elements.

Figure 5.1 shows a 1D graphical representation of a variety of ways in which basis elements can be assigned an LOR. If all elements are refined the same number of times as in option 1, say, with a LOR of the order two or more, the transition from the refined grid to the coarse grid can be quite abrupt. In particular when the band of refinement consists only of the intersected elements and their direct neighbors, this choice could have a negative impact on the effectiveness of the refinement. The same argument holds for options 2, 3 and 6.

It is also best avoided to assign the highest LOR to just the intersected elements, as in options 2 and 4. When the moving interface advances to adjacent elements, a loss of accuracy would be the result of the adjacent elements being coarser. This loss of accuracy can be easily avoided by at least giving the directly neighboring elements the same LOR as the intersected elements.

In short, the fifth and seventh option are the most favorable. Of these two alternatives, option 5 seems to be the best choice, because it will lead to the least total number of elements in the refined mesh.

Next, it has to be decided how to realize the actual refinement of the marked elements. Three different refinement strategies can be employed:

S1 Refine each basis element the assigned number of times at once. Consider the basis elements one at a time, sequentially.

S2 Perform a single refinement of all basis elements that need to be refined at least once. Next, refine the subelements of the basis elements that are marked to be refined twice or more. Repeat this procedure until all basis elements are refined the assigned number of times.

S3 Perform a single refinement of all basis elements that have the highest assigned LOR. Then refine each element that has an LOR that is one smaller. Continue this process until all marked basis elements are refined as intended.

To make these refinement strategies more illustrative, let us look at a 2D example. Consider a segment of a basis mesh consisting of four elements, as shown in Figure 5.2. For simplicity, let us assume that element 3 has been marked to be refined twice and the adjacent elements 2 and 4 are marked to be refined once.

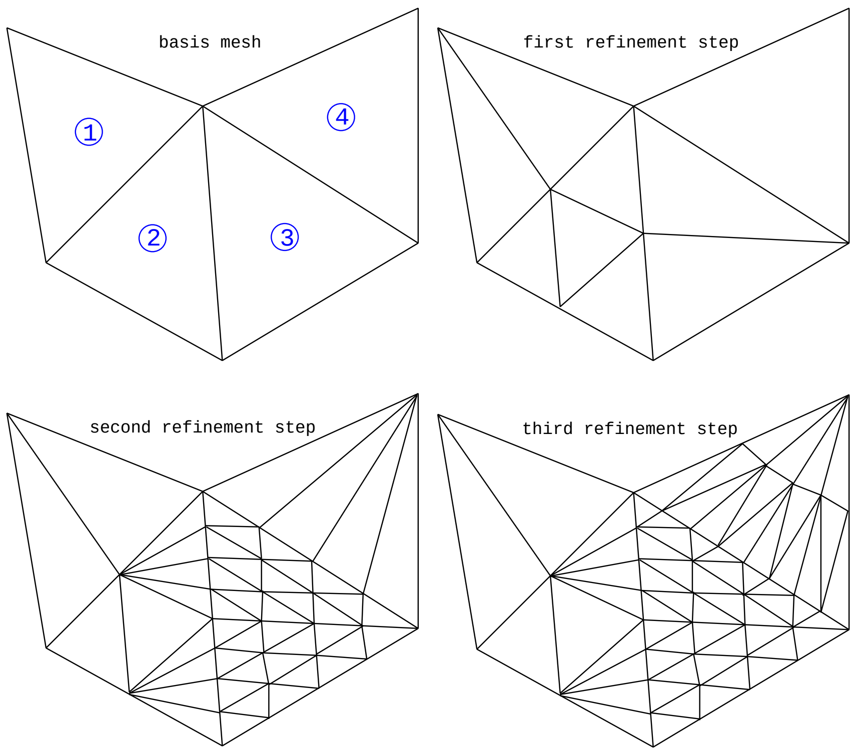


Figure 5.2: Refinement strategy S1. Elements are refined successively. Subdivided elements that are created to avoid hanging nodes inherit the LOR of the original basis element.

Remark that as a result of the refinement of some of the basis elements, neighboring elements need to be subdivided, even when they are not marked for refinement, in order to avoid hanging nodes. The way in which these hanging nodes are avoided can have a major effect on the final mesh. Let us consider the first refinement strategy. In case the hanging nodes are corrected for in

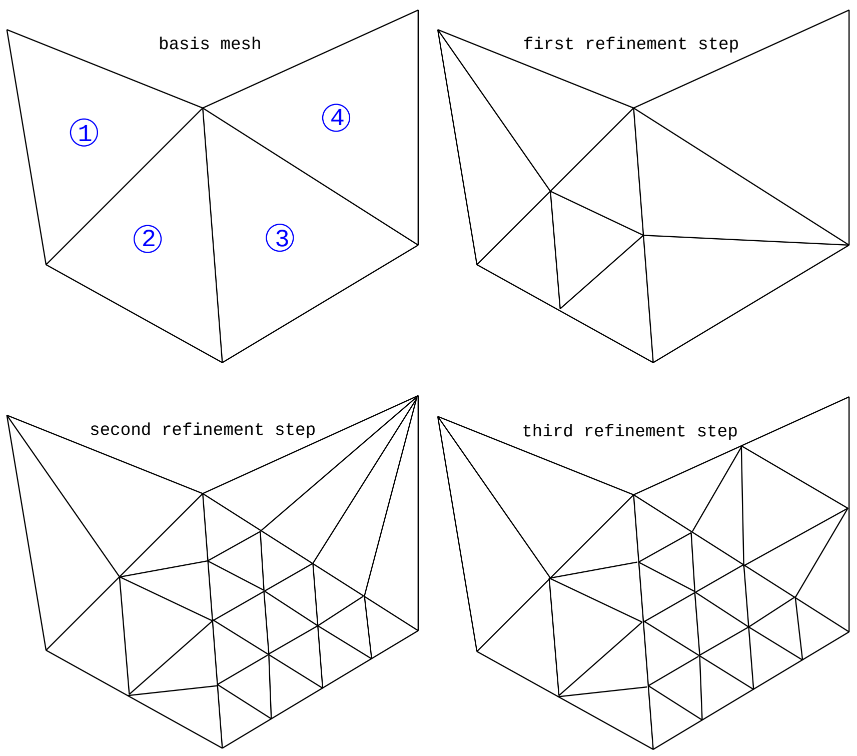


Figure 5.3: Refinement strategy S1. Elements are refined successively. Subdivided elements that are created to avoid hanging nodes are removed. Refinement proceeds from the associated basis element onward.

adjacent elements each time a basis element is refined to the appointed LOR, the quality of the resulting mesh deteriorates considerably, as is shown in Figure 5.2.

A better approach is to remove the subelements that are the result of previous corrections of hanging nodes, then to refine the original basis element the required number of times, and finally to deal with newly created hanging nodes, see Figure 5.3. Of course, this is very impractical with respect to the implementation, because a lot of precise bookkeeping is required.

Therefore, it is advisable to choose either the second or third mesh refinement strategy, as shown in Figure 5.4. Of these two options, strategy S2 is the best alternative. Firstly, the refinement of the mesh changes the most gradually. Secondly, the total number of elements (and unknowns) of the final mesh is the smallest. Furthermore, unnecessary interpolations during transitions from one mesh to another are avoided, since all the nodes of refined basis elements with a low LOR are always also nodes of subdivided basis elements with a higher LOR.

Finally, a decision has to be made on how to implement the transition from an existing locally refined mesh to the next. Two basic approaches can be distinguished:

- First refine the elements that need further refining. Then coarsen those elements that are assigned a lower LOR than before.
- Start from the basis mesh and refine the mesh employing refinement strategy S2 or S3.

Each of these approaches has been illustrated in Figure 5.5. As a starting mesh, a refined mesh obtained using strategy S3 is considered (cf. Figure 5.4). It is assumed that for the mesh to be created, element 2 has been marked to be refined twice and the adjacent elements 1 and 3 are marked to be refined once. This could for instance occur when a moving interface advances from element 3 to element 2.

From Figure 5.5 it immediately becomes clear that building a new locally refined mesh is best done starting from the basis mesh. The number of elements, and thus the number of unknowns,

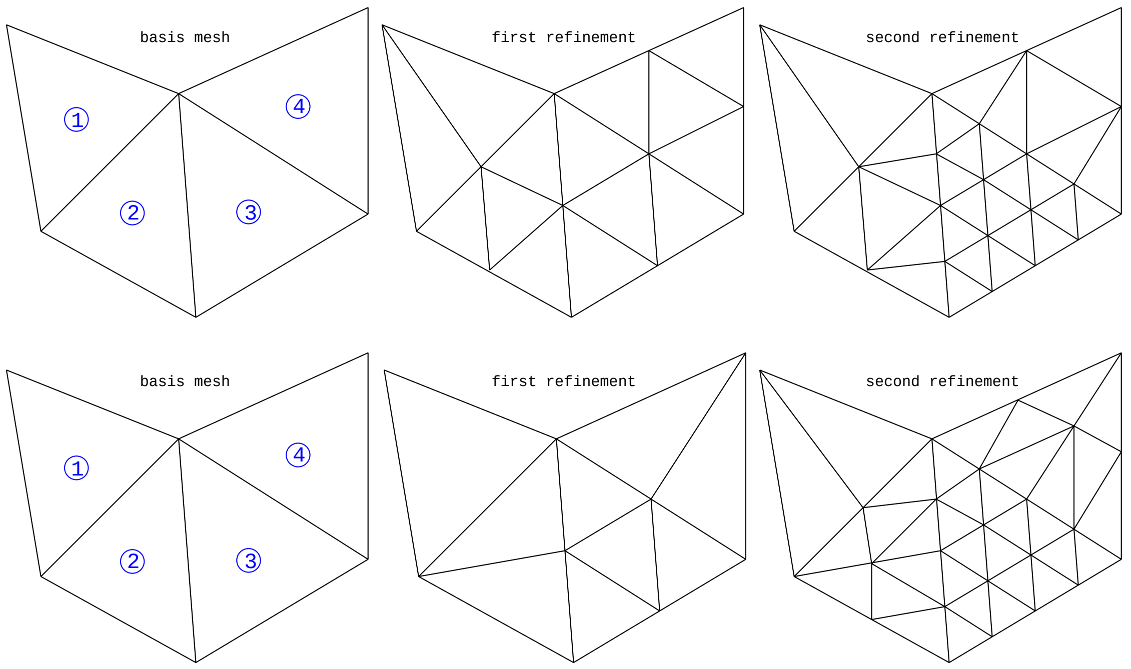


Figure 5.4: Refinement strategies S2 (top) and S3 (bottom). Refinement proceeds per LOR. Subdivided elements that are created to avoid hanging nodes are removed. Refinement proceeds from the associated basis element onward.

is less. Besides, no irregularly shaped elements appear, as a result of compensating for what otherwise would have been hanging nodes. Furthermore, the mapping of data between meshes becomes almost trivial.

Finally, it is important to stress that the coarseness of the basis mesh is restricted to be such that the accuracy of the solution to a problem without latent heat is guaranteed on this mesh. If not, there is no use in refining the mesh locally.

