Finite Element Discretisation of a Problem in Nonlinear Non-Fickian Viscoelastic Diffusion Using a Discontinuous Galerkin Method in Space

Roswitha Bultmann & Simon Shaw

November 2005
www.brunel.ac.uk/bicom
Technical Report 05/5

Note added by supervisor

The following pages contain the MSc dissertation completed, under my supervision, by Roswitha Bultmann in summer 2005. Our aim was to create a resource for DG FEM similar to the "*50 lines of matlab*" implementation of the CG FEM in [ACF99]. However, note that due to the more intricate nature of the DG FEM, we did not try for a similarly small number of lines.

The reason for including this preliminary note is that this report, and in particular the numerical results in Chapter 5, needs to be referenced by [RS] where an error estimate for the nonlinear non-Fickian diffusion problem is proven. However, the numerical results given in Tables 5.6, 5.7, 5.9 and 5.10 are misleading in that they don't convincingly show second-order convergence in time. In hindsight, and as is so often the case, the reason for this was obvious but neither of us were able to find it until after the course submission deadline.

The 'correct' versions of the tables are shown, in the same order, in the next section.

Simon Shaw, Brunel University, November 2005.

Revised numerics

It is not stated anywhere in the following chapters, but the time interval for computations was taken as $I = (0, T) := (0, 1)$.

In the practical implementation the quantity γ_E^*, defined in Section 5.2, is computed element-by-element by quadrature. To improve the numerical results all that was required was to increase the value of M so that the time discretisation error dominated this quadrature error.

The new versions of the tables are shown in Tables 1, 2, 3 and 4.

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	0.088458		0.088458	
2	0.025616	1.7879	0.025616	1.7879
4	0.006599	1.9568	0.006599	1.9568
8	0.001661	1.9903	0.001661	1.9903
16	0.000417	1.9951	0.000417	1.9951
32	0.000105	1.9889	0.000105	1.9889
64	0.000027	1.9549	0.000027	1.9549

Table 1: Tabulated errors for $u(\boldsymbol{x}, t) = t^3 x$, $u_{RG} = 0.5$, $M = 8$, $\delta = 10^4$ and $\beta = 2$ (revised version of Table 5.6).

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	0.088458		0.088458	
2	0.025617	1.7879	0.025617	1.7879
4	0.006607	1.9551	0.006607	1.9551
8	0.001664	1.9889	0.001664	1.9889
16	0.000417	1.9958	0.000417	1.9958
32	0.000105	1.9929	0.000105	1.9929
64	0.000027	1.9684	0.000027	1.9684

Table 2: Tabulated errors for $u(\boldsymbol{x}, t) = t^3 x$, $u_{RG} = 0.9$, $M = 8$, $\delta = 10^4$ and $\beta = 2$ (revised version of Table 5.7).

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	0.088491		0.088491	
2	0.025631	1.7877	0.025631	1.7877
4	0.006604	1.9565	0.006604	1.9565
8	0.001673	1.9810	0.001673	1.9810
16	0.000452	1.8877	0.000452	1.8877
32	0.000153	1.5638	0.000153	1.5638
64	0.000032	2.2775	0.000032	2.2775

Table 3: Tabulated errors for $u(\boldsymbol{x}, t) = (t + 1)^3 x$, $u_{RG} = 4$, $M = 16$, $\delta = 10^4$ and $\beta = 2$ (revised version of Table 5.9).

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	0.088491		0.088491	
2	0.025624	1.7880	0.025624	1.7880
4	0.006609	1.9551	0.006609	1.9551
8	0.001665	1.9889	0.001665	1.9889
16	0.000417	1.9964	0.000417	1.9964
32	0.000105	1.9952	0.000105	1.9952
64	0.000027	1.9581	0.000027	1.9581

Table 4: Tabulated errors for $u(\boldsymbol{x}, t) = (t+1)^3 x$, $u_{RG} = 7.9$, $M = 16$, $\delta = 10^4$ and $\beta = 2$ (revised version of Table 5.10).

Finite Element Discretisation of a Problem in Nonlinear Non-Fickian Viscoelastic Diffusion Using a Discontinuous Galerkin Method in Space

A dissertation submitted for the degree of

Master of Science
in
Computational Mathematics with Modelling

by

Roswitha Bultmann

November 2, 2005

Department of Mathematical Sciences

Brunel University, UK

Abstract

A MATLAB 7.0.1 implementation for a (spatially) discontinuous Galerkin finite element method is developed. After having implemented the numerical solution to Poisson's equation, the code is extended to the heat equation, a linear diffusion problem and, finally, to a nonlinear non-Fickian diffusion problem. Numerical experiments demonstrate the accuracy and flexibility of the code. Several possibilities for the graphical representation of the numerical solution are also given.

Contents

Acknowledgements

I would like to thank my supervisor Dr. Simon Shaw for his excellent guidance and advice throughout this dissertation. Furthermore, I would like to thank my family and friends for their love and their (financial and moral) support. This would not have been possible without you!

Chapter 1

Introduction

1.1 The Problem

In this dissertation we want to develop a MATLAB 7.0.1 implementation for a spatially discontinuous Galerkin finite element method (DG FEM). This work is based on [ACF99], which provides a MATLAB implementation for the continuous finite element method. The problem we want to solve using the DG FEM is a nonlinear non-Fickian diffusion problem. Motivated by a model proposed by Cohen *et al.* in [CJW95], Rivière and Shaw consider a nonlinear model of non-Fickian diffusion in viscoelastic polymers in [RS]. Their model is as follows.

For $\Omega \subset \mathbb{R}^d$, $d = 2$ or 3, and $I := (0, T)$, $T > 0$, find $u : \Omega \times I \to \mathbb{R}$ and $\boldsymbol{\sigma} : \Omega \times I \to \mathbb{R}^d$ such that in $\Omega \times I$

$$u_t(\boldsymbol{x}, t) - \nabla \cdot D\nabla u(\boldsymbol{x}, t) = f(\boldsymbol{x}, t) + \nabla \cdot K\boldsymbol{\sigma}(\boldsymbol{x}, t), \tag{1.1}$$

$$\boldsymbol{\sigma}_t(\boldsymbol{x}, t) + \gamma(u)\boldsymbol{\sigma}(\boldsymbol{x}, t) = \mu\nabla u(\boldsymbol{x}, t), \tag{1.2}$$

subject to the initial conditions

$$u(\boldsymbol{x}, 0) = \breve{u}(\boldsymbol{x}) \quad \text{and} \quad \boldsymbol{\sigma}(\boldsymbol{x}, 0) = \breve{\boldsymbol{\sigma}}(\boldsymbol{x}), \tag{1.3}$$

and the boundary conditions

$$u(\boldsymbol{x}, t) = 0 \text{ on } \Gamma_D \times I \quad \text{and} \quad (D\nabla u(\boldsymbol{x}, t) + K\boldsymbol{\sigma}(\boldsymbol{x}, t)) \cdot \nu(\boldsymbol{x}) = g(\boldsymbol{x}, t) \text{ on } \Gamma_N \times I, \tag{1.4}$$

where $\Gamma_D \subset \partial\Omega$ is closed with positive surface measure, $\Gamma_N = \partial\Omega \backslash \Gamma_D$ and $\nu(\boldsymbol{x})$ is the unit outward normal vector at $\boldsymbol{x} \in \Gamma_N$. Here D, K and μ are constant and positive and, for constants $\gamma_R \gg \gamma_G > 0$, $\Delta > 0$ and $u_{RG} \in \mathbb{R}$,

$$\gamma(u) = \frac{1}{2}(\gamma_R + \gamma_G) + \frac{1}{2}(\gamma_R - \gamma_G) \tanh\left(\frac{u - u_{RG}}{\Delta}\right). \tag{1.5}$$

Rivière and Shaw provide numerical schemes using both the symmetric and the non-symmetric interior penalty DG FEM in space and a Crank-Nicolson type discretisation in time. Furthermore, they propose two different approaches to handling the nonlinearity, $\gamma(u)$, one leading to a nonlinear numerical scheme, the other leading to a linear numerical scheme. We will only consider the linear scheme for the

case $\Omega \subset \mathbb{R}^2$. Beginning with the simple problem of Poisson's equation we will extend the code step by step to the diffusion problem. The functions u and $\boldsymbol{\sigma}$ will be approximated by, respectively, piece-wise linear and piece-wise constant functions.

The next section gives a short overview of definitions and some notation we will need for the DG FEM approximations of the problems. Chapter 2 describes the implementation of Poisson's equation in two dimensions. In Chapter 3 this code is extended to the heat equation. Then, in Chapter 4 and 5, the implementation of the linear and nonlinear diffusion problem is described. The conclusions in Chapter 6 complete this dissertation.

For details on the DG FEM see, for example, [Bau00], [RW99] or, for a DG time stepping method, [Tho97]. For information on MATLAB and the pre-defined functions used in this dissertation refer to [Mat04] and [MH03].

1.2 The Discontinuous Galerkin Method - Preliminaries and Notation

Let $\{\mathcal{E}_h\}$ denote a family of triangular meshes, which we require to be nondegenerate and quasiuniform. Let $\mathcal{E}_h = \{E_1, \ldots, E_{N_h}\}$ and $h = \max_{j\in\{1,\ldots,N_h\}} \operatorname{diam}(E_j)$. Also, let $\mathcal{S}_h = \{e_1, \ldots, e_{P_h}, e_{P_h+1}, \ldots, e_{M_h}\}$ be the set of edges, where $\Gamma_h = \{e_1, \ldots, e_{P_h}\}$ is the set of interior edges, and let $\mathcal{N}_h = \{\boldsymbol{n}^1, \ldots, \boldsymbol{n}^{Q_h}\}$ denote the set of nodes. With each edge, e, we associate a unit normal vector, ν_e, which is taken to be the unit outward vector normal to $\partial\Omega$ if the edge is on the boundary. Define the broken spaces

$$\begin{aligned}
\mathcal{D}_r(\mathcal{E}_h) &:= \big\{v \in L^2(\Omega) : \ v|_{E_j} \ \text{ is a polynomial of degree } \leq r \\
&\qquad \forall j \in \{1, \ldots, N_h\}\big\}, \\
\boldsymbol{\mathcal{D}}_r(\mathcal{E}_h) &:= \mathcal{D}_r(\mathcal{E}_h)^d, \\
\mathcal{H}^s(\mathcal{E}_h) &:= \big\{v \in L^2(\Omega) : \ v|_{E_j} \in H^s(E_j) \quad \forall j \in \{1, \ldots, N_h\}\big\},
\end{aligned}$$

and the broken norms associated with these spaces

$$|||v|||_s := \left(\sum_{j=1}^{N_h} \|v\|^2_{s,E_j}\right)^{\frac{1}{2}}.$$

Note that the functions which belong to the broken spaces may have discontinuities along interior edges of the mesh. Finally, if E_i^e and E_j^e are neighbouring elements, sharing the edge e, such that ν_e points from E_i^e to E_j^e, define the jump of the function v on e as

$$[v] := \begin{cases} (v|_{E_i^e})|_e - (v|_{E_j^e})|_e & \text{if } e \subset \Omega, \\ (v|_{E^e})|_e & \text{if } e \subset \partial\Omega, \end{cases}$$

and the average of v on e as

$$\{v\} := \begin{cases} \frac{1}{2}(v|_{E_i^e})|_e + \frac{1}{2}(v|_{E_j^e})|_e & \text{if } e \subset \Omega, \\ (v|_{E^e})|_e & \text{if } e \subset \partial\Omega. \end{cases}$$

Chapter 2

Poisson's Equation

2.1 The Model Problem and Variational Formulation

In this section we summarise some theoretical results from [Bau00]. The notation is taken from [RS]. For more details and proofs of the given results refer to [Bau00].

The problem considered in this chapter is Poisson's equation in two dimensions with homogeneous Dirichlet and nonhomogeneous Neumann boundary conditions: find $u \in H^2(\Omega)$ such that

$$\begin{aligned} -\nabla \cdot D\nabla u &= f \qquad \text{in } \Omega, && (2.1)\\ u &= 0 \qquad \text{on } \Gamma_D, && (2.2)\\ D\nabla u \cdot \nu &= g \qquad \text{on } \Gamma_N, && (2.3) \end{aligned}$$

where $\Omega \subset \mathbb{R}^2$, $\Gamma_D \subset \partial\Omega$ is closed and of positive length, $\Gamma_N = \partial\Omega \backslash \Gamma_D$, ν is the outward unit normal vector, $f \in L^2(\Omega)$ and $g \in L^2(\Gamma_N)$. We require $u \in H^2(\Omega)$ to ensure that the solution u is a continuous function and observe that if $u \in H^2(\Omega)$ satisfies equations (2.1)–(2.3), then, for any $v \in \mathcal{H}^1(\mathcal{E}_h)$,

$$\sum_{E\in\mathcal{E}_h} \int_E D\nabla u \cdot \nabla v \, d\boldsymbol{x} - \sum_{e\in\Gamma_h\cup\Gamma_D} \int_e \{D\nabla u \cdot \nu_e\}[v] \, ds = \int_\Omega fv \, d\boldsymbol{x} + \int_{\Gamma_N} gv \, ds. \quad (2.4)$$

Equation (2.4) looks like a weak formulation of (2.1)–(2.3). Nonetheless, we add a penalty term and 'add zero' to the left hand side.

For $\delta > 0$ and $\beta \geq 1$, define the bilinear form $A: \ \mathcal{H}^2(\mathcal{E}_h) \times \mathcal{H}^2(\mathcal{E}_h) \to \mathbb{R}$ by

$$\begin{aligned} A(w,v) := &\sum_{E\in\mathcal{E}_h} \int_E D\nabla w \cdot \nabla v \, d\boldsymbol{x} + \sum_{e\in\Gamma_h\cup\Gamma_D} \frac{\delta}{|e|^\beta} \int_e [w][v] \, ds \\ &- \sum_{e\in\Gamma_h\cup\Gamma_D} \int_e \{D\nabla w \cdot \nu_e\}[v] \, ds + \kappa \sum_{e\in\Gamma_h\cup\Gamma_D} \int_e \{D\nabla v \cdot \nu_e\}[w] \, ds, \quad (2.5) \end{aligned}$$

where κ can be chosen to be 1 or -1 to obtain a non-symmetric or symmetric bilinear form, respectively. Define also the linear form $L:\ \mathcal{H}^1(\mathcal{E}_h) \to \mathbb{R}$ by

$$L(v) = \int_\Omega fv\ d\boldsymbol{x} + \int_{\Gamma_N} gv\ ds.$$

Using this notation we get the following result. If $u \in H^2(\Omega)$ satisfies equations (2.1)–(2.3), then, for any $v \in \mathcal{H}^2(\mathcal{E}_h)$,

$$A(u, v) = L(v).$$

Substituting u_h for u and $\mathcal{D}_1(\mathcal{E}_h)$ for $H^2(\Omega)$ and $\mathcal{H}^2(\mathcal{E}_h)$, we get the discrete formulation of problem (2.1)–(2.3): find $u_h \in \mathcal{D}_1(\mathcal{E}_h)$ such that, for all $v \in \mathcal{D}_1(\mathcal{E}_h)$,

$$A(u_h, v) = L(v). \tag{2.6}$$

2.2 Assembling the Stiffness Matrix and Computing the Load Vector

2.2.1 The Linear System

Before we define a basis for our finite element space $\mathcal{D}_1(\mathcal{E}_h)$ we need to introduce some more notation (cf. [Bau00, Definition 3.1]). Define the functions

$$\sigma:\ \{1, \ldots, N_h\} \times \{1, 2, 3\} \to \{1, \ldots, Q_h\}$$

and

$$\alpha:\ \{1, \ldots, N_h\} \times \{1, 2, 3\} \to \{1, \ldots, M_h\}$$

such that $\boldsymbol{n}^{\sigma(m,j)}$, $j \in \{1, 2, 3\}$, are the nodes of element E_m and the edge $e_{\alpha(m,j)}$ of E_m lies opposite the node $\boldsymbol{n}^{\sigma(m,j)}$. The nodes and edges of E_m are numbered counterclockwise (see Figure 2.1).

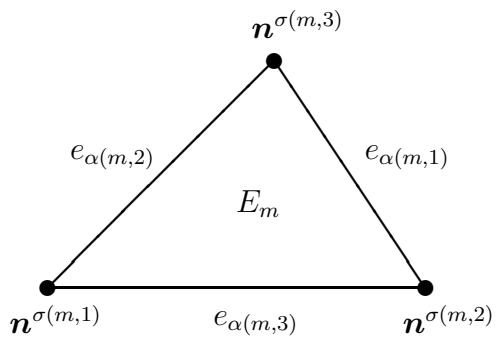

Figure 2.1: Example of an element.

Let $\phi_{mj},\ m \in \{1,\dots,N_h\},\ j \in \{1,2,3\}$, be functions in $\mathcal{D}_1(\mathcal{E}_h)$ such that

$\phi_{mj} \neq 0$ only on E_m for $j = 1,2,3,$ and

$\left(\phi_{mj}\big|_{E_m}\right)\big|_{\boldsymbol{n}^{\sigma(m,k)}} = \delta_{jk}$ for $j,k = 1,2,3.$

Obviously, $\{\phi_{mj}\}$ is a basis of $\mathcal{D}_1(\mathcal{E}_h)$ and the solution u_h to the discrete problem (2.6) can be written as

$$u_h = \sum_{m=1}^{N_h}\sum_{j=1}^{3} U_{3(m-1)+j}\ \phi_{mj}.$$

Define the matrix $\underline{A} = (A_{jk})_{k=1,\dots,3N_h}^{j=1,\dots,3N_h}$ and the vector $\boldsymbol{L} = (L_i)_{i=1,\dots,3N_h}$ by

$$A_{3(n-1)+k,3(m-1)+j} = A(\phi_{mj},\phi_{nk}) \quad \text{and} \quad L_{3(m-1)+j} = L(\phi_{mj}).$$

Then, problem (2.6) can be written as

$$\text{find } \boldsymbol{U} \in \mathbb{R}^{3N_h} \text{ such that } \underline{A}\boldsymbol{U} = \boldsymbol{L}. \tag{2.7}$$

2.2.2 Mesh Data

Suppose Ω is a polygonal domain and $\mathcal{E}_h$ is a regular triangulation of Ω in the sense that there exists a constant $\tau > 0$ such that

$$\frac{h}{\operatorname{diam}(E)} \leq \tau \quad \forall\ E \in \mathcal{E}_h.$$

Five matrices are needed to describe $\mathcal{E}_h$. The columns of the matrix `coordinates` are the x- and y-coordinates of the nodes of the given mesh. The number of each row corresponds to the number of the node. For each element the matrix `elements` contains the numbers of its vertices. Here, the number of the row corresponds to the number of the element. The matrices `dirichlet`, `neumann` and `interior` describe the edges which lie in Γ_D, Γ_N and Γ_h, respectively. Each row contains the numbers of the edge's endpoints, the number of the element it belongs to and its number (1, 2 or 3) with respect to the element (cf. Figure 2.1). For interior edges, which belong to two elements, we use the format

node1 node2 element# edge# element# edge#.

To illustrate this, consider the square $(0,1)\times(0,1) \subset \mathbb{R}^2$. The domain is subdivided into eight triangles as shown in Figure 2.2. The nodes are numbered from 1 to 9 and the elements are numbered from 1 to 8. The matrices now look like:

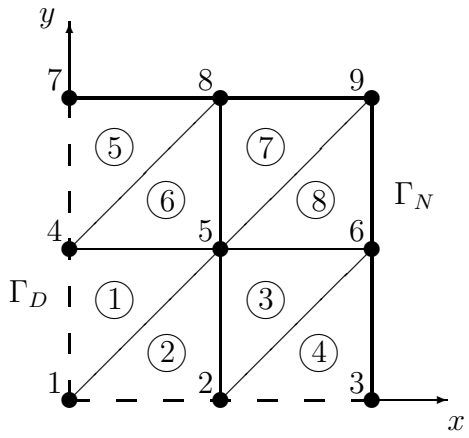

Figure 2.2: Example of a mesh.

```
coordinates
  0    0
0.5    0
  1    0
  0  0.5
0.5  0.5
  1  0.5
  0    1
0.5    1
  1    1
```

```
elements
1  5  4
1  2  5
2  6  5
2  3  6
4  8  7
4  5  8
5  9  8
5  6  9
```

```
dirichlet
7  4  5  2
4  1  1  2
1  2  2  3
2  3  4  3
```

```
neumann
3  6  4  1
6  9  8  1
9  8  7  1
8  7  5  1
```

```
interior
1  5  1  3  2  2
2  5  2  1  3  2
2  6  3  3  4  2
4  8  5  3  6  2
5  8  6  1  7  2
5  9  7  3  8  2
4  5  1  1  6  3
5  6  3  1  8  3
```

2.2.3 The Stiffness Matrix

Consider

$$A(\phi_{mj},\phi_{nk}) = \sum_{E\in\mathcal{E}_h}\int_E D\nabla\phi_{mj}\cdot\nabla\phi_{nk}\,d\boldsymbol{x} + \sum_{e\in\Gamma_h\cup\Gamma_D}\frac{\delta}{|e|^\beta}\int_e[\phi_{mj}][\phi_{nk}]\,ds$$

$$-\sum_{e\in\Gamma_h\cup\Gamma_D}\int_e\{D\nabla\phi_{mj}\cdot\nu_e\}[\phi_{nk}]\,ds+\kappa\sum_{e\in\Gamma_h\cup\Gamma_D}\int_e\{D\nabla\phi_{nk}\cdot\nu_e\}[\phi_{mj}]\,ds. \quad (2.8)$$

We want to express $A(\phi_{mj},\phi_{nk})$ in terms of local basis functions $\widetilde{\phi}_i,\ i=1,2,3$, on the standard triangle $T=\{(s,t)\in\mathbb{R}^2:\ s,t>0,\ s+t<1\}$. Only the results are

given here. For more details on the calculations refer to [Bau00, Section 3.4]. The

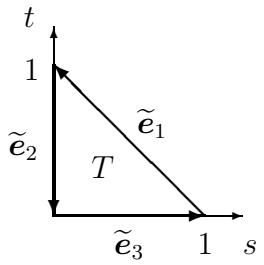

Figure 2.3: The standard triangle.

local basis functions $\widetilde{\phi}_i : \mathbb{R}^2 \to \mathbb{R}$ are defined as

$$\widetilde{\phi}_1(s,t) = 1 - s - t, \qquad \widetilde{\phi}_2(s,t) = s \qquad \text{and} \qquad \widetilde{\phi}_3(s,t) = t.$$

We can map from the standard triangle to any element E_m using

$$\boldsymbol{x}(s,t) = J_m \begin{pmatrix} s \\ t \end{pmatrix} + \boldsymbol{d}_m,$$

where

$$J_m = \left(\boldsymbol{n}^{\sigma(m,2)} - \boldsymbol{n}^{\sigma(m,1)}, \quad \boldsymbol{n}^{\sigma(m,3)} - \boldsymbol{n}^{\sigma(m,1)}\right) \quad \text{and} \quad \boldsymbol{d}_m = \boldsymbol{n}^{\sigma(m,1)}.$$

Thus, we have

$$\phi_{mj}(\boldsymbol{x}(s,t)) = \begin{cases} \widetilde{\phi}_j(s,t) & \text{if } (s,t) \in T, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\nabla \phi_{mj} = J_m^{-T} \nabla \widetilde{\phi}_j.$$

Let

$$\widetilde{\boldsymbol{e}}_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \quad \widetilde{\boldsymbol{e}}_2 = \begin{pmatrix} 0 \\ -1 \end{pmatrix} \quad \text{and} \quad \widetilde{\boldsymbol{e}}_3 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

denote the three edges of the standard triangle (cf. Figure 2.3) and define

$$D_{\pi/2} := \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}.$$

With this set-up we get for the first term in (2.8)

$$\sum_{E \in \mathcal{E}_h} \int_E D \nabla \phi_{mj} \cdot \nabla \phi_{nk} \, d\boldsymbol{x} \;=\; \frac{D}{2} \delta_{mn} |\det J_m| \left(\nabla \widetilde{\phi}_j\right)^T J_m^{-1} J_m^{-T} \nabla \widetilde{\phi}_k.$$

Defining

$$\left(\nabla\widetilde{\phi}\right) := \left(\nabla\widetilde{\phi}_1 \quad \nabla\widetilde{\phi}_2 \quad \nabla\widetilde{\phi}_3\right),$$

we may write

$$\sum_{E\in\mathcal{E}_h}\int_E D\Big(\nabla\phi_{mj}\cdot\nabla\phi_{nk}\Big)_{\substack{j=1,2,3\\k=1,2,3}} d\boldsymbol{x} \;=\; \frac{D}{2}\delta_{mn}|\det J_m|\left(\nabla\widetilde{\phi}\right)^T J_m^{-1}J_m^{-T}\left(\nabla\widetilde{\phi}\right).$$

A MATLAB implementation of this last term is straightforward. In the following code segments `coord` is a matrix like the matrix `coordinates` defined in the previous section, `elements`, `dirichlet`, `neumann` and `interior` correspond, respectively, to `elements`, `dirichlet`, `neumann` and `interior` in the previous section.

```
s = size(elements,1); j = 1:3;
A1 = sparse(3*s,3*s);
for m = 1:s
    Jm = [coord(elements(m,2),:)-coord(elements(m,1),:);...
        coord(elements(m,3),:)-coord(elements(m,1),:)]';
    B = inv(Jm'*Jm);

    A1(3*(m-1)+j,3*(m-1)+j) = A1(3*(m-1)+j,3*(m-1)+j) + ...
        gphi()'*B*gphi()*det(Jm)*D/2;
end
```

Here, the function `gphi()` returns the matrix $(\nabla\widetilde{\phi})$.

```
function gphi = gphi()
gphi = [-1, 1, 0; -1, 0, 1];
```