In conclusion, a practical strategy for locally refining a mesh has the following characteristics:

- Always construct the refined mesh from the basis mesh.
- Always refine the intersected basis elements and the direct neighbors (CFL-like condition).
- Mark the intersected basis elements and the direct neighbors to get the highest LOR. The remaining marked elements get a lower LOR, which gradually reduces with the 'distance' of the elements from the intersected elements.
- Perform a single refinement of all basis elements that need to be refined at least once. Next, refine the subelements of the basis elements that are marked to be refined twice or more. Repeat this procedure until all basis elements are refined the assigned number of times.

5.4 Implementation of the local mesh refinement

In the previous section we have derived an optimal strategy for locally refining a mesh. In this section, the implementation of this refinement strategy is discussed. The backbone of the local refinement code is the so-called *outer loop*. The structure of the outer loop can be split up into five parts:

1. initialization of the data structure

2. solution of the governing equation for a single time step on the current (refined) mesh
3. if needed, construction of new refined mesh
4. if the mesh has been adapted, mapping of the data to the new mesh
5. if needed, adaptation of the time step size

Each step of the outer loop, except of course the initialization of the data structure, which is only needed at the very beginning of a computation, is processed during each time level of the numerical time integration process. Next, we will address steps 3 to 5 of the outer loop in more detail. In pseudo-code, the structure of the outer loop is as follows:

Outer refinement loop [**prenthrefine**]

```
extract parameters from input [prenthrefrdin]

if (first time this routine is called) then
   create and initialize auxiliary arrays [mshrefinit/prfillrefvec]
endif

do while (no convergence and not iterations > maxiter)
   solve governing equation [prenthalphy]
   if (no convergence) then
      reset time and data to previous time level
      modify time step size
   endif
enddo

based on new solution, determine necessity of refinement, and mark elements [prsetrefine]

if (refinement necesarry) then
   fill auxiliary arrays [mshrefupdatend]
   construct new refined mesh [mshrefupdate]
endif

if (interface outside refinement) then
   reset time and data to previous time level
   possibly modify time step size
else
   map data to new mesh [mshrefmapping]
   update auxiliary arrays [prenthrefine1]
endif
```

5.4.1 Marking elements for refinement

When the numerical solution to the governing equation has been computed at the new time level on a previously refined mesh, this solution is used to determine whether the *old mesh* should be adapted, and if so, how. To this end, first an array *update* is created of which the length is equal to the number of elements of the old mesh. If an element of the old mesh is intersected, the corresponding entry in array *update* is set equal to '1', if it is not intersected, the entry is set equal to '0', see Figure 5.6.

Next, an array *refinenew* is filled, of which the length is equal to the number of elements of the basis mesh. At this stage, array *refinenew* has essentially the same functionality as array *update*, but now for the basis mesh. If the old mesh is equal to the basis mesh then filling array *refinenew*

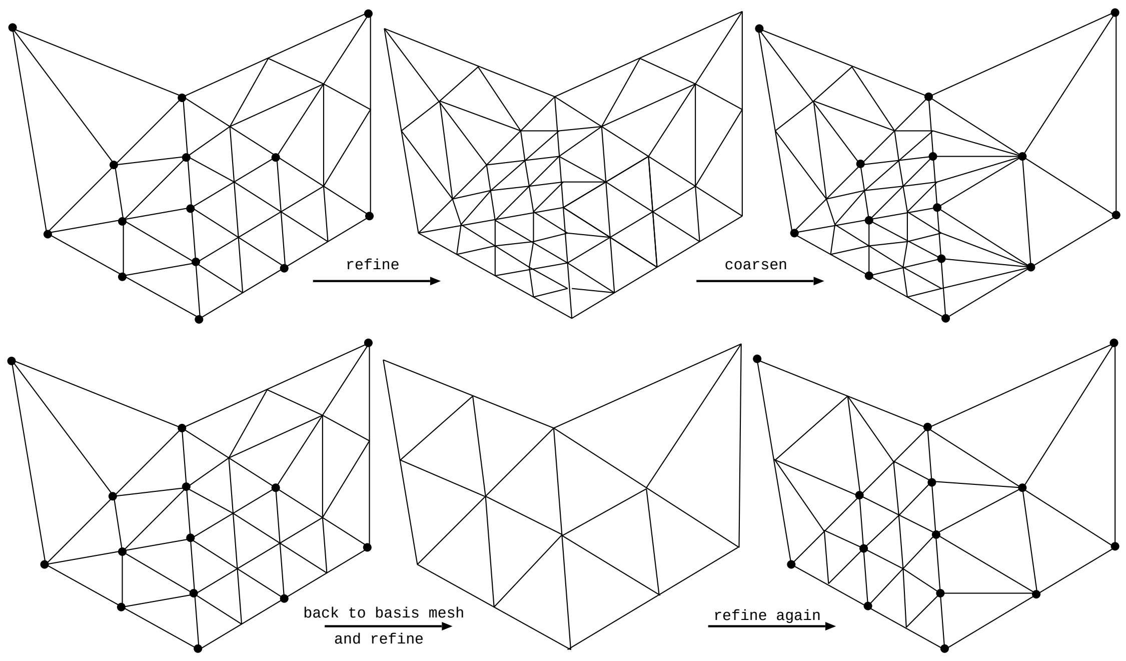


Figure 5.5: Two ways of implementing the transition from one mesh to another. Top: first refine and then coarsen. Bottom: rebuild mesh from basis mesh. The nodes indicated with a '●' are common to all final meshes.

is trivial: *refinew* becomes an exact copy of array *update*. In case the old mesh differs from the basis mesh, a auxiliary array called *refmapold* is used, which contains mapping information on how the elements of the refined old mesh relate to those of the basis mesh. The addition 'old' in the name 'refmapold' indicates that the array contains information concerning the old mesh. In the same way, the addition 'new' in the name refinenew indicates that the array contains information with respect to the new mesh. For a detailed explanation of array *refmap* we refer to Section 5.4.3. If one or more elements of the old mesh, that are subelements of the same basis element, are marked for refinement, the corresponding entry of array *refinenew* is also marked.

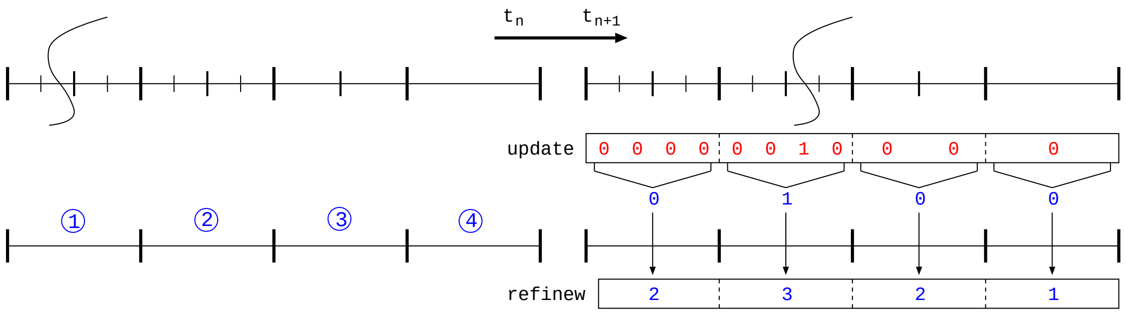


Figure 5.6: Filling of the refinement vector. Entries of array *update* that correspond to an intersected element of the old mesh are assigned a value of '1'. Basis elements are marked for refinement if at least one of its subelements is marked. The corresponding positions in array *refinenew* are assigned a value LOR$_{max}$. Finally, neighboring elements are assigned LOR values in accordance with the employed refinement strategy.

Now that all basis elements that are intersected by the moving interface have been marked, the band of refinement surrounding the intersected elements is created. First the entries of array refinenew that contain a '1' are replaced by a number equal to the desired LOR plus one. In this way, the intersected elements can later be distinguished from the adjacent elements. The entries of array refinenew that correspond to the direct neighbors are assigned a value equal to the LOR. Next, the elements adjacent to these elements are given a lower LOR, which gradually reduces with the 'distance' of these elements from the intersected elements.

In pseudo-code, the marking of the elements proceeds as follows:

Marking of elements **[prsetrefine]**

```
fill array update [prfillrefvec]

fill array refinenew [prfillrefvec2]

do (for each LOR)
   mark neighboring elements
enddo
```

5.4.2 Construction of the refined mesh

Obviously, by far the most important part of the mesh refinement code considers the actual construction of the refined mesh. This process consists of an outer loop over all LOR, and an inner loop over all elements of the basis mesh. The outer loop is straightforward:

mesh construction outer loop **[mshrefupdate]**

```
start with basis mesh

create and initialize auxiliary arrays

do (for each LOR)
   construct intermediate mesh [mshreflocalent]
enddo

finalize new mesh
```

At this point it is important to explain up to some degree of detail the way in which a mesh is build up and stored in the Sepran finite element package [11]. To illustrate the construction process of a mesh in Sepran, let us consider a 2D triangulation. Initially, one starts with a single triangular element. This element is labeled to be element number '1', and the vertices and edges are each assigned labels '1' to '3'. Information on the numbering of the vertices, edges and elements is stored in separate arrays. In addition, information is stored on the connections between vertices and edges, vertices and elements, and edges and elements.

When a second element is added, first a new node is added to the list of vertices. This node is assigned the label '4'. Then, two edges, numbered '5' and '6' are added to the list of edges. Next, element number '2' is added to the array of elements. Finally, the information on the connections between the vertices, edges and elements is updated in accordance with new mesh. Each successive element is added accordingly.

Let us return to the refinement of an existing mesh. As explained in Section 5.3, a refined mesh is built up starting from the basis mesh. Then, for each LOR, the following procedure is followed:

mesh construction inner loop **[mshlocalent]**

```
do (for each edge)
   if (edge is intersected)
```

```
      mark edges to be subdivided
   endif
enddo

do (for each element)
   if (at least one edge is marked)
      appoint required subdivision of element
   endif
enddo

do (for each marked element)
   remove old element
   create new nodes
   create new edges
   add new elements [mshelemref]
   fill auxiliary arrays
enddo
```

In short, the general idea behind the inner loop is thus to sequentially expand the basis mesh by taking out basis elements that are to be refined, and adding the appropriate subelements to the new mesh. The described procedure is best illustrated by means of an example. Consider a basis mesh consisting of three elements as shown in Figure 5.7, and assume that element '2' is marked to be refined once.

In this situation, the contents of the array in which the numbering of the elements is stored is: '1', '2' and '3'. The vertices are numbered '1' to '5', and the edges are labeled '1' to '7'. Since element '2' is assumed to be refined, the edges '3', '4' and '5' are marked to be subdivided. Because edge '3' is shared by elements '1' and '2', element '1' is appointed to be subdivided into two subelements. Elements '2' is labeled to be divided into 4 subelements, since all its edges are marked to be subdivided. For element '3' the same argument for subdivision holds as for element '1', because of edge number '4'.

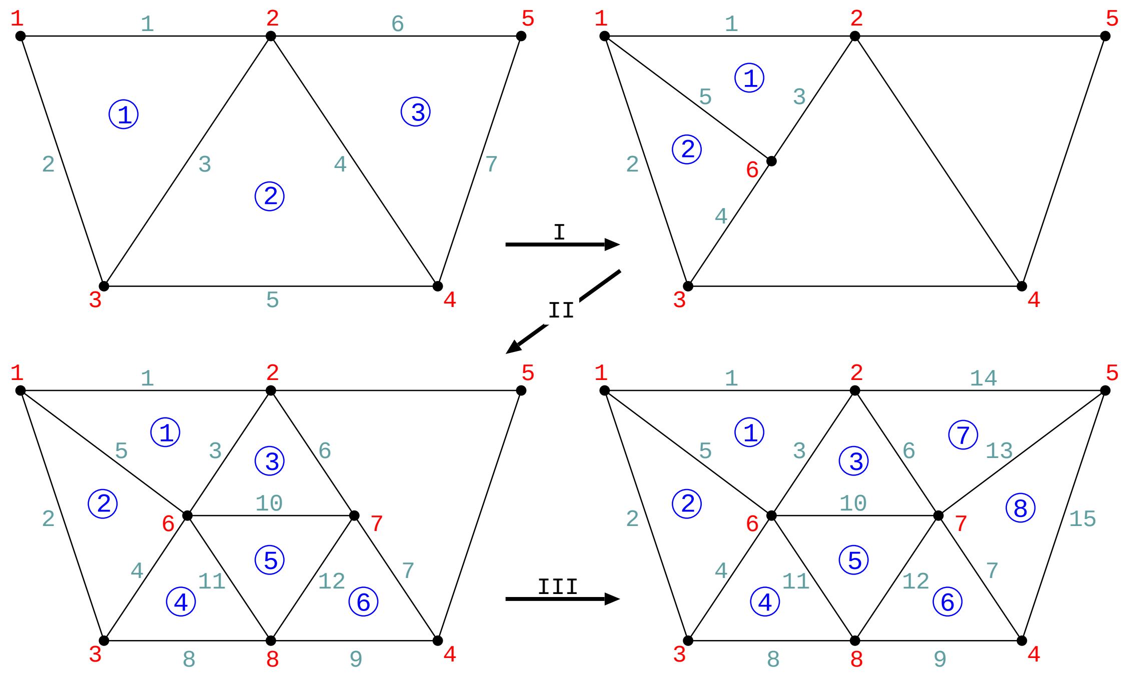


Figure 5.7: Sketch of the refinement procedure. Basis element '2' is to be refined once.

During the refinement process, as illustrated in Figure 5.7, several auxiliary arrays are filled. A new version of each of these arrays is created for each newly adapted mesh. A version of each of these arrays associated with a previously refined mesh is always stored, since they are useful for the mapping of the data or when the time level is reset to the previous time level.

Arrays *intnodes* and *parentnodes* are later used for the mapping of the data from the previous mesh to the newly created mesh. Array *refmap* is used to fill array *update*, see Section 5.4.1, and to fill array *intnodes*. The auxiliary arrays that are discussed here are:

- *refmap*: contains mapping information on how the elements of the new mesh relate to those of the basis mesh.
- *intnodes*: contains for each basis element the node numbers of all the vertices that are added during the subdivision, for all LOR. These added nodes are called *internal nodes*.
- *parentnodes*: contains for each added node, the node numbers of the end points of the edge that this node was added on during the subdivision. These two end nodes are referred to as *parent nodes* of the added node.

For each element of the refined mesh, array *refmap* contains two integer values. The first integer is the number of the basis element, the *parent element*, of which this element of the refined mesh is a subelement. The second number is the assigned LOR of the parent element. The total length of array *refmap* is equal to two times the number of elements of the refined mesh.

If, for instance, a 1D refined mesh as sketched in Figure 5.8 is considered, then array *refmap* has the following contents: 1, 1 | 1, 1 | 2, 2 | 2, 2 | 2, 2 | 2, 2 | 3, 2 | 3, 2 etc.

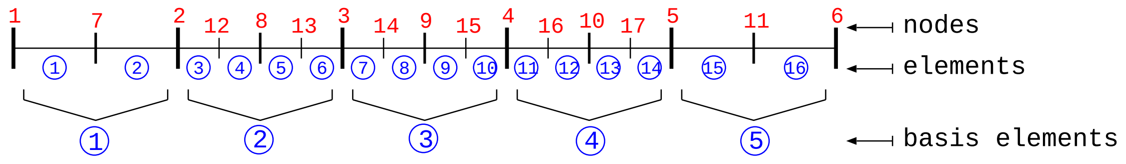


Figure 5.8: Example of a 1D refined mesh consisting of 5 basis elements.

By far the most difficult array to construct, is auxiliary array *intnodes*. The contents of array *intnodes* consists of two parts. Let N_{basis} be the number of elements of the basis mesh. The first N_{basis} entries of array *intnodes* contain pointers to positions larger than N_{basis} of the array, where the numbers of the internal nodes for each basis element are stored. These node numbers per basis element form the second part of array *intnodes*. Because the required LOR for each basis element has been determined beforehand, the number of internal nodes per basis element to be refined can easily be calculated. The number of internal nodes for each dimension and required LOR is computed as follows:

- $2^{\text{LOR}} - 1$ $\quad(1D)$
- $(2^{\text{LOR}} + 1) * (2^{\text{LOR}-1} + 1) - 3$ $\quad(2D)$
- $(2^{\text{LOR}} + 1) * (2^{\text{LOR}} + 2) * (2^{\text{LOR}} + 3)/6 - 4$ $\quad(3D)$

The number of internal nodes for the most relevant situations are shown in Table 5.1. The length of array *intnodes* is thus equal to N_{basis} plus the total number of internal nodes. In the second part of array *intnodes*, the node numbers of the internal nodes are stored in the order in which the nodes are added to the new mesh. This ordering of the nodes numbers is particularly important for the mapping of data from one mesh to another. Information concerning the parent element, such as the assigned LOR, is obtained from array *refmap*.

For the 1D refined mesh as shown in Figure 5.8, array *intnodes* has the following contents: 6, 7, 10, 13, 16 || 7 | 8, 12, 13 | 9, 14, 15 | 10, 16, 17 | 11. This array was constructed and filled as follows. Assume that each basis element has been assigned a LOR: basis elements '1' and '5' are refined once, the other basis elements twice. The length of array *intnodes* is thus equal to 5 + 2*1 + 3*3 = 16. The first five entries of array *intnodes*, the pointers to the starting positions of the information for each basis element, can now be filled. Then, a loop over all elements that need to be refined at least once is performed. This fills the first position in each segment of the array corresponding to the various basis elements. After a second loop, this time over all elements that have an LOR equal to 2, the remaining entries are filled.

Table 5.1: The number of internal nodes per dimension and LOR.

LOR:	1	2	3
1D	1	3	7
2D	3	12	42
3D	6	31	161

The structure of array *parentnodes* is simple. Let n_{basis} be the number of nodes of the basis mesh. Then the first two positions of array *parentnodes* contain the node numbers of the parent nodes of internal node number $n_{\text{basis}}+1$, the third and fourth position of array *parentnodes* contain the node numbers of the parent nodes of internal node number $n_{\text{basis}}+2$, etc. The length of array *parentnodes* is therefore equal to two times the number of internal nodes. For the 1D refined mesh as shown in Figure 5.8, array *parentnodes* has the following contents: 1, 2 | 2, 3 | 3, 4 | 4, 5 | 5, 6 | 2, 8 | 8, 3 | 3, 9 | 9, 4 | 4, 10 | 10, 5.

5.4.3 Mapping of the data vectors

The final step of the mesh refinement considers the mapping of the data vectors from the old to the newly refined mesh. In case the temperature method is considered, a typical data vector contains the temperature. The mapping takes advantage of the manner in which the refined mesh is constructed. The most important aspect of course is the fact that each consecutive LOR further refines the subelements that results from a previous refinement. In this way, a newly refined basis element with a certain LOR always shares all its nodes with the same basis element at a previous time level and with a higher LOR. If the newly refined element has a higher LOR than before, only a few (linear) interpolations are needed. The mapping of the data has the following structure:

Mapping of data vectors **[mshrefmapping]**

```
do (for each node of the basis mesh)
   make a one-on-one copy of the data
enddo

do (for each basis element)
   if (LOR of old mesh >= LOR of new mesh)
      do (for each common node unequal to those of the basis mesh)
         make a one-on-one copy of the data
      enddo
   else
      do (for each common node unequal to those of the basis mesh)
         make a one-on-one copy of the data
      enddo
      do (for each remaining node of the new mesh)
         do (for each LOR)
            interpolate data
         enddo
      enddo
   endif
enddo
```

To illustrate the mapping procedure, consider the transition from an old refined mesh to a newly refined mesh, as shown in Figure 5.9. In the first step of the mapping, data is copied for nodes '1' to '4'. These nodes are the vertices of the basis elements, and are therefore always part of each mesh. The numbering of these nodes is also always the same. Next is the mapping of basis element '1'. Because in the old mesh the LOR of this basis element is one order higher than that of the new mesh, it suffices to simply perform a one-on-one mapping of the data of nodes '5' to '7' of the old mesh to the nodes '6', '5' and '7' of the new mesh.

At this stage, it is really important to note that the numbering of the internal nodes in the new mesh can be completely different from the numbering of the internal nodes of the old mesh, even though it geometrically concerns the same nodes. This of course is a direct consequence of the possible differences in the assigned LOR of each basis element for each mesh at each time level. It only requires one basis element with a different LOR than before to completely alter the order in which the added nodes are numbered. The essential link between the node numbers of the internal nodes of the old and new mesh is obtained via arrays *intnodesold* and *intnodesnew*.

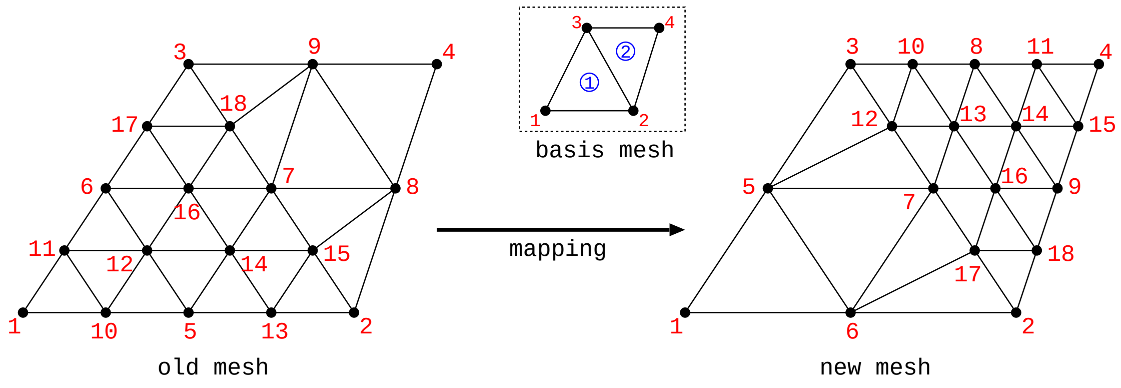


Figure 5.9: Transition from one mesh to another. Due to the way the meshes are constructed, the number of nodes and the number of interpolations is minimal.

The mapping of the data to the internal nodes of the second basis element is slightly more complicated. In the same way as for element '1', first the nodes are mapped that are contained in both the new as well as the old mesh. In case of our example, this concerns nodes '7' to '9'. Surely, the possible difference in numbering between the two meshes, which follows from comparing the corresponding entries in arrays *intnodesold* and *intnodesnew*, is taken into account.