The other terms of equation (2.8) are more involved. We consider the case $e \in \Gamma_D$ first. In this case we have

$$\sum_{e\in\Gamma_D}\frac{\delta}{|e|^\beta}\int_e[\phi_{mj}][\phi_{mk}]\;ds \;=\; \sum_{p\in\{1,2,3\}\setminus\{j,k\}}\mathbb{I}(e_{\alpha(m,p)}\in\Gamma_D)\delta|e_{\alpha(m,p)}|^{1-\beta}\frac{2^{\delta_{jk}}}{6}, \quad (2.9)$$

$$-\sum_{e\in\Gamma_D}\int_e\{D\nabla\phi_{mj}\cdot\nu_e\}[\phi_{mk}]\;ds \;=$$

$$\sum_{p=1}^{3}\mathbb{I}(e_{\alpha(m,p)}\in\Gamma_D)D\left(\nabla\widetilde{\phi}_j\right)^T J_m^{-1}D_{\pi/2}J_m\widetilde{\boldsymbol{e}}_p\int_0^1\widetilde{\phi}_k|_{\widetilde{\boldsymbol{e}}_p}\;ds \quad (2.10)$$

and

$$\sum_{e\in\Gamma_D}\int_e\{D\nabla\phi_{mk}\cdot\nu_e\}[\phi_{mj}]\;ds \;=$$

$$-\sum_{p=1}^{3}\mathbb{I}(e_{\alpha(m,p)}\in\Gamma_D)D\left(\nabla\widetilde{\phi}_k\right)^T J_m^{-1}D_{\pi/2}J_m\widetilde{\boldsymbol{e}}_p\int_0^1\widetilde{\phi}_j|_{\widetilde{\boldsymbol{e}}_p}\;ds, \quad (2.11)$$

where

$$\mathbb{I}(I) = \begin{cases} 1 & \text{if the proposition } I \text{ is true,} \\ 0 & \text{if the proposition } I \text{ is false.} \end{cases}$$

In terms of vectors and matrices the expression on the right hand side of equation (2.9) may be written as a linear combination of matrices $\mathrm{P}^p \in \mathbb{R}^{3\times 3}$ such that, for $j, k = 1, 2, 3$,

$$(\mathrm{P}^p)_{jk} = \begin{cases} \frac{1}{3} & \text{if } j = k \text{ and } j, k \neq p, \\ \frac{1}{6} & \text{if } j \neq k \text{ and } j, k \neq p, \\ 0 & \text{otherwise.} \end{cases}$$

So, if we define $\boldsymbol{\phi}_m = \begin{pmatrix} \phi_{m1} & \phi_{m2} & \phi_{m3} \end{pmatrix}$ and apply the jump operator to each component, we get

$$\sum_{e\in\Gamma_D} \frac{\delta}{|e|^\beta} \int_e [\boldsymbol{\phi}_m]^T [\boldsymbol{\phi}_m] \, ds = \sum_{p\in\{1,2,3\}\setminus\{j,k\}} \mathbb{I}(e_{\alpha(m,p)} \in \Gamma_D)\delta |e_{\alpha(m,p)}|^{1-\beta} \mathrm{P}^p.$$

Similarly, since

$$\int_0^1 \widetilde{\phi}_k|_{\widetilde{\boldsymbol{e}}_p} \, ds = \frac{1}{2}(1 - \delta_{kp}),$$

we can define a vector $\boldsymbol{P}^p \in \mathbb{R}^{1\times 3}$ such that, for $k = 1, 2, 3$,

$$(\boldsymbol{P}^p)_k = \begin{cases} \frac{1}{2} & \text{if } k \neq p, \\ 0 & \text{otherwise,} \end{cases}$$

and we may write equation (2.10) as

$$-\sum_{e\in\Gamma_D} \int_e \left\{ D \begin{pmatrix} \nabla\phi_{m1} & \nabla\phi_{m2} & \nabla\phi_{m3} \end{pmatrix}^T \nu_e \right\} [\boldsymbol{\phi}_m] \, ds \; = \; \sum_{p=1}^{3} \mathbb{I}(e_{\alpha(m,p)} \in \Gamma_D) D \left(\left(\nabla\widetilde{\phi} \right)^T J_m^{-1} D_{\pi/2} J_m \widetilde{\boldsymbol{e}}_p \right) \boldsymbol{P}^p.$$

For equation (2.11) we get

$$\sum_{e\in\Gamma_D} \int_e \left(\left\{ D \begin{pmatrix} \nabla\phi_{m1} & \nabla\phi_{m2} & \nabla\phi_{m3} \end{pmatrix}^T \nu_e \right\} [\boldsymbol{\phi}_m] \right)^T \, ds \; = \; -\sum_{p=1}^{3} \mathbb{I}(e_{\alpha(m,p)} \in \Gamma_D) D \left(\left(\left(\nabla\widetilde{\phi} \right)^T J_m^{-1} D_{\pi/2} J_m \widetilde{\boldsymbol{e}}_p \right) \boldsymbol{P}^p \right)^T.$$

In the MATLAB code the matrices `A2a`, `A3a` and `A4a` contain, respectively, the contributions of (2.9), (2.10) and (2.11). The matrix `rotmat` describes the rotation $D_{\pi/2}$ and the columns of `standedges` are the edges of the standard triangle.

```
rotmat = [0, -1; 1, 0]; standedges = [-1, 0, 1; 1, -1, 0];
for l = 1:size(dirichlet,1)
    e = norm(coord(dirichlet(l,1),:)-...
        coord(dirichlet(l,2),:))^(1-beta);
    m = dirichlet(l,3); p = dirichlet(l,4);

    Jm = [coord(elements(m,2),:)-coord(elements(m,1),:); ...
        coord(elements(m,3),:)-coord(elements(m,1),:)]';

    P = ones(2)/6; P(1:3:4) = 2*P(1:3:4);

    A2a(3*(m-1)+j(j~=p),3*(m-1)+j(j~=p)) = ...
        A2a(3*(m-1)+j(j~=p),3*(m-1)+j(j~=p)) + delta*e*P;

    P = ones(1,3)/2; P(p) = 0;
    Q = gphi()'*inv(Jm)*rotmat*Jm*standedges(:,p)*P*D;

    A3a(3*(m-1)+(1:3),3*(m-1)+(1:3)) = ...
        A3a(3*(m-1)+(1:3),3*(m-1)+(1:3)) + Q;
    A4a(3*(m-1)+(1:3),3*(m-1)+(1:3)) = ...
        A4a(3*(m-1)+(1:3),3*(m-1)+(1:3)) + Q';
end
```

For $e \in \Gamma_h$ we have

$$\sum_{e\in\Gamma_h} \frac{\delta}{|e|^\beta} \int_e [\phi_{mj}][\phi_{nk}] = \begin{cases} \sum_{p\in\{1,2,3\}\setminus\{j,k\}} \mathbb{I}(e_{\alpha(m,p)} \in \Gamma_h)\delta |e_{\alpha(m,p)}|^{1-\beta} \frac{2^{\delta_{jk}}}{6} \\ \qquad\qquad \text{if } m = n, \\ -\mathbb{I}(j \neq p \text{ and } k \neq q)\delta |\bar{e}|^{1-\beta} \frac{2^{\mathbb{I}(\boldsymbol{n}^{\sigma(m,j)} = \boldsymbol{n}^{\sigma(n,k)})}}{6} \\ \qquad\qquad \text{if } \partial E_m \cap \partial E_n = e_{\alpha(m,p)} = e_{\alpha(n,q)} = \bar{e}, \\ 0 \qquad\qquad \text{otherwise,} \end{cases} \tag{2.12}$$

$$-\sum_{e\in\Gamma_h} \int_e \{D\nabla\phi_{mj}\cdot\nu_e\}[\phi_{nk}] = \begin{cases} \sum_{p=1}^3 \mathbb{I}(e_{\alpha(m,p)} \in \Gamma_h) \frac{D}{2} \left(\nabla\widetilde{\phi}_j\right)^T J_m^{-1} D_{\pi/2} J_m \widetilde{\boldsymbol{e}}_p \int_0^1 \widetilde{\phi}_k|_{\widetilde{\boldsymbol{e}}_p} \\ \qquad\qquad \text{if } m = n, \\ -\frac{D}{2} \left(\nabla\widetilde{\phi}_j\right)^T J_m^{-1} D_{\pi/2} J_m \widetilde{\boldsymbol{e}}_p \int_0^1 \widetilde{\phi}_k|_{\widetilde{\boldsymbol{e}}_q} \\ \qquad\qquad \text{if } \partial E_m \cap \partial E_n = e_{\alpha(m,p)} = e_{\alpha(n,q)} = \bar{e}, \\ 0 \qquad\qquad \text{otherwise,} \end{cases} \tag{2.13}$$

and

$$\sum_{e\in\Gamma_h}\int_e\{D\nabla\phi_{nk}\cdot\nu_e\}[\phi_{mj}] = \begin{cases} -\sum_{p=1}^3 \mathbb{I}(e_{\alpha(m,p)}\in\Gamma_h)\frac{D}{2}\left(\nabla\widetilde{\phi}_k\right)^T J_m^{-1}D_{\pi/2}J_m\widetilde{\boldsymbol{e}}_p\int_0^1\widetilde{\phi}_j|_{\widetilde{\boldsymbol{e}}_p} \\ \qquad\qquad \text{if } m=n, \\ \frac{D}{2}\left(\nabla\widetilde{\phi}_k\right)^T J_n^{-1}D_{\pi/2}J_n\widetilde{\boldsymbol{e}}_q\int_0^1\widetilde{\phi}_j|_{\widetilde{\boldsymbol{e}}_p} \\ \qquad\qquad \text{if } \partial E_m\cap\partial E_n = e_{\alpha(m,p)} = e_{\alpha(n,q)} = \bar{e}, \\ 0 \qquad\qquad \text{otherwise.} \end{cases} \tag{2.14}$$

The case $m=n$ is similar to the case $e\in\Gamma_D$. For the case $\partial E_m\cap\partial E_n = e_{\alpha(m,p)} = e_{\alpha(n,q)} = \bar{e}$ consider first the term

$$-\mathbb{I}(j\neq p \text{ and } k\neq q)\delta|\bar{e}|^{1-\beta}\frac{2^{\mathbb{I}(\boldsymbol{n}^{\sigma(m,j)}=\boldsymbol{n}^{\sigma(n,k)})}}{6}\ .$$

Again, we want to express this in terms of a matrix $\mathrm{P}^{p,q}\in\mathbb{R}^{3\times3}$ with entries 1/3, 1/6 and 0. We have

$$(P^{p,q})_{jk} = \begin{cases} \frac{1}{3} & \text{if } j\neq p,\ k\neq q \text{ and } \boldsymbol{n}^{\sigma(m,j)} = \boldsymbol{n}^{\sigma(n,k)}, \\ \frac{1}{6} & \text{if } j\neq p,\ k\neq q \text{ and } \boldsymbol{n}^{\sigma(m,j)} \neq \boldsymbol{n}^{\sigma(n,k)}, \\ 0 & \text{otherwise.} \end{cases}$$

In MATLAB the contribution of the interior edge with the number l is as follows.

```
e = norm(coord(interior(l,1),:)-...
    coord(interior(l,2),:))^(1-beta);
m = interior(l,3); p = interior(l,4);
n = interior(l,5); q = interior(l,6);

P = zeros(3,3); P(j(j~=p),j(j~=q)) = ones(2)/6;
cnodes = intersect(elements(m,:),elements(n,:));
i = [find(elements(m,:)==cnodes(1)), ...
    find(elements(m,:)==cnodes(2))];
k = [find(elements(n,:)==cnodes(1)), ...
    find(elements(n,:)==cnodes(2))];
P(i(1),k(1)) = 2*P(i(1),k(1));
P(i(2),k(2)) = 2*P(i(2),k(2));

A2b(3*(m-1)+j(j~=p),3*(n-1)+j(j~=q)) = ...
    A2b(3*(m-1)+j(j~=p),3*(n-1)+j(j~=q)) + ...
    delta*e*P(j(j~=p),j(j~=q));
A2b(3*(n-1)+j(j~=q),3*(m-1)+j(j~=p)) = ...
    A2b(3*(n-1)+j(j~=q),3*(m-1)+j(j~=p)) + ...
    delta*e*P(j(j~=p),j(j~=q));
```

The last two terms we have to consider are (cf. (2.13) and (2.14))

$$-\frac{D}{2}\left(\nabla\widetilde{\phi}_j\right)^T J_m^{-1} D_{\pi/2} J_m \widetilde{\boldsymbol{e}}_p \int_0^1 \widetilde{\phi}_k|_{\widetilde{\boldsymbol{e}}_q}\, ds$$

and

$$\frac{D}{2}\left(\nabla\widetilde{\phi}_k\right)^T J_n^{-1} D_{\pi/2} J_n \widetilde{\boldsymbol{e}}_q \int_0^1 \widetilde{\phi}_j|_{\widetilde{\boldsymbol{e}}_p}\, ds.$$

Their handling is similar to what we have done before for the Dirichlet boundary. We only have to take care of the indices.

```
Jm = [coord(elements(m,2),:)-coord(elements(m,1),:);...
    coord(elements(m,3),:)-coord(elements(m,1),:)]';
Jn = [coord(elements(n,2),:)-coord(elements(n,1),:);...
    coord(elements(n,3),:)-coord(elements(n,1),:)]';

P = ones(1,3)/2; P(q) = 0;
Q = gphi()'*inv(Jm)*rotmat*Jm*standedges(:,p)*P*D/2;

A3b(3*(m-1)+j,3*(n-1)+j) = ...
    A3b(3*(m-1)+j,3*(n-1)+j) + Q;
A4b(3*(n-1)+j,3*(m-1)+j) = ...
    A4b(3*(n-1)+j,3*(m-1)+j) + Q';

P = ones(1,3)/2; P(p) = 0;
Q = gphi()'*inv(Jn)*rotmat*Jn*standedges(:,q)*P*D/2;

A3b(3*(n-1)+(1:3),3*(m-1)+(1:3)) = ...
    A3b(3*(n-1)+(1:3),3*(m-1)+(1:3)) + Q;
A4b(3*(m-1)+(1:3),3*(n-1)+(1:3)) = ...
    A4b(3*(m-1)+(1:3),3*(n-1)+(1:3)) + Q';
```

All that is left to do now is summing up.

```
B = A1 + A3a - kappa*A4a + A2a;
C = A3b - kappa*A4b + A2b;
A = B' - C';
```

Note the transpose at the end. This is because

$$A_{3(n-1)+k,3(m-1)+j} = A(\phi_{mj}, \phi_{nk}).$$

The stiffness matrix A can now be obtained with a call to the function `stiff.m`.

```
A = stiff(kappa,D,beta,delta,coordinates,elements,...
    dirichlet,interior);
```

For the full source code of `stiff.m` refer to Appendix A.

2.2.4 The Load Vector

In order to compute the right hand side of (2.7) we first need to decide on a quadrature rule. We use a three-point Gaussian quadrature formula for functions on the interval $[0,1]$ and a seven-point Gaussian quadrature formula for functions on the standard triangle T. Both quadrature rules are exact for polynomials of up to degree five.

$$\int_0^1 f(s)\ ds \approx \sum_{n=1}^{3} w_n f(y_n). \qquad \int_T f(\boldsymbol{y})\ d\boldsymbol{y} \approx \sum_{n=1}^{7} \hat{w}_n f(\hat{\boldsymbol{y}}_n).$$

For tables giving the quadrature points and weights, see [Bau00, Section 3.5.1]. We have

$$\begin{aligned}
L(\phi_{mj}) &= \int_\Omega f\phi_{mj}\ d\boldsymbol{x} + \int_{\Gamma_N} g\phi_{mj}\ ds \\
&= \int_{E_m} f\phi_{mj}\ d\boldsymbol{x} \\
&\qquad + \sum_{p=1}^{3} \mathbb{I}(e_{\alpha(m,p)} \in \Gamma_N \text{ and } j \neq p) \int_{e_{\alpha(m,p)}} g\ \left(\phi_{mj}|_{E_m}\right)|_{e_{\alpha(m,p)}}\ ds \\
&= |\det J_m| \iint_T f(\boldsymbol{x}(s,t))\widetilde{\phi}_j(s,t)\ dsdt \\
&\qquad + \sum_{p\in\{1,2,3\}\setminus\{j\}} \mathbb{I}(e_{\alpha(m,p)} \in \Gamma_N)|e_{\alpha(m,p)}| \cdot \\
&\qquad\qquad \int_0^1 g(\boldsymbol{n}^{\sigma(m,6-j-p)} + s(\boldsymbol{n}^{\sigma(m,j)} - \boldsymbol{n}^{\sigma(m,6-j-p)}))s\ ds \\
&\approx |\det J_m| \sum_{n=1}^{7} \hat{w}_n f(\boldsymbol{x}(\hat{\boldsymbol{y}}_n))\widetilde{\phi}_j(\hat{\boldsymbol{y}}_n) \\
&\qquad + \sum_{p\in\{1,2,3\}\setminus\{j\}} \mathbb{I}(e_{\alpha(m,p)} \in \Gamma_N)|e_{\alpha(m,p)}| \cdot \\
&\qquad\qquad \sum_{n=1}^{3} w_n y_n g(\boldsymbol{n}^{\sigma(m,6-j-p)} + y_n(\boldsymbol{n}^{\sigma(m,j)} - \boldsymbol{n}^{\sigma(m,6-j-p)})).
\end{aligned}$$

Here, $\boldsymbol{n}^{\sigma(m,j)}$ is the node for which $\phi_{mj}(\boldsymbol{n}^{\sigma(m,j)}) = 1$. Thus, $\phi_{mj}(\boldsymbol{n}^{\sigma(m,6-j-p)}) = 0$.

Using the functions `myquad` and `mydblquad` (see Appendix A, p.73), which implement the three- and seven-point Gaussian quadrature rules, a MATLAB routine which returns the right hand side of equation (2.7) is as follows.

```
function L = L(coord,elements,neumann)
M = zeros(3*size(elements,1),1);
for m = 1:size(elements,1)
    M(3*(m-1)+(1:3)) = det([1,1,1;coord(elements(m,:),:)'])*...
```

```
        mydblquad(@(x) f(map(coord(elements(m,:),:)',x))*...
        phi(x),[0 0; 1 0; 0 1]);
end
for l = 1:size(neumann,1)
    m = neumann(l,3); p = neumann(l,4);
    if p == 2, j = 3:-1:1; else j = 1:3; end
    e = norm(coord(neumann(l,1),:)-...
        coord(neumann(l,2),:));
    P1 = myquad(@(x) g(neumann(l,1:2),x), coord(neumann(l,1),:), ...
        coord(neumann(l,2),:));
    P2 = myquad(@(x) g(neumann(l,1:2),x), coord(neumann(l,2),:), ...
        coord(neumann(l,1),:));
    M(3*(m-1)+j(j~=p)) = M(3*(m-1)+j(j~=p)) + [P1 P2]';
end
L = M;
```

The function `phi.m` returns a row vector containing the values of the local basis functions $\widetilde{\phi}_j$ at the point (s,t),

```
function phi = phi(y)
phi = [1-y(1)-y(2), y(1), y(2)];
```

and the function `map.m` maps from the standard triangle to the element described by the matrix `vertices`.

```
function M = map(vertices,x)
J = [vertices(:,2)-vertices(:,1), vertices(:,3)-vertices(:,1)];
M = J*x + vertices(:,1);
```

2.3 Numerical Experiments and Graphical Representation

As a model problem consider

$$\begin{aligned} -\Delta u &= f \qquad \text{in } \Omega, \\ u &= 0 \qquad \text{on } \Gamma_D, \\ \frac{\partial u}{\partial n} &= g \qquad \text{on } \Gamma_N, \end{aligned}$$

where

$$\Omega = (0,1)\times(0,1) \subset \mathbb{R}^2, \quad \Gamma_D = \{\boldsymbol{x} \in \mathbb{R}^2 : \ x = 0 \text{ or } y = 0\},$$

and

$$\begin{aligned} f(\boldsymbol{x}) &= 8\pi^2 \sin(2\pi x)\sin(2\pi y), \\ g(\boldsymbol{x}) &= \begin{cases} 2\pi\cos(2\pi x)\sin(2\pi y) & \text{if } x = 1, \\ 2\pi\sin(2\pi x)\cos(2\pi y) & \text{if } y = 1. \end{cases} \end{aligned}$$

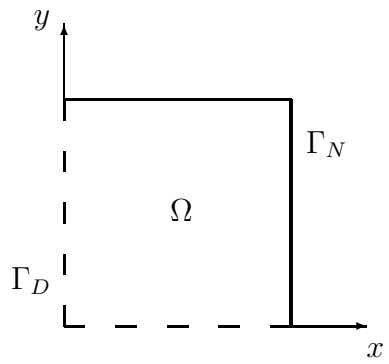

Figure 2.4: The domain Ω.

The exact solution to the problem is

$$u(\boldsymbol{x}) = \sin(2\pi x)\sin(2\pi y).$$

We use meshes as shown in Figure 2.2. Each mesh is described by a parameter $M \in \mathbb{N},\ M \geq 1$, which gives the number of subintervals into which each side of the square is divided. The mesh is refined by increasing M. The values of f and g are obtained with calls to the functions `f.m` and `g.m`.

```
function f = f(x)
f = 8*pi^2*sin(2*pi*x(1))*sin(2*pi*x(2));
```

```
function g = g(edge,x)
global g1
if isG(edge,g1)
    g = 2*pi*cos(2*pi*x(1))*sin(2*pi*x(2));
else
    g = 2*pi*sin(2*pi*x(1))*cos(2*pi*x(2));
end
    function b = isG(x,G)
    if size(setdiff(G,x,'rows'),1) < size(G,1) || ...
        size(setdiff(G,x(end:-1:1),'rows'),1) < size(G,1)
        b = 1;
    else
        b = 0;
    end
    end
end
```

The matrix `g1` contains the edges $e \in \Gamma_N$ which lie on $\{\boldsymbol{x} \in \mathbb{R}^2 :\ x = 1,\ 0 \leq y \leq 1\}$. In the case $M = 2$, as shown in Figure 2.2, `g1` is given by:

```
g1
 3   6
 6   9
```

Using a MATLAB script `mesh.m`, which generates the mesh for a given parameter M, we obtain the numerical solution to our model problem as follows.

```
%% Initialisation
M = 2;
mesh
D = 1;
beta = 2; delta = 10^4;
% kappa = 1 gives a non-symmetric scheme,
% kappa = -1 gives a symmetric scheme
kappa = -1;

%% Assembly
A = stiff(kappa,D,beta,delta,coordinates,elements,...
    dirichlet,interior);

%% Computing the load vector
b = L(coordinates,elements,neumann);

%% Computing the solution
U = A\b;
```

From the definition of the bilinear form $A(\cdot,\cdot)$ follows the definition of the norm

$$\|v\|_{\mathcal{A}} := \left(\sum_{E \in \mathcal{E}_h} \int_E D\nabla v \cdot \nabla v \; d\boldsymbol{x} + \sum_{e \in \Gamma_h \cup \Gamma_D} \frac{\delta}{|e|^\beta} [v]^2 \; ds \right)^{\frac{1}{2}}.$$

In the tables below, approximate values of $\|u - u_h\|_{\mathcal{A}}$ and the estimated order of convergence for arbitrarily chosen values of δ and β are given. The estimated order of convergence is computed as follows.

$$\text{EOC} = \log_2 \left(\frac{\|u - u_{2h}\|_{\mathcal{A}}}{\|u - u_h\|_{\mathcal{A}}} \right).$$

The tables suggest that EOC = 1, i.e. that the error decreases as $O(h)$, where $h = \sqrt{2}/M$ is the maximum diameter of the mesh.

To plot the numerical solution we use the `trisurf` function. The `trisurf` function is used with the following syntax.

```
trisurf(Tri,x,y,z)
```

Here, the vectors `x`, `y` and `z` contain the x-, y- and z-coordinates of points in $\mathbb{R}^3$. The rows of the matrix `Tri` index into these vectors and thus describe triangular faces. Since our numerical solutions are discontinuous, each node of the mesh is associated with more than one function value. Hence, we need to arrange the rows of the matrix `coordinates` in such a way that they correspond to the entries in the vector `U`. Entries one to three of `U` correspond to the first element, entries four to six

	$\kappa = -1$		$\kappa = 1$	
M	$\|u - u_h\|_{\mathcal{A}}$	EOC	$\|u - u_h\|_{\mathcal{A}}$	EOC
1	3.6093		3.6090	
2	3.5498	0.0240	3.5498	0.0239
4	3.0042	0.2408	3.0042	0.2408
8	1.6735	0.8441	1.6735	0.8441
16	0.8630	0.9554	0.8630	0.9554
32	0.4350	0.9884	0.4350	0.9884

Table 2.1: Tabulated errors for $\delta = 10^4$ and $\beta = 2$.

	$\kappa = -1$		$\kappa = 1$	
M	$\|u - u_h\|_{\mathcal{A}}$	EOC	$\|u - u_h\|_{\mathcal{A}}$	EOC
1	6.2702		6.2698	
2	4.1030	0.6118	4.1023	0.6120
4	2.5561	0.6827	2.5561	0.6825
8	1.3044	0.9706	1.3044	0.9706
16	0.6569	0.9896	0.6569	0.9897
32	0.3283	1.0009	0.3282	1.0011

Table 2.2: Tabulated errors for $\delta = 10^{-4}$ and $\beta = 2$.

	$\kappa = -1$		$\kappa = 1$	
M	$\|u - u_h\|_{\mathcal{A}}$	EOC	$\|u - u_h\|_{\mathcal{A}}$	EOC
1	5.0810		4.6159	
2	3.6165	0.4905	3.6511	0.3383
4	3.2211	0.1670	3.2440	0.1706
8	1.7375	0.8906	1.7383	0.9001
16	0.8792	0.9827	0.8791	0.9835
32	0.4389	1.0023	0.4389	1.0022

Table 2.3: Tabulated errors for $\delta = 10$ and $\beta = 2$.

	$\kappa = -1$		$\kappa = 1$	
M	$\|u - u_h\|_{\mathcal{A}}$	EOC	$\|u - u_h\|_{\mathcal{A}}$	EOC
1	5.0810		4.6159	
2	3.5848	0.5032	3.5897	0.3628
4	3.0246	0.2451	3.0244	0.2472
8	1.6747	0.8529	1.6746	0.8528
16	0.8631	0.9563	0.8631	0.9563
32	0.4350	0.9885	0.4350	0.9885

Table 2.4: Tabulated errors for $\delta = 10$ and $\beta = 4$.