Next, the remaining internal nodes of the newly refined second basis element are mapped by means of (linear) interpolation. In order to determine the correct node numbers of the parent nodes, array *parentnodes* is used. Some caution is advised, in particular when the difference in LOR between the old and the new mesh is larger than one, because the order in which the nodes are interpolated then becomes important. This is the reason why the node numbers of the internal nodes are stored in sequence of increasing LOR in auxiliary array *intnodes*.

5.4.4 Adaptation of the time step size

The performance of both the numerical scheme and the local refinement code is affected by the size of the time step. Clearly, the time step size should not be chosen too large, because this can lead to instability of the numerical scheme (for instance in case an explicit time integration method is used) or it could affect the capturing of physical phenomena. Setting the time step size to too small a value is mostly impractical and often unnecessary. Therefore, even though the choice of the initial time step size is reasonable, it is important to include some means of time step control into the code.

In case divergence of the iterative solver is detected, a sound approach is to attempt to solve the governing equation again after the the time step size has been halved and the time level has been reset to a previous time instance. Of course, one cannot halve the time step indefinitely. Therefore, the maximum number of refinements is set to be equal to 16. If the time step has become too small, the computation is terminated.

If the (initial) time step size is too large, a situation can arise in which (part of) the moving interface has advanced to elements having a lower LOR than the elements that were previously intersected. Inevitably, this results in a loss of information and thus reduces the accuracy of the numerical solution. This so-called *interface outside refinement* problem is addressed next.

5.4.5 Interface outside refinement

At any given moment during a computation, a situation may arise in which the mesh, and possibly the time step size, are to be adapted. If for instance the time step size is relatively large, a moving interface might advance outside the band of refinement. Another possibility is that a new interface is created due to a source term or a change in boundary conditions. In any of these circumstances, proper actions need to be taken to limit any possible loss of accuracy.

Consider the solution to a moving boundary problem at time level $t = t_n$, on a refined grid as shown in Figure 5.10a. The time step size is taken to be Δt. Next, the numerical solution at time level $t_{n+1} = t + \Delta t$ is computed on the given grid. In case the obtained solution at $t = t_{n+1}$ is such that the moving interface remained within the same basis element as before (and no new interface has appeared), as shown in Figure 5.10b, neither the mesh nor the time step size need to be modified. If the moving interface advances to a neighboring basis element which has an equal LOR as the previously intersected basis element, a local re-meshing is sufficient (Figure 5.10c).

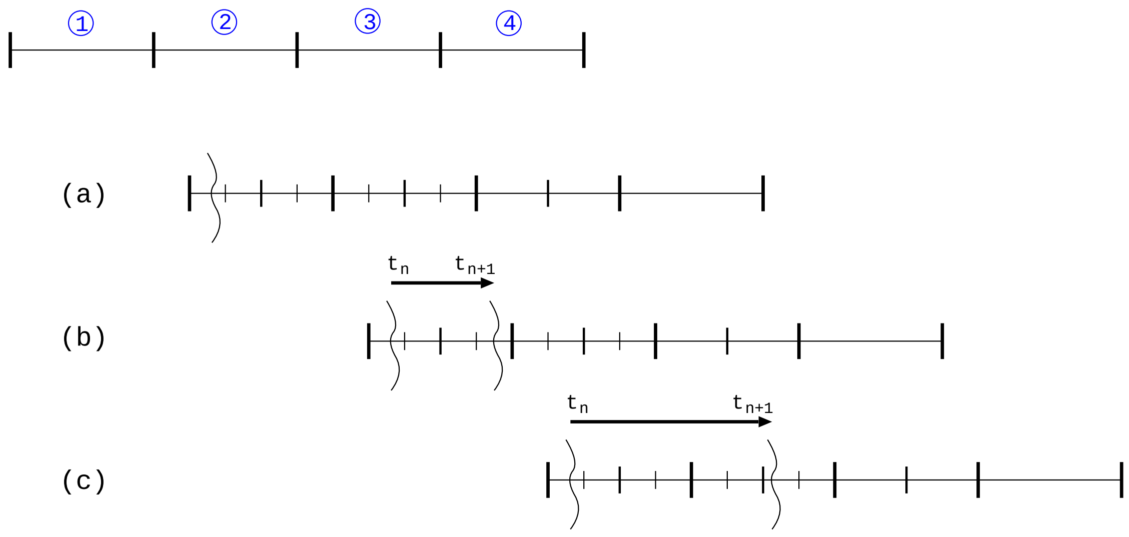


Figure 5.10: Basic interface progression. (a) Initial interface position. (b) The moving interface remains within the same basis element. (c) The moving interface advances to a neighboring basis element with equal LOR.

However, in any of the following three instances, illustrated in Figure 5.11, a more strategic approach is required:

I the moving interface advances to a basis element which has a lower LOR than before (either still within the band of refinement, or worse, outside the band of refinement)

II the moving interface remains in a basis element which has the same LOR as before, but a new interface has appeared in a basis element with a lower LOR

III the moving interface advances to a basis element which has a lower LOR than before and a new interface has appeared in a basis element with a lower LOR

Each of the above 'interface outside refinement' situations can easily be detected. All that is required is a comparison of the (at most) N_{basis} entries of the arrays *refineold* and *refinenew*. As soon as an entry of array *refinenew* contains a value equal to LOR$_{max}+1$ (which indicates that this particular basis element is now intersected by a moving interface, see Section 5.4.1), while the corresponding entry in array *refineold* contains a lower value, the search is terminated, and a 'interface outside refinement' warning is given.

Although an 'interface outside refinement' situation is easily detected, it is far from trivial to determine whether its cause is an advancing or a newly created moving front. Indeed, a means of keeping track of existing front positions, in combination with estimates of the front velocity field, could probably be implemented. However, this will most likely be rather costly.

Another problem is that in either situation, preferably, different actions have to be taken. In case of an existing front, a good strategy is to halve the time step size and then to retry to solve the governing equation on the existing mesh, in order to keep the number of elements (and thus the number of unknowns) limited. If a new front appears, there is no real need to modify the time step size. But, in order to resolve the new interface position at the highest resolution possible, a local re-meshing is required. Of course, in either case, the time level is to be reset to the previous time instance.

At first glance, a reasonable strategy might seem to just perform both a re-meshing and a time step size reduction, whenever an 'interface outside refinement' situation is detected. In this way, the level of accuracy is sustained, although at a fairly high price: an increase in both the number of unknowns as well as the number of time steps to be taken.

Unfortunately, besides the increase in costs, there is another drawback to this strategy. For instance, assume that due to a source term a new moving interface appears in a previously unrefined mesh, see Figure 5.12. If we now proceed according to the above strategy, we reduce the time step size, reset the time level to the previous time instance, perform a re-meshing and solve the governing equation on the now locally refined mesh.

As a result of the higher resolution of the mesh, and the smaller time step size, it is not unlikely that at this new time instance, we find a solution for which a front does not yet appear. Since there no longer exists a moving front, it is then determined that a local mesh refinement is not needed. The mesh is once more coarsened and the governing equation is again solved on the coarsest grid. And so, once again a new front is detected, etc. In the worst case, this process of repeatedly refining and coarsening might even cause the computation to be terminated because the time step size is reduced too often.

Hypothetically, an appropriate remedy to this problem is to always maintain the local refinement for at least two or three consecutive time steps and not to reduce the time step size. If during these successive time steps new modifications to the mesh are required, the desired refinement can be obtained by combining arrays *refinenew* and *refineold*. A modified array refinenew is then constructed by setting each entry of the new array refinenew equal to the maximum value of the old refinenew and the corresponding value of array refineold.

However, although in this example it is clear that we are dealing with a new interface and not an advancing front, recall that in general this distinction is not easily made. The problem with this remedy lies in the time step size, which is not reduced. If, for instance, we have a moving front as sketched in Figure 5.12, that has advanced considerably within one time step, say, outside the band of refinement, it definitely not suffices to simply perform a local refinement and not to reduce the time step size. The reason is that in this way physical phenomena, such as the influence of latent heat, are not properly captured.

At this point, it can be concluded that more information is needed in order to derive a generally applicable strategy. Luckily, for our mark formation problem, this additional piece of information

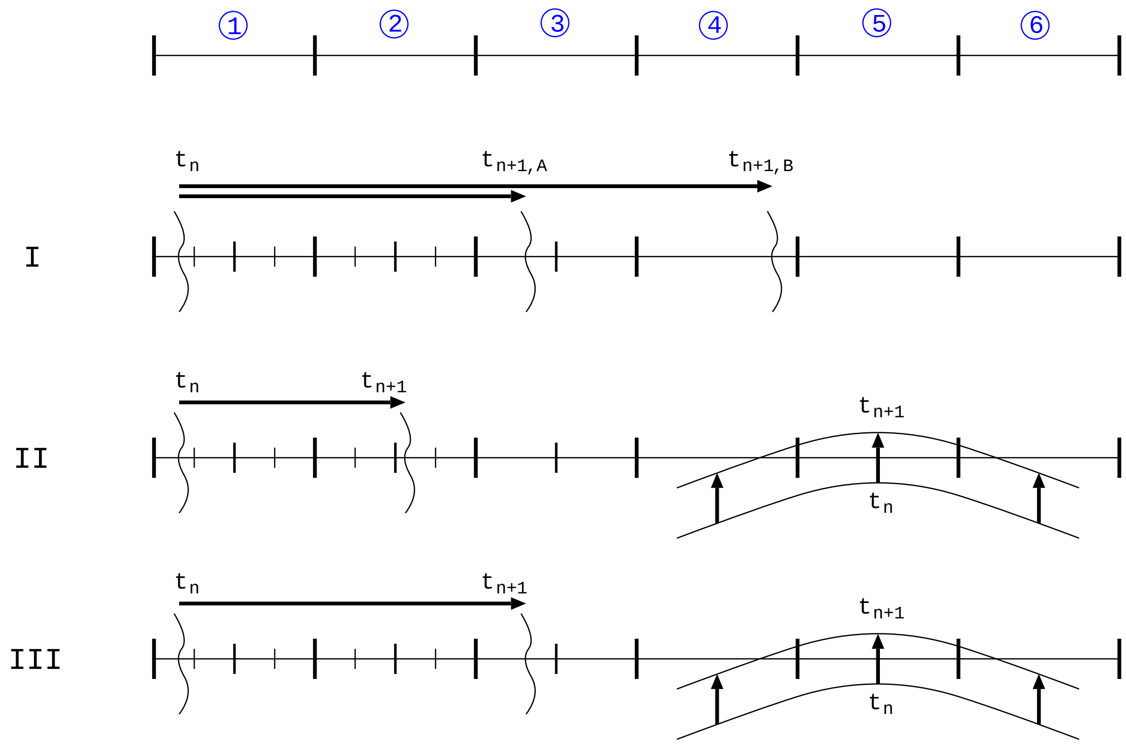


Figure 5.11: More complicated interface progression. (I) The moving interface advances to an element A) within the band of refinement; B) outside the band of refinement. (II)/(III) Besides an advancing existing front, a new interface appears (for instance due to a source).

follows from a basic observation: whenever a moving interface skips one or more elements during a single time step, this automatically means that these skipped elements have undergone a full phase transition. That is, they either went from being fully solid to fully liquid, or vice versa.