	$\kappa = -1$		$\kappa = 1$	
M	$\|u - u_h\|_{\mathcal{A}}$	EOC	$\|u - u_h\|_{\mathcal{A}}$	EOC
1	3.8469		3.8192	
2	3.5583	0.1125	3.5586	0.1020
4	3.0119	0.2405	3.0118	0.2407
8	1.6744	0.8470	1.6744	0.8470
16	0.8631	0.9560	0.8631	0.9560
32	0.4350	0.9885	0.4350	0.9885

Table 2.5: Tabulated errors for $\delta = 10^2$ and $\beta = 3$.

of `U` correspond to the second element, and so forth. In general, entries $3(m-1)+1$ to $3(m-1)+3$ of `U` correspond to the m-th element. Hence, the m-th row of our matrix `Tri` is $(3(m-1)+1,\ 3(m-1)+2,\ 3(m-1)+3)$. The following script plots the numerical solution and adjusts and labels the axes.

```
Y = elements';
Z = zeros(3,size(elements,1)); Z(:) = 1:3*size(elements,1);
trisurf(Z',coordinates(Y(:),1),coordinates(Y(:),2),U,...
    'facecolor','interp')
axis([0 1 0 1 min(U(:)) max(U(:))]), xlabel('x'), ylabel('y')
```

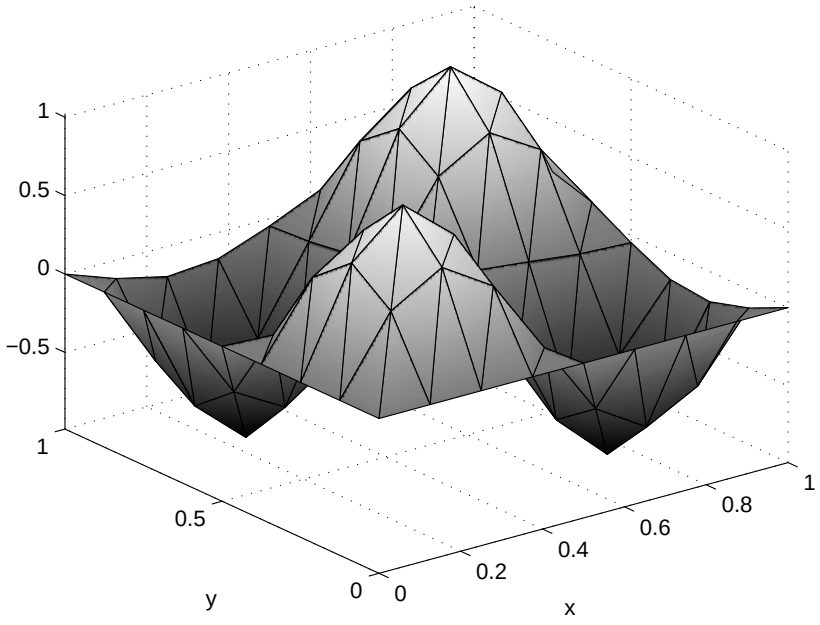

Figure 2.5: Numerical solution for $M = 8$, $\kappa = -1$, $\delta = 10^4$ and $\beta = 2$.

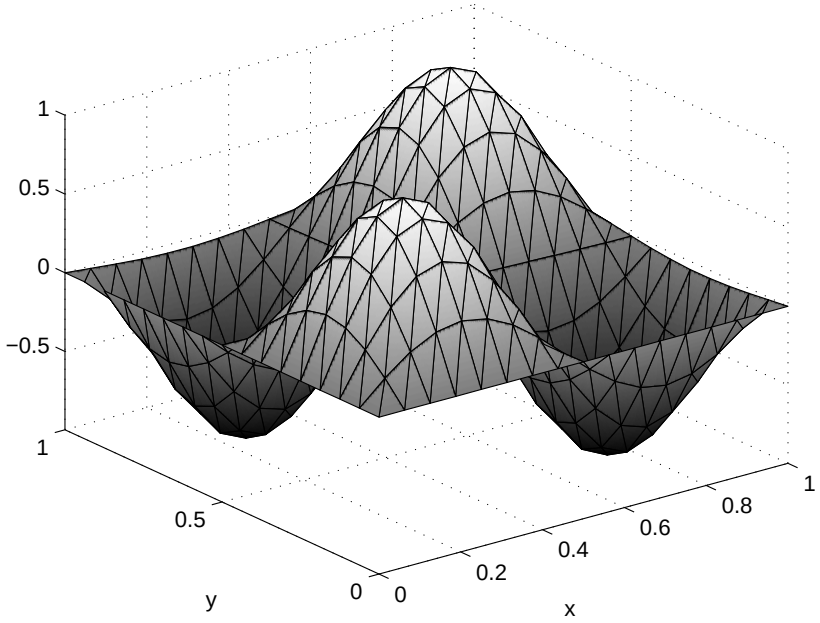

Figure 2.6: Numerical solution for $M = 16$, $\kappa = -1$, $\delta = 10^4$ and $\beta = 2$.

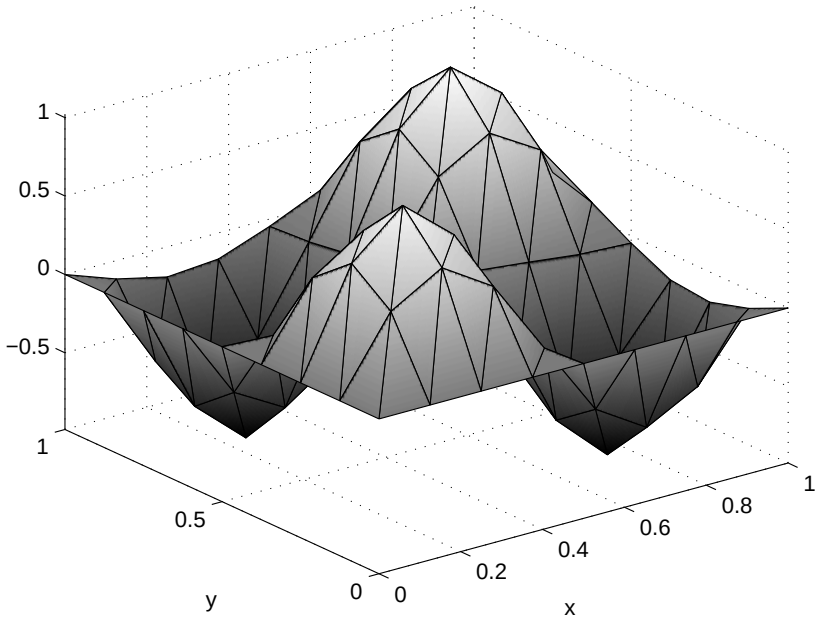

Figure 2.7: Numerical solution for $M = 8$, $\kappa = 1$, $\delta = 10^4$ and $\beta = 2$.

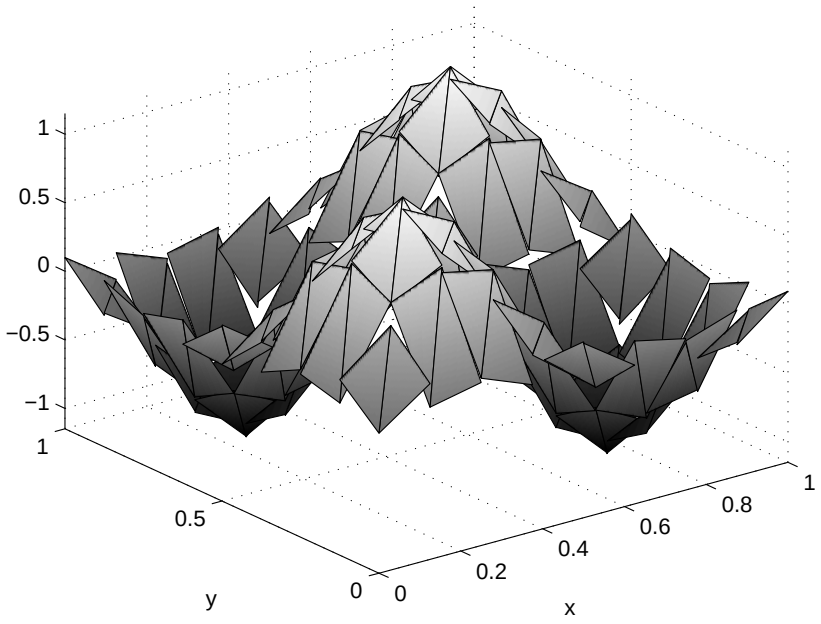

Figure 2.8: Numerical solution for $M = 8$, $\kappa = -1$, $\delta = 10^{-4}$ and $\beta = 2$.

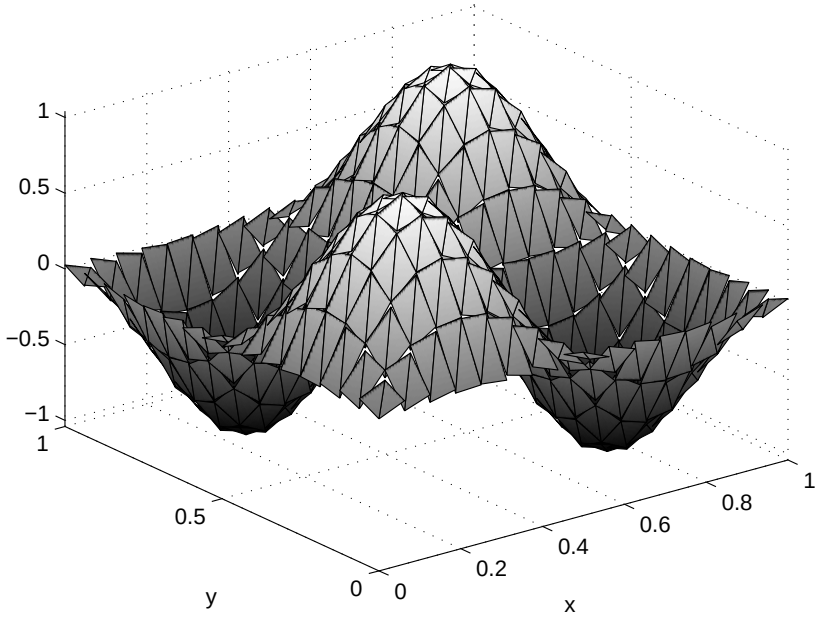

Figure 2.9: Numerical solution for $M = 16$, $\kappa = -1$, $\delta = 10^{-4}$ and $\beta = 2$.

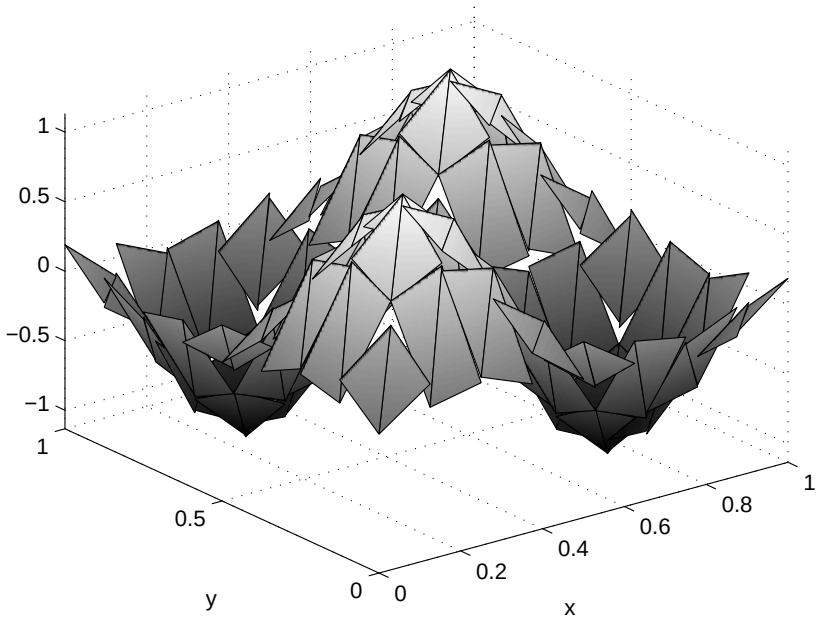

Figure 2.10: Numerical solution for $M = 8$, $\kappa = 1$, $\delta = 10^{-4}$ and $\beta = 2$.

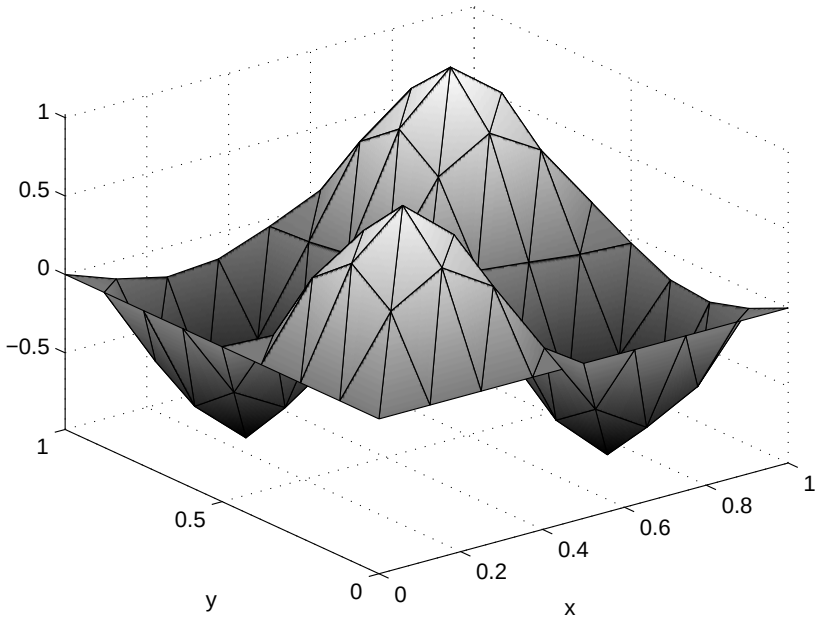

Figure 2.11: Numerical solution for $M = 8$, $\kappa = -1$, $\delta = 10^2$ and $\beta = 3$.

Chapter 3

The Heat Equation

3.1 The Model Problem and Variational Formulation

In this chapter we consider the ordinary heat equation: find u such that

$$\begin{aligned} u_t - \nabla \cdot D\nabla u &= f \qquad \text{in } \Omega \times I, && (3.1)\\ u &= 0 \qquad \text{on } \Gamma_D \times I, && (3.2)\\ D\nabla u \cdot \nu &= g \qquad \text{on } \Gamma_N \times I, && (3.3)\\ u(\boldsymbol{x}, 0) &= \breve{u}(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in \Omega. && (3.4) \end{aligned}$$

The domain Ω, the Dirichlet boundary Γ_D and the Neumann boundary Γ_N are defined as in Section 2.1, and we have $f(t) \in L^2(\Omega)$ and $g(t) \in L^2(\Gamma_N)$ for all $t \in I = (0, T)$. Multiplying equation (3.1) by a test function $v \in V = \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_D\}$ and integrating over the domain, we get

$$\int_\Omega u_t(t) v \, d\boldsymbol{x} + \int_\Omega D\nabla u(t) \cdot \nabla v \, d\boldsymbol{x} = \int_\Omega f(t) v \, d\boldsymbol{x} + \int_{\Gamma_N} g(t) v \, ds \qquad \forall v \in V.$$

However, since we work with broken spaces, we have, for $u(t) \in C(\bar{\Omega})\ \forall t \in I$ and for all $v \in \mathcal{D}_1(\mathcal{E}_h)$,

$$(u_t(t), v) + A(u(t), v) = L(t, v),$$

where

$$\begin{aligned} (u_t(t), v) &:= \int_\Omega u_t(t) v \, d\boldsymbol{x}\\ L(t, v) &:= \int_\Omega f(t) v \, d\boldsymbol{x} + \int_{\Gamma_N} g(t) v \, ds. \end{aligned}$$

and $A(\cdot, \cdot)$ is the bilinear form defined in (2.5). Using a time discretisation of Crank-Nicolson type and the notation

$$\begin{aligned} \partial_t w_i &:= \frac{w(t_i) - w(t_{i-1})}{k},\\ \bar{w}_i &:= \frac{w(t_i) + w(t_{i-1})}{2}, \end{aligned}$$

and

$$L_i(v) := \frac{1}{2}\big(L(t_i, v) + L(t_{i-1}, v)\big),$$

where $k = t_i - t_{i-1}$, the fully discrete formulation of equations (3.1)–(3.4) is as follows.

For $i = 1, \ldots, N$, find $u_i^h \in \mathcal{D}_1(\mathcal{E}_h)$ such that, for all $v \in \mathcal{D}_1(\mathcal{E}_h)$,

$$(\partial_t u_i^h, v) + A(\bar{u}_i^h, v) = L_i(v) \tag{3.5}$$

subject to the discrete initial condition

$$(u_0^h, v) = (\breve{u}, v).$$

3.2 Adding the Time Dependence

3.2.1 The Linear System

We can rewrite equation (3.5) as

$$2(u_i^h - u_{i-1}^h, v) + kA(u_i^h + u_{i-1}^h, v) = 2kL_i(v),$$

and, therefore,

$$2(u_i^h, v) + kA(u_i^h, v) = 2kL_i(v) + 2(u_{i-1}^h, v) - kA(u_{i-1}^h, v).$$

Define $\underline{B} = (B_{jk})_{k=1,\ldots,3N_h}^{j=1,\ldots,3N_h}$, $\boldsymbol{L}^i = (L_j^i)_{j=1,\ldots,3N_h}$ and $\breve{\boldsymbol{U}} = (\breve{U}_j)_{j=1,\ldots,3N_h}$ by

$$B_{3(n-1)+k,3(m-1)+j} = (\phi_{mj}, \phi_{nk}),$$

$$L^i_{3(m-1)+j} = L(t_i, \phi_{mj}) + L(t_{i-1}, \phi_{mj})$$

and

$$\breve{U}_{3(m-1)+j} = (\breve{u}, \phi_{mj}),$$

and remember the definition of the matrix $\underline{A} = (A_{jk})_{k=1,\ldots,3N_h}^{j=1,\ldots,3N_h}$ in Section 2.2.1,

$$A_{3(n-1)+k,3(m-1)+j} = A(\phi_{mj}, \phi_{nk}).$$

Then, at every time step $i = 1, \ldots, N$, we have to solve the linear system

$$(k\underline{A} + 2\underline{B})\boldsymbol{U}^i = k\boldsymbol{L}^i + (2\underline{B} - k\underline{A})\boldsymbol{U}^{i-1},$$
$$\underline{B}\boldsymbol{U}^0 = \breve{\boldsymbol{U}}.$$

3.2.2 The Mass Matrix and Initial Conditions

Since the stiffness matrix $\underline{A}$ is the same as before, we may use the function `stiff` from Section 2.2.3. Adding the time variable to the argument lists of `L`, `f` and `g`, we may also use the function `L` from Section 2.2.4.

```
function L = L(t,coord,elements,neumann)
M = zeros(3*size(elements,1),1);
for m = 1:size(elements,1)
    M(3*(m-1)+(1:3)) = det([1,1,1;coord(elements(m,:),:)'])*...
        mydblquad(@(x) f(t,map(coord(elements(m,:),:)',x))*...
        phi(x),[0 0; 1 0; 0 1]);
end
for l = 1:size(neumann,1)
    m = neumann(l,3); p = neumann(l,4);
    if p == 2, j = 3:-1:1; else j = 1:3; end
    P1 = myquad(@(x) g(t,neumann(l,1:2),x), coord(neumann(l,1),:), ...
        coord(neumann(l,2),:));
    P2 = myquad(@(x) g(t,neumann(l,1:2),x), coord(neumann(l,2),:), ...
        coord(neumann(l,1),:));
    M(3*(m-1)+j(j~=p)) = M(3*(m-1)+j(j~=p)) + [P1 P2]';
end
L = M;
```

For the computation of the mass matrix $\underline{B}$, observe that

$$\begin{aligned}
(\phi_{nk},\phi_{mj}) &= (\phi_{mj},\phi_{nk}),\\
&= \int_\Omega \phi_{mj}\phi_{nk}\; d\boldsymbol{x},\\
&= \delta_{mn}\int_{E_m} \phi_{mj}\phi_{mk}\; d\boldsymbol{x},\\
&= \delta_{mn}|\det J_m|\int_T \widetilde{\phi}_j\widetilde{\phi}_k\; d\boldsymbol{y},\\
&= \delta_{mn}|\det J_m|\frac{2^{\delta_{jk}}}{24}.
\end{aligned}$$

So, for each element, we obtain the matrix

$$\frac{|\det J_m|}{24}\begin{pmatrix}2&1&1\\1&2&1\\1&1&2\end{pmatrix}.$$

```
B = sparse(3*size(elements,1),3*size(elements,1));
for m = 1:size(elements,1)
    B(3*(m-1)+(1:3),3*(m-1)+(1:3)) = ...
        det([1,1,1;coordinates(elements(m,:),:)']) * ...
        [2,1,1;1,2,1;1,1,2]/24;
end
```

To obtain the vector $\boldsymbol{U}^0$, we also need to compute the vector $\breve{\boldsymbol{U}}$. Now,

$$\begin{aligned}(\breve{u}, \phi_{mj}) &= \int_\Omega \breve{u}\phi_{mj}\, d\boldsymbol{x}, \\ &= \int_{E_m} \breve{u}\phi_{mj}\, d\boldsymbol{x}, \\ &= |\det J_m| \iint_T \breve{u}(\boldsymbol{x}(s,t))\widetilde{\phi}_j(s,t)\, dsdt.\end{aligned}$$

Thus, for each element, we get a vector

$$(\breve{u}, \boldsymbol{\phi}_m^T) = |\det J_m| \iint_T \breve{u}(\boldsymbol{x}(s,t))\widetilde{\boldsymbol{\phi}}(s,t)^T\, dsdt,$$

where $\boldsymbol{\phi}_m = \begin{pmatrix}\phi_{m1} & \phi_{m2} & \phi_{m3}\end{pmatrix}$ and $\widetilde{\boldsymbol{\phi}} = \begin{pmatrix}\widetilde{\phi}_1 & \widetilde{\phi}_2 & \widetilde{\phi}_3\end{pmatrix}$.

```
function b = b_0(coord,elements)
y = zeros(3*size(elements,1),1);
for m = 1:size(elements,1)
        y(3*(m-1)+(1:3)) = det([1,1,1;coord(elements(m,:),:)'])*...
        mydblquad(@(x) ubreve(map(coord(elements(m,:),:)',x))*...
        phi(x),[0 0; 1 0; 0 1]);
end
b = y;
```

Here, `ubreve` implements the function $\breve{u}$. The numerical solution to the discrete problem (3.5) can now be obtained with the following MATLAB script.

```
%% Initialisation
M = 2;
mesh
N = 5;
T = 1; dt = T/N;
D = 1; beta = 2; delta = 10^4;
% kappa = 1 gives a non-symmetric scheme,
% kappa = -1 gives a symmetric scheme
kappa = -1;
U = zeros(3*size(elements,1),N+1);

%% Assembly
A = stiff(kappa,D,beta,delta,coordinates,elements,...
    dirichlet,interior);
B = sparse(3*size(elements,1),3*size(elements,1));
for m = 1:size(elements,1)
    B(3*(m-1)+(1:3),3*(m-1)+(1:3)) = ...
        det([1,1,1;coordinates(elements(m,:),:)'])*...
```

```
        [2,1,1;1,2,1;1,1,2]/24;
end

%% Initial condition
U(:,1) = B\b_0(coordinates,elements);

%% Time steps
for i = 2:N+1
    % Computing the load vector
    b = L(dt*(i-1),coordinates,elements,neumann) + ...
        L(dt*(i-2),coordinates,elements,neumann);
    b = dt*b + (2*B - dt*A)*U(:,i-1);

    % Computing the solution
    U(:,i) = (dt*A + 2*B)\b;
end
```

3.3 Numerical Experiments

Consider first the test problem

$$
\begin{aligned}
u_t - \Delta u &= f \qquad \text{in } \Omega \times I, && (3.6)\\
u &= 0 \qquad \text{on } \Gamma_D \times I, && (3.7)\\
\frac{\partial u}{\partial n} &= g \qquad \text{on } \Gamma_N \times I, && (3.8)\\
u(\boldsymbol{x}, 0) &= 0 \qquad \forall \boldsymbol{x} \in \Omega, && (3.9)
\end{aligned}
$$

where

$$
\begin{aligned}
&\Omega = (0,1) \times (0,1) \in \mathbb{R}^2, \quad \Gamma_D = \{\boldsymbol{x} \in \mathbb{R}^2 : \ x = 0 \text{ or } y = 0\},\\
&f(\boldsymbol{x}, t) = (1 + 8\pi^2 t)\sin(2\pi x)\sin(2\pi y),\\
&g(\boldsymbol{x}, t) = \begin{cases} 2\pi t \cos(2\pi x)\sin(2\pi y) & \text{if } x = 1,\\ 2\pi t \sin(2\pi x)\cos(2\pi y) & \text{if } y = 1. \end{cases}
\end{aligned}
$$

The solution to this problem is

$$u(\boldsymbol{x}, t) = t \sin(2\pi x)\sin(2\pi y).$$

The values of f, g and $\breve{u}$ are obtained with calls to the functions `f.m`, `g.m` and `ubreve.m`.

```
function f = f(t,x)
f = (1+8*pi^2*t)*sin(2*pi*x(1))*sin(2*pi*x(2));
```

```
function g = g(t,edge,x)
global g1
if isG(edge,g1)
    g = 2*pi*t*cos(2*pi*x(1))*sin(2*pi*x(2));
else
    g = 2*pi*t*sin(2*pi*x(1))*cos(2*pi*x(2));
end

function y = ubreve(x)
y = 0;
```

The function `isG` and the matrix `g1` are the same as in Section 2.3. In the tables below, the parameter M describes the mesh size and the parameter N gives the number of time steps. Since the solution to the test problem is linear in time, we expect the error to be constant as we fix the mesh size and increase the number of time steps. The error also seems to decrease with EOC $= 1$ as we fix the number of time steps and refine the mesh. The error is measured as a weighted sum of averages over time. For $k = T/N$,

$$\bar{e}_{\mathcal{A}} := k \sum_{i=1}^{N} \|\bar{u}_i - \bar{u}_i^h\|_{\mathcal{A}}.$$

	$\kappa = -1$		$\kappa = 1$	
M	$\bar{e}_{\mathcal{A}}$	EOC	$\bar{e}_{\mathcal{A}}$	EOC
1	1.7899		1.7899	
2	1.7752	0.0119	1.7752	0.0119
4	1.5016	0.2415	1.5016	0.2415
8	0.8366	0.8438	0.8366	0.8438
16	0.4315	0.9553	0.4315	0.9553
32	0.2175	0.9883	0.2175	0.9883

Table 3.1: Tabulated errors for $N = 2$, $\delta = 10^4$ and $\beta = 2$.

MATLAB comes with some very convenient tools for visualising the behaviour of a function in time. A short script which creates a movie showing the time dependence of the numerical solution is as follows.

```
G = moviein(N+1); Y = elements';
Z = zeros(3,size(elements,1)); Z(:) = 1:3*size(elements,1);
for i = 1:N+1
    trisurf(Z',coordinates(Y(:),1),coordinates(Y(:),2),U(:,i),...
        'facecolor','interp')
    axis([0 1 0 1 min(U(:)) max(U(:))]), xlabel('x'), ylabel('y')
    G(i) = getframe;
end
```

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A}}$	EOC	$\bar{e}_{\mathcal{A}}$	EOC
1	1.7751		1.7751	
2	1.7752	$-5.3523 \cdot 10^{-5}$	1.7752	$-5.3542 \cdot 10^{-5}$
4	1.7752	$-3.2055 \cdot 10^{-5}$	1.7752	$-3.2048 \cdot 10^{-5}$
8	1.7752	$-2.1644 \cdot 10^{-5}$	1.7753	$-2.1641 \cdot 10^{-5}$
16	1.7753	$-1.2707 \cdot 10^{-5}$	1.7753	$-1.2707 \cdot 10^{-5}$
32	1.7753	$-5.8906 \cdot 10^{-6}$	1.7753	$-5.8906 \cdot 10^{-6}$

Table 3.2: Tabulated errors for $M = 2$, $\delta = 10^4$ and $\beta = 2$.