Thus, one final array is added to the set of auxiliary arrays. This array, called *phasenew* or *phaseold*, depending on which mesh it refers to, has a length equal to the number of elements of the basis mesh N_{basis}, and each of its entries is assigned an integer value corresponding to the phase of the associated basis element. For instance, if a basis element is fully solid, the position in the array is assigned a value of '0', if the element is fully liquid it contains the value '1'. Otherwise, its value is set to '2'. In order to detect whether at least one basis element skipped its phase transition, all that is needed is a one-on-one comparison of the entries of arrays *phasenew* and *phaseold*.

The general refinement strategy to deal with 'interface outside refinement' situations (which is also applicable to moving boundaries initiated by Dirichlet boundary conditions at $t = t_0$), takes on the following form:

Interface outside refinement procedure

```
solve governing equation on current grid

if ( basis element skipped phase transition ) then
   reduce time step size
endif

if ( interface outside refinement ) then
   if needed, combine refineold and refinenew
   refine mesh
```

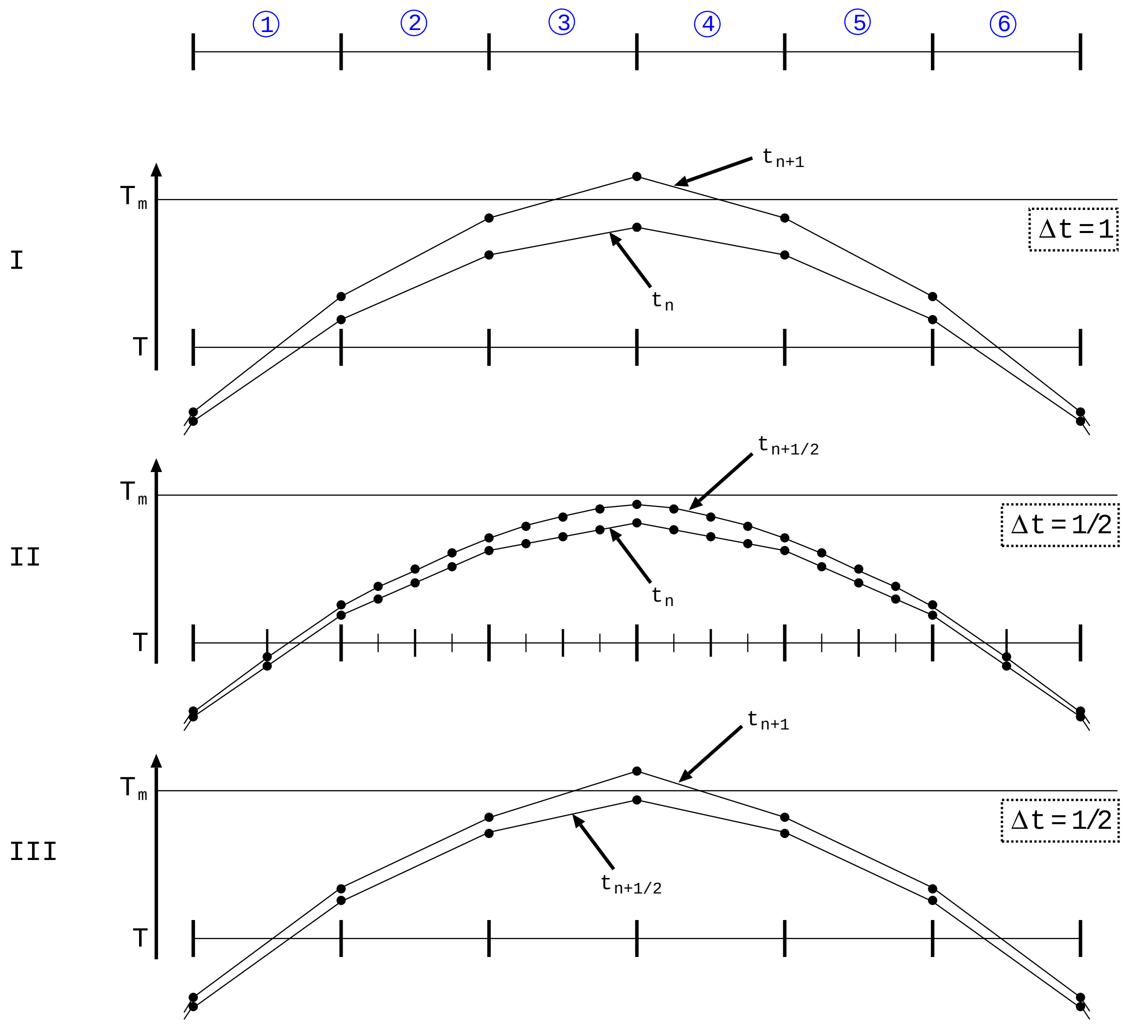


Figure 5.12: Inefficient capturing of a source induced interface. (I) A moving interface is detected at $t = t_{n+1} = t_n + \Delta t$. (II) The time step size Δt is halved and the time level is reset to the previous time instance t_n. The mesh is refined and the numerical solution is computed for $t = t_{n+1/2}$. (III) Because no moving interface is detected, the mesh is coarsened and the numerical solution is computed for $t = t_{n+1}$.

```
endif
```

5.5 Conclusions

In this chapter a local refinement code has been described which is capable of accurately capturing changes in the geometry or the displacements of a moving interface, while preserving the quality of the basis mesh. The data structure developed, and the implementation of the local refinement in the Sepran package, are both efficient and easily understood.

A good strategy for refining the mesh is to always start constructing the new mesh from the basis mesh onward. The intersected elements, as well as the direct neighboring ones, are preferable assigned the highest LOR, while the remaining elements are appointed an LOR which gradually reduces with the distance from the intersected elements. The actual construction of the refined mesh proceeds by first performing a single refinement of all basis elements that are assigned to be refined at least once, then to refine all elements that are appointed an LOR of at least two, etc. In this way, the number of interpolations of the data field are kept at a minimum.

With respect to the basis mesh it is important that the coarseness is such that for a smooth solution an acceptable accuracy is guaranteed. If not, there is no use in refining the mesh locally.

In case solving the governing equation fails within a given maximum number of iterations, a new attempt can be made with a reduced time step size. Of course, the number of reductions of the time step size is to be limited to a reasonable number of times. If for several successive time steps the time step size is not reduced, it can be attempted to increase the time step size.

In order to ensure that physical phenomena are properly captured, it is always first checked whether a basis element did not skip its phase transition. If an element indeed underwent a full change of phase, the time step size is reduced. Whenever a moving interface advances to elements that are assigned a LOR that is lower than LOR$_{max}$, or when a new interface appears, an 'interface outside refinement' situation is created. In order to prevent unnecessary inaccuracies as a result of interpolations, the mesh is locally refined. Finally, in case either an element skipped its phase transition and/or an 'interface outside refinement' situation has occurred, the time level is reset to the previous time instance, and the governing equation is solved anew under the modified conditions.

Chapter 6

Numerical results

In this chapter a selection of numerical results is presented. The main focus of Section 6.2 will be on the differences in performance of the enthalpy and the temperature approach, which have been discussed in Section 3.3 and Chapter 4, respectively. The remainder of this chapter concentrates on the temperature approach. Of special interest are the inclusion of an artificial mushy region, as introduced in Section 4.4, and the application of the local adaptive mesh refinement discussed in Chapter 5.

6.1 Introduction

Throughout this chapter, the emphasis will be on the 'performance' of certain numerical methods. But before a judgment can be made on how well a method performs, it is worthwhile to define the quantities on which to base our assessment.

Since the temperature T is the quantity of interest for our mark formation model, it is interesting to know how accurate the numerical solution actually is. To this end, the numerical solution is compared to a reference solution T_{ref}, which is either a known similarity solution or a numerically obtained solution. The error in the approximated solution $T(x,t)$ is measured as

$$\text{Err}_{(i[,j])} = \frac{\|T_{\text{ref}} - T(x,t)\|_i}{\|T_{\text{ref}}\|_i} \times 100\%. \tag{6.1}$$

The type of norm that is used to measure the error is indicated by the subscript i. Both the 2-norm ($i = 2$) and ∞-norm ($i = \infty$) will be considered. The optional second subscript j is used to distinguish between either the temperature as a function of position ($j = x$) and the temperature as function of time ($j = t$). Errors are always evaluated for $t = t_{\text{end}}$.

Other interesting quantities include the total number of time taken to complete a computation, the total number of (pseudo) Newton iterations, and the average number of iterations per time step. In tables, the abbreviation '*av. #iters*' is used to indicate the average number of Newton iterations. In case the time step size is adapted at least once during a computation, the total number of time steps taken at the end of the computation is given within brackets.

Unless indicated otherwise, time is in seconds, and all temperatures are given in $10^{3\circ}C$, except for Section 6.2, where temperatures are given in $^\circ C$.

6.2 Comparison: Fachinotti vs Nedjar

The performance of the Nedjar method and the Fachinotti scheme has been evaluated using three test cases. The first two test cases consider the melting of a 1D line segment, and are taken from Chun and Park [5]. The third test problem considers the melting of a 2D square region, which consists of a phase-change material, and is embedded on two sides by a second, non-melting material.

Since we are comparing two numerical methods that are essentially quite different, it is appropriate to first address the validity of the comparison.

6.2.1 Validity of the comparison

In the method of Nedjar we solve a linear system for the temperature update ΔT. In the Fachinotti method, we solve a non-linear system for T^{m+1}. Both systems are solved iteratively, but the stopping criterion for each is different. The Fachinotti method is said to have converged when $\|\Psi(T_i^{m+1})\|_\infty < \epsilon \|\Psi(T^m)\|_\infty$, where ϵ is taken to be equal to 10^{-6}. The method of Nedjar is said to have converged when $\|\Delta T_i\|_\infty < \epsilon$, and corresponding results are presented in this section. The reason for this choice of stopping criterion for the Nedjar method is that results for using absolute criterion $\|\Delta T_i\|_\infty < \epsilon \|\Delta T_0\|_\infty$ are found to be equal up to the third or higher significant digits. However, slightly more iterations (and thus more time) are required.

Although the solution methods and their corresponding stopping criterion are essentially different, we believe that the comparison presented in this section is valid. The reason is that for the given value of ϵ, the computed errors with respect to the similarity solution, in both the 2-norm and ∞-norm, are of the same order of magnitude. This argument holds for the temperature as function of time as well as position. In particular, it holds for the test case where the physical parameters in for both phases are equal, most likely because the temperatures are of the order of one. For the test case 'unequal' the errors of the Fachinotti method are even less, for the given tolerance ϵ, which supports why the Fachinotti method is to be preferred.

The reason for choosing the ∞-norm for the measures used in the stopping criteria, is that for melting problems in composite domains, this choice of norm ensures that the solution method converges up to desired accuracy level, in those regions where the actual melting occurs. In particular when these regions are small with respect to the computational domain, the use of the 2-norm would negatively affect the accuracy of the solution, because of too much averaging of the error.

6.2.2 Test case 1: 'equal'

In this first 1D test problem, the thermal properties for the liquid and the solid phase are taken to be constant, and equal for both phases. The diffusivity $\kappa = 2W/m°C$, $c = 2.5 \times 10^6 J/kg°C$ and density $\rho = 1kg/m^3$. The latent heat $L = 1 \times 10^8 J/kg$. The initial temperature at $t = 0$ of the whole domain is $2°C$, the melting temperature $T_m = 0°C$. On one end, a Dirichlet boundary condition is imposed, $T = -4°C$, and on the other end we have the insulation condition $\partial T/\partial t = 0$. The time step size is kept fixed at $\Delta t = 21600s$ (6 hours) and the computation ends at $t = t_{\text{end}} = 2.592.000s$ (30 days).

The computations for the original test case are performed on a spatial grid consisting of 100 elements, with $\Delta x = 0.1m$. The temperature as a function of time for both methods is plotted in Figure 6.1. Notice that the solution, as computed using the method by Nedjar, clearly shows 'staircasing', which is characteristic for enthalpy methods. The temperature method exhibits a similar type of behavior.

Results with respect to a global refinement of the spatial grid are listed in Table 6.1. A benefit of this test case, as well a test case 2, is the existence of a corresponding similarity solution [5], which is used to estimate the errors in the approximated solution. The table clearly shows that in general the error for the temperature as a function of time ($\text{Err}_{(.,t)}$, evaluated at a fixed node located at $x = 0.3m$) is about ten times larger (in both norms) than the error for the temperature as a function of position ($\text{Err}_{(.,x)}$). In addition, the reduction of the errors is linear when the spatial grid size is doubled, as expected. Also, note that for the Nedjar method for $n = 400$ too low a value is found for the interface position x_m, as a result of wiggles.

Table 6.1: Test case 1: 'equal'. The effect of a global refinement of the spatial grid. The 'exact' position of the moving interface at $t = t_{\text{end}}$ is $x_m = 0.587$.

method	n	time	av. #iters	x_m	Err$_{(2,t)}$ (Err$_{(\infty,t)}$)	Err$_{(2,x)}$ (Err$_{(\infty,x)}$)
Nedjar	100	12	28	0.600	12 (25)	0.89 (2.2)
	200	18	55	0.600	5.5 (13)	0.60 (2.0)
	400	44	116	0.575	2.6 (8.0)	0.43 (1.8)
	800	91	137	0.594	1.9 (3.1)	0.22 (0.70)
Fachinotti	100	10	3.2	0.601	8.1 (17)	1.3 (1.7)
	200	10	3.5	0.593	4.2 (7.1)	0.67 (0.98)
	400	13	4.0	0.589	2.2 (3.6)	0.38 (0.76)
	800	18	4.6	0.587	1.3 (2.1)	0.15 (0.19)

6.2.3 Test case 2: 'unequal'

For the second 1D test case, the values of κ and c are taken to be constant, but different for each phase. The thermal properties are: $\kappa_s = 2.22 W/m°C, \kappa_l = 0.556 W/m°C, c_s = 1.762 \times 10^6 J/kg°C, c_l = 4.226 \times 10^6 J/kg°C, L = 3.38 \times 10^8 J/kg$, and $\rho = \rho_s = \rho_l = 1 kg/m^3$. On the whole domain the initial temperature is now $10°C$, while the melting temperature is once more set to be $T_m = 0°C$. The temperature at the left boundary is kept at $-20°C$, the right end is again insulated. Although t_{end} is also set to 30 days, a smaller time step size is taken: $\Delta t = 2000s$.

The results obtained for this second test case are listed in Table 6.2. In comparison to the previous test problem, the overall performance of both methods is comparable. Remark that for both test cases, it holds that the computational load, expressed by the average number of (pseudo)-Newton iterations and the total computational time, for the Nedjar method scales with the increase of the number of elements. For the temperature based method, the computational load hardly increases when the mesh is globally refined.

An interesting observation can be made from Table 6.3, in which the performance of both methods is compared for varying Stefan numbers $St = c_l(T_0 - T_m)/L$, where c_l is the conductivity of the liquid phase and T_0 the ambient temperature. For problems for which the melting front moves very rapidly, i.e., the Stefan number $St \approx \mathcal{O}(10^2)$, the enthalpy based method seems to be the method of choice. However, for the phase-change materials that are used in optical rewritable recording, a Stefan number of $\mathcal{O}(10^{-3})$ is commonly found. These problems can thus best be resolved using the method of Fachinotti.

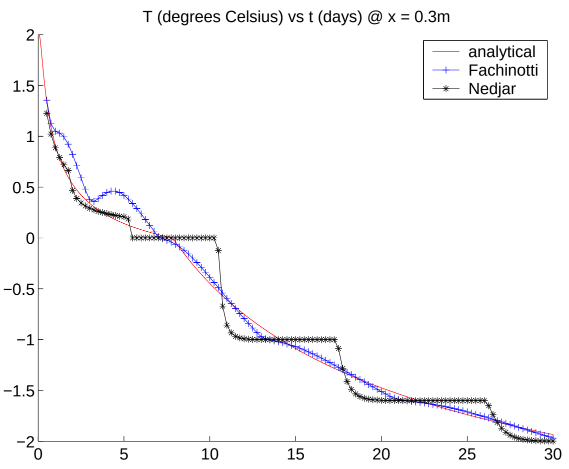


Figure 6.1: Test case 1: 'equal'. Difference between Nedjar and Fachinotti for the temperature as function of time in a point $x = 0.3$m. $n = 100$, $\epsilon = 10^{-6}$.

Table 6.2: Test case 2: 'unequal'. The effect of a global refinement of the spatial grid. The 'exact' position of the moving interface at $t = t_{\text{end}}$ is $x_m = 0.742$.

method	n	time	av. #iters	x_m	$\text{Err}_{(2,t)}$ ($\text{Err}_{(\infty,t)}$)	$\text{Err}_{(2,x)}$ ($\text{Err}_{(\infty,x)}$)
Nedjar	100	107	11	0.700	11 (30)	0.98 (2.1)
	200	131	19	0.750	4.7 (18)	0.56 (2.0)
	400	223	36	0.725	2.0 (9.4)	0.31 (1.4)
	800	564	72	0.738	0.85 (5.3)	0.06 (0.64)
Fachinotti	100	102	3.0	0.748	2.1 (6.0)	0.49 (1.4)
	200	110	3.1	0.744	0.88 (3.2)	0.15 (0.39)
	400	128	3.5	0.743	0.41 (1.6)	0.08 (0.25)
	800	171	4.0	0.743	0.20 (0.75)	0.04 (0.19)

Table 6.3: Test case 2: 'unequal'. Performance with respect to Stefan number St, $\epsilon = 10^{-3}$, $\Delta t = 2000$, $n = 800$, $t_{\text{end}} = 20$ days.

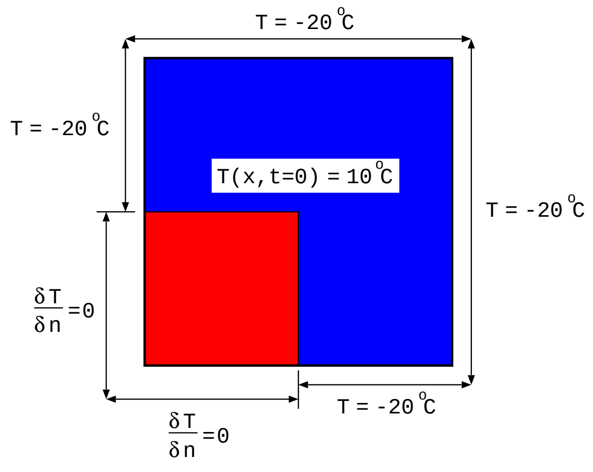
Stefan number St		$\approx 5 \times 10^{-4}$	$\approx 5 \times 10^{-2}$	$\approx 5 \times 10^{0}$	$\approx 5 \times 10^{2}$
total #Newton	Fachinotti	2322	2829	4957	6603
iterations	Nedjar	78947	43045	14396	6271
total time	Fachinotti	23	29	45	58
	Nedjar	355	195	132	32

6.2.4 Test case 3: '2D corner'

The third test problem considers the melting of a square region, occupied by the same material as that of test case 2: 'unequal', which is on two sides enclosed by a non-melting material, see Figure 6.2. For the embedding material, $\rho = 1$ kg/m^3, $C = 1{\times}10^6$ J/kg°C, $\kappa = 0.5$ W/m°C, $L = 2{\times}10^4$ J/kg and $T_m = 20$ °C.

Table 6.4 shows that the computational demand of the Fachinotti method does not increase as rapidly for smaller values of the tolerance ϵ then that of the Nedjar method. This is caused by the damping factor μ in the pseudo Newton iteration process.

In Figure 6.3, contour levels are plotted of the temperature within the composite domain. The figure illustrates the difference in the captured interface position. Although the moving interface, represented by contour level 5, is very smooth when using the temperature approach, Nedjar



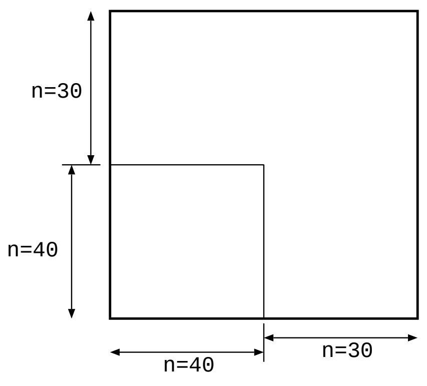


Figure 6.2: Test case 3: '2D corner'. Initial and boundary conditions (left). Number of grid points used (right).

clearly shows oscillations.

Table 6.4: Test case 3: '2D corner'. Computational load for varying tolerance ϵ: $\Delta t = 2000$, $t_{\text{end}} = 20$ days.

		$\epsilon = 10^{-6}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-2}$
total #Newton	Fachinotti	3376	2599	1734
iterations	Nedjar	88964	46020	10107
total time	Fachinotti	208	166	116
	Nedjar	2030	1093	266

6.2.5 Nedjar: Newton-Cotes vs Gauss rule

The results shown for the above test cases have all been obtained using first order Newton-Cotes integration. As such, the plot of the solution for test case 'equal', i.e., Figure 6.1, for the original mesh size ($n = 100$), is found to be equal to results presented by, for instance, Chun and Park [5]. However, the solution obtained on the same grid using a 3-points Gauss rule, has quite a different shape, see Figure 6.4.

Because of the observed difference, the computations for the various grid spacings have been performed anew for test cases 1 and 2, using Gauss integration. The results obtained are listed in Tables 6.5 and 6.6. When compared to the results that are found using Newton-Cotes integration, i.e., Tables 6.1 and 6.2, it can be concluded that in general, the use of a Gauss rule is much more costly: both the average number of iterations as well as the total time are (much) higher. With respect to the level of accuracy, not much difference between both integration rules can be discerned.

6.2.6 Conclusions

Based on the results, the temperature based method seems to be the method of choice for small and medium range Stefan numbers. Not only can it easily be integrated into existing finite element codes for conduction problems in composite domains, it also outperforms the enthalpy based method with respect to accuracy, stability and computational load. For melting problems with Stefan number of $O(10^2)$ or larger, the enthalpy method might be the better choice.

With respect to spatial integration, the use of first order Newton-Cotes is preferred over the use of a 3-point Gauss rule.