	$\kappa = -1$		$\kappa = 1$	
M	$\bar{e}_{\mathcal{A}}$	EOC	$\bar{e}_{\mathcal{A}}$	EOC
1	2.5137		2.2640	
2	1.8152	0.4697	1.8454	0.2950
4	1.7017	0.0931	1.7749	0.0562
8	0.9534	0.8359	0.9772	0.8610
16	0.4926	0.9525	0.4976	0.9737
32	0.2480	0.9900	0.2490	0.9986

Table 3.3: Tabulated errors for $N = 2$, $\delta = 10$ and $\beta = 1$.

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A}}$	EOC	$\bar{e}_{\mathcal{A}}$	EOC
1	1.8151		1.8452	
2	1.8152	$-8.5185 \cdot 10^{-5}$	1.8454	-0.00012193
4	1.8152	$-1.1958 \cdot 10^{-5}$	1.8454	$2.4715 \cdot 10^{-6}$
8	1.8152	$-1.3282 \cdot 10^{-5}$	1.8454	$-4.977 \cdot 10^{-6}$
16	1.8152	$-9.7771 \cdot 10^{-6}$	1.8454	$-8.6381 \cdot 10^{-6}$
32	1.8152	$-6.6169 \cdot 10^{-6}$	1.8454	$-6.7613 \cdot 10^{-6}$

Table 3.4: Tabulated errors for $M = 2$, $\delta = 10$ and $\beta = 1$.

To play the movie, use the command `movie(G)`. The plots below show the numerical solution for $N = 10$, $M = 8$, $\kappa = -1$, $\delta = 10$ and $\beta = 1$ at time levels t_1, t_5 and t_{10}, where $t_i = i/N$.

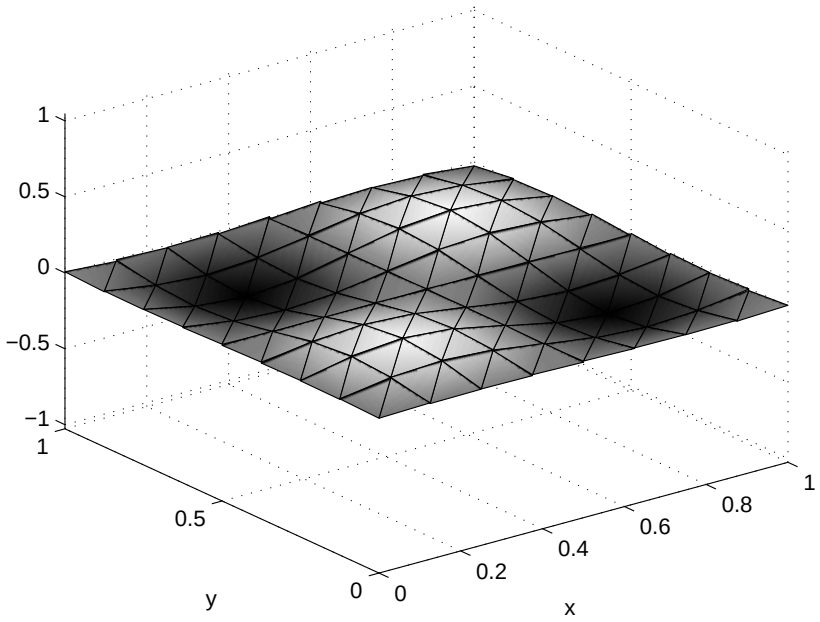

Figure 3.1: Numerical solution at t_1 for $N = 10$, $M = 8$, $\kappa = -1$, $\delta = 10$ and $\beta = 1$.

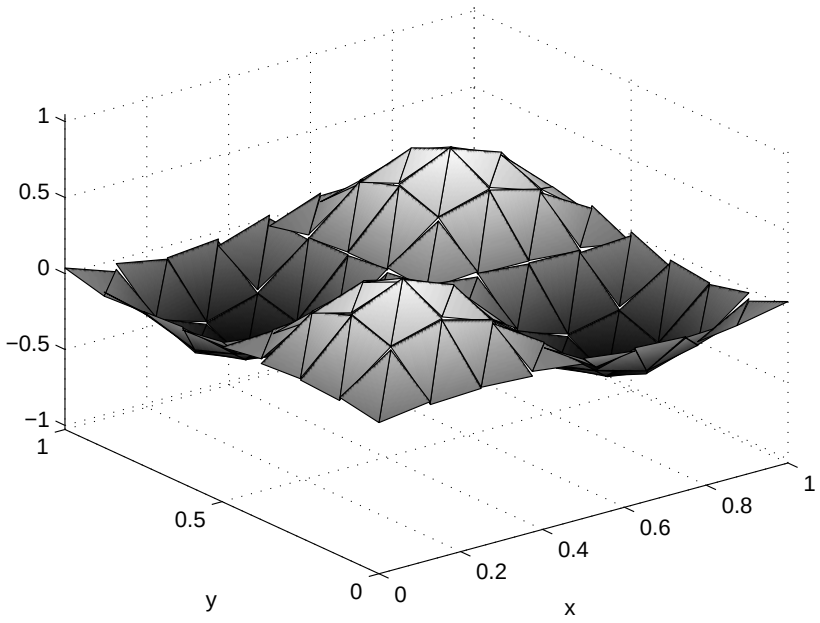

Figure 3.2: Numerical solution at t_5 for $N = 10$, $M = 8$, $\kappa = -1$, $\delta = 10$ and $\beta = 1$.

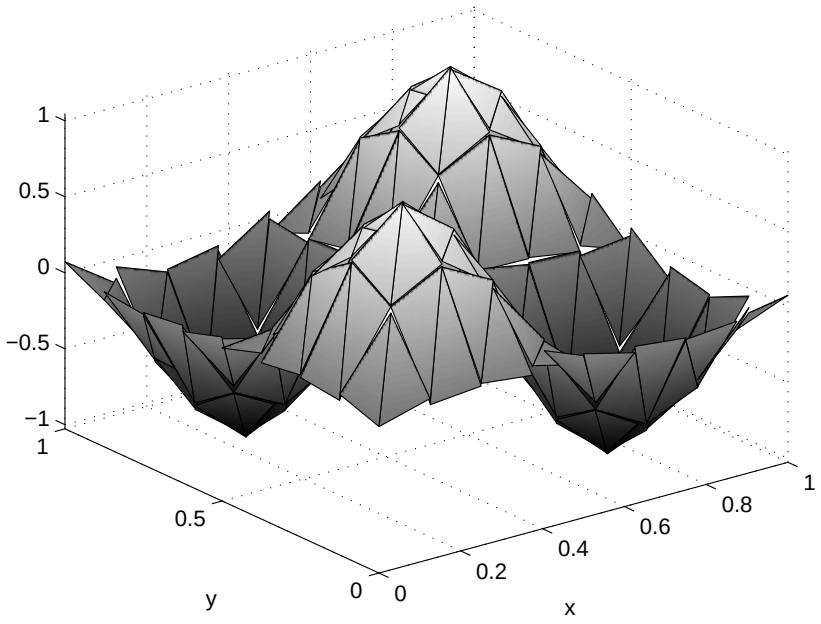

Figure 3.3: Numerical solution at t_{10} for $N = 10$, $M = 8$, $\kappa = -1$, $\delta = 10$ and $\beta = 1$.

Consider now problem (3.6)–(3.9) with a solution which is linear in space and nonlinear in time. Let

$$\begin{aligned} f(\boldsymbol{x}, t) &= 3(t+1)^2 x, \\ g(\boldsymbol{x}, t) &= \begin{cases} (t+1)^3 & \text{if } x = 1, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

and

$$\Omega = (0,1) \times (0,1) \subset \mathbb{R}^2, \qquad \Gamma_D = \{(x,y) \in \mathbb{R}^2 : \ x = 0\}.$$

Then, the exact solution is

$$u(\boldsymbol{x}, t) = (t+1)^3 x.$$

We now expect the error to be constant as we fix the number of time steps and refine the mesh. In the tables below we also see that the error seems to decrease with EOC = 2 as we fix the mesh size and increase the number of time steps. The plots show u_i^h at time levels t_1, t_5, t_7 and t_{10} for $N = 10$, $M = 8$, $\kappa = 1$, $\delta = 1$ and $\beta = 4$.

	$\kappa = -1$		$\kappa = 1$	
M	$\bar{e}_{\mathcal{A}}$	EOC	$\bar{e}_{\mathcal{A}}$	EOC
1	0.0281		0.0281	
2	0.0282	-0.0046741	0.0282	-0.0045908
4	0.0282	-0.0017259	0.0282	-0.0017177
8	0.0282	-0.00054789	0.0282	-0.00054689
16	0.0282	-0.00014903	0.0282	-0.00014892
32	0.0282	$-3.5319 \cdot 10^{-5}$	0.0282	$-3.7942 \cdot 10^{-5}$

Table 3.5: Tabulated errors for $N = 2$, $\delta = 10^4$ and $\beta = 2$.

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A}}$	EOC	$\bar{e}_{\mathcal{A}}$	EOC
1	0.1007		0.1007	
2	0.0282	1.8376	0.0282	1.8376
4	0.0071	1.9804	0.0071	1.9804
8	0.0018	1.9954	0.0018	1.9954
16	0.0004	1.9988	0.0004	1.9988
32	0.0001	1.9997	0.0001	1.9997

Table 3.6: Tabulated errors for $M = 2$, $\delta = 10^4$ and $\beta = 2$.

	$\kappa = -1$		$\kappa = 1$	
M	$\bar{e}_{\mathcal{A}}$	EOC	$\bar{e}_{\mathcal{A}}$	EOC
1	0.0257		0.0286	
2	0.0280	-0.12717	0.0286	0.00077727
4	0.0282	-0.0095152	0.0282	0.018384
8	0.0282	-0.00023742	0.0282	0.00057948
16	0.0282	-0.00014043	0.0282	-0.00011335
32	0.0282	$-3.8325 \cdot 10^{-5}$	0.0282	$-3.7286 \cdot 10^{-5}$

Table 3.7: Tabulated errors for $N = 2$, $\delta = 1$ and $\beta = 4$.

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A}}$	EOC	$\bar{e}_{\mathcal{A}}$	EOC
1	0.1002		0.1023	
2	0.0280	1.8387	0.0286	1.8389
4	0.0071	1.9800	0.0072	1.9796
8	0.0018	1.9953	0.0018	1.9952
16	0.0004	1.9988	0.0005	1.9988
32	0.0001	1.9997	0.0001	1.9997

Table 3.8: Tabulated errors for $M = 2$, $\delta = 1$ and $\beta = 4$.

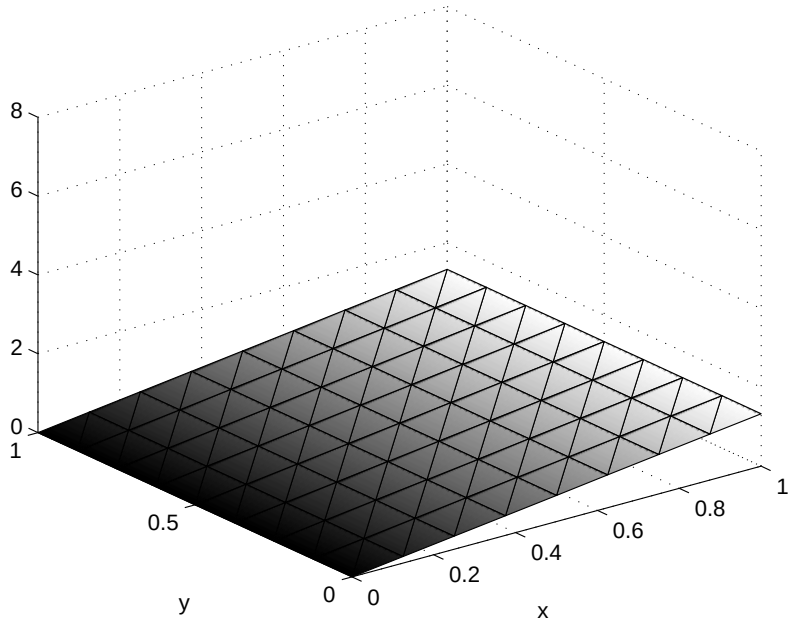

Figure 3.4: Numerical solution at t_1 for $N = 10$, $M = 8$, $\kappa = 1$, $\delta = 1$ and $\beta = 4$.

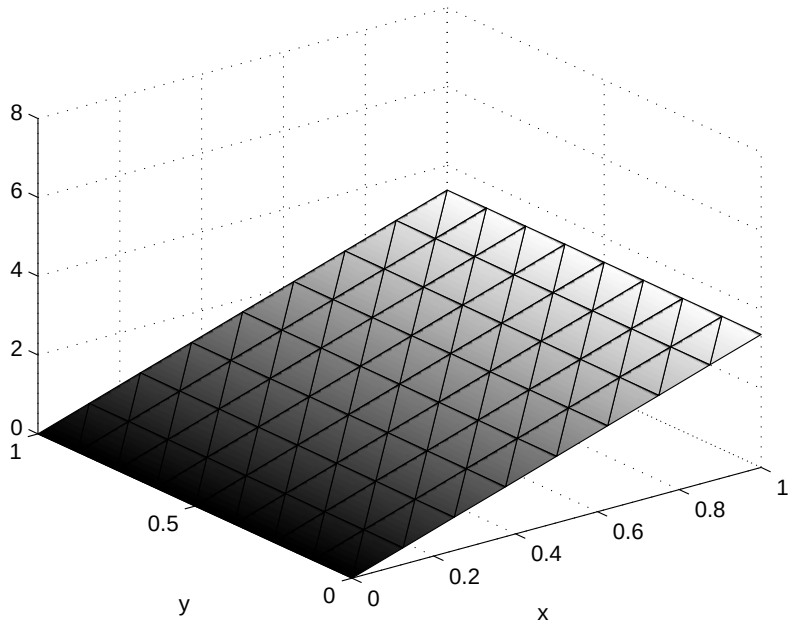

Figure 3.5: Numerical solution at t_5 for $N = 10$, $M = 8$, $\kappa = 1$, $\delta = 1$ and $\beta = 4$.

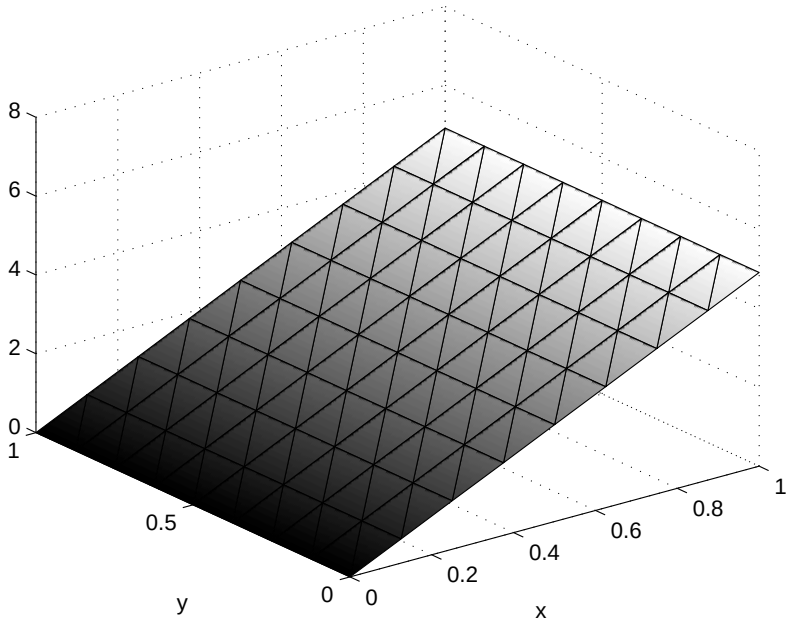

Figure 3.6: Numerical solution at t_7 for $N = 10$, $M = 8$, $\kappa = 1$, $\delta = 1$ and $\beta = 4$.

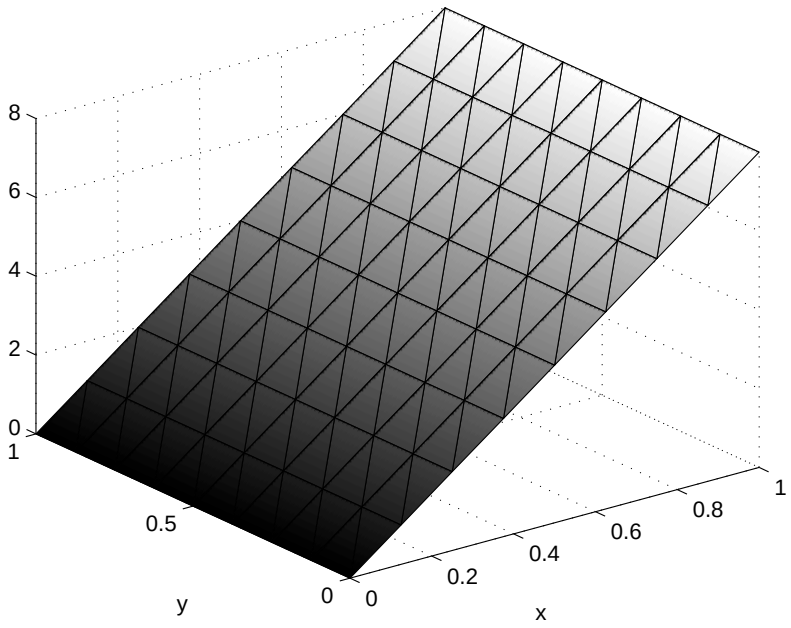

Figure 3.7: Numerical solution at t_{10} for $N = 10$, $M = 8$, $\kappa = 1$, $\delta = 1$ and $\beta = 4$.

Chapter 4

A Linear Diffusion Problem

4.1 The Model Problem and Variational Formulation

Before we consider the nonlinear diffusion problem, we implement a linear version of problem (1.1)–(1.4). The problem we consider now is as follows. Find $u: \Omega \times I \to \mathbb{R}$ and $\boldsymbol{\sigma}: \Omega \times I \to \mathbb{R}^2$ such that in $\Omega \times I$

$$\begin{aligned} u_t(\boldsymbol{x},t) - \nabla \cdot D\nabla u(\boldsymbol{x},t) &= f(\boldsymbol{x},t) + \nabla \cdot K\boldsymbol{\sigma}(\boldsymbol{x},t), & (4.1)\\ \boldsymbol{\sigma}_t(\boldsymbol{x},t) + \gamma\boldsymbol{\sigma}(\boldsymbol{x},t) &= \mu\nabla u(\boldsymbol{x},t) + \boldsymbol{h}(\boldsymbol{x},t), & (4.2) \end{aligned}$$

subject to

$$\begin{aligned} &u(\boldsymbol{x},0) = \breve{u}(\boldsymbol{x}), & (4.3)\\ &\boldsymbol{\sigma}(\boldsymbol{x},0) = \breve{\boldsymbol{\sigma}}(\boldsymbol{x}), & (4.4)\\ &u(\boldsymbol{x},t) = 0 \quad \text{on } \Gamma_D \times I, & (4.5) \end{aligned}$$

and

$$(D\nabla u(\boldsymbol{x},t) + K\boldsymbol{\sigma}(\boldsymbol{x},t)) \cdot \nu(\boldsymbol{x}) = g(\boldsymbol{x},t) \quad \text{on } \Gamma_N \times I, \tag{4.6}$$

where $\Omega \subset \mathbb{R}^2$, $I = (0,T)$ for some $T > 0$ and γ is a constant. If $u(t) \in C(\bar{\Omega})$, we have, using the notation defined in Section 3.1,

$$\begin{aligned} (u_t(t), v) + A(u(t), v) = L(t,v) &- \sum_{E \in \mathcal{E}_h} (K\boldsymbol{\sigma}(t), \nabla v)_E \\ &+ \sum_{e \in \Gamma_h \cup \Gamma_D} \int_e \{K\boldsymbol{\sigma}(t) \cdot \nu_e\}[v]\, ds \qquad \forall v \in \mathcal{D}_1(\mathcal{E}_h) \end{aligned}$$

and

$$(\boldsymbol{\sigma}_t(t) + \gamma\boldsymbol{\sigma}(t), \boldsymbol{w}) = \sum_{E \in \mathcal{E}_h} (\mu\nabla u(t), \boldsymbol{w})_E + (\boldsymbol{h}(t), \boldsymbol{w}) \qquad \forall \boldsymbol{w} \in \boldsymbol{\mathcal{D}}_0(\mathcal{E}_h).$$

Using again the Crank-Nicolson method for the discretisation in time, the fully discrete formulation of problem (4.1)–(4.6) is as follows.

For $i = 1, \ldots, N$, find $u_i^h \in \mathcal{D}_1(\mathcal{E}_h)$ and $\boldsymbol{\sigma}_i^h \in \boldsymbol{\mathcal{D}}_0(\mathcal{E}_h)$ such that, for all $v \in \mathcal{D}_1(\mathcal{E}_h)$ and $\boldsymbol{w} \in \boldsymbol{\mathcal{D}}_0(\mathcal{E}_h)$,

$$(\partial_t u_i^h, v) + A(\bar{u}_i^h, v) = L_i(v) - \sum_{E \in \mathcal{E}_h} (K\bar{\boldsymbol{\sigma}}_i^h, \nabla v)_E + \sum_{e \in \Gamma_h \cup \Gamma_D} \int_e \{K\bar{\boldsymbol{\sigma}}_i^h \cdot \nu_e\}[v]\, ds \quad (4.7)$$

and

$$(\partial_t \boldsymbol{\sigma}_i^h + \gamma \bar{\boldsymbol{\sigma}}_i^h, \boldsymbol{w}) = \sum_{E \in \mathcal{E}_h} (\mu \nabla \bar{u}_i^h, \boldsymbol{w})_E + (\bar{\boldsymbol{h}}_i, \boldsymbol{w}), \quad (4.8)$$

subject to

$$(u_0^h, v) = (\breve{u}, v),$$
$$(\boldsymbol{\sigma}_0^h, \boldsymbol{w}) = (\breve{\boldsymbol{\sigma}}, \boldsymbol{w}).$$

4.2 Solving for u and $\boldsymbol{\sigma}$

4.2.1 The Linear System

Rewrite equation (4.8) as

$$(2(\boldsymbol{\sigma}_i^h - \boldsymbol{\sigma}_{i-1}^h) + \gamma k(\boldsymbol{\sigma}_i^h + \boldsymbol{\sigma}_{i-1}^h), \boldsymbol{w}) = \sum_{E \in \mathcal{E}_h} (\mu k(\nabla u_i^h + \nabla u_{i-1}^h), \boldsymbol{w})_E + (k(\boldsymbol{h}_i + \boldsymbol{h}_{i-1}), \boldsymbol{w}),$$

and, thus,

$$(\boldsymbol{\sigma}_i^h, \boldsymbol{w}) = \frac{2 - \gamma k}{2 + \gamma k}(\boldsymbol{\sigma}_{i-1}^h, \boldsymbol{w}) + \sum_{E \in \mathcal{E}_h} \frac{\mu k}{2 + \gamma k}(\nabla u_i^h + \nabla u_{i-1}^h, \boldsymbol{w})_E + \frac{k}{2 + \gamma k}(\boldsymbol{h}_i + \boldsymbol{h}_{i-1}, \boldsymbol{w}). \quad (4.9)$$

Hence, we have, for all elements $E \in \mathcal{E}_h$,

$$\boldsymbol{\sigma}_i^h\big|_E = \frac{2 - \gamma k}{2 + \gamma k}\boldsymbol{\sigma}_{i-1}^h\big|_E + \frac{\mu k}{2 + \gamma k}(\nabla u_i^h + \nabla u_{i-1}^h)\big|_E + \frac{k}{2 + \gamma k}(\boldsymbol{h}_i + \boldsymbol{h}_{i-1})\big|_E. \quad (4.10)$$

Using (4.9) and (4.10) in (4.7), we get

$$
\begin{aligned}
&2(u_i^h - u_{i-1}^h, v) + kA(u_i^h + u_{i-1}^h, v) = 2kL_i(v) - \sum_{E\in\mathcal{E}_h} k(K\boldsymbol{\sigma}_{i-1}^h, \nabla v)_E \\
&- \sum_{E\in\mathcal{E}_h} k\frac{2-\gamma k}{2+\gamma k}(K\boldsymbol{\sigma}_{i-1}^h, \nabla v)_E - \sum_{E\in\mathcal{E}_h} k\frac{\mu k}{2+\gamma k}(K(\nabla u_i^h + \nabla u_{i-1}^h), \nabla v)_E \\
&- \sum_{E\in\mathcal{E}_h} k\frac{k}{2+\gamma k}(K(\boldsymbol{h}_i + \boldsymbol{h}_{i-1}), \nabla v)_E + \sum_{e\in\Gamma_h\cup\Gamma_D}\int_e k\frac{4}{2+\gamma k}\{K\boldsymbol{\sigma}_{i-1}^h\cdot\nu_e\}[v]\,ds \\
&+ \sum_{e\in\Gamma_h\cup\Gamma_D}\int_e k\frac{\mu k}{2+\gamma k}\{K(\nabla u_i^h + \nabla u_{i-1}^h)\cdot\nu_e\}[v]\,ds \\
&\qquad\qquad + \sum_{e\in\Gamma_h\cup\Gamma_D}\int_e k\frac{k}{2+\gamma k}\{K(\boldsymbol{h}_i + \boldsymbol{h}_{i-1})\cdot\nu_e\}[v]\,ds.
\end{aligned}
$$