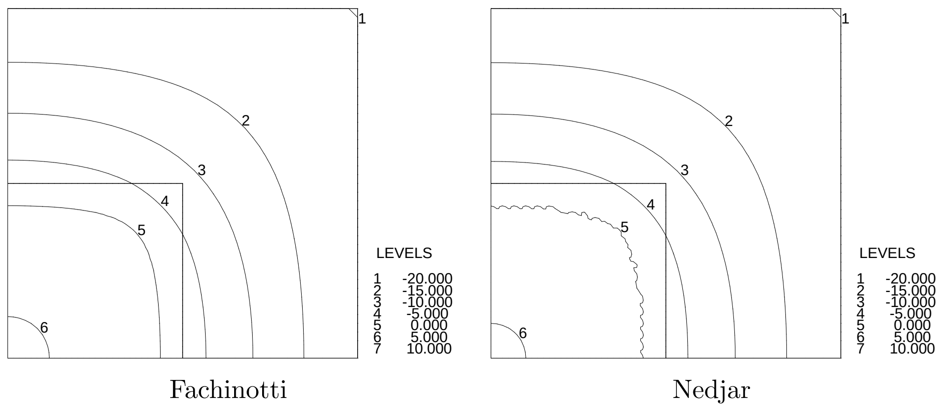


Figure 6.3: Test case 3: '2D corner'. Fachinotti: no wiggles; Nedjar: wiggles. $\epsilon = 10^{-3}$, $\Delta t = 2000$, $t_{\text{end}} = 20$ days.

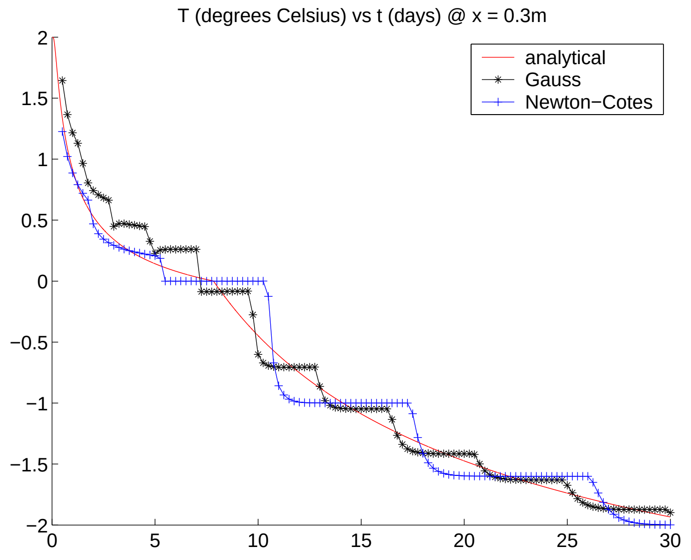


Figure 6.4: Test case 1: 'equal'. Influence of the choice of integration rule for the Nedjar method on the temperature as function of time in a point $x = 0.3$m. $n = 100$, $\epsilon = 10^{-6}$.

Table 6.5: Test case 1: 'equal'. Gauss rule. The 'exact' value of $x_m = 0.587$.

n	time	av. #iters	x_m	$\text{Err}_{(2,t)}$ $(\text{Err}_{(\infty,t)})$	$\text{Err}_{(2,x)}$ $(\text{Err}_{(\infty,x)})$
100	17	79	0.621	11 (18)	1.5 (1.9)
200	32	143	0.589	5.2 (8.6)	0.72 (1.2)
400	54	144	0.607	2.8 (6.5)	0.36 (0.69)
800	84	122	0.581	2.3 (4.4)	0.35 (1.4)

6.3 Temperature approach

This section is dedicated to the Fachinotti method. Numerical results will be discussed concerning the introduction of an artificial mushy region and the application of adaptive local refinement of the mesh.

6.3.1 Line search

In Section 4.3 it is explained how convergence of the Newton-Raphson method can be improved, by means a line searching algorithm. Of particular interest is the order of the polynomial that approximates the function $g(\alpha)$ as defined in (4.26), and which is used to determine the linesearch parameter α. In practice, the use of a linear polynomial sometimes leads to non-convergence of the Newton scheme. No noticeable difference in performance can be discerned when using either a cubic or quadratic polynomial. Since the cubic approximation is slightly more costly, the use of a quadratic approximation polynomial is preferred.

Table 6.6: Test case 2: 'unequal'. Gauss rule. The 'exact' value of $x_m = 0.742$.

n	time	av. #iters	x_m	$\text{Err}_{(2,t)}$ $(\text{Err}_{(\infty,t)})$	$\text{Err}_{(2,x)}$ $(\text{Err}_{(\infty,x)})$
100	152	54	0.721	5.4 (15)	1.2 (2.7)
200	244	80	0.682	2.4 (9.0)	0.22 (0.68)
400	572	139	0.745	1.4 (8.5)	0.14 (0.34)
800	1831	264	0.737	1.0 (4.1)	0.16 (0.72)

Table 6.7: Definitions of the various source distributions and values for the position of the moving interface and the maximum temperature.

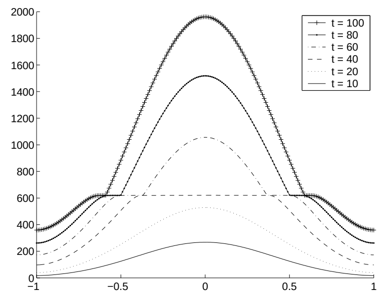
short notation	actual function ($A = 3.5 \cdot 10^2$)	x_m	$T_{\max}$
exp2	$A \exp\left(-\frac{1}{2}x^2/\sigma^2\right)$	0.58890	1.96416
exp4	$A \exp\left(-\frac{1}{2}x^2/\sigma^4\right)$	0.25950	1.38633
2exp4	$A\left[\exp\left(-\frac{1}{2}(x-\frac{1}{4})^2/\sigma^4\right) + \exp\left(-\frac{1}{2}(x+\frac{1}{4})^2/\sigma^4\right)\right]$	0.51344	1.61716
const	A	X	2.22568

6.3.2 Artificial mushy region

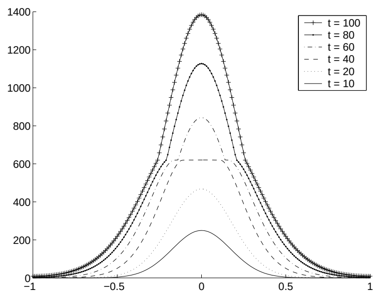
Let us consider the performance of the Fachinotti method in case a source is added to the problem formulation. Because the isothermal approach fails for this type of problems, an artificial mushy region is introduced, see Section 4.4. A pivotal parameter in this, is the 'radius' of the mushy zone T_Δ. In this section, we will limit our study to 1D problems. The source functions that are used in the experiments are listed in Table 6.7.

The physical parameters of the test problems considered are taken such that they resemble the values used in simulations of the recording of a DVD[9]. The values for these parameters are: $\kappa_s = \kappa_l = 0.006$, $c_s = c_l = 12.85$, $L = 6400$ and $T_m = 620$. (These values are scaled to be used in the computer code, and therefore can differ from those in Reference [9]). At $t = 0$, the whole domain [-1, 1] is at a temperature of 0 $°C$, thus solid, and the boundaries are taken to be adiabatic. For this configuration, the reference solutions, corresponding to the four source functions from Table 6.7, are plotted in Figure 6.5 at selected time instances between $t = 0$ and $t = t_{\text{end}} = 100$.

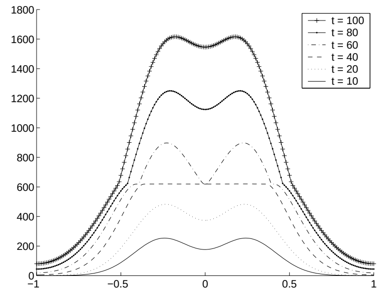
In case of isothermal problems that are driven by a non-zero source term, the mushy parameter T_Δ is preferable chosen such that the solution to the regularized problem resembles the isothermal solution as much as possible. In order to get a good impression on how to choose T_Δ, a parameter study has been performed, of which the results are listed in Table 6.8. In this table, as in future



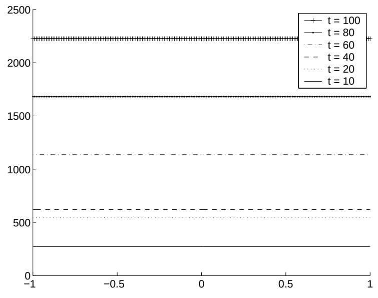
(a)



(b)



(c)



(d)

Figure 6.5: Source functions. (a) exp2, (b) exp4, (c) 2exp4, (d) const. (For definitions, see Table 6.7).

Table 6.8: The effect of varying the mushy parameter T_Δ (in % of the melting temperature T_m). $\Delta t = 1s$, $n = 100$, $\epsilon = 10^{-6}$.

function	T_Δ	time	av. #iters	x_m	$\text{Err}_{(2,x)}$ ($\text{Err}_{(\infty,x)}$)	$T_{\max}$
exp2	10	7.8	2.4	0.602	1.089 (1.92)	1.96018
	0.1	8.3	4.7	0.595	0.120 (0.319)	1.96236
	0.001	11	9.9 (102)	0.621	0.119 (0.342)	1.96242
exp4	10	7.8	2.7	0.263	0.739 (0.730)	1.37975
	0.1	8.4	3.9	0.261	0.277 (0.264)	1.38267
	0.001	8.5	4.9	0.261	0.277 (0.265)	1.38265
2exp4	10	7.9	2.8	0.518	0.416 (0.509)	1.61298
	0.1	8.3	4.7	0.516	0.165 (0.229)	1.61520
	0.001	10	7.4	0.516	0.165 (0.230)	1.61522
const	10	7.5	1.0	X	0 (0)	2.22568
	0.1	7.5	1.2	X	0 (0)	2.22568
	0.001	7.6	1.2	X	0 (0)	2.22568

Table 6.9: The effect of varying the threshold parameter ϵ. $\Delta t = 1s$, $n = 100$, $T_\Delta = 0.1\%$ of T_m.

function	ϵ	time	av. #iters	x_m	$\text{Err}_{(2,x)}$ ($\text{Err}_{(\infty,x)}$)	$T_{\max}$
exp2	10^{-4}	8.5	5.2	0.595	0.12 (0.32)	1.96236
	10^{-2}	8.2	4.2	0.595	0.12 (0.32)	1.96234
	10^{-1}	8.0	3.2	0.596	0.20 (0.32)	1.96106
exp4	10^{-4}	8.0	3.4	0.261	0.28 (0.26)	1.38267
	10^{-2}	7.8	2.7	0.261	0.29 (0.28)	1.38246
	10^{-1}	7.8	2.2	0.260	0.49 (0.44)	1.38031
2exp4	10^{-4}	8.3	4.1	0.516	0.17 (0.23)	1.61520
	10^{-2}	8.1	3.5	0.516	0.17 (0.24)	1.61521
	10^{-1}	7.9	2.9	0.515	0.29 (0.33)	1.61403
const	10^{-4}	7.5	1.2	X	0 (0)	2.22568
	10^{-2}	7.5	1.2	X	0 (0)	2.22568
	10^{-1}	7.5	1.2	X	0.05 (0.05)	2.22686

tables, x_m denotes the left most point at which the moving interface intersects the melting temperature. For the 'exact' values of x_m and the maximum temperature $T_{\max}$ at $t = t_{\text{end}}$ we refer to Table 6.7.