Hence,

$$
\begin{aligned}
&2(u_i^h, v) + kA(u_i^h, v) + \sum_{E\in\mathcal{E}_h}\frac{\mu k^2 K}{2+\gamma k}(\nabla u_i^h, \nabla v)_E - \sum_{e\in\Gamma_h\cup\Gamma_D}\int_e \frac{\mu k^2 K}{2+\gamma k}\{\nabla u_i^h\cdot\nu_e\}[v]\,ds \\
&= k(L(t_i, v) + L(t_{i-1}, v)) + 2(u_{i-1}^h, v) - kA(u_{i-1}^h, v) \\
&- \sum_{E\in\mathcal{E}_h}\frac{\mu k^2 K}{2+\gamma k}(\nabla u_{i-1}^h, \nabla v)_E + \sum_{e\in\Gamma_h\cup\Gamma_D}\int_e \frac{\mu k^2 K}{2+\gamma k}\{\nabla u_{i-1}^h\cdot\nu_e\}[v]\,ds \\
&- \sum_{E\in\mathcal{E}_h}\frac{4kK}{2+\gamma k}(\boldsymbol{\sigma}_{i-1}^h, \nabla v)_E + \sum_{e\in\Gamma_h\cup\Gamma_D}\int_e \frac{4kK}{2+\gamma k}\{\boldsymbol{\sigma}_{i-1}^h\cdot\nu_e\}[v]\,ds \\
&- \sum_{E\in\mathcal{E}_h}\frac{k^2 K}{2+\gamma k}(\boldsymbol{h}_i + \boldsymbol{h}_{i-1}, \nabla v)_E + \sum_{e\in\Gamma_h\cup\Gamma_D}\int_e \frac{k^2 K}{2+\gamma k}\{(\boldsymbol{h}_i + \boldsymbol{h}_{i-1})\cdot\nu_e\}[v]\,ds.
\end{aligned}
$$

Define the matrices $\underline{A}$ and $\underline{B}$ and the vector $\boldsymbol{L}$ as in Section 3.2.1. Let $\underline{C} = (C_{jk})_{k=1,\ldots,3N_h}^{j=1,\ldots,3N_h}$, $\underline{E} = (E_{jk})_{k=1,\ldots,2N_h}^{j=1,\ldots,3N_h}$ and $\boldsymbol{H}^i = (H_j^i)_{j=1,\ldots,3N_h}$, where

$$
\begin{aligned}
C_{3(n-1)+k,3(m-1)+j} &= \sum_{E\in\mathcal{E}_h}\int_E \nabla\phi_{mj}\cdot\nabla\phi_{nk}\,d\boldsymbol{x} - \sum_{e\in\Gamma_h\cup\Gamma_D}\int_e\{\nabla\phi_{mj}\cdot\nu_e\}[\phi_{nk}]\,ds, \\
E_{3(n-1)+k,2(m-1)+j} &= \sum_{E\in\mathcal{E}_h}\int_E \boldsymbol{\psi}_{mj}\cdot\nabla\phi_{nk}\,d\boldsymbol{x} - \sum_{e\in\Gamma_h\cup\Gamma_D}\int_e\{\boldsymbol{\psi}_{mj}\cdot\nu_e\}[\phi_{nk}]\,ds,
\end{aligned}
$$

and

$$
H_{3(m-1)+j}^i = \sum_{E\in\mathcal{E}_h}\int_E 2\bar{\boldsymbol{h}}_i\cdot\nabla\phi_{mj}\,d\boldsymbol{x} - \sum_{e\in\Gamma_h\cup\Gamma_D}\int_e\{2\bar{\boldsymbol{h}}_i\cdot\nu_e\}[\phi_{mj}]\,ds.
$$

Here, $\{\boldsymbol{\psi}_{mj}\}$ is a basis of $\boldsymbol{\mathcal{D}}_0(\mathcal{E}_h)$ such that, for $m = 1, \ldots, N_h$,

$$
\boldsymbol{\psi}_{m1}(\boldsymbol{x}) = \begin{cases} \begin{pmatrix}1\\0\end{pmatrix} & \text{if } \boldsymbol{x}\in E_m, \\ \mathbf{0} & \text{otherwise,}\end{cases} \qquad \text{and} \qquad \boldsymbol{\psi}_{m2}(\boldsymbol{x}) = \begin{cases} \begin{pmatrix}0\\1\end{pmatrix} & \text{if } \boldsymbol{x}\in E_m, \\ \mathbf{0} & \text{otherwise,}\end{cases}
$$

and

$$\boldsymbol{\sigma}_i^h = \sum_{m=1}^{N_h} \sum_{j=1}^{2} S_{2(m-1)+j}^i \ \boldsymbol{\psi}_{mj}.$$

Then, we get, at each time step, the following equation system, which we have to solve for $\boldsymbol{U}^i$.

$$\left(k\underline{A} + 2\underline{B} + \frac{\mu k^2 K}{2+\gamma k}\underline{C}\right)\boldsymbol{U}^i = \\ k\boldsymbol{L}^i + \left(2\underline{B} - k\underline{A} - \frac{\mu k^2 K}{2+\gamma k}\underline{C}\right)\boldsymbol{U}^{i-1} - \frac{4kK}{2+\gamma k}\underline{E}\boldsymbol{S}^{i-1} - \frac{k^2 K}{2+\gamma k}\boldsymbol{H}^i.$$

Once we know $\boldsymbol{U}^i$, we can go back to equation (4.9). From

$$u_i^h = \sum_{m=1}^{N_h} \sum_{j=1}^{3} U_{3(m-1)+j}^i \ \phi_{mj}$$

follows

$$\nabla u_i^h|_{E_m} = \sum_{j=1}^{3} U_{3(m-1)+j}^i \ \nabla\phi_{mj},$$

and, thus,

$$(\nabla u_i^h + \nabla u_{i-1}^h)|_{E_m} = \sum_{j=1}^{3} (U_{3(m-1)+j}^i + U_{3(m-1)+j}^{i-1}) \ \nabla\phi_{mj}.$$

Define $\underline{F} = (F_{jk})_{k=1,\ldots,2N_h}^{j=1,\ldots,2N_h}$, $\underline{G} = (G_{jk})_{k=1,\ldots,3N_h}^{j=1,\ldots,2N_h}$ and $\boldsymbol{J}^i = (J_j^i)_{j=1,\ldots,2N_h}$ by

$$F_{2(n-1)+k,2(m-1)+j} = \int_\Omega \boldsymbol{\psi}_{mj} \cdot \boldsymbol{\psi}_{nk} \ d\boldsymbol{x},$$
$$G_{2(n-1)+k,3(m-1)+j} = \sum_{E\in\mathcal{E}_h} \int_E \nabla\phi_{mj} \cdot \boldsymbol{\psi}_{nk} \ d\boldsymbol{x},$$

and

$$J_{2(m-1)+j}^i = \int_\Omega 2\bar{\boldsymbol{h}}_i \cdot \boldsymbol{\psi}_{mj} \ d\boldsymbol{x}.$$

Then, at each time step, we have to solve the following equation system for $\boldsymbol{S}^i$.

$$\underline{F}\boldsymbol{S}^i = \frac{2-\gamma k}{2+\gamma k}\underline{F}\boldsymbol{S}^{i-1} + \frac{\mu k}{2+\gamma k}\underline{G}(\boldsymbol{U}^i + \boldsymbol{U}^{i-1}) + \frac{k}{2+\gamma k}\boldsymbol{J}^i.$$

From the initial conditions, we derive two equation systems,

$$\underline{B}\boldsymbol{U}^0 = \breve{\boldsymbol{U}} \qquad \text{and} \qquad \underline{F}\boldsymbol{S}^0 = \breve{\boldsymbol{S}},$$

where

$$\breve{U}_{3(m-1)+j} = (\breve{u}, \phi_{mj}) \qquad \text{and} \qquad \breve{S}_{2(m-1)+k} = (\breve{\boldsymbol{\sigma}}, \boldsymbol{\psi}_{mk}),$$

for $m = 1, \ldots, N_h$, $j = 1, 2, 3$ and $k = 1, 2$.

Summarising, at each time step, we obtain the solution to the discrete problem (4.7)–(4.8) by consecutively solving the linear systems

$$\left(k\underline{A} + 2\underline{B} + \frac{\mu k^2 K}{2+\gamma k}\underline{C}\right)\boldsymbol{U}^i = k\boldsymbol{L}^i + \left(2\underline{B} - k\underline{A} - \frac{\mu k^2 K}{2+\gamma k}\underline{C}\right)\boldsymbol{U}^{i-1} - \frac{4kK}{2+\gamma k}\underline{E}\boldsymbol{S}^{i-1} - \frac{k^2 K}{2+\gamma k}\boldsymbol{H}^i$$

and

$$\underline{F}\boldsymbol{S}^i = \frac{2-\gamma k}{2+\gamma k}\underline{F}\boldsymbol{S}^{i-1} + \frac{\mu k}{2+\gamma k}\underline{G}(\boldsymbol{U}^i + \boldsymbol{U}^{i-1}) + \frac{k}{2+\gamma k}\boldsymbol{J}^i,$$

subject to

$$\underline{B}\boldsymbol{U}^0 = \breve{\boldsymbol{U}} \qquad \text{and} \qquad \underline{F}\boldsymbol{S}^0 = \breve{\boldsymbol{S}}.$$

4.2.2 Extending the Code

A MATLAB function `matC` which returns the matrix $\underline{C}$ can easily be obtained by modifying the function `stiff.m`. Simply remove the lines of code which are not needed (compare the definitions of the matrices $\underline{A}$ and $\underline{C}$) and set $D = 1$. For the source code of `matC.m` refer to Appendix B. For the matrix $\underline{E}$ we have, similar to Section 2.2.3,

$$\sum_{E\in\mathcal{E}_h}\int_E \boldsymbol{\psi}_{mj}\cdot\nabla\phi_{nk}\,d\boldsymbol{x} = \delta_{mn}\frac{|\det J_m|}{2}\widetilde{\boldsymbol{\psi}}_j\cdot J_m^{-T}\nabla\widetilde{\phi}_k,$$

$$-\sum_{e\in\Gamma_D}\int_e\{\boldsymbol{\psi}_{mj}\cdot\nu_e\}[\phi_{nk}]\,ds = \sum_{p=1}^{3}\delta_{mn}\mathbb{I}(e_{\alpha(m,p)}\in\Gamma_D)\widetilde{\boldsymbol{\psi}}_j\cdot D_{\pi/2}J_m\widetilde{\boldsymbol{e}}_p\int_0^1\widetilde{\phi}_k|_{\widetilde{\boldsymbol{e}}_p}\,ds$$

and

$$-\sum_{e\in\Gamma_h}\int_e\{\boldsymbol{\psi}_{mj}\cdot\nu_e\}[\phi_{nk}]\,ds = \begin{cases} \sum_{p=1}^{3}\mathbb{I}(e_{\alpha(m,p)}\in\Gamma_h)\frac{1}{2}\widetilde{\boldsymbol{\psi}}_j\cdot D_{\pi/2}J_m\widetilde{\boldsymbol{e}}_p\int_0^1\widetilde{\phi}_k|_{\widetilde{\boldsymbol{e}}_p}\,ds \\ \qquad\qquad \text{if } m = n, \\ -\frac{1}{2}\widetilde{\boldsymbol{\psi}}_j\cdot D_{\pi/2}J_m\widetilde{\boldsymbol{e}}_p\int_0^1\widetilde{\phi}_k|_{\widetilde{\boldsymbol{e}}_q}\,ds \\ \qquad\qquad \text{if } \partial E_m\cap\partial E_n = e_{\alpha(m,p)} = e_{\alpha(n,q)} = \bar{e}, \\ 0 \qquad\qquad \text{otherwise}, \end{cases}$$

where $\widetilde{\boldsymbol{\psi}}_j(s,t) = \boldsymbol{\psi}_{mj}(\boldsymbol{x}(s,t))$ are the local basis functions on the standard triangle T. A function `matE.m` which returns the matrix $\underline{E}$ can be obtained by modifying the function `matC.m` (again, compare the definitions of the matrices). See Appendix B for the source code of `matE.m`.

Using the results (cf. [Bau00, Section 3.3.6])

$$[\phi_{mj}] = s_{mp}\left(\phi_{mj}|_{E_m}\right)|_{e_{\alpha(m,p)}} \qquad \text{and} \qquad s_{mp}|e_{\alpha(m,p)}|\nu_{\alpha(m,p)} = -D_{\pi/2}J_m\widetilde{\boldsymbol{e}}_p,$$

where

$$s_{mp} = \begin{cases} 1 & \text{if } \nu_{\alpha(m,p)} \text{ is exterior to } E_m, \\ -1 & \text{otherwise,} \end{cases}$$

we get for the vector $\boldsymbol{H}^i$

$$\begin{aligned} \sum_{E\in\mathcal{E}_h}\int_E 2\bar{\boldsymbol{h}}_i\cdot\nabla\phi_{mj}\;d\boldsymbol{x} &= \int_{E_m} 2\bar{\boldsymbol{h}}_i\cdot\nabla\phi_{mj}\;d\boldsymbol{x}, \\ &= \iint_T 2|\det J_m|\bar{\boldsymbol{h}}_i(\boldsymbol{x}(s,t))^T J_m^{-T}\nabla\widetilde{\phi}_j\;dsdt, \\ &= |\det J_m|\left(\nabla\widetilde{\phi}_j\right)^T J_m^{-1}\iint_T 2\bar{\boldsymbol{h}}_i(\boldsymbol{x}(s,t))\;dsdt, \quad (4.11) \end{aligned}$$

$$\begin{aligned} &-\sum_{e\in\Gamma_D}\int_e\{2\bar{\boldsymbol{h}}_i\cdot\nu_e\}[\phi_{mj}]\;ds \\ &\quad= -\sum_{p=1}^{3}\mathbb{I}(e_{\alpha(m,p)}\in\Gamma_D)\int_{e_{\alpha(m,p)}} 2\bar{\boldsymbol{h}}_i\cdot\nu_{\alpha(m,p)}s_{mp}\phi_{mj}|_{E_m}\;ds, \\ &\quad= -\sum_{p\in\{1,2,3\}\setminus\{j\}}\mathbb{I}(e_{\alpha(m,p)}\in\Gamma_D)\int_0^1 2s\bar{\boldsymbol{h}}_i(\boldsymbol{z}(s))^T(s_{mp}|e_{\alpha(m,p)}|\nu_{\alpha(m,p)})\;ds, \\ &\quad= \sum_{p\in\{1,2,3\}\setminus\{j\}}\mathbb{I}(e_{\alpha(m,p)}\in\Gamma_D)\int_0^1 2s\bar{\boldsymbol{h}}_i(\boldsymbol{z}(s))^T D_{\pi/2}J_m\widetilde{\boldsymbol{e}}_p\;ds, \qquad (4.12) \end{aligned}$$

and

$$\begin{aligned} -\int_{\bar{e}}\{2\bar{\boldsymbol{h}}_i\cdot\nu_{\bar{e}}\}[\phi_{mj}]\;ds &= -\int_{\bar{e}} 2\bar{\boldsymbol{h}}_i\cdot\nu_{\bar{e}}s_{mp}\phi_{mj}|_{E_m}\;ds, \\ &= -\mathbb{I}(j\neq p)\int_0^1 2s\bar{\boldsymbol{h}}_i(\boldsymbol{z}(s))^T\left(s_{mp}|\bar{e}|\nu_{\bar{e}}\right)\;ds, \\ &= \mathbb{I}(j\neq p)\int_0^1 2s\bar{\boldsymbol{h}}_i(\boldsymbol{z}(s))^T D_{\pi/2}J_m\widetilde{\boldsymbol{e}}_p\;ds, \qquad (4.13) \end{aligned}$$

if $\partial E_m\cap\partial E_n = e_{\alpha(m,p)} = e_{\alpha(n,q)} = \bar{e}$. Here,

$$\boldsymbol{z}(s) = \boldsymbol{n}^{\sigma(m,6-j-p)} + s(\boldsymbol{n}^{\sigma(m,j)} - \boldsymbol{n}^{\sigma(m,6-j-p)}).$$

We obtain $\boldsymbol{H}^i$ with a call to the function `conth_a.m`.

```
conth_a(dt,i,coordinates,elements,dirichlet,interior)
```

The source code of `conth_a.m` can be found on page 77 in Appendix B. Lines 6–13 implement (4.11). Lines 15–29 add (4.12). Finally, lines 31–58 add (4.13).

For the matrix $\underline{F}$ we have

$$\int_\Omega \boldsymbol{\psi}_{mj} \cdot \boldsymbol{\psi}_{nk} \, d\boldsymbol{x} = \delta_{mn}\delta_{jk}\frac{|\det J_m|}{2}.$$

```
F = sparse(2*size(elements,1),2*size(elements,1));
for m = 1:size(elements,1)
    F(2*(m-1)+(1:2),2*(m-1)+(1:2)) = ...
        det([1,1,1;coordinates(elements(m,:),:)'])*eye(2)/2;
end
```

For the matrix $\underline{G}$ we get

$$\begin{aligned}\sum_{E\in\mathcal{E}_h}\int_E \nabla\phi_{mj}\cdot\boldsymbol{\psi}_{nk}\,d\boldsymbol{x} &= \delta_{mn}\int_{E_m}\nabla\phi_{mj}\cdot\boldsymbol{\psi}_{mk}\,d\boldsymbol{x},\\ &= \delta_{mn}\frac{|\det J_m|}{2}\left(\nabla\widetilde{\phi}_j\right)^T J_m^{-1}\widetilde{\boldsymbol{\psi}}_k,\end{aligned}$$

and for the vector $\boldsymbol{J}^i$ we have

$$\begin{aligned}\int_\Omega 2\bar{\boldsymbol{h}}_i\cdot\boldsymbol{\psi}_{mj}\,d\boldsymbol{x} &= \int_{E_m} 2\bar{\boldsymbol{h}}_i\cdot\boldsymbol{\psi}_{mj}\,d\boldsymbol{x},\\ &= |\det J_m|\widetilde{\boldsymbol{\psi}}_j\cdot\iint_T 2\bar{\boldsymbol{h}}_i(\boldsymbol{x}(s,t))\,dsdt.\end{aligned}$$

The functions `matG` and `conth_b` implement these terms.

```
function A = matG(coord,elements)
A1 = sparse(2*size(elements,1),3*size(elements,1));
for m = 1:size(elements,1)
    Jm = [coord(elements(m,2),:)-coord(elements(m,1),:); ...
        coord(elements(m,3),:)-coord(elements(m,1),:)]';
    A1(2*(m-1)+(1:2),3*(m-1)+(1:3)) = inv(Jm)'*gphi()*det(Jm)/2;
end
A = A1;

function f = conth_b(dt,i,coord,elements)
g = zeros(2*size(elements,1),1);
for m = 1:size(elements,1)
    g(2*(m-1)+(1:2)) = ...
        mydblquad(@(x)(h(dt*(i-1),x)+h(dt*(i-2),x)), ...
        coord(elements(m,:),:));
end
f = g;
```

To obtain $\breve{\boldsymbol{U}}$, we use the function `b_0` as described in Section 3.2.2. Modifying it slightly, we can also use it to obtain $\breve{\boldsymbol{S}}$. Simply change the function name to `s_0` and substitute the call to the function `ubreve` by a call to the function `sbreve`, which implements the function $\breve{\boldsymbol{\sigma}}$.

```
function s = s_0(coord,elements)
y = zeros(2*size(elements,1),1);
for m = 1:size(elements,1)
    y(2*(m-1)+(1:2)) = det([1,1,1;coord(elements(m,:),:)'])*...
        mydblquad(@(x) sbreve(map(coord(elements(m,:),:)',x)), ...
        [0 0; 1 0; 0 1]);
end
s = y;
```

The following MATLAB script now computes the numerical solutions for u and $\boldsymbol{\sigma}$.

```
%% Initialisation
M = 4;
mesh
N = 10;
T = 1; dt = T/N;
D = 1; K = 1; mu = 1; gamma = 1;
beta = 2; delta = 10^4;
% kappa = 1 gives a non-symmetric scheme,
% kappa = -1 gives a symmetric scheme
kappa = 1;
U = zeros(3*size(elements,1),N+1);
S = zeros(2*size(elements,1),N+1);

%% Assembly
A = stiff(kappa,D,beta,delta,coordinates,elements,...
    dirichlet,interior);
B = sparse(3*size(elements,1),3*size(elements,1));
F = sparse(2*size(elements,1),2*size(elements,1));
for m = 1:size(elements,1)
    B(3*(m-1)+(1:3),3*(m-1)+(1:3)) = ...
        det([1,1,1;coordinates(elements(m,:),:)'])*...
        [2,1,1;1,2,1;1,1,2]/24;
    F(2*(m-1)+(1:2),2*(m-1)+(1:2)) = ...
        det([1,1,1;coordinates(elements(m,:),:)'])*eye(2)/2;
end
C = matC(coordinates,elements,dirichlet,interior);
E = matE(coordinates,elements,dirichlet,interior);
G = matG(coordinates,elements);

%% Initial condition
U(:,1) = B\b_0(coordinates,elements);
```

```
S(:,1) = F\s_0(coordinates,elements);

%% Time steps
for i = 2:N+1
    % Computing the load vector
    b = L(dt*(i-1),coordinates,elements,neumann) + ...
        L(dt*(i-2),coordinates,elements,neumann);
    b = dt*b + (2*B - dt*A - dt^2*mu*K*C/(2+dt*gamma))*U(:,i-1) - ...
        (4*dt*K*E*S(:,i-1) + dt^2*K*conth_a(dt,i,...
        coordinates,elements,dirichlet,interior))/(2+dt*gamma);

    % Computing the solution
    U(:,i) = (dt*A + 2*B + dt^2*mu*K*C/(2+dt*gamma))\b;
    S(:,i) = ((2+dt*gamma)*F)\((2-dt*gamma)*F*S(:,i-1) + ...
        dt*mu*G*(U(:,i)+U(:,i-1)) + ...
        dt*conth_b(dt,i,coordinates,elements));
end
```

4.3 Numerical Experiments and Graphical Representation

Consider the problem

$$u_t - \Delta u = f + \nabla \cdot \boldsymbol{\sigma}, \tag{4.14}$$

$$\boldsymbol{\sigma}_t + \boldsymbol{\sigma} = \nabla u + \boldsymbol{h}, \tag{4.15}$$

subject to

$$u = 0 \qquad \text{on } \Gamma_D \times I, \tag{4.16}$$

$$(\nabla u + \boldsymbol{\sigma}) \cdot \nu = g \qquad \text{on } \Gamma_N \times I, \tag{4.17}$$

$$u(\boldsymbol{x}, 0) = \breve{u}(\boldsymbol{x}), \tag{4.18}$$

$$\boldsymbol{\sigma}(\boldsymbol{x}, 0) = 0, \tag{4.19}$$

where

$$\Omega = (0,1) \times (0,1) \in \mathbb{R}^2, \quad \Gamma_D = \{\boldsymbol{x} \in \mathbb{R}^2 : \ x = 0 \text{ or } y = 0\},$$

and

$$f(\boldsymbol{x}, t) = (1 + 8\pi^2 t)\sin(2\pi x)\sin(2\pi y) - 2\pi t\big(\cos(2\pi x) - \sin(2\pi y)\big),$$

$$\boldsymbol{h}(\boldsymbol{x}, t) = (1+t)\begin{pmatrix}\sin(2\pi x)\\ \cos(2\pi y)\end{pmatrix} - 2\pi t\begin{pmatrix}\cos(2\pi x)\sin(2\pi y)\\ \sin(2\pi x)\cos(2\pi y)\end{pmatrix},$$

$$g(\boldsymbol{x}, t) = \begin{cases} 2\pi t\cos(2\pi x)\sin(2\pi y) + t\sin(2\pi x) & \text{if } x = 1, \\ 2\pi t\sin(2\pi x)\cos(2\pi y) + t\cos(2\pi y) & \text{if } y = 1, \end{cases}$$

$$\breve{u}(\boldsymbol{x}) = 0.$$

The exact solution to the problem is

$$u(\boldsymbol{x},t) = t\sin(2\pi x)\sin(2\pi y), \qquad \boldsymbol{\sigma}(\boldsymbol{x},t) = t\begin{pmatrix}\sin(2\pi x)\\ \cos(2\pi y)\end{pmatrix}. \tag{4.20}$$

Define the functions `f.m` and `g.m` as in the previous chapters. The values of the function $\boldsymbol{h}$ are given by the function `h.m`.

```
function h = h(t,x)
h = (1+t)*[sin(2*pi*x(1)); cos(2*pi*x(2))] - ...
    2*pi*t*[cos(2*pi*x(1))*sin(2*pi*x(2));...
    sin(2*pi*x(1))*cos(2*pi*x(2))];
```

As before in Section 3.3, we expect the error to be constant as we fix the mesh size and increase the number of time steps and observe that the error seems to decrease with EOC = 1 as we fix the time step and refine the mesh. The error is measured as follows.

$$\bar{e}_{\mathcal{A},0} := k\sum_{i=1}^{N}\left(\|\bar{u}_i - \bar{u}_i^h\|_{\mathcal{A}}^2 + \|\bar{\boldsymbol{\sigma}}_i - \bar{\boldsymbol{\sigma}}_i^h\|_0^2\right)^{\frac{1}{2}}. \tag{4.21}$$

	$\kappa=-1$		$\kappa=1$	
M	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	1.8579		1.8579	
2	1.8191	0.0304	1.8191	0.0304
4	1.5142	0.2646	1.5142	0.2646
8	0.8544	0.8256	0.8544	0.8256
16	0.4426	0.9491	0.4426	0.9491
32	0.2234	0.9864	0.2234	0.9864

Table 4.1: Tabulated errors for $N = 2$, $\delta = 10^4$ and $\beta = 2$.

	$\kappa=-1$		$\kappa=1$	
N	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	1.8243		1.8243	
2	1.8191	0.004133	1.8191	0.004133
4	1.8185	0.00046525	1.8185	0.0004653
8	1.8185	$4.8346\cdot10^{-5}$	1.8185	$4.8369\cdot10^{-5}$
16	1.8185	$-2.3911\cdot10^{-7}$	1.8185	$-2.2765\cdot10^{-7}$
32	1.8185	$-3.1664\cdot10^{-6}$	1.8185	$-3.1606\cdot10^{-6}$

Table 4.2: Tabulated errors for $M = 2$, $\delta = 10^4$ and $\beta = 2$.

	$\kappa = -1$		$\kappa = 1$	
M	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	2.5844		2.3714	
2	1.8399	0.4902	1.8412	0.3651
4	1.5184	0.2771	1.5184	0.2782
8	0.8545	0.8294	0.8545	0.8294
16	0.4425	0.9492	0.4425	0.9492
32	0.2234	0.9864	0.2234	0.9864

Table 4.3: Tabulated errors for $N = 2$, $\delta = 10$ and $\beta = 5$.

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	1.8472		1.8486	
2	1.8399	0.0056504	1.8412	0.0057413
4	1.8388	0.00086007	1.8401	0.00090042
8	1.8387	0.00012474	1.8399	0.00014203
16	1.8387	$-2.9015 \cdot 10^{-6}$	1.8399	$4.5563 \cdot 10^{-6}$
32	1.8387	$-1.7458 \cdot 10^{-5}$	1.8399	$-1.4116 \cdot 10^{-5}$

Table 4.4: Tabulated errors for $M = 2$, $\delta = 10$ and $\beta = 5$.

Plots of u_i^h can be found in Section 3.3. For the graphical representation of $\boldsymbol{\sigma}_i^h$, we offer two possibilities. The first possibility is to plot a vector field using the function `quiver`. `quiver(x,y,u,v,s)` plots arrows with components (`u`,`v`) at the points (`x`,`y`) and scales them by a factor `s`. First, we split the vector $\boldsymbol{S}^i$ into two vectors containing the first and second components of $\boldsymbol{\sigma}_i^h|_E$ for each element E at time level t_i.

```
i = N;
S1 = S(mod(1:size(S,1),2)==1,i+1);
S2 = S(mod(1:size(S,1),2)==0,i+1);
```

Then, we compute the coordinates of the centroid of each element and a scaling factor s.

```
z = zeros(size(elements,1),2);
for m = 1:size(elements,1)
    z(m,:) = sum(coordinates(elements(m,:),:))/3;
end
s = M/max(S(:))/8;
```

A short script plots the mesh.

```
plot([0 1 1 0 0],[0 0 1 1 0])
hold on
```

```
for m = 0:(M-1)
    X = [m*(M+1)+(1:M); (m+1)*(M+1)+(2:(M+1))];
    plot(coordinates(X(:),1),coordinates(X(:),2))
    plot(coordinates(m*(M+1)+((M+1):-1:1),1),...
        coordinates(m*(M+1)+((M+1):-1:1),2))
end
axis([-0.1 1.1 -0.1 1.1]), xlabel('x'), ylabel('y')
```

Finally, the following command plots the vector field.

```
quiver(z(:,1),z(:,2),S1,S2,s)
hold off
```

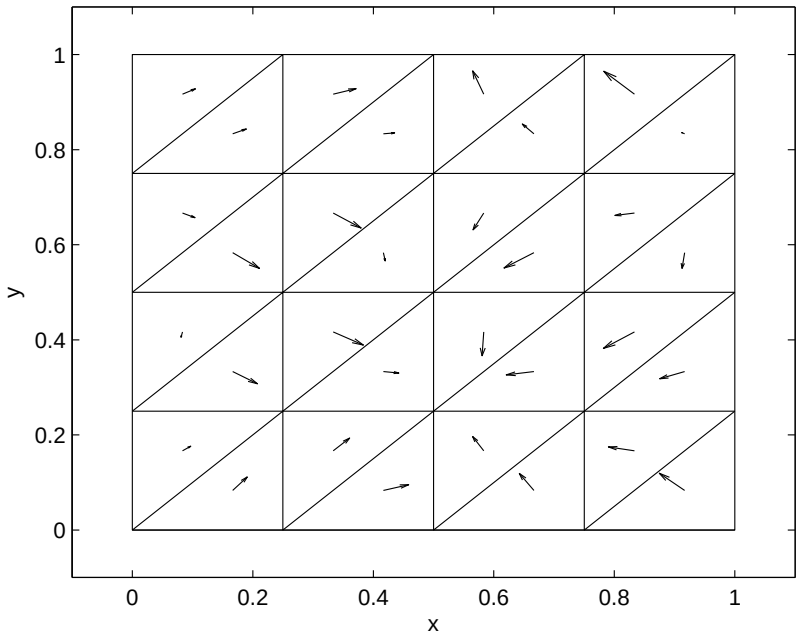

Figure 4.1: 'Quiver plot' at t_5 for $N = 10$, $M = 4$, $\kappa = 1$, $\delta = 10^4$ and $\beta = 2$.

The second possibility we offer is to plot each component of $\boldsymbol{\sigma}_i^h$ using the `trisurf` function. First, we define a vector containing the component we want to plot. We choose to plot the first component.

```
i = N;
S1 = S(mod(1:size(S,1),2)==1,i+1);
```

Define the matrix `Z` and arrange the rows of the matrix `coordinates` exactly as we did on page 22 in Section 2.3. Then, we need to define a vector $\boldsymbol{Q}$ such that

$$Q_{3(m-1)+j} = S^i_{2(m-1)+1} \qquad \text{for } m = 1, \ldots, N_h,\ j = 1, 2, 3.$$

In MATLAB this is easily done by defining

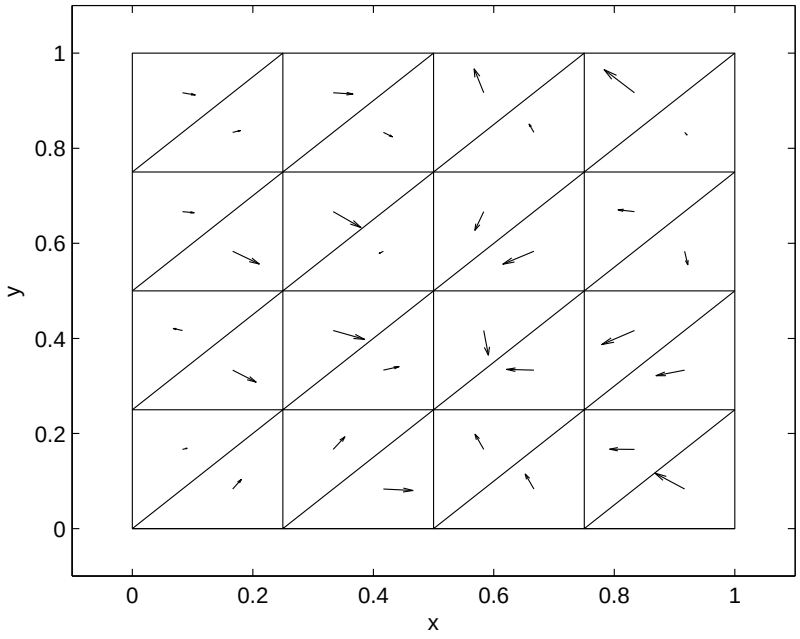

Figure 4.2: 'Quiver plot' at t_{10} for $N = 10$, $M = 4$, $\kappa = 1$, $\delta = 10^4$ and $\beta = 2$.

```
Q = [S1 S1 S1]';
```

The command `Q(:)` now gives the vector $\boldsymbol{Q}$. The following script plots the first component of $\boldsymbol{\sigma}_i^h$.

```
Y = elements';
Z = zeros(3,size(elements,1)); Z(:) = 1:3*size(elements,1);
trisurf(Z',coordinates(Y(:),1),coordinates(Y(:),2),Q(:),...
    'facecolor','interp')
axis([0 1 0 1 min(S(:)) max(S(:))])
xlabel('x'), ylabel('y')
```

To plot the second component of $\boldsymbol{\sigma}_i^h$, replace

```
S1 = S(mod(1:size(S,1),2)==1,i+1);
```

by

```
S2 = S(mod(1:size(S,1),2)==0,i+1);
```

and replace

```
Q = [S1 S1 S1]';
```

by

```
Q = [S2 S2 S2]';
```

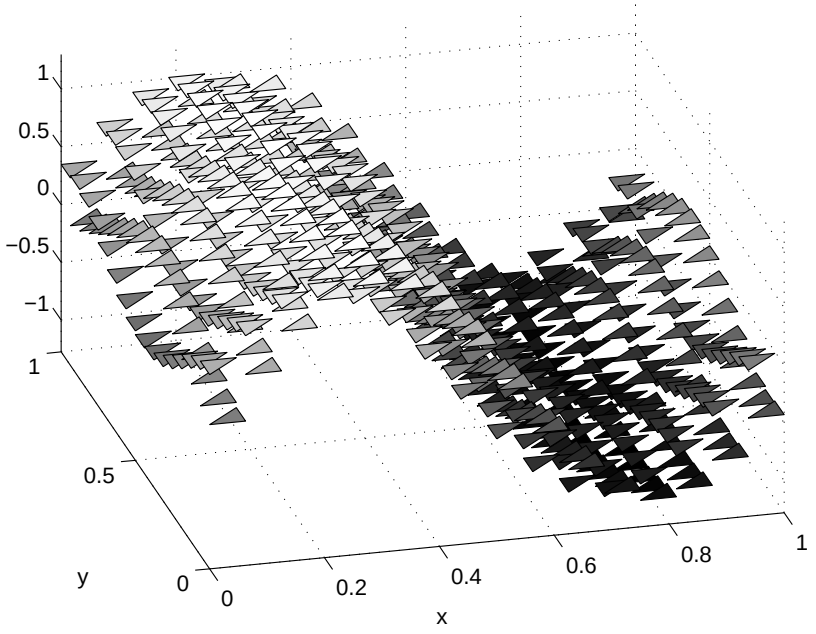

Figure 4.3: 'Trisurf plot' of the first component of $\boldsymbol{\sigma}_i^h$ at t_5 for $N = 5$, $M = 16$, $\kappa = -1$, $\delta = 10^4$ and $\beta = 2$.

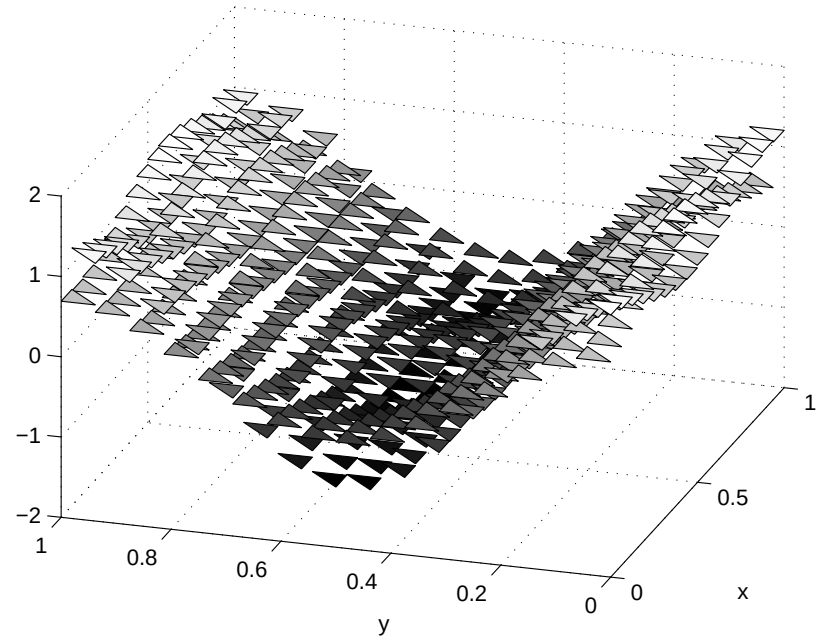

Figure 4.4: 'Trisurf plot' of the second component of $\boldsymbol{\sigma}_i^h$ at t_5 for $N = 5$, $M = 16$, $\kappa = -1$, $\delta = 10^4$ and $\beta = 2$.

Consider problem (4.14)–(4.19) with

$$\Omega = (0,1) \times (0,1) \subset \mathbb{R}^2, \qquad \Gamma_D = \{(x,y) \in \mathbb{R}^2 : \ x = 0\},$$

and

$$f(\boldsymbol{x},t) = 3(t+1)^2 x,$$

$$\boldsymbol{h}(\boldsymbol{x},t) = \begin{pmatrix} 2t + t^2 - (t+1)^3 \\ 2t + t^2 \end{pmatrix},$$

$$g = \begin{cases} -t^2 & \text{if } y = 0, \\ (t+1)^3 + t^2 & \text{if } x = 1, \\ t^2 & \text{if } y = 1, \end{cases}$$

$$\breve{u}(\boldsymbol{x}) = x.$$

Then, the exact solution is

$$u(\boldsymbol{x},t) = (t+1)^3 x,$$

$$\boldsymbol{\sigma}(\boldsymbol{x},t) = t^2 \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

The tables below show that the error is constant as we keep the timestep constant and refine the mesh and seems to decrease with EOC = 2 as we fix the size of the mesh and increase the number of time steps.

For plots of u_i^h, again see Section 3.3. A 'quiver plot' and a 'trisurf plot' of $\boldsymbol{\sigma}_i^h$ are shown below.

	$\kappa = -1$		$\kappa = 1$	
M	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	0.0253		0.0253	
2	0.0256	-0.017657	0.0256	-0.017563
4	0.0257	-0.0053676	0.0257	-0.0053582
8	0.02575	-0.001498	0.02575	-0.0014969
16	0.02576	-0.00039106	0.02576	-0.00039091
32	0.02576	$-9.9772 \cdot 10^{-5}$	0.02576	$-9.9712 \cdot 10^{-5}$

Table 4.5: Tabulated errors for $N = 2$, $\delta = 10^4$ and $\beta = 2$.

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	0.0893		0.0893	
2	0.0256	1.8011	0.0256	1.8011
4	0.0065	1.9688	0.0065	1.9688
8	0.0016	1.9931	0.0016	1.9931
16	0.0004	1.9983	0.0004	1.9983
32	0.0001	1.9996	0.0001	1.9996

Table 4.6: Tabulated errors for $M = 2$, $\delta = 10^4$ and $\beta = 2$.

	$\kappa = -1$		$\kappa = 1$	
M	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	0.023357		0.026328	
2	0.025682	-0.13693	0.026213	0.0063141
4	0.025741	-0.0033003	0.025758	0.025227
8	0.025751	-0.0005306	0.025751	0.0003916
16	0.025757	-0.00036168	0.025757	-0.00033106
32	0.025759	$-9.8975 \cdot 10^{-5}$	0.025759	$-9.798 \cdot 10^{-5}$

Table 4.7: Tabulated errors for $N = 2$, $\delta = 1$ and $\beta = 4$.

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	0.0897		0.0915	
2	0.0257	1.8037	0.0262	1.8031
4	0.0066	1.9686	0.0067	1.9679
8	0.0016	1.9930	0.0017	1.9928
16	0.0004	1.9983	0.0004	1.9982
32	0.0001	1.9995	0.0001	1.9995

Table 4.8: Tabulated errors for $M = 2$, $\delta = 1$ and $\beta = 4$.

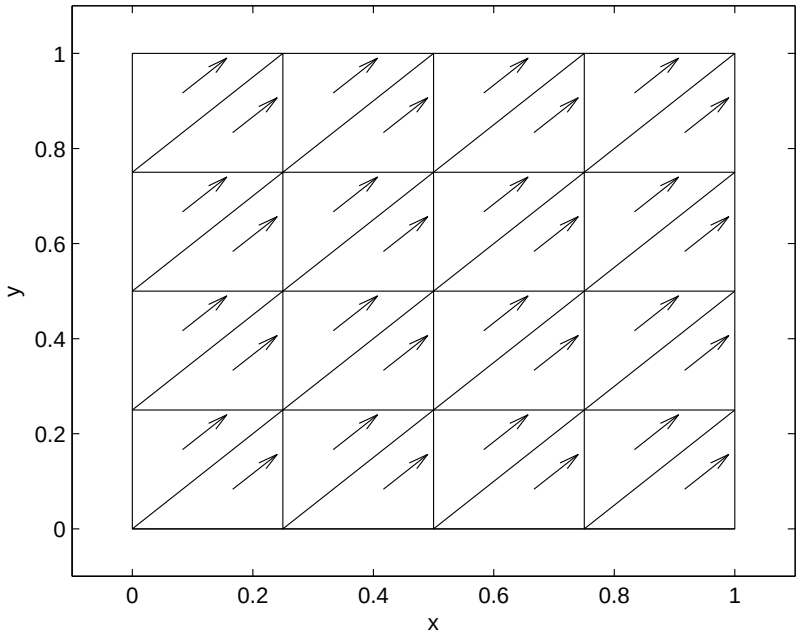

Figure 4.5: 'Quiver plot' at t_5 for $N = 5$, $M = 4$, $\kappa = 1$, $\delta = 10^4$ and $\beta = 2$.

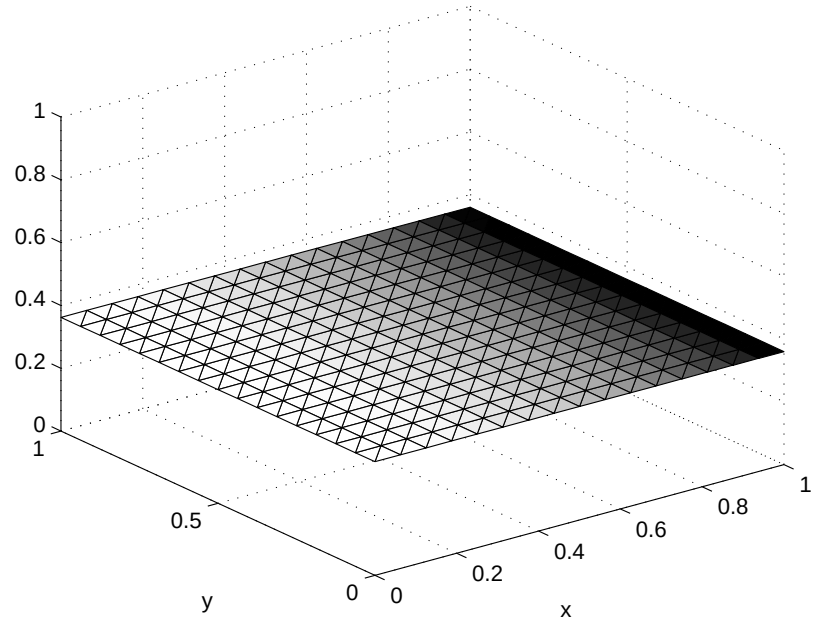

Figure 4.6: 'Trisurf plot' of the first component of $\boldsymbol{\sigma}_i^h$ at t_3 for $N = 5$, $M = 16$, $\kappa = -1$, $\delta = 10^4$ and $\beta = 2$.

Chapter 5

The Nonlinear Diffusion Problem

5.1 The Problem and Variational Formulation

Consider now the following problem. Find $u: \Omega \times I \to \mathbb{R}$ and $\boldsymbol{\sigma}: \Omega \times I \to \mathbb{R}^2$ such that in $\Omega \times I$

$$u_t(\boldsymbol{x},t) - \nabla \cdot D\nabla u(\boldsymbol{x},t) = f(\boldsymbol{x},t) + \nabla \cdot K\boldsymbol{\sigma}(\boldsymbol{x},t), \tag{5.1}$$

$$\boldsymbol{\sigma}_t(\boldsymbol{x},t) + \gamma(u)\boldsymbol{\sigma}(\boldsymbol{x},t) = \mu\nabla u(\boldsymbol{x},t) + \boldsymbol{h}(\boldsymbol{x},t), \tag{5.2}$$

subject to

$$u(\boldsymbol{x},0) = \breve{u}(\boldsymbol{x}),$$
$$\boldsymbol{\sigma}(\boldsymbol{x},0) = \breve{\boldsymbol{\sigma}}(\boldsymbol{x}),$$
$$u(\boldsymbol{x},t) = 0 \quad \text{on } \Gamma_D \times I,$$

and

$$(D\nabla u(\boldsymbol{x},t) + K\boldsymbol{\sigma}(\boldsymbol{x},t)) \cdot \nu(\boldsymbol{x}) = g(\boldsymbol{x},t) \quad \text{on } \Gamma_N \times I.$$

The domain Ω is again a subset of $\mathbb{R}^2$ and $I = (0,T)$ for some $T > 0$. The function $\gamma(u)$ is defined by

$$\gamma(u) = \frac{1}{2}(\gamma_R + \gamma_G) + \frac{1}{2}(\gamma_R - \gamma_G)\tanh\left(\frac{u - u_{RG}}{\Delta}\right),$$

where $\gamma_R \gg \gamma_G > 0$, $\Delta > 0$ and $u_{RG} \in \mathbb{R}$. For $\boldsymbol{h}(\boldsymbol{x},t) = 0$, this is problem (1.1)–(1.4). Similar to Section 4.1, we have, for $u(t) \in C(\bar{\Omega})\ \forall t \in I$,

$$(u_t(t), v) + A(u(t), v) = L(t,v) - \sum_{E \in \mathcal{E}_h} (K\boldsymbol{\sigma}(t), \nabla v)_E + \sum_{e \in \Gamma_h \cup \Gamma_D} \int_e \{K\boldsymbol{\sigma}(t) \cdot \nu_e\}[v]\ ds \qquad \forall v \in \mathcal{D}_1(\mathcal{E}_h)$$

and

$$(\boldsymbol{\sigma}_t(t) + \gamma(u)\boldsymbol{\sigma}(t), \boldsymbol{w}) = \sum_{E \in \mathcal{E}_h} (\mu\nabla u(t), \boldsymbol{w})_E + (\boldsymbol{h}(t), \boldsymbol{w}) \qquad \forall \boldsymbol{w} \in \boldsymbol{\mathcal{D}}_0(\mathcal{E}_h).$$

We can choose between two possibilities of approximating $\gamma(u)$. First, we can approximate $\gamma(u)$ by $\gamma(\bar{u}_i^h)$, where $\bar{u}_i^h = (u_i^h + u_{i-1}^h)/2$ is the true average of the discrete solution. This will lead to a nonlinear numerical scheme. Second, we can approximate $\gamma(u)$ by $\gamma(\mathcal{E}_i u^h)$, where

$$\mathcal{E}_i u^h = \begin{cases} u_0^h & i = 1, \\ \frac{3}{2}u_{i-1}^h - \frac{1}{2}u_{i-2}^h & i = 2, \ldots, N. \end{cases}$$

The second approach will lead to a linear numerical scheme. We will consider only the linear scheme. The discrete formulation is as follows.

For $i = 1, \ldots, N$, find $u_i^h \in \mathcal{D}_1(\mathcal{E}_h)$ and $\boldsymbol{\sigma}_i^h \in \boldsymbol{\mathcal{D}}_0(\mathcal{E}_h)$ such that, for all $v \in \mathcal{D}_1(\mathcal{E}_h)$ and $\boldsymbol{w} \in \boldsymbol{\mathcal{D}}_0(\mathcal{E}_h)$,

$$(\partial_t u_i^h, v) + A(\bar{u}_i^h, v) = L_i(v) - \sum_{E \in \mathcal{E}_h} (K\bar{\boldsymbol{\sigma}}_i^h, \nabla v)_E + \sum_{e \in \Gamma_h \cup \Gamma_D} \int_e \{K\bar{\boldsymbol{\sigma}}_i^h \cdot \nu_e\}[v]\, ds$$

and

$$(\partial_t \boldsymbol{\sigma}_i^h + \gamma(\mathcal{E}_i u^h)\bar{\boldsymbol{\sigma}}_i^h, \boldsymbol{w}) = \sum_{E \in \mathcal{E}_h} (\mu \nabla \bar{u}_i^h, \boldsymbol{w})_E + (\bar{\boldsymbol{h}}_i, \boldsymbol{w}),$$

subject to

$$(u_0^h, v) = (\breve{u}, v),$$
$$(\boldsymbol{\sigma}_0^h, \boldsymbol{w}) = (\breve{\boldsymbol{\sigma}}, \boldsymbol{w}).$$

5.2 Adapting the MATLAB Code

Let

$$\gamma_E^* := \frac{1}{|E|} \int_E \gamma(u)\, d\boldsymbol{x},$$

where $|E| = \int_E d\boldsymbol{x}$. Then, all the calculations in Sections 4.1 and 4.2 are still valid, if we replace γ by γ_E^* on each element E. At each time level, we have to solve the linear equation systems

$$\left(k\underline{A} + 2\underline{B} + \mu k^2 K \underline{C}^1\right) \boldsymbol{U}^i = k\boldsymbol{L}^i + \left(2\underline{B} - k\underline{A} - \mu k^2 K \underline{C}^1\right) \boldsymbol{U}^{i-1} - 4kK\underline{E}^1 \boldsymbol{S}^{i-1} - k^2 K \boldsymbol{H}^i$$

and

$$\underline{F}^1 \boldsymbol{S}^i = \underline{F}^2 \boldsymbol{S}^{i-1} + \mu k \underline{G}(\boldsymbol{U}^i + \boldsymbol{U}^{i-1}) + k\boldsymbol{J}^i,$$

where $\underline{A}$, $\underline{B}$, $\underline{G}$ and $\boldsymbol{J}^i$ are defined as in Section 3.2.1, and $\underline{C}^1$, $\underline{E}^1$, $\underline{F}^1$, $\underline{F}^2$ and $\boldsymbol{H}^i$ are defined by

$$C^1_{3(n-1)+k,3(m-1)+j} = \sum_{E \in \mathcal{E}_h} \int_E \frac{1}{2 + k\gamma_E^*} \nabla\phi_{mj} \cdot \nabla\phi_{nk}\, d\boldsymbol{x} - \sum_{e \in \Gamma_h \cup \Gamma_D} \int_e \left\{ \frac{1}{2 + k\gamma_E^*} \nabla\phi_{mj} \cdot \nu_e \right\} [\phi_{nk}]\, ds,$$

$$E^1_{3(n-1)+k,2(m-1)+j} = \sum_{E\in\mathcal{E}_h}\int_E \frac{1}{2+k\gamma^*_E}\boldsymbol{\psi}_{mj}\cdot\nabla\phi_{nk}\;d\boldsymbol{x} - \sum_{e\in\Gamma_h\cup\Gamma_D}\int_e\left\{\frac{1}{2+k\gamma^*_E}\boldsymbol{\psi}_{mj}\cdot\nu_e\right\}[\phi_{nk}]\;ds,$$

$$F^1_{2(n-1)+k,2(m-1)+j} = \sum_{E\in\mathcal{E}_h}\int_E (2+k\gamma^*_E)\boldsymbol{\psi}_{mj}\cdot\boldsymbol{\psi}_{nk}\;d\boldsymbol{x},$$

$$F^2_{2(n-1)+k,2(m-1)+j} = \sum_{E\in\mathcal{E}_h}\int_E (2-k\gamma^*_E)\boldsymbol{\psi}_{mj}\cdot\boldsymbol{\psi}_{nk}\;d\boldsymbol{x}$$

and

$$H^i_{3(m-1)+j} = \sum_{E\in\mathcal{E}_h}\int_E \frac{2}{2+k\gamma^*_E}\bar{\boldsymbol{h}}_i\cdot\nabla\phi_{mj}\;d\boldsymbol{x} - \sum_{e\in\Gamma_h\cup\Gamma_D}\int_e\left\{\frac{2}{2+k\gamma^*_E}\bar{\boldsymbol{h}}_i\cdot\nu_e\right\}[\phi_{mj}]\;ds. \quad (5.3)$$