An interesting observation that can be made from Table 6.8 is that the approximated solutions are in general rather accurate (the errors are less than 1% in both norms). Furthermore, the mushy parameter T_Δ can be taken surprisingly large, without much loss of accuracy. It seems that the shape of the source function plays an important role in the performance of the method: in case of multiple appearing fronts (2exp4) or functions with steep gradients near the front (exp2), the performance of the temperature method is affected more for smaller values of T_Δ, than in case the other source functions are applied. It can be concluded that $T_\Delta = 0.1\%$ of T_m is a save choice for most problems.

In the previous test, the threshold parameter ϵ, which is used in the stopping criterion for the Newton scheme, has been kept exceptionally small ($\epsilon = 10^{-6}$). In Table 6.9 results are listed on the performance for increasing values of ϵ. The mushy parameter T_Δ is taken to be equal to 0.1% of T_m. From the table it follows that for values of ϵ smaller than 10^{-2}, the gain in accuracy becomes negligible. Therefore, a value of 10^{-3} is a reasonable choice for parameter ϵ.

In case both ϵ and T_Δ are kept fixed, it is interesting to see how the performance is affected when the spatial and the time grid are globally refined at the same. From Table 6.10 it can be concluded that the reduction of the error of the approximated solution is linear with respect to halving both the time step and the number of elements. As expected, the total amount of time

Table 6.10: Fachinotti: the effect of varying the time step size Δt and number of elements n. $T_\Delta = 0.1\%$ of T_m, $\epsilon = 10^{-3}$.

function	Δt	n	time	av. #iters	x_m	$\text{Err}_{2,x}$ ($\text{Err}_{\infty,x}$)	$T_{\max}$
exp2	1	100	8.4	4.7	0.595	0.12 (0.32)	1.96236
	0.5	200	18	4.0	0.590	0.05 (0.14)	1.96320
	0.25	400	42	3.6	0.590	0.03 (0.10)	1.96367
exp4	1	100	7.9	3.1	0.261	0.28 (0.26)	1.38266
	0.5	200	17	2.9	0.260	0.14 (0.15)	1.38450
	0.25	400	39	3.0	0.260	0.06 (0.07)	1.38549
2exp4	1	100	8.2	3.9	0.516	0.17 (0.23)	1.61519
	0.5	200	18	3.8	0.515	0.09 (0.19)	1.61613
	0.25	400	42	3.3	0.514	0.04 (0.05)	1.61664
const	1	100	7.6	1.2	X	0 (0)	2.22568
	0.5	200	16	1.0	X	0 (0)	2.22568
	0.25	400	34	1.1	X	0 (0)	2.22568

Table 6.11: Nedjar: the effect of varying the time step size Δt and number of elements n. $\epsilon = 10^{-3}$, Gauss-rule.

function	Δt	n	time	av. #iters	x_m	$\text{Err}_{(2,x)}$ ($\text{Err}_{(\infty,x)}$)	$T_{\max}$
exp2	1	100	11	31	0.638	0.12 (0.34)	1.96239
	0.5	200	28	32	0.635	0.06 (0.15)	1.96323
	0.25	400	69	25	0.631	0.04 (0.12)	1.96370
exp4	1	100	9.4	23	0.264	0.38 (0.84)	1.38265
	0.5	200	23	24	0.262	0.16 (0.41)	1.38448
	0.25	400	54	17	0.259	0.09 (0.26)	1.38542
2exp4	1	100	9.9	25	0.516	0.16 (0.28)	1.61518
	0.5	200	25	29	0.513	0.12 (0.42)	1.61613
	0.25	400	58	22	0.515	0.06 (0.26)	1.61665
const	1	100	8.5	29	X	0 (0)	2.22568
	0.5	200	20	37	X	0 (0)	2.22568
	0.25	400	47	34	X	0 (0)	2.22568

required for the computation also scales accordingly. The average number of Newton iterations is almost constant for each source function. The same effect on the performance is observed for the Nedjar method, see Table 6.11. Not surprisingly, after our comparison made in Section 6.2, the enthalpy approach shows to be computationally more demanding and less accurate.

One might wonder to what extent the latent heat affects the maximum value of the temperature distribution when compared to the same problem, but with no latent heat. For the given parameters, this difference appears to be of the order of 20%, see Table 6.12.

6.3.3 Adaptive local mesh refinement

The adaptive local mesh refinement code, discussed in Chapter 5 has been tested using test cases 1 and 2 from Section 6.2. The results can be found in Tables 6.13 and 6.14. The number of elements of the basis mesh is taken to be $n = 100$. In the tables, the maximum LOR assigned to the basis elements intersected by the moving interface is denoted by *reflvl*. Since $n = 100$ for the basis mesh, *reflvl* $= 2$ means that locally, the grid spacing is the same as for $n = 400$. The maximum number of elements used during the computation is denoted by $n_{\max}$.

Even though in some cases the total number of time steps taken slightly increases (as a result of time step size reductions), there is a noticeable gain in computational time. Moreover, the obtained accuracy in case of *local* refinement is almost identical to the level of accuracy acquired

Table 6.12: Fachinotti: the difference in maximum temperature $T_{\max}$ for latent heat L = 6400 and L = 0. $\Delta t = 1s$, $n = 100$, $T_\Delta = 0.1\%$ of T_m, $\epsilon = 10^{-3}$.

function	$T_{\max}$ $(L = 6400)$	$T_{\max}$ $(L = 0)$	difference
exp2	1.96236	2.40911	19 %
exp4	1.38266	1.71917	20 %
2exp4	1.61519	2.05155	21 %
const	2.22568	2.72374	18 %

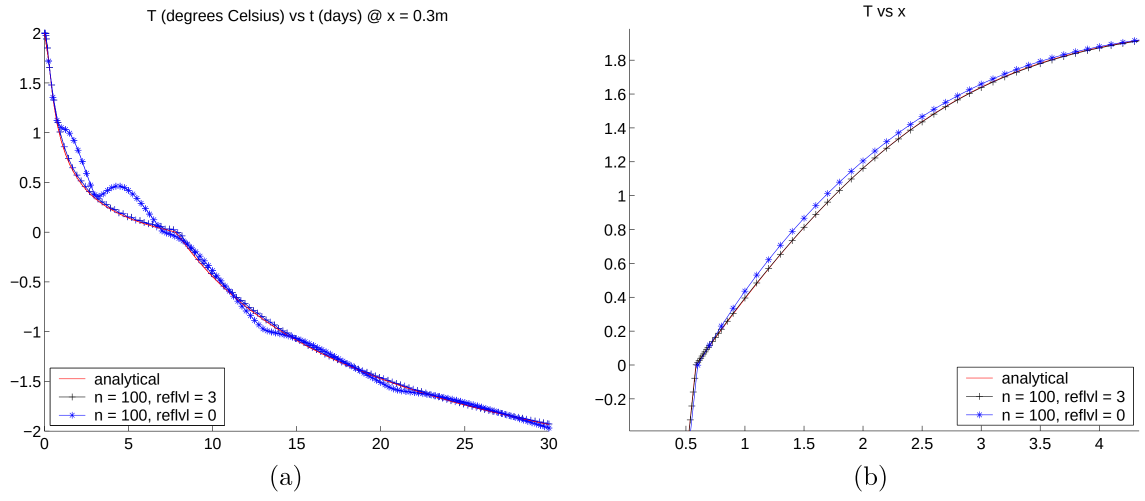


Figure 6.6: Test case 1: 'equal'. The influence of the local adaptive mesh refinement for $n = 100$, using the Fachinotti method. (a) T as a function of time (in days) at the point $x = 0.3$m , (b) T as a function of position x (zoomed in) at time $t = t_{\text{end}}$.

when applying a *global* refinement. Even more interesting is that an increase of the level of refinement hardly affects the computational load. We refer to Figure 6.6 for an illustration of the effect of the mesh refinement for a basis mesh of 100 elements and a *reflvl* of three.

Table 6.13: Test case 1: 'equal'. The effect of adaptive local mesh refinement. The 'exact' value of $x_m = 0.587$. The maximum number of elements used during the computation is denoted by $n_{\max}$. The bottom row shows the results in case it is not checked whether the moving interface skips one or more elements during a single time step.

$n_{\max}$	reflvl	time	av. #iters	x_m	$\text{Err}_{(2,t)}$ $(\text{Err}_{(\infty,t)})$	$\text{Err}_{(2,x)}$ $(\text{Err}_{(\infty,x)})$
100	0	9.6	3.2	0.601	8.1 (17)	1.3 (1.7)
103	1	9.8	3.6 (123)	0.593	4.2 (7.5)	0.74 (0.98)
111	2	10	4.1 (127)	0.589	2.2 (3.9)	0.54 (0.78)
129	3	11	4.5 (131)	0.587	1.4 (2.8)	0.20 (0.20)
129	3	10	4.5	0.587	1.6 (2.6)	0.22 (0.19)

Table 6.14: Test case 2: 'unequal'. The effect of adaptively refining the mesh. The 'exact' value of $x_m = 0.742$. The maximum number of elements used during the computation is denoted by $n_{\max}$.

$n_{\max}$	reflvl	time	av. #iters	x_m	$\mathrm{Err}_{(2,t)}$ ($\mathrm{Err}_{(\infty,t)}$)	$\mathrm{Err}_{(2,x)}$ ($\mathrm{Err}_{(\infty,x)}$)
100	0	102	3.0	0.748	2.1 (6.0)	0.49 (1.4)
103	1	103	3.1	0.744	0.91 (3.2)	0.19 (0.39)
111	2	105	3.5	0.743	0.42 (1.6)	0.14 (0.25)
129	3	108	4.1	0.743	0.21 (0.76)	0.10 (0.20)

Bibliography

[1] V. Alexiades and A.D. Solomon. *Mathematical modeling of melting and freezing processes.* Hemisphere Publishing Corporation, 1993.

[2] J.H. Brusche. Numerical modeling of optical phase-change recording. Internal report 04-04, Technische Universiteit Delft, 2004.

[3] J.H. Brusche, A. Segal, and H.P. Urbach. Finite-element model for phase-change recording. *J. Opt. Soc. Am. A*, 22:773–786, 2005.

[4] J.H. Brusche, A. Segal, and C. Vuik. A study of enthalpy methods for mark formation modeling. Internal report 06-04, Technische Universiteit Delft, 2006.

[5] C.K. Chun and S.O. Park. A fixed-grid finite-difference method for phase-change problems. *Num. Heat Transf., Part B*, 38:59–73, 2000.

[6] L.A. Crivelli and S.R. Idelson. A temperature-based finite element solution for phase-change problems. *Int. J. Numer. Meth. Engng.*, 23:99–119, 1986.

[7] V.D. Fachinotti, A. Cardano, and A.E. Huespe. A fast convergent and accurate temperature model for phase-change heat conduction. *Int. J. Numer. Meth. Engng.*, 44:1863–1884, 1999.

[8] B. Nedjar. An enthalpy-based finite element method for nonlinear heat problems involving phase change. *Computers and Structures*, 80:9–21, 2002.

[9] C. Peng, L. Cheng, and M. Mansuripur. Experimental and theoretical investigations of laser-induced crystallization and amorphization in phase-change optical recording media. *J. Appl. Phys.*, 82(9):4183–4191, 1997.

[10] Numerical Recipes Software. *Numerical recipes in fortran 77: the art of scientific computing.* Cambridge university press, 1992. ISBN 0-521-43064-X.

[11] SEPRAN is a finite element package of SEPRA (Ingenieursbureau), Den Haag, the Netherlands. (www.sepra.nl).