We have

$$C^1_{3(n-1)+k,3(m-1)+j} = \frac{1}{2+k\gamma^*_{E_m}}C_{3(n-1)+k,3(m-1)+j},$$

$$E^1_{3(n-1)+k,2(m-1)+j} = \frac{1}{2+k\gamma^*_{E_m}}E_{3(n-1)+k,2(m-1)+j},$$

$$F^1_{2(n-1)+k,2(m-1)+j} = (2+k\gamma^*_{E_m})F_{2(n-1)+k,2(m-1)+j}$$

and

$$F^2_{2(n-1)+k,2(m-1)+j} = (2-k\gamma^*_{E_m})F_{2(n-1)+k,2(m-1)+j},$$

where $\underline{C} = (C_{jk})^{j=1,\ldots,3N_h}_{k=1,\ldots,3N_h}$, $\underline{E} = (E_{jk})^{j=1,\ldots,3N_h}_{k=1,\ldots,2N_h}$ and $\underline{F} = (F_{jk})^{j=1,\ldots,2N_h}_{k=1,\ldots,2N_h}$ are defined as in Section 3.2.1. The value of γ^*_E is given by the function `gammaStar.m`.

```
function g = gammaStar(U,gr,gg,urg,d)
g = 2*dblquad(@(x,y) gamma(phi(x,y)*U,gr,gg,urg,d).*...
    (x'+y<1),0,1,0,1);
```

Here, we use the pre-defined function `dblquad.m` because it is more accurate than the function `mydblquad.m`. For this we need to adapt the function `phi.m`.

```
function phi = phi(x,y)
if nargin == 2
    phi = [1-x'-y, x', y*ones(size(x,2),1)];
else
    phi = [1-x(1)-x(2), x(1), x(2)];
end
```

We compute $\underline{C}$, $\underline{E}$ and $\underline{F}$, and initialise $\underline{C}^1$, $\underline{E}^1$, $\underline{F}^1$ and $\underline{F}^2$ once.

```
F = sparse(2*size(elements,1),2*size(elements,1));
F1 = sparse(2*size(elements,1),2*size(elements,1));
F2 = sparse(2*size(elements,1),2*size(elements,1));
for m = 1:size(elements,1)
    F(2*(m-1)+(1:2),2*(m-1)+(1:2)) = ...
        det([1,1,1;coordinates(elements(m,:),:)'])*eye(2)/2;
end
C = matC(coordinates,elements,dirichlet,interior);
C1 = sparse(size(C,1),size(C,2));
E = matE(coordinates,elements,dirichlet,interior);
E1 = sparse(size(E,1),size(E,2));
```

Then, at each time step, we define a vector which contains the values of $\mathcal{E}_i u^h$ at each node and obtain $\underline{C}^1$, $\underline{E}^1$, $\underline{F}^1$ and $\underline{F}^2$ with the following script.

```
if i == 2
    Eu = U(:,1);
else
    Eu = (3*U(:,i-1)-U(:,i-2))/2;
end

for m = 1:size(elements,1)
    gmm = gammaStar(Eu(3*(m-1)+(1:3)),gr,gg,urg,d);
    C1(:,3*(m-1)+(1:3)) = C(:,3*(m-1)+(1:3))/(2+dt*gmm);
    E1(:,2*(m-1)+(1:2)) = E(:,2*(m-1)+(1:2))/(2+dt*gmm);
    F1(:,2*(m-1)+(1:2)) = (2+dt*gmm)*F(:,2*(m-1)+(1:2));
    F2(:,2*(m-1)+(1:2)) = (2-dt*gmm)*F(:,2*(m-1)+(1:2));
end
```

To obtain the vector $\boldsymbol{H}^i$, we need to adapt the function `conth_a.m`. First, we define functions $\boldsymbol{h}_1$ and $\boldsymbol{h}_2$ such that

$$\boldsymbol{h}(\boldsymbol{x},t) = \boldsymbol{h}_1(\boldsymbol{x},t) + \gamma(u)\boldsymbol{h}_2(\boldsymbol{x},t).$$

The values of $\boldsymbol{h}_1$ and $\boldsymbol{h}_2$ are given by the MATLAB functions `h1.m` and `h2.m`. Then, we define a MATLAB function `hbar.m`, which returns the vector

$$\big(\boldsymbol{h}_1(\boldsymbol{x},t_i) + \boldsymbol{h}_1(\boldsymbol{x},t_{i-1})\big) + \gamma(\mathcal{E}_i u)\big(\boldsymbol{h}_2(\boldsymbol{x},t_i) + \boldsymbol{h}_2(\boldsymbol{x},t_{i-1})\big).$$

```
function z = hbar(dt,i,x,U,gr,gg,urg,d,coord)
z = h1(dt*(i-1),x) + h1(dt*(i-2),x) + ...
    gamma(phi(invmap(coord',x))*U,gr,gg,urg,d)*...
    (h2(dt*(i-1),x) + h2(dt*(i-2),x));
```

Here, the function `invmap.m` maps from any given element to the standard triangle.

```
function y = invmap(vertices,x)
invJ = inv([vertices(:,2)-vertices(:,1), ...
    vertices(:,3)-vertices(:,1)]);
y = invJ*(x-vertices(:,1));
```

With this set-up, make the following changes to the code of `conth_a.m` on page 77. First, change lines 10–12 to:

```
gmm = gammaStar(U(3*(m-1)+(1:3)),gr,gg,urg,d);
g(3*(m-1)+(1:3)) = gphi()'*invJm*...
    mydblquad(@(x) hbar(dt,i,x,U(3*(m-1)+(1:3)),...
    gr,gg,urg,d,coord(elements(m,:),:)), ...
    coord(elements(m,:),:))/(2+dt*gmm);
```

Second, change lines 20–28 to:

```
gmm = gammaStar(U(3*(m-1)+(1:3)),gr,gg,urg,d);
P1 = myquad(@(x) hbar(dt,i,x',U(3*(m-1)+(1:3)),...
    gr,gg,urg,d,coord(elements(m,:),:))'*rotmat*...
    Jm*standedges(:,p), coord(dirichlet(l,2),:), ...
    coord(dirichlet(l,1),:));
P2 = myquad(@(x) hbar(dt,i,x',U(3*(m-1)+(1:3)),...
    gr,gg,urg,d,coord(elements(m,:),:))'*rotmat*...
    Jm*standedges(:,p), coord(dirichlet(l,1),:), ...
    coord(dirichlet(l,2),:));
g(3*(m-1)+j(j~=p)) = g(3*(m-1)+j(j~=p)) + ...
    [P1 P2]'/norm(coord(dirichlet(l,1),:)-...
    coord(dirichlet(l,2),:))/(2+dt*gmm);
```

Third, change lines 40–47 to:

```
gmm1 = gammaStar(U(3*(m-1)+(1:3)),gr,gg,urg,d);
gmm2 = gammaStar(U(3*(n-1)+(1:3)),gr,gg,urg,d);
P1 = myquad(@(x) hbar(dt,i,x',U(3*(m-1)+(1:3)),...
    gr,gg,urg,d,coord(elements(m,:),:))'*rotmat*...
    Jm*standedges(:,p), coord(interior(l,2),:), ...
    coord(interior(l,1),:));
P2 = myquad(@(x) hbar(dt,i,x',U(3*(m-1)+(1:3)),...
    gr,gg,urg,d,coord(elements(m,:),:))'*rotmat*...
    Jm*standedges(:,p), coord(interior(l,1),:), ...
    coord(interior(l,2),:));
g(3*(m-1)+j(j~=p)) = g(3*(m-1)+j(j~=p)) + [P1 P2]'/...
    norm(coord(interior(l,1),:)-coord(interior(l,2),:))*...
    (1/(2+dt*gmm1)+1/(2+dt*gmm2))/2;
```

Finally, change lines 50–57 to:

```
P1 = myquad(@(x) hbar(dt,i,x',U(3*(n-1)+(1:3)),...
    gr,gg,urg,d,coord(elements(n,:),:))'*rotmat*...
    Jn*standedges(:,q), coord(interior(l,1),:), ...
    coord(interior(l,2),:));
P2 = myquad(@(x) hbar(dt,i,x',U(3*(n-1)+(1:3)),...
    gr,gg,urg,d,coord(elements(n,:),:))'*rotmat*...
    Jn*standedges(:,q), coord(interior(l,2),:), ...
    coord(interior(l,1),:));
g(3*(n-1)+j(j~=q)) = g(3*(n-1)+j(j~=q)) + [P1 P2]'/...
    norm(coord(interior(l,1),:)-coord(interior(l,2),:))*...
    (1/(2+dt*gmm1)+1/(2+dt*gmm2))/2;
```

See Appendix C for the full adapted MATLAB code.

In order to compute the vector $\boldsymbol{J}^i$, we have to change the function `conth_b.m` to the following function.

```
function y = conth_b(dt,i,U,coord,elements,gr,gg,urg,d)
g = zeros(2*size(elements,1),1);
for m = 1:size(elements,1)
    g(2*(m-1)+(1:2)) = ...
        mydblquad(@(x) hbar(dt,i,x,U(3*(m-1)+(1:3)),...
        gr,gg,urg,d,coord(elements(m,:),:)), ...
        coord(elements(m,:),:));
end
y = g;
```

Now, the following script computes the numerical solutions u_i^h and $\boldsymbol{\sigma}_i^h$.

```
%% Initialisation
M = 1;
mesh
N = 1;
T = 1; dt = T/N;
D = 1; K = 1; mu = 1;
gr = 10; gg = 0.1; urg = 0.5; d = 0.1;
beta = 2; delta = 10^4;
% kappa = 1 gives a non-symmetric scheme,
% kappa = -1 gives a symmetric scheme
kappa = 1;

U = zeros(3*size(elements,1),N+1);
S = zeros(2*size(elements,1),N+1);

%% Assembly
A = stiff(kappa,D,beta,delta,coordinates,elements,...
    dirichlet,interior);
B = sparse(3*size(elements,1),3*size(elements,1));
```

```
F = sparse(2*size(elements,1),2*size(elements,1));
F1 = sparse(2*size(elements,1),2*size(elements,1));
F2 = sparse(2*size(elements,1),2*size(elements,1));
for m = 1:size(elements,1)
    B(3*(m-1)+(1:3),3*(m-1)+(1:3)) = ...
        det([1,1,1;coordinates(elements(m,:),:)'])*...
        [2,1,1;1,2,1;1,1,2]/24;
    F(2*(m-1)+(1:2),2*(m-1)+(1:2)) = ...
        det([1,1,1;coordinates(elements(m,:),:)'])*eye(2)/2;
end
C = matC(coordinates,elements,dirichlet,interior);
C1 = sparse(size(C,1),size(C,2));
E = matE(coordinates,elements,dirichlet,interior);
E1 = sparse(size(E,1),size(E,2));
G = matG(coordinates,elements);

%% Initial condition
U(:,1) = B\b_0(coordinates,elements);
S(:,1) = F\s_0(coordinates,elements);

%% Time steps
for i = 2:N+1
    if i == 2
        Eu = U(:,1);
    else
        Eu = (3*U(:,i-1)-U(:,i-2))/2;
    end
    ch = conth_a(dt,i,Eu,gr,gg,urg,d,...
        coordinates,elements,dirichlet,interior);
    for m = 1:size(elements,1)
        gmm = gammaStar(Eu(3*(m-1)+(1:3)),gr,gg,urg,d);
        C1(:,3*(m-1)+(1:3)) = C(:,3*(m-1)+(1:3))/(2+dt*gmm);
        E1(:,2*(m-1)+(1:2)) = E(:,2*(m-1)+(1:2))/(2+dt*gmm);
        F1(:,2*(m-1)+(1:2)) = (2+dt*gmm)*F(:,2*(m-1)+(1:2));
        F2(:,2*(m-1)+(1:2)) = (2-dt*gmm)*F(:,2*(m-1)+(1:2));
    end

    % Computing the load vector
    b = L(dt*(i-1),coordinates,elements,neumann) + ...
        L(dt*(i-2),coordinates,elements,neumann);
    b = dt*b + (2*B - dt*A - dt^2*mu*K*C1)*U(:,i-1) - ...
        4*dt*K*E1*S(:,i-1) - dt^2*K*ch;

    % Computing the solution
    U(:,i) = (dt*A + 2*B + dt^2*mu*K*C1)\b;
```

```
    S(:,i) = F1\(F2*S(:,i-1) + ...
        dt*mu*G*(U(:,i)+U(:,i-1)) + ...
        dt*conth_b(dt,i,Eu,coordinates,elements,gr,gg,urg,d));
end
```

5.3 Numerical Experiments

Consider first the problem

$$u_t - \Delta u = f + \nabla \cdot \boldsymbol{\sigma}, \tag{5.4}$$
$$\boldsymbol{\sigma}_t + \gamma(u)\boldsymbol{\sigma} = \nabla u + \boldsymbol{h}, \tag{5.5}$$

subject to

$$u = 0 \qquad \text{on } \Gamma_D \times I, \tag{5.6}$$
$$(\nabla u + \boldsymbol{\sigma}) \cdot \nu = g \qquad \text{on } \Gamma_N \times I, \tag{5.7}$$
$$u(\boldsymbol{x}, 0) = 0, \tag{5.8}$$
$$\boldsymbol{\sigma}(\boldsymbol{x}, 0) = 0, \tag{5.9}$$

where

$$\Omega = (0,1) \times (0,1) \in \mathbb{R}^2, \quad \Gamma_D = \{\boldsymbol{x} \in \mathbb{R}^2 : \ x = 0 \text{ or } y = 0\},$$

and

$$f(\boldsymbol{x}, t) = (1 + 8\pi^2 t)\sin(2\pi x)\sin(2\pi y) - 2\pi t\big(\cos(2\pi x) - \sin(2\pi y)\big),$$

$$\boldsymbol{h}(\boldsymbol{x}, t) = \begin{pmatrix} \sin(2\pi x) - 2\pi t \cos(2\pi x)\sin(2\pi y) \\ \cos(2\pi y) - 2\pi t \sin(2\pi x)\cos(2\pi y) \end{pmatrix} + \gamma(u) \begin{pmatrix} t\sin(2\pi x) \\ t\cos(2\pi y) \end{pmatrix},$$

$$g(\boldsymbol{x}, t) = \begin{cases} 2\pi t \cos(2\pi x)\sin(2\pi y) + t\sin(2\pi x) & \text{if } x = 1, \\ 2\pi t \sin(2\pi x)\cos(2\pi y) + t\cos(2\pi y) & \text{if } y = 1, \end{cases}$$

For the parameters $\gamma_R,\ \gamma_G,\ u_{RG}$ and Δ we choose

$$\gamma_R = 10, \quad \gamma_G = 0.1, \quad u_{RG} = 0.5 \quad \text{and} \quad \Delta = 0.1.$$

The exact solution to the problem is again (cf. (4.20))

$$u(\boldsymbol{x}, t) = t\sin(2\pi x)\sin(2\pi y), \qquad \boldsymbol{\sigma}(\boldsymbol{x}, t) = t \begin{pmatrix} \sin(2\pi x) \\ \cos(2\pi y) \end{pmatrix}.$$

Again, the error seems to decrease with EOC = 1 in space and is constant in time. This result is consistent with the theory in [RS]. Rivière and Shaw provide the following error estimate.

$$\|u(t_N) - u_N^h\|_0 + \|\boldsymbol{\sigma}(t_N) - \boldsymbol{\sigma}_N^h\|_0 + k\sum_{i=1}^{N} \left(\|\bar{u}_i - \bar{u}_i^h\|_{\mathcal{A}}^2 + \|\bar{\boldsymbol{\sigma}}_i - \bar{\boldsymbol{\sigma}}_i^h\|_0^2\right)^{\frac{1}{2}} \leq C(h + k^2).$$

It follows that

$$\bar{e}_{\mathcal{A},0} \leq C(h + k^2),$$

where $\bar{e}_{\mathcal{A},0}$ is defined as in (4.21). So, for a solution which is linear in time and nonlinear in space, we expect the error to be constant in time and decrease as $O(h)$ in space.

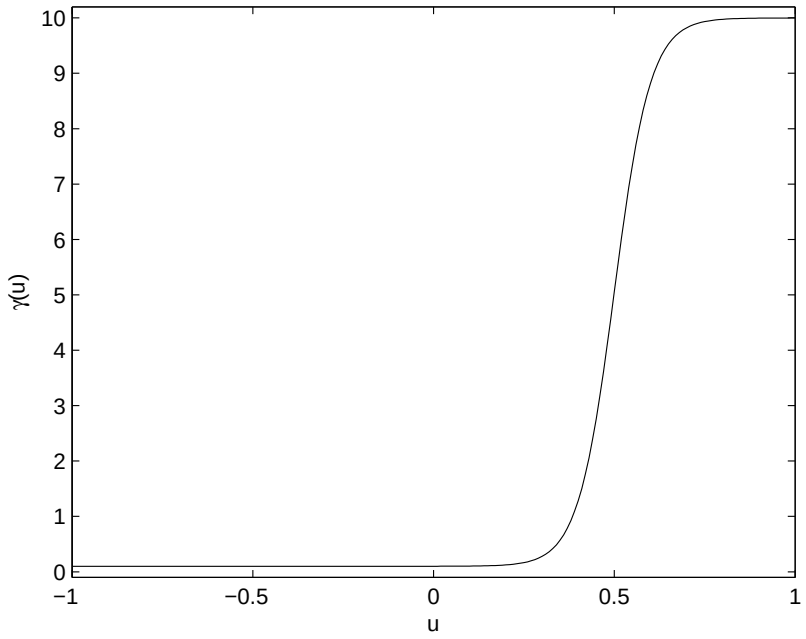

Figure 5.1: The function $\gamma(u)$ for $\gamma_R = 10,\ \gamma_G = 0.1,\ u_{RG} = 0.5$ and $\Delta = 0.1$.

	$\kappa = -1$		$\kappa = 1$	
M	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	2.0320		2.0266	
2	1.8373	0.1453	1.8375	0.1413
4	1.5323	0.2619	1.5324	0.2620
8	0.8639	0.8268	0.8639	0.8269
16	0.4476	0.9486	0.4476	0.9486
32	0.2260	0.9862	0.2260	0.9862

Table 5.1: Tabulated errors for $N = 2$, $\delta = 10^2$ and $\beta = 3$.

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	1.8557		1.8560	
2	1.8373	0.014358	1.8375	0.014412
4	1.8343	0.0023291	1.8346	0.0023315
8	1.8338	0.00039565	1.8341	0.0003987
16	1.8337	$8.054 \cdot 10^{-5}$	1.8340	$8.1997 \cdot 10^{-5}$
32	1.8337	$1.4729 \cdot 10^{-5}$	1.8339	$1.5379 \cdot 10^{-5}$

Table 5.2: Tabulated errors for $M = 2$, $\delta = 10^2$ and $\beta = 3$.

	$\kappa = -1$		$\kappa = 1$	
M	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	1.8633		1.8633	
2	1.8273	0.0281	1.8273	0.0281
4	1.5257	0.2603	1.5257	0.2603
8	0.8631	0.8218	0.8631	0.8218
16	0.4475	0.9476	0.4475	0.9476
32	0.2259	0.9860	0.2259	0.9860

Table 5.3: Tabulated errors for $N = 2$, $\delta = 10^4$ and $\beta = 5$.

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	1.8438		1.8438	
2	1.8273	0.012979	1.8273	0.012979
4	1.8249	0.0019157	1.8249	0.0019157
8	1.8245	0.00028299	1.8245	0.00028299
16	1.8245	$5.2201 \cdot 10^{-5}$	1.8245	$5.22 \cdot 10^{-5}$
32	1.8245	$9.3898 \cdot 10^{-6}$	1.8245	$9.3889 \cdot 10^{-6}$

Table 5.4: Tabulated errors for $M = 2$, $\delta = 10^4$ and $\beta = 5$.

Consider now problem (5.4)–(5.9) with

$$\Omega = (0,1) \times (0,1) \subset \mathbb{R}^2, \qquad \Gamma_D = \{(x,y) \in \mathbb{R}^2 : \ x = 0\},$$

and

$$f(\boldsymbol{x}, t) = 3t^2 x,$$

$$\boldsymbol{h}(\boldsymbol{x}, t) = \begin{pmatrix} 2t - t^3 \\ 2t \end{pmatrix} + \gamma(u) \begin{pmatrix} t^2 \\ t^2 \end{pmatrix},$$

$$g = \begin{cases} -t^2 & \text{if } y = 0, \\ t^3 + t^2 & \text{if } x = 1, \\ t^2 & \text{if } y = 1, \end{cases}$$

Again,

$$\gamma_R = 10, \quad \gamma_G = 0.1, \quad u_{RG} = 0.5 \quad \text{and} \quad \Delta = 0.1.$$

The exact solution is

$$u(\boldsymbol{x}, t) = t^3 x, \qquad \boldsymbol{\sigma}(\boldsymbol{x}, t) = t^2 \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

We expect the error to be constant in space and decrease as $O(h^2)$ in time. The tables show that the error is constant in space, but there seems to be a problem with convergence in time. For up to $N = 16$ we have EOC ≈ 2 (see Table 5.6), but then the EOC falls below 1 for $M = 64$. We get slightly better results if we choose $u_{RG} = 0.9$.

Now, change the function u to

$$u(\boldsymbol{x}, t) = (t + 1)^3 x,$$

that is, consider (5.4)–(5.7) and (5.9) with

$$u(\boldsymbol{x}, 0) = x,$$

$$f(\boldsymbol{x}, t) = 3(t + 1)^2 x,$$

$$\boldsymbol{h}(\boldsymbol{x}, t) = \begin{pmatrix} 2t - (t + 1)^3 \\ 2t \end{pmatrix} + \gamma(u) \begin{pmatrix} t^2 \\ t^2 \end{pmatrix},$$

$$g(\boldsymbol{x}, t) = \begin{cases} -t^2 & \text{if } y = 0, \\ (t + 1)^3 + t^2 & \text{if } x = 1, \\ t^2 & \text{if } y = 1. \end{cases}$$

The error converges to a nonzero constant in time for $u_{RG} = 4$. We get better results for $u_{RG} = 7.9$, but we still see the EOC decrease as we keep refining the time step. In space the error converges to a nonzero constant as expected.

	$\kappa = -1$		$\kappa = 1$	
M	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	0.025116		0.025118	
2	0.025476	-0.02054	0.025477	-0.020444
4	0.025586	-0.00618	0.025586	-0.006168
8	0.025616	-0.00171	0.025616	-0.001709
16	0.025624	-0.00045	0.025624	-0.000445
32	0.025626	-0.00011	0.025626	-0.000113

Table 5.5: Tabulated errors for $u(\boldsymbol{x}, t) = t^3 x$, $u_{RG} = 0.5$, $N = 2$, $\delta = 10^4$ and $\beta = 2$.

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	0.087875		0.087876	
2	0.025476	1.7863	0.025477	1.7863
4	0.006567	1.9558	0.006568	1.9557
8	0.001702	1.9477	0.001703	1.9477
16	0.000436	1.9637	0.000436	1.9637
32	0.000145	1.5926	0.000145	1.5926
64	0.000082	0.8153	0.000082	0.8154

Table 5.6: Tabulated errors for $u(\boldsymbol{x}, t) = t^3 x$, $u_{RG} = 0.5$, $M = 2$, $\delta = 10^4$ and $\beta = 2$.

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	0.087875		0.087875	
2	0.025477	1.7863	0.025477	1.7863
4	0.006566	1.9561	0.006566	1.9561
8	0.001653	1.9901	0.001653	1.9901
16	0.000425	1.9600	0.000425	1.9600
32	0.000116	1.8757	0.000116	1.8757
64	0.000040	1.5345	0.000040	1.5345

Table 5.7: Tabulated errors for $u(\boldsymbol{x}, t) = t^3 x$, $u_{RG} = 0.9$, $M = 2$, $\delta = 10^4$ and $\beta = 2$.

	$\kappa = -1$		$\kappa = 1$	
M	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	0.070118		0.070119	
2	0.040633	0.78715	0.040633	0.78716
4	0.027018	0.58870	0.027019	0.58871
8	0.025649	0.07501	0.025649	0.07502
16	0.025631	0.00106	0.025631	0.00106
32	0.025632	-0.00010	0.025632	-0.00010

Table 5.8: Tabulated errors for $u(\boldsymbol{x}, t) = (t+1)^3 x$, $u_{RG} = 4$, $N = 2$, $\delta = 10^4$ and $\beta = 2$.

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	0.0879		0.0879	
2	0.0406	1.1128	0.0406	1.1128
4	0.0139	1.5441	0.0139	1.5441
8	0.0066	1.0831	0.0066	1.0831
16	0.0037	0.8164	0.0037	0.8165
32	0.0028	0.4234	0.0028	0.4234
64	0.0030	-0.1145	0.0030	-0.1145

Table 5.9: Tabulated errors for $u(\boldsymbol{x}, t) = (t+1)^3 x$, $u_{RG} = 4$, $M = 2$, $\delta = 10^4$ and $\beta = 2$.

	$\kappa = -1$		$\kappa = 1$	
N	$\bar{e}_{\mathcal{A},0}$	EOC	$\bar{e}_{\mathcal{A},0}$	EOC
1	0.087875		0.087875	
2	0.025477	1.7863	0.025477	1.7863
4	0.006566	1.9561	0.006566	1.9561
8	0.001654	1.9892	0.001654	1.9892
16	0.000414	1.9969	0.000414	1.9969
32	0.000112	1.8901	0.000112	1.8901
64	0.000042	1.4286	0.000042	1.4286

Table 5.10: Tabulated errors for $u(\boldsymbol{x}, t) = (t+1)^3 x$, $u_{RG} = 7.9$, $M = 2$, $\delta = 10^4$ and $\beta = 2$.

Chapter 6

Conclusions

In the previous chapters we have provided a MATLAB 7.0.1 implementation for the discontinuous Galerkin finite element method. We have chosen several model problems to demonstrate the flexibility of the method. These model problems included Poisson's equation, the heat equation, a linear and a nonlinear diffusion problem.

The DG FEM depends on the parameters δ and β. The choice of δ and β affects the convergence of the method. We have provided several choices of δ and β for which the method is convergent. But not all values of δ and β give reasonable results. Table 6.1 shows the error and EOC for the problem described in Section 2.3 with $\delta = 10^8$ and $\beta = 5$. Here, the stiffness matrix is badly conditioned for $M = 32$.

	$\kappa = -1$		$\kappa = 1$	
M	$\|u - u_h\|_{\mathcal{A}}$	EOC	$\|u - u_h\|_{\mathcal{A}}$	EOC
1	3.6066		3.6066	
2	3.5496	0.0230	3.5496	0.0230
4	3.0039	0.2408	3.0039	0.2408
8	1.6735	0.8440	1.6735	0.8440
16	0.8634	0.9547	0.8630	0.9553
32	1.9970	-1.2096	1.1333	-0.3931

Table 6.1: Tabulated errors for $\delta = 10^8$ and $\beta = 5$.

We need loops to compute the matrices and vectors which describe the linear systems we need to solve. Vectorisation speeds up the MATLAB code, but is not always possible. We cannot avoid element-wise computation, which is the reason for the code being slow.

Apart from the code for the nonlinear diffusion problem being very slow, the results in Section 5.3 have not been understood completely by the time of writing this dissertation. The sudden decrease of the EOC as the time step is refined cannot be explained yet.

Further work on the following topics is suggested:

- Determine a reasonable range for the values of δ and β.

- Try to speed up the execution of the code.
- Try to explain the numerical results in Section 5.3.
- Implement the nonlinear scheme for the nonlinear diffusion problem.

Bibliography

[ACF99] Jochen Alberty, Carsten Carstensen, and Stefan A. Funken. Remarks around 50 lines of Matlab: short finite element implementation. *Numerical Algorithms*, 20:117–137, 1999.

[Bau00] Norbert Bauermeister. A Discontinuous Galerkin Finite Element Method for Elliptic Partial Differential Equations. Master's thesis, Brunel University, 2000.

[CJW95] Donald S. Cohen, Andrew B. White Jr., and Thomas P. Witelski. Shock Formation in a Multidimensional Viscoelastic Diffusive System. *SIAM J. Appl. Math.*, 55:348–368, 1995.

[Mat04] The MathWorks, Inc. *MATLAB 7.0.1 Help*, 2004.

[MH03] Patrick Marchand and O. Thomas Holland. *Graphics and GUIs with MATLAB*. Chapman & Hall/CRC, Boca Raton, Florida, USA, third edition, 2003.

[RS] Béatrice Rivière and Simon Shaw. Discontinuous Galerkin finite element approximation of nonlinear non-Fickian diffusion in viscoelastic polymers. In preparation.

[RW99] Béatrice Rivière and Mary F. Wheeler. A Discontinuous Galerkin method applied to nonlinear parabolic equations. Technical Report 99–26, TICAM, University of Texas, Austin, 1999. (see www.ticam.utexas.edu/research/reports/1999).

[Tho97] Vidar Thomée. *Galerkin Finite Element Methods for Parabolic Problems*. Springer Series in Computational Mathematics. Springer-Verlag, Berlin, Heidelberg, 1997.

Appendix A

Poisson's Equation

A.1 The Stiffness Matrix

```
 1  function A = stiff(kappa,D,beta,delta,coord,elements,...
 2      dirichlet,interior)
 3
 4  rotmat = [0, -1; 1, 0]; standedges = [-1, 0, 1; 1, -1, 0];
 5  s = size(elements,1); j = 1:3;
 6  A1 = sparse(3*s,3*s);
 7  A2a = sparse(3*s,3*s); A2b = sparse(3*s,3*s);
 8  A3a = sparse(3*s,3*s); A3b = sparse(3*s,3*s);
 9  A4a = sparse(3*s,3*s); A4b = sparse(3*s,3*s);
10
11  for m = 1:s
12      Jm = [coord(elements(m,2),:)-coord(elements(m,1),:);...
13          coord(elements(m,3),:)-coord(elements(m,1),:)]';
14      B = inv(Jm'*Jm);
15
16      A1(3*(m-1)+j,3*(m-1)+j) = A1(3*(m-1)+j,3*(m-1)+j) + ...
17          gphi()'*B*gphi()*det(Jm)*D/2;
18  end
19
20  for l = 1:size(dirichlet,1)
21      e = norm(coord(dirichlet(l,1),:)-...
22          coord(dirichlet(l,2),:))^(1-beta);
23      m = dirichlet(l,3); p = dirichlet(l,4);
24
25      Jm = [coord(elements(m,2),:)-coord(elements(m,1),:); ...
26          coord(elements(m,3),:)-coord(elements(m,1),:)]';
27
28      P = ones(2)/6; P(1:3:4) = 2*P(1:3:4);
29
30      A2a(3*(m-1)+j(j~=p),3*(m-1)+j(j~=p)) = ...
```

```
31          A2a(3*(m-1)+j(j~=p),3*(m-1)+j(j~=p)) + delta*e*P;
32
33      P = ones(1,3)/2; P(p) = 0;
34      Q = gphi()'*inv(Jm)*rotmat*Jm*standedges(:,p)*P*D;
35
36      A3a(3*(m-1)+(1:3),3*(m-1)+(1:3)) = ...
37          A3a(3*(m-1)+(1:3),3*(m-1)+(1:3)) + Q;
38      A4a(3*(m-1)+(1:3),3*(m-1)+(1:3)) = ...
39          A4a(3*(m-1)+(1:3),3*(m-1)+(1:3)) + Q';
40  end
41
42  for l = 1:size(interior,1)
43      e = norm(coord(interior(l,1),:)-...
44          coord(interior(l,2),:))^(1-beta);
45      m = interior(l,3); p = interior(l,4);
46      n = interior(l,5); q = interior(l,6);
47
48      Jm = [coord(elements(m,2),:)-coord(elements(m,1),:);...
49          coord(elements(m,3),:)-coord(elements(m,1),:)]';
50      Jn = [coord(elements(n,2),:)-coord(elements(n,1),:);...
51          coord(elements(n,3),:)-coord(elements(n,1),:)]';
52
53      P = ones(2)/6; P(1:3:4) = 2*P(1:3:4);
54
55      A2a(3*(m-1)+j(j~=p),3*(m-1)+j(j~=p)) = ...
56          A2a(3*(m-1)+j(j~=p),3*(m-1)+j(j~=p)) + delta*e*P;
57      A2a(3*(n-1)+j(j~=q),3*(n-1)+j(j~=q)) = ...
58          A2a(3*(n-1)+j(j~=q),3*(n-1)+j(j~=q)) + delta*e*P;
59
60      P = ones(1,3)/2; P(p) = 0;
61      Q = gphi()'*inv(Jm)*rotmat*Jm*standedges(:,p)*P*D/2;
62
63      A3a(3*(m-1)+j,3*(m-1)+j) = ...
64          A3a(3*(m-1)+j,3*(m-1)+j) + Q;
65      A4a(3*(m-1)+j,3*(m-1)+j) = ...
66          A4a(3*(m-1)+j,3*(m-1)+j) + Q';
67
68      P = ones(1,3)/2; P(q) = 0;
69      Q = gphi()'*inv(Jn)*rotmat*Jn*standedges(:,q)*P*D/2;
70
71      A3a(3*(n-1)+j,3*(n-1)+j) = ...
72          A3a(3*(n-1)+j,3*(n-1)+j) + Q;
73      A4a(3*(n-1)+j,3*(n-1)+j) = ...
74          A4a(3*(n-1)+j,3*(n-1)+j) + Q';
75
```

```
76      P = zeros(3,3); P(j(j~=p),j(j~=q)) = ones(2)/6;
77      cnodes = intersect(elements(m,:),elements(n,:));
78      i = [find(elements(m,:)==cnodes(1)), ...
79          find(elements(m,:)==cnodes(2))];
80      k = [find(elements(n,:)==cnodes(1)), ...
81          find(elements(n,:)==cnodes(2))];
82      P(i(1),k(1)) = 2*P(i(1),k(1));
83      P(i(2),k(2)) = 2*P(i(2),k(2));
84
85      A2b(3*(m-1)+j(j~=p),3*(n-1)+j(j~=q)) = ...
86          A2b(3*(m-1)+j(j~=p),3*(n-1)+j(j~=q)) + ...
87          delta*e*P(j(j~=p),j(j~=q));
88      A2b(3*(n-1)+j(j~=q),3*(m-1)+j(j~=p)) = ...
89          A2b(3*(n-1)+j(j~=q),3*(m-1)+j(j~=p)) + ...
90          delta*e*P(j(j~=p),j(j~=q));
91
92      P = ones(1,3)/2; P(q) = 0;
93      Q = gphi()'*inv(Jm)*rotmat*Jm*standedges(:,p)*P*D/2;
94
95      A3b(3*(m-1)+j,3*(n-1)+j) = ...
96          A3b(3*(m-1)+j,3*(n-1)+j) + Q;
97      A4b(3*(n-1)+j,3*(m-1)+j) = ...
98          A4b(3*(n-1)+j,3*(m-1)+j) + Q';
99
100     P = ones(1,3)/2; P(p) = 0;
101     Q = gphi()'*inv(Jn)*rotmat*Jn*standedges(:,q)*P*D/2;
102
103     A3b(3*(n-1)+(1:3),3*(m-1)+(1:3)) = ...
104         A3b(3*(n-1)+(1:3),3*(m-1)+(1:3)) + Q;
105     A4b(3*(m-1)+(1:3),3*(n-1)+(1:3)) = ...
106         A4b(3*(m-1)+(1:3),3*(n-1)+(1:3)) + Q';
107 end
108
109 B = A1 + A3a - kappa*A4a + A2a;
110 C = A3b - kappa*A4b + A2b;
111 A = B' - C';
```

A.2 Numerical Quadrature

```
1  function a = myquad(fun,a,b)
2
3  quadpoints = [(5-sqrt(15))/10; 1/2; (5+sqrt(15))/10];
4  quadweights = [5/18; 4/9; 5/18];
5  A = 0;
6  for j = 1:3
```

```
7       A = A + quadweights(j)*quadpoints(j)*...
8           fun(a+quadpoints(j)*(b-a));
9   end
10  a = A*norm(a-b);
```

```
1   function a = mydblquad(fun,coord)
2
3   cubpoints = [(6-sqrt(15))/21, (6-sqrt(15))/21;...
4       (9+2*sqrt(15))/21, (6-sqrt(15))/21;...
5       (6-sqrt(15))/21, (9+2*sqrt(15))/21;...
6       (6+sqrt(15))/21, (6+sqrt(15))/21;...
7       (9-2*sqrt(15))/21, (6+sqrt(15))/21;...
8       (6+sqrt(15))/21, (9-2*sqrt(15))/21;...
9       1/3, 1/3];
10  cubweights = [(155-sqrt(15))/2400; (155-sqrt(15))/2400;...
11      (155-sqrt(15))/2400; (155+sqrt(15))/2400;...
12      (155+sqrt(15))/2400; (155+sqrt(15))/2400;...
13      9/80];
14  A = 0;
15  for j = 1:7
16      A = A + cubweights(j)*fun(map(coord',cubpoints(j,:)'));
17  end
18  A = A*det([1,1,1;coord']);
19  a = A;
```

Appendix B

A Linear Diffusion Problem

B.1 The Matrix C

```
 1  function A = matC(coord,elements,dirichlet,interior)
 2
 3  rotmat = [0, -1; 1, 0]; standedges = [-1, 0, 1; 1, -1, 0];
 4  s = size(elements,1); j = 1:3;
 5  A1 = sparse(3*s,3*s);
 6  A3a = sparse(3*s,3*s); A3b = sparse(3*s,3*s);
 7
 8  for m = 1:s
 9      Jm = [coord(elements(m,2),:)-coord(elements(m,1),:); ...
10          coord(elements(m,3),:)-coord(elements(m,1),:)]';
11      B = inv(Jm'*Jm);
12      A1(3*(m-1)+j,3*(m-1)+j) = A1(3*(m-1)+j,3*(m-1)+j) + ...
13          gphi()'*B*gphi()*det(Jm)/2;
14  end
15
16  for l = 1:size(dirichlet,1)
17      m = dirichlet(l,3); p = dirichlet(l,4);
18      Jm = [coord(elements(m,2),:)-coord(elements(m,1),:); ...
19          coord(elements(m,3),:)-coord(elements(m,1),:)]';
20      P = ones(1,3)/2; P(p) = 0;
21      A3a(3*(m-1)+(1:3),3*(m-1)+(1:3)) = ...
22          A3a(3*(m-1)+(1:3),3*(m-1)+(1:3)) + ...
23          gphi()'*inv(Jm)*rotmat*Jm*standedges(:,p)*P;
24  end
25
26  for l = 1:size(interior,1)
27      m = interior(l,3); p = interior(l,4);
28      n = interior(l,5); q = interior(l,6);
29      Jm = [coord(elements(m,2),:)-coord(elements(m,1),:); ...
30          coord(elements(m,3),:)-coord(elements(m,1),:)]';
```

```
31      Jn = [coord(elements(n,2),:)-coord(elements(n,1),:); ...
32          coord(elements(n,3),:)-coord(elements(n,1),:)]';
33
34      P = ones(1,3)/2; P(p) = 0;
35      A3a(3*(m-1)+j,3*(m-1)+j) = ...
36          A3a(3*(m-1)+j,3*(m-1)+j) + ...
37          gphi()'*inv(Jm)*rotmat*Jm*standedges(:,p)*P/2;
38      A3b(3*(n-1)+(1:3),3*(m-1)+(1:3)) = ...
39          A3b(3*(n-1)+(1:3),3*(m-1)+(1:3)) + ...
40          gphi()'*inv(Jn)*rotmat*Jn*standedges(:,q)*P/2;
41
42      P = ones(1,3)/2; P(q) = 0;
43      A3a(3*(n-1)+j,3*(n-1)+j) = ...
44          A3a(3*(n-1)+j,3*(n-1)+j) + ...
45          gphi()'*inv(Jn)*rotmat*Jn*standedges(:,q)*P/2;
46      A3b(3*(m-1)+j,3*(n-1)+j) = ...
47          A3b(3*(m-1)+j,3*(n-1)+j) + ...
48          gphi()'*inv(Jm)*rotmat*Jm*standedges(:,p)*P/2;
49  end
50
51  A = A1' + A3a' - A3b';
```

B.2 The Matrix E

```
 1  function A = matE(coord,elements,dirichlet,interior)
 2
 3  rotmat = [0, -1; 1, 0]; standedges = [-1, 0, 1; 1, -1, 0];
 4  s = size(elements,1);
 5  A1 = sparse(3*s,2*s);
 6  A3a = sparse(3*s,2*s); A3b = sparse(3*s,2*s);
 7
 8  for m = 1:size(elements,1)
 9      Jm = [coord(elements(m,2),:)-coord(elements(m,1),:); ...
10          coord(elements(m,3),:)-coord(elements(m,1),:)]';
11      A1(3*(m-1)+(1:3),2*(m-1)+(1:2)) = ...
12          A1(3*(m-1)+(1:3),2*(m-1)+(1:2)) + ...
13            gphi()'*inv(Jm)*det(Jm)/2;
14  end
15
16  for l = 1:size(dirichlet,1)
17      m = dirichlet(l,3); p = dirichlet(l,4);
18      Jm = [coord(elements(m,2),:)-coord(elements(m,1),:); ...
19          coord(elements(m,3),:)-coord(elements(m,1),:)]';
20      P = ones(1,3)/2; P(p) = 0;
21      A3a(3*(m-1)+(1:3),2*(m-1)+(1:2)) = ...
```

```
22          A3a(3*(m-1)+(1:3),2*(m-1)+(1:2)) + ...
23          (rotmat*Jm*standedges(:,p)*P)';
24  end
25
26  for l = 1:size(interior,1)
27      m = interior(l,3); p = interior(l,4);
28      n = interior(l,5); q = interior(l,6);
29      Jm = [coord(elements(m,2),:)-coord(elements(m,1),:); ...
30          coord(elements(m,3),:)-coord(elements(m,1),:)]';
31      Jn = [coord(elements(n,2),:)-coord(elements(n,1),:); ...
32          coord(elements(n,3),:)-coord(elements(n,1),:)]';
33
34      P = ones(1,3)/2; P(p) = 0;
35      A3a(3*(m-1)+(1:3),2*(m-1)+(1:2)) = ...
36          A3a(3*(m-1)+(1:3),2*(m-1)+(1:2)) + ...
37          (rotmat*Jm*standedges(:,p)*P)'/2;
38      A3b(3*(m-1)+(1:3),2*(n-1)+(1:2)) = ...
39          A3b(3*(m-1)+(1:3),2*(n-1)+(1:2)) + ...
40        (rotmat*Jn*standedges(:,q)*P)'/2;
41
42      P = ones(1,3)/2; P(q) = 0;
43      A3a(3*(n-1)+(1:3),2*(n-1)+(1:2)) = ...
44          A3a(3*(n-1)+(1:3),2*(n-1)+(1:2)) + ...
45          (rotmat*Jn*standedges(:,q)*P)'/2;
46      A3b(3*(n-1)+(1:3),2*(m-1)+(1:2)) = ...
47          A3b(3*(n-1)+(1:3),2*(m-1)+(1:2)) + ...
48          (rotmat*Jm*standedges(:,p)*P)'/2;
49  end
50
51  A = A1 + A3a - A3b;
```

B.3 The Vector $\boldsymbol{H}^i$

```
 1  function f = conth_a(dt,i,coord,elements,dirichlet,interior)
 2
 3  rotmat = [0, -1; 1, 0]; standedges = [-1, 0, 1; 1, -1, 0];
 4  g = zeros(3*size(elements,1),1);
 5
 6  for m = 1:size(elements,1)
 7      invJm = inv(...
 8          [coord(elements(m,2),:)-coord(elements(m,1),:); ...
 9          coord(elements(m,3),:)-coord(elements(m,1),:)]');
10      g(3*(m-1)+(1:3)) = g(3*(m-1)+(1:3)) + gphi()'*invJm*...
11          mydblquad(@(x)(h(dt*(i-1),x)+h(dt*(i-2),x)), ...
12          coord(elements(m,:),:));
```

```
13  end
14
15  for l = 1:size(dirichlet,1)
16      m = dirichlet(l,3); p = dirichlet(l,4);
17      Jm = [coord(elements(m,2),:)-coord(elements(m,1),:); ...
18          coord(elements(m,3),:)-coord(elements(m,1),:)]';
19      if p == 2, j = 3:-1:1; else j = 1:3; end
20      P1 = myquad(@(x) (h(dt*(i-1),x)+h(dt*(i-2),x))'*...
21          rotmat*Jm*standedges(:,p), coord(dirichlet(l,2),:), ...
22          coord(dirichlet(l,1),:));
23      P2 = myquad(@(x) (h(dt*(i-1),x)+h(dt*(i-2),x))'*...
24          rotmat*Jm*standedges(:,p), coord(dirichlet(l,1),:), ...
25          coord(dirichlet(l,2),:));
26      g(3*(m-1)+j(j~=p)) = g(3*(m-1)+j(j~=p)) + ...
27          [P1 P2]'/norm(coord(dirichlet(l,1),:)-...
28          coord(dirichlet(l,2),:));
29  end
30
31  for l = 1:size(interior,1)
32      m = interior(l,3); p = interior(l,4);
33      n = interior(l,5); q = interior(l,6);
34      Jm = [coord(elements(m,2),:)-coord(elements(m,1),:); ...
35          coord(elements(m,3),:)-coord(elements(m,1),:)]';
36      Jn = [coord(elements(n,2),:)-coord(elements(n,1),:); ...
37          coord(elements(n,3),:)-coord(elements(n,1),:)]';
38
39      if p == 2, j = 3:-1:1; else j = 1:3; end
40      P1 = myquad(@(x) (h(dt*(i-1),x)+h(dt*(i-2),x))'*...
41          rotmat*Jm*standedges(:,p), coord(interior(l,2),:), ...
42          coord(interior(l,1),:));
43      P2 = myquad(@(x) (h(dt*(i-1),x)+h(dt*(i-2),x))'*...
44          rotmat*Jm*standedges(:,p), coord(interior(l,1),:), ...
45          coord(interior(l,2),:));
46      g(3*(m-1)+j(j~=p)) = g(3*(m-1)+j(j~=p)) + [P1 P2]'/...
47          norm(coord(interior(l,1),:)-coord(interior(l,2),:));
48
49      if q == 2, j = 3:-1:1; else j = 1:3; end
50      P1 = myquad(@(x) (h(dt*(i-1),x)+h(dt*(i-2),x))'*...
51          rotmat*Jn*standedges(:,q), coord(interior(l,1),:), ...
52          coord(interior(l,2),:));
53      P2 = myquad(@(x) (h(dt*(i-1),x)+h(dt*(i-2),x))'*...
54          rotmat*Jn*standedges(:,q), coord(interior(l,2),:), ...
55          coord(interior(l,1),:));
56      g(3*(n-1)+j(j~=q)) = g(3*(n-1)+j(j~=q)) + [P1 P2]'/...
57          norm(coord(interior(l,1),:)-coord(interior(l,2),:));
```

```
58  end
59
60  f = g;
```

Appendix C

The Nonlinear Diffusion Problem

C.1 The Vector $\boldsymbol{H}^i$

```
 1  function f = conth_a(dt,i,U,gr,gg,urg,d,...
 2      coord,elements,dirichlet,interior)
 3
 4  rotmat = [0, -1; 1, 0]; standedges = [-1, 0, 1; 1, -1, 0];
 5  g = zeros(3*size(elements,1),1);
 6
 7  for m = 1:size(elements,1)
 8      invJm = inv(...
 9          [coord(elements(m,2),:)-coord(elements(m,1),:); ...
10          coord(elements(m,3),:)-coord(elements(m,1),:)]');
11      gmm = gammaStar(U(3*(m-1)+(1:3)),gr,gg,urg,d);
12      g(3*(m-1)+(1:3)) = gphi()'*invJm*...
13          mydblquad(@(x) h(dt,i,x,U(3*(m-1)+(1:3)),...
14          gr,gg,urg,d,coord(elements(m,:),:)), ...
15          coord(elements(m,:),:))/(2+dt*gmm);
16  end
17
18  for l = 1:size(dirichlet,1)
19      m = dirichlet(l,3); p = dirichlet(l,4);
20      Jm = [coord(elements(m,2),:)-coord(elements(m,1),:); ...
21          coord(elements(m,3),:)-coord(elements(m,1),:)]';
22      if p == 2, j = 3:-1:1; else j = 1:3; end
23      gmm = gammaStar(U(3*(m-1)+(1:3)),gr,gg,urg,d);
24      P1 = myquad(@(x) h(dt,i,x',U(3*(m-1)+(1:3)),...
25          gr,gg,urg,d,coord(elements(m,:),:))'*rotmat*...
26          Jm*standedges(:,p), coord(dirichlet(l,2),:), ...
27          coord(dirichlet(l,1),:));
28      P2 = myquad(@(x) h(dt,i,x',U(3*(m-1)+(1:3)),...
29          gr,gg,urg,d,coord(elements(m,:),:))'*rotmat*...
30          Jm*standedges(:,p), coord(dirichlet(l,1),:), ...
```

```
31          coord(dirichlet(l,2),:));
32      g(3*(m-1)+j(j~=p)) = g(3*(m-1)+j(j~=p)) + ...
33          [P1 P2]'/norm(coord(dirichlet(l,1),:)-...
34          coord(dirichlet(l,2),:))/(2+dt*gmm);
35  end
36
37  for l = 1:size(interior,1)
38      m = interior(l,3); p = interior(l,4);
39      n = interior(l,5); q = interior(l,6);
40      Jm = [coord(elements(m,2),:)-coord(elements(m,1),:); ...
41          coord(elements(m,3),:)-coord(elements(m,1),:)]';
42      Jn = [coord(elements(n,2),:)-coord(elements(n,1),:); ...
43          coord(elements(n,3),:)-coord(elements(n,1),:)]';
44      gmm1 = gammaStar(U(3*(m-1)+(1:3)),gr,gg,urg,d);
45      gmm2 = gammaStar(U(3*(n-1)+(1:3)),gr,gg,urg,d);
46
47      if p == 2, j = 3:-1:1; else j = 1:3; end
48      P1 = myquad(@(x) h(dt,i,x',U(3*(m-1)+(1:3)),...
49          gr,gg,urg,d,coord(elements(m,:),:))'*rotmat*...
50          Jm*standedges(:,p), coord(interior(l,2),:), ...
51          coord(interior(l,1),:));
52      P2 = myquad(@(x) h(dt,i,x',U(3*(m-1)+(1:3)),...
53          gr,gg,urg,d,coord(elements(m,:),:))'*rotmat*...
54          Jm*standedges(:,p), coord(interior(l,1),:), ...
55          coord(interior(l,2),:));
56      g(3*(m-1)+j(j~=p)) = g(3*(m-1)+j(j~=p)) + [P1 P2]'/...
57          norm(coord(interior(l,1),:)-coord(interior(l,2),:))*...
58          (1/(2+dt*gmm1)+1/(2+dt*gmm2))/2;
59
60      if q == 2, j = 3:-1:1; else j = 1:3; end
61      P1 = myquad(@(x) h(dt,i,x',U(3*(n-1)+(1:3)),...
62          gr,gg,urg,d,coord(elements(n,:),:))'*rotmat*...
63          Jn*standedges(:,q), coord(interior(l,1),:), ...
64          coord(interior(l,2),:));
65      P2 = myquad(@(x) h(dt,i,x',U(3*(n-1)+(1:3)),...
66          gr,gg,urg,d,coord(elements(n,:),:))'*rotmat*...
67          Jn*standedges(:,q), coord(interior(l,2),:), ...
68          coord(interior(l,1),:));
69      g(3*(n-1)+j(j~=q)) = g(3*(n-1)+j(j~=q)) + [P1 P2]'/...
70          norm(coord(interior(l,1),:)-coord(interior(l,2),:))*...
71          (1/(2+dt*gmm1)+1/(2+dt*gmm2))/2;
72  end
73
74  f = g;
```

Appendix D

How to Run the Code

For each problem there is a folder which contains all the files needed to solve the problem numerically. There are folders *poisson*, *heat*, *linear_diffusion* and *nonlinear_diffusion* to solve Poisson's equation, the heat equation, the linear and the nonlinear diffusion problem, respectively. Each folder contains a script which computes the numerical solution and the error. The script has the same name as the folder. To obtain a numerical solution (for a problem with known solution), first adapt the files `f.m`, `g.m`, `u.m`, `gradu.m`, `mesh.m` and, in the case of a diffusion problem, `s.m` and `h.m` (or `h1.m` and `h2.m`). Then, choose values for the mesh size `M`, the number of time steps `N` and all other parameters and run the script *foldername*. To run the script, make the folder in question your current path and either type the name of the script at the MATLAB prompt or open the script in the MATLAB editor and press the run button (alternatively, press F5). After the script has been executed, the variable `ee`, giving the error, and the matrices `U` and, in the case of a diffusion problem, `S`, giving the numerical solution, can be found in the workspace.

For producing tables similar to those given in this thesis, the folder *poisson* contains the script *main* and all other folders contain the scripts *mainSpace* and *mainTime*. The script *mainSpace* varies the value of `M` at a fixed time step while the script *mainTime* varies the value of `N` at a fixed mesh size. To obtain a table, disable the allocations of `M` and `N` in the script *foldername* and run *main*/*mainSpace*/*mainTime*.

Each folder also contains a script called *graphics*, which produces plots of the numerical solution. To plot the numerical solution first run the script *foldername* and then run the script *graphics*